Sheffield Hallam University

Cyber Resilience and Privacy in Healthcare Information Systems Using Machine Learning

AMEH, Jude <http://orcid.org/0009-0001-4523-5204>

Available from the Sheffield Hallam University Research Archive (SHURA) at:

https://shura.shu.ac.uk/38057/

A Sheffield Hallam University thesis

This thesis is protected by copyright which belongs to the author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Please visit https://shura.shu.ac.uk/38057/ and http://shura.shu.ac.uk/information.html for further details about copyright and re-use permissions.

Cyber Resilience and Privacy in Healthcare Information Systems Using Machine Learning

Jude Enenche Ameh

A thesis submitted in partial fulfilment of the requirements of Sheffield Hallam University for the Degree of Doctor of Philosophy

June 2026

Declaration

I hereby declare that:

1. I have not been enrolled for another award of the University, or other academic or professional organisation, whilst undertaking my research degree.
2. None of the material contained in the thesis has been used in any other submission for an academic award.
3. I certify that this thesis is my own work. The use of all published or other sources of material consulted have been properly and fully acknowledged. I confirm that I have sought and obtained copyright permission for any third-party materials included in this thesis.
   This work was produced with AI assistance at Level 2 (AI for Shaping) of the Artificial Intelligence Transparency Scale (AITS). ChatGPT was used solely to support initial brainstorming and to suggest a structural outline during the early planning stage. No AI tool was used to generate, draft, or edit any part of the written content.
   All arguments were independently developed, all research was conducted, and the full text was written by me.
4. The work undertaken towards the thesis has been conducted in accordance with the SHU Principles of Integrity in Research and the SHU Research Ethics Policy, and ethics approval has been granted for all research studies in the thesis, as shown in the table below.

Ethics review reference number	**Title of research study**	**Approval date**	**Date of any post-approval amendments (if applicable)**
ER82757456	Cyber Resilience and Privacy of Healthcare Information Systems During Software Development	18/09/2025	N/A
ER54973679	Cyber Resilience and Privacy of Healthcare Information Systems During Software Development	22/05/2023	N/A

5. The word count of the thesis is 79,756.

Name	Jude Enenche Ameh
Date of Submission	June 2026
Award	Doctor of Philosophy
College	Business Technology and Engineering
Director of Studies	Abayomi Otebolaku

Abstract

Healthcare software systems occupy a uniquely sensitive intersection of patient safety, personal data protection, and regulatory compliance, making the security of their code a matter of technical and ethical urgency. Although privacy engineering and automated vulnerability detection have progressed over the years, both have mostly developed separately. Privacy frameworks address concerns at the architectural and design level, while vulnerability detection tools focus on code, lacking a systematic way to connect the two. This separation creates gaps in secure software development, as privacy threats found during system design cannot be linked to specific weaknesses in code. Additionally, vulnerability detection tools are not tailored or tested for healthcare software, meaning development teams lack integrated, domain-specific tools that their sensitive systems require.

This thesis addresses that gap through five interconnected contributions. First, the LINDDUN privacy threat modelling methodology is systematically extended by constructing a mapping between all seven LINDDUN threat categories (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of Information, Unawareness, and Non-compliance) and their corresponding Common Weakness Enumeration (CWE) entries and thus operationalising the translation from design-time privacy threats to code-level weaknesses. The mapping was validated through structured expert assessment, achieving substantial inter-rater agreement (Cohen's $\kappa = 0.78$), establishing it as a reliable analytical instrument. Second, a large-scale, healthcare-specific vulnerability dataset is constructed comprising 6,308,314 annotated C/C++ functions drawn from DiverseVul, ReposVul, and StarCoder, spanning 773 distinct CWE identifiers and enriched with LINDDUN privacy threat labels. This makes it the first dataset to integrate privacy threat taxonomy with software vulnerability classification at this scale and domain specificity. Third, a comprehensive comparative evaluation of 14 machine learning models across three paradigms (traditional ML, graph neural networks (GNNs), and transformer-based models) was conducted on this dataset. Transformer models achieve the highest detection performance, with CodeGPT-small recording an F1 score of 0.9976 and a false positive rate of 0.09%, and DistilBERT achieving the highest recall of 0.9985. Among GNN approaches, Reveal attains an F1 of 0.9958 with an FPR of 0.15%, while Devign yields the highest GNN recall at 0.9977. Among traditional methods, XGBoost delivers an F1 of 0.9926

with an FPR of 0.40%, demonstrating that computationally efficient classical models remain highly competitive and represent a viable alternative where GPU infrastructure is unavailable. Multi-class CWE classification proves substantially more challenging across all paradigms, with Random Forest achieving the best macro-F1 of 0.676, confirming that fine-grained vulnerability typing remains an open problem. Fourth, a novel hybrid detection architecture is developed, combining deterministic template matching for known vulnerability patterns with a machine learning fallback for novel pattern detection and safety rule overrides that provide hard false-negative guarantees unattainable by purely statistical approaches. Fifth, this architecture is realised as a validated, deployment-feasible prototype. Deployed as a real-time REST API with CPU-only inference feasibility, the system achieves an overall accuracy of 99.11% and delivers CWE-specific remediation guidance alongside each detection decision and is validated through user acceptance testing with industry practitioners and academic researchers.

This work presents the first end-to-end, traceable framework that connects privacy threat identification during design with automated vulnerability detection in code, tailored specifically for healthcare. By using healthcare-specific CWE profiles and LINDDUN threat categories, it offers targeted tools for healthcare software teams that support security needs in regulated environments. Additionally, the results show that traditional ML and hybrid models can perform competitively with transformers, suggesting resource-efficient alternatives for effective vulnerability detection in healthcare and similar contexts.

Dedication

This research work is dedicated to the Almighty God who gave me strength, resilience and the opportunity to go through this process and saw me finish it.

To my amazing wife, Olubukola, the wind beneath my wings. Your passionate advocacy and support that always assured me that it is possible and it will come to pass.

My 3 amazing kids, being your father and doing research was an experience that I will always cherish. I appreciate you for keeping me accountable always.

My mother who continues to pray for me every day by name, my siblings who have supported me both in cash and kind, I say thank you ever so much.

This is a dedication to you the summary of my scientific effort.

Jude

Acknowledgments

This thesis would not have been possible without the support, guidance, and generosity of several individuals and institutions to whom I owe a profound debt of gratitude.

First and foremost, I am deeply grateful to my Director of Studies, Dr Abayomi Otebolaku, whose intellectual rigour, patient mentorship, and unwavering commitment to my development as a researcher have been the constant anchor of this doctoral journey. His ability to ask the right question at the right moment, and his willingness to engage seriously with every stage of this work, has shaped both the thesis and the researcher I have become. I am equally grateful to my supervisory team for their constructive challenge, breadth of expertise, and support throughout the research process.

I wish to express my sincere appreciation to the members of my supervisory team, Professor Alex Shenfield and Dr. Augustine Ikpehai who gave me advice and feedback that helped shape this research effort. Special appreciations also go out to Dr Da Silva Carlos and Dr Da Costa Abreu Marjory who in no small way gave me moral and mental support to push through even in very difficult times. To the school of Computing and Digital Technologies for providing the institutional environment, resources, and community within which this research was conducted. The support of colleagues and fellow researchers within the department has been a source of both intellectual stimulation and personal encouragement throughout.

I am grateful to the seven participants who gave their time to take part in the user acceptance testing evaluation. Their professional insights and candid feedback were invaluable in validating the practical utility of the prototype system and in grounding the research contributions in real-world relevance.

My gratitude also extends to the academic and industry professionals who participated in the expert validation study for the LINDDUN-to-CWE mapping. Their rigorous independent review strengthened the credibility and reliability of one of the central contributions of this thesis.

To my mother, my siblings and my wider family, thank you for your love, your prayers, and your enduring belief in me through every season of this journey. The sacrifices you have made and the encouragement you have given, often from a distance, have meant more than words can adequately express.

Finally, I am grateful to the community of S2 Connect, whose fellowship, prayers, and consistent support have sustained me in ways that extend well beyond the academic. Faith and friendship have been the foundation beneath all this work, and I do not take either for granted.

Jude Enenche Ameh Sheffield, 2026

Table of Contents

Declaration ii
Abstract iii
Dedication v
Acknowledgments vi
List of Publications xi
List of Tables xii
List of Figures xiv
List of Abbreviations xvi
1 Chapter 1: Introduction 1
1.1 Motivation 5
1.2 The Significance of this Research 5
1.3 Problem Statement 8
1.3.1 Research Hypotheses 11
1.4 Aim and Objectives 12
1.5 Contributions 13
1.6 Thesis Scope 16
1.7 Thesis Structure 16
2 Chapter 2: Literature Review 19
2.1 Background 19
2.2 An Overview of Privacy and Security for Information Systems in Healthcare 20
2.3 Secured Software Development Lifecycle 24
2.4 Secure Software Development Standards, Guidelines, Best Practices and Procedures 33
2.4.1 International Standards for Secure Development 33
2.4.2 National and Sector-Specific Security Frameworks 37
2.4.3 Secure Coding Standards and Practices 41
2.4.4 DevSecOps and Continuous Integration and Continuous Delivery/Deployment (CI/CD) Integration 44
2.5 Privacy Engineering and Privacy-by-Design Methods 46
2.5.1 LINDDUN Privacy Threat Modelling 46
2.5.2 Other Threat Modelling Frameworks 49
2.5.3 Framework Selection Rationale for Healthcare Contexts 51
2.5.4 Software Assurance Maturity Models (SAMM and BSIMM) 54
2.5.5 Alignment with Security-by-Design and Privacy-by-Design Principles 57
2.6 Healthcare Information Systems: Architecture, Privacy, and Security 58
2.6.1 Privacy Implications of Healthcare Components 61
2.7 Application Security Testing 65
2.8 Machine Learning in Static Application Security Testing (SAST) 68
2.8.1 Rule-Based vs. ML-Based SAST Approaches 69
2.8.2 Evolution of ML Techniques Applied to SAST 70
2.8.3 ML Techniques for Code Vulnerability Detection 72
2.8.4 Challenges in the State-of-the-Art of ML in SAST 74
2.9 Security-Relevant Features and Vulnerabilities in C/C++ 77
2.9.1 C/C++ in Critical Systems and Legacy Software 78
2.9.2 Common Vulnerability Types in C/C++ Software 79
2.9.3 Impact of C/C++ Vulnerabilities and Datasets Reflecting Their Prevalence 80
2.9.4 Focusing NLP and SAST Methods on C/C++ in Healthcare Contexts 81
2.10 Common Weakness Enumeration (CWE) and Software Vulnerability Taxonomies 82
2.10.1 CWE Usage Across Domains and Common Weaknesses in Healthcare Software 83

2.10.2 Role of CWE in Secure Software Development and Vulnerability Detection 85
2.10.3 Relationship of CWE with CVE and CAPEC 86
2.10.4 Mapping Threats and Misuse Cases to CWE Types: Efforts and Challenges 88
2.10.5 From Privacy Threat to Privacy Harm: A Conceptual Model 90
2.11 NLP for Software Vulnerability Detection 92
2.11.1 Natural Language vs Programming Language Processing 93
2.11.2 Transformer-Based Code Models: CodeBERT, GraphCodeBERT, CodeT5 94
2.11.3 Key Challenges in Learning-Based Vulnerability Detection 96
2.11.4 Datasets and Generalizability in Software Security Tasks 98
2.12 Systematic Review Methodology (Ameh et al., 2025) 101
2.13 Summary of Identified Gaps 102
3 Chapter 3: Enhanced LINDDUN Privacy Threat Methodology for Healthcare Software Systems 108
3.1 Background 108
3.1.1 Privacy-by-Design and Traceable Privacy Risks 109
3.1.2 Language and Implementation Context 113
3.2 Methods 115
3.2.1 Context 115
3.2.2 Constructing the Domain Data Flow Diagram 119
3.2.3 Systematic Threat Elicitation per LINDDUN Class 121
3.2.4 Semantics of the LINDDUN-to-CWE Mapping 132
3.2.5 Systematic Mapping of LINDDUN Privacy Threats to Common Weakness Enumeration (CWE) Entries 133
3.2.6 Traceability and Reproducibility Artifacts 136
3.2.7 Expert Review and Validation Protocol 138
3.3 Outcomes of applying the enhanced LINDDUN-based methodology to the healthcare software system 141
3.3.1 Threat Coverage Across the DFD 142
3.3.2 Representative Mappings and Examples 143
3.3.3 Quantitative Validation Outcomes 146
3.3.4 Actionability and Integration Readiness 151
3.4 Discussion 152
3.4.1 Systematic Threat Elicitation Across LINDDUN Categories 153
3.4.2 LINDDUN to CWE Mapping Contributions 155
3.4.3 Expert Review and Validation Protocol Outcomes 156
3.4.4 Bridging Design and Implementation 157
3.4.5 Healthcare Dataset Creation for Machine Learning 158
3.4.6 Practical Implications for Healthcare Organizations 159
3.4.7 Methodological Limitations and Constraints 160
3.4.8 Contribution to Healthcare Cybersecurity 161
4 Chapter 4: Dataset Preparation and Classification 164
4.1 Introduction 164
4.1.1 Motivation for a Healthcare Cybersecurity Natural Language Processing Dataset 165
4.1.2 Gap in Healthcare-Relevant Code Datasets 166
4.1.3 Challenges in Code Vulnerability Classification 167
4.2 Dataset Construction 169
4.2.1 Source datasets and selection rationale 169
4.2.2 Integration into a healthcare privacy-oriented dataset 171
4.2.3 Privacy filtering and healthcare domain alignment 173

4.2.4 Synthetic augmentation and coverage of rare weaknesses 175
4.2.5 Resulting Dataset Characteristics and Justification 179
4.3 Dataset Preparation and Sampling Strategy 189
4.4 Experimental Setup 189
4.4.1 Hardware Configuration 190
4.4.2 Software Environment 190
4.4.3 Training Configuration Summary 191
4.5 Dataset Evaluation 191
4.5.1 Traditional Machine Learning Approaches for Healthcare Code Vulnerability Detection 191
4.5.2 Graph Neural Network Approaches for Vulnerability Detection 201
4.5.3 Transformer-Based Approaches for Vulnerability Detection 205
4.6 Evaluation Results 213
4.6.1 Evaluation Results for Traditional Machine Learning 213
4.6.2 Evaluation Results for Graph Neural Network 220
4.6.3 Transformer-Based Model Results 225
4.6.4 Cross-paradigm consistency 232
4.7 Discussion of Evaluation Results 233
4.7.1 Comparative Performance Analysis 233
4.7.2 Representation Learning Across Paradigms 234
4.7.3 Computational Trade-offs and Practical Considerations 236
4.7.4 Critical Observations and Implications for Healthcare Software Security 238
4.7.5 Model Selection for Prototype Development 240
4.7.6 Statistical versus Practical Significance 241
5 Chapter 5: From Evaluation to Deployment: Design and Implementation of a Real-Time Vulnerability Detection Prototype for Healthcare Software Systems 243
5.1 Introduction 243
5.2 System Architecture 249
5.2.1 Technology Stack 251
5.2.2 Request Processing Pipeline 253
5.3 Multi-Modal Feature Extraction 254
5.3.1 Semantic Features via CodeT5 Embeddings 254
5.3.2 Structural Features via Abstract Syntax Tree Analysis 255
5.3.3 Statistical Features via Pattern Counting 256
5.3.4 Feature Fusion and Dimensionality Reduction 256
5.3.5 Healthcare Domain Adaptation 256
5.4 Hybrid Classification Architecture 257
5.4.1 Template Matching Phase 259
5.4.2 Machine Learning Fallback 259
5.5 Safety Rule System 260
5.5.1 Dangerous Pattern Detection 260
5.5.2 Override Mechanism 261
5.6 API and Frontend Implementation 262
5.6.1 REST API Design 262
5.6.2 Frontend User Interface 263
5.6.3 Security and Privacy Analysis of the Prototype 264
5.7 End-to-End Integration Demonstration 266
5.7.1 Example 1 — Linkability Threat to CWE-200 Detection 267
5.7.2 Example 2 — Disclosure of Information Threat to CWE-319 Detection 269
5.8 User Acceptance Testing 271

5.8.1 Participant Recruitment 272
5.8.2 Evaluation Tasks 273
5.8.3 Evaluation Results 274
5.8.4 Qualitative Feedback 275
5.9 Limitations and Future Work 275
6 Chapter 6: Conclusion and Future Work 277
6.1 Introduction 277
6.2 Summary of Research 277
6.3 Achievement of Research Objectives 279
6.3.1 Objective 1: LINDDUN to CWE Mapping 279
6.3.2 Objective 2: Healthcare Cybersecurity Dataset 280
6.3.3 Objective 3: Reliable NLP Method for Static Code Security Testing 280
6.3.4 Objective 4: Vulnerability Identification Solution with Corrective Actions 281
6.4 Evaluation of Research Hypotheses 282
6.5 Summary of Key Findings 285
6.6 Research Contributions 289
6.7 Implications for Theory and Practice 291
6.7.1 Theoretical Implications 291
6.7.2 Practical Implications for Healthcare 292
6.8 Limitations 294
6.9 Recommendations 297
6.9.1 Recommendations for Healthcare Organisations 297
6.9.2 Recommendations for Security Practitioners 298
6.9.3 Recommendations for Researchers 298
6.10 Future Work 299
6.10.1 Short-Term Extensions 299
6.10.2 Medium-Term Developments 301
6.10.3 Long-Term Research Agenda 301
6.11 Concluding Remarks 301
7 References 304
8 Appendices 355
8.1 Appendix A: Threat Trees 355
8.2 Appendix B: Threat to CWE Mapping 358
8.3 Appendix C - Expert Evaluation Form and Review Instructions 361
8.4 Appendix D - Expert Evaluation Form and Review Instructions 361
8.5 Appendix E – Transformer Models Architecture Adapted for Binary Vulnerability Classification 361
8.6 Appendix F: Vulnerability detection system prototype high-level architecture 364
8.7 Appendix G: Participant Information Sheet and Consent Form 364
8.8 Appendix H: AI Vulnerability System User Evaluation Form 364
8.9 Appendix I: Synthetic Data Generation: Per-CWE Candidate Sample Counts 365

List of Publications

The following publications have been produced during this doctoral research. They represent the peer-reviewed dissemination of the principal contributions described in this thesis.

Published Works

1. Ameh, J. E., Otebolaku, A., Shenfield A. & Ikpehai A., C3-VULMAP: A Dataset for Privacy-Aware Vulnerability Detection in Healthcare Systems, *Electronics (MDPI),* Volume 14, Issue 13, Article 2703. Special Issue on Machine Learning for Cybersecurity. DOI: 10.3390/electronics14132703
2. Ameh, J. E., Otebolaku, A., Shenfield A. & Ikpehai A., Performance Analysis of Lightweight Transformer Models for Healthcare Application Privacy Threat Detection, *Lecture Notes in Computer Science, Volume 16231,* ESORICS 2025 International Workshops on Computer Security. Springer. DOI: 10.1007/978-3-032-16089-8_12

Manuscripts Under Review

3. Ameh, J. E., & Otebolaku, A., Shenfield A., Ikpehai A & Sule D., Machine Learning Approaches in Software Vulnerability Detection: A Systematic Review and Analysis of Contemporary Methods, *Artificial Intelligence Review (Springer),* Preprint available via Research Square. Manuscript under review at Artificial Intelligence Review., DOI: 10.21203/rs.3.rs-5975490/v1
4. Ameh, J. E., & Otebolaku, A., A Hybrid Machine Learning Architecture for Real-Time Vulnerability Detection: Combining Template Matching with Neural Networks, *ACM Transactions on Software Engineering and Methodology (TOSEM),* Submitted to the New Frontiers track.
5. Ameh, J. E., Otebolaku, A., Shenfield A. & Ikpehai A., Addressing the Deployment Gap: Hybrid Symbolic-Statistical Vulnerability Detection in Safety-Critical C/C++ Systems, *IEEE Empirical Software Engineering,* Preprint

List of Tables

Table 2-1: Comparative Summary of Threat Modelling Frameworks: Key Features, Strengths, and Weaknesses 51
Table 3-1: Showing the DFD elements and mapping the LINDDUN components to the possible points of compromise. 121
Table 3-2: Formal Notation and DFD Element Definitions for LINDDUN Threat Category Representations 131
Table 3-3: Distribution of Identified Privacy Threat Instances Across Healthcare Information System Components by LINDDUN Category 143
Table 3-4: Privacy & Security Threat Mapping: 965 Candidate CWEs Screened Across Seven LINDDUN Threat Categories 145
Table 4-1 - Summary of constituted datasets 171
Table 4-2: CWEs with Missing or Few Representations in the Pre-Augmentation Merged Dataset 177
Table 4-3: Top Twenty CWE Identifiers by Label Occurrence Count (spanning vulnerable and patched non-vulnerable functions) 178
Table 4-4: Core Dataset Schema Showing Harmonised Fields 180
Table 4-5: Label Distribution Across All 6.3 Million Functions 181
Table 4-6: Distribution of The Largest CWE Categories and Their Unique Subtypes 187
Table 4-7: Distribution of Mapped LINDDUN Privacy Threat Types 188
Table 4-8: Hardware Specification of the Experimental Platform 190
Table 4-9: Software Library Versions Used Across All Experimental Paradigms 191
Table 4-10: Rare versus Common CWE Performance Gap Across Traditional ML Models, Measured as F1-Score 200
Table 4-11: Binary Classification Cross-Validation Performance (5-Fold) 214
Table 4-12: Binary Classification Hold-out Test Set Performance 214
Table 4-13: Multi-class CWE Classification Cross-Validation Performance 220
Table 4-14: GNN Model Test Set Performance 221
Table 4-15: McNemar's Test Contingency Table Comparing Devign and Reveal Predictions on the Shared Fixed Test Set 221
Table 4-16: Rare versus Common CWE Performance Gap, GNN Models 223
Table 4-17: GNN Training Summary 225
Table 4-18: Transformer Model Test Set Performance 226
Table 4-19: Transformer Training Summary 228
Table 4-20: Transformer Model Error Analysis 230
Table 4-21: Rare versus Common CWE Performance Gap, Transformer Models 231
Table 4-22: Consolidated Cross-Paradigm Performance Comparison 234
Table 4-23: Computational Requirements Comparison 236
Table 5-1: Summary of the Fourteen Iterative Development Phases 244
Table 5-2: Classification Performance Metrics of the Hybrid Vulnerability Detection System 248
Table 5-3: Technology Stack Components used in the prototype 252
Table 5-4: Request Processing Pipeline Timing 253
Table 5-5: Classification Architecture Approaches Evaluated 258

Table 5-6: Safety Rule Pattern Mappings 260
Table 5-7: UAT Participant Demographics 272
Table 5-8: User Acceptance Testing Ratings (n=7) 274
Table 6-1: Research Objectives Achievement Summary 279
Table 6-2: Mapping of Literature Review Gaps to Research Contributions 286
Table 6-3: Summary of Key Research Findings 287
Table 8-1: Stage One: GPT-3.5 Turbo Generation for Critically Underrepresented CWEs 365
Table 8-2: Stage Two: GPT-3.5 Turbo Generation for CWE Category Balancing 367

List of Figures

Figure 1-1: The four dimensions of cyber resilience (Ross et al., 2021). 5
Figure 1-2: The core problem — privacy threat analysis (LINDDUN) and software vulnerability detection (CWE) operate as disconnected disciplines in healthcare software development. 7
Figure 1-3: Research Objectives: Enhanced LINDDUN Methodology 13
Figure 1-4: The five research contributions as a connected pipeline 14
Figure 2-1: The two disciplines governing healthcare software protection operate independently, creating a gap that leaves privacy risks unconnected to code-level vulnerabilities. 19
Figure 2-2: Evolution of Secured SDLC Frameworks 28
Figure 2-3: The four principal challenges in ML-based static application security testing. 74
Figure 2-4: Overview of Section 2.9: the case for applying NLP and SAST to C/C++ healthcare software. 78
Figure 2-5: Relationship between CAPEC, CWE and CVE 88
Figure 2-6: Conceptual model of the causal chain from a design-level LINDDUN privacy threat to actual privacy harm 91
Figure 2-7: PRISMA-style flow diagram of the study selection process for the systematic review underpinning the gap analysis in this section 102
Figure 3-1: Healthcare Information System (HIS) High-level Data Flow Diagram (DFD) 117
Figure 3-2 (a–c): Inter-rater agreement from the three-reviewer expert validation of the LINDDUN-to-CWE mappings. 147
Figure 3-3: Heatmap of Expert Acceptance Rates Across LINDDUN Threat Categories and CWE Groups 149
Figure 3-4: Expert Reviewer Acceptance Rate by LINDDUN Threat Category 150
Figure 3-5: Distribution of Accepted Mappings Across CWE Groups in the Expert-Reviewed LINDDUN-to-CWE Mapping Matrix 151
Figure 3-6: The Enhanced LINDDUN Privacy Threat Modelling Process for Healthcare Software Systems, Showing the Systematic Pathway from Data Flow Analysis to CWE Mapping (Stages i–iv) and its Role in Informing the C3-VULMAP Dataset Construction Pipeline (Stages v–vi, Detailed in Chapter 4) 162
Figure 4-1: The motivation for a healthcare-aligned vulnerability dataset 168
Figure 4-2: Dataset Integration Pipeline — Source Combination, Harmonisation, and Binary Labelling 173
Figure 4-3: Two-Stage Privacy Filtering and Healthcare Domain Alignment Pipeline 174
Figure 4-4: Class Imbalance Distribution in C3-VULMAP — Proportional Split (left) and Absolute Counts on Log Scale (right) 177
Figure 4-5: Calibration Curves (Reliability Diagrams) for Five Traditional ML Models Relative to the Perfectly Calibrated Baseline 199
Figure 4-6: Code Property Graph construction — C/C++ source function to multi-edge graph via Tree-sitter 203
Figure 4-7: Taxonomy of the seven transformer-based models evaluated for binary vulnerability detection across three architectural paradigms 207
Figure 4-8: Confusion Matrices (Using hold-out test set) 216

Figure 4-9: Per-fold F1 stability....216
Figure 4-10: ROC Curves 217
Figure 4-11: Precision-Recall Curves....218
Figure 4-12: Calibration Curves....219
Figure 4-13: Confusion Matrices for GNN Models 222
Figure 4-14: Training Curves - GNN Models 224
Figure 4-15: ROC Curves - GNN Models....225
Figure 4-16: Confusion Matrices - Transformer Models 229
Figure 4-17: Training Curves - Transformer Models Figure 4-18: Validation F1-Score - Transformer Models 231
Figure 5-1: Use Case Diagram for Vulnerability Detection Prototype System 250
Figure 5-2: Component diagram — internal structure, interfaces, and data store dependencies of the vulnerability detection prototype 251
Figure 5-3: Front-end web interface of developed prototype 264
Figure 8-1: BERT-base architecture Figure 8-2: DistilBERT architecture 361
Figure 8-3: DistilRoBERTa architecture Figure 8-4: CodeBERT bimodal architecture 362
Figure 8-5: GraphCodeBERT graph-augmented architecture Figure 8-6: CodeGPT-small causal decoder architecture 362
Figure 8-7: CodeT5-small encoder-decoder architecture....363
Figure 8-8: high-level architecture of the vulnerability detection prototype system 364

List of Abbreviations

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
AST	Abstract Syntax Tree
ASVS	Application Security Verification Standard (OWASP)
AUC-ROC	Area Under the Receiver Operating Characteristic Curve
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory Network
BSIMM	Building Security in Maturity Model
C3-VULMAP	C/C++ Vulnerability Mapping Dataset for Privacy-Aware Healthcare Systems
CAPEC	Common Attack Pattern Enumeration and Classification
CDN	Content Delivery Network
CI/CD	Continuous Integration / Continuous Deployment
CIA Triad	Confidentiality, Integrity, and Availability
CLASP	Comprehensive Lightweight Application Security Process
CNN	Convolutional Neural Network
CodeBERT	Code Bidirectional Encoder Representations from Transformers
CORS	Cross-Origin Resource Sharing
CTI	Cyber Threat Intelligence
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DAST	Dynamic Application Security Testing
DFD	Data Flow Diagram
DICOM	Digital Imaging and Communications in Medicine
DPIA	Data Protection Impact Assessment
DSP	Data Security and Protection (NHS)
EHR	Electronic Health Record
FHIR	Fast Healthcare Interoperability Resources
FPR	False Positive Rate
GDPR	General Data Protection Regulation
GNN	Graph Neural Network
GPT	Generative Pre-trained Transformer
HIPAA	Health Insurance Portability and Accountability Act
HIS	Healthcare Information System
HL7	Health Level Seven (interoperability standard)
HSE	Health Service Executive (Ireland)
IAST	Interactive Application Security Testing
IoMT	Internet of Medical Things
IoT	Internet of Things
LINDDUN	Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, Non-compliance

LIS	Laboratory Information System
LLM	Large Language Model
LSTM	Long Short-Term Memory Network
ML	Machine Learning
MITRE	MITRE Corporation (CVE/CWE maintainer)
NCSC	National Cyber Security Centre (UK)
NDG	National Data Guardian (UK)
NHS	National Health Service (UK)
NIST	National Institute of Standards and Technology (US)
NLP	Natural Language Processing
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
PACS	Picture Archiving and Communication System
PbD	Privacy by Design
PCA	Principal Component Analysis
PET	Privacy-Enhancing Technology
PHI	Protected Health Information
PR-AUC	Precision-Recall Area Under the Curve
RASP	Runtime Application Self-Protection
REST	Representational State Transfer
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimised BERT Pre-training Approach
SAMM	Software Assurance Maturity Model (OWASP)
SAST	Static Application Security Testing
SbD	Security by Design
SCA	Software Composition Analysis
SCAP	Security Content Automation Protocol
SDL	Security Development Lifecycle (Microsoft)
SDLC	Software Development Lifecycle
SHAP	SHapley Additive exPlanations
SMOTE	Synthetic Minority Over-sampling Technique
SSDF	Secure Software Development Framework (NIST SP 800-218)
SSDLC	Secure Software Development Lifecycle
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
TOSEM	ACM Transactions on Software Engineering and Methodology
UAT	User Acceptance Testing
XSS	Cross-Site Scripting

1 Chapter 1: Introduction

In 2021, the Health Service Executive (HSE) (National Health Service, NHS UK equivalent in Ireland) had its national and local information technology systems attacked by a gang who installed a ransomware called Conti, stole patient details, records of employees, financial information and demanded a ransom of €16.5m to prevent them from releasing the data to other threat actors. This breach, as with others, was because of a vulnerability in software that led to an exploitation of a back door (NHS Digital, 2023). Considering that the NHS has an enormous attack surface based on its many charities, services and points of contact (outside edge), it has a lot of devices running all kinds of software, and these are handled by a lot of people. The software applications that these devices run are either proprietary or third-party, which brings to question issues of privacy and security during their development. These situations make the NHS and indeed all healthcare providers globally a significant target for cyber criminals who are motivated to exploit private health data, demand money after encrypting data, denial of service or just an indiscriminate cyberattack like the WannaCry Ransom outbreak of 2017 (National Cyber Security Centre, 2018). Moreso, the ransomware attack on the HSE in Ireland brought down healthcare delivery operations and subsequently an estimated €100 million in remediation cost (McNally, 2021).

The last decade saw a concerning proliferation of vulnerabilities in software, and this is evidenced by the high number of publicly disclosed flaws. For instance, over 18,000 new vulnerabilities were documented in 2020 alone (Gueye et al., 2021), which reflects a sustained upward trend since the early 2010s (Srinivasan & Rothacker, 2022). Many of the reported high-profile cyber incidents have been linked to the exploitations of known vulnerabilities in widely used software (Shaon & Akter, 2025), with annual Common Vulnerabilities and Exposures (CVEs) rising from only about 6,400 in 2015 to over 40,000 in 2024, representing a more than 500% increase (MITRE Corporation, 2024; National Institute of Standards and Technology [NIST], 2025). The most recurrent threats include cross-site scripting, out-of-bounds write, and SQL injection, while code injection has shown the steepest climb (MITRE Corporation, 2024).

Healthcare cybersecurity failures carry a disproportionate financial burden relative to other industries. For the thirteenth consecutive year, the sector recorded the highest average breach cost of any industry globally, at approximately $10.93–11 million per incident in 2023 which is more than double the cross-industry average of $4.45 million (IBM Security, 2024; Alder, 2023). This figure reflects direct costs including detection, containment, notification, and post-breach response, but does not capture the broader economic impact of disrupted care delivery, regulatory penalties, and reputational damage that compound the financial toll of healthcare cybersecurity failures. Moreover, attackers are exploiting vulnerabilities much faster, with the median time to remediation occurring in 32 days, while exploitation of the weakness often occurs within just five days of disclosure or discovery (CISA, 2025; Verizon, 2025).

The healthcare industry particularly has faced an escalation in cyberattacks and software security failures in the last ten years. Seh et al. (2020) reported that healthcare industry data are increasingly seen as valuable, and their study concluded that deliberate external forms of cyberattacks, such as ransomware, denial of service, phishing and data theft, are the most frequently occurring. A situation of great concern because of the sensitive patient data that organisations in this sector hold and the critical services they operate, a quagmire that makes them prime targets for threat actors. Li et al. (2025) reported that there was a sharp increase in cyber threat incidents between 2016 and 2017, and in the last decade, the reported breaches in the healthcare sector have more than tripled. In the United States, the annual healthcare breach disclosures grew from 216 to over 700 per year between 2010 and the 2020s, and hacking incidents were a prevailing destructive threat (Neprash et al., 2022). Similarly, the United Kingdom (UK) and Europe also have these concerning attacks. The WannaCry ransom outbreak of 2017, known to have exploited a software vulnerability in Windows operating systems, disrupted almost a third of National Health Service (NHS)

hospitals in the UK, a situation that led to approximately 19,000 cancelled appointments and an estimated £92 million lost in damages and information technology (IT) recovery costs[1][2].

An EU-wide analysis found that around 76% of healthcare cybersecurity incidents in 2021 traced back to basic web application attacks, system intrusions, or human error (van Kessel et al., 2023), evidence that lapses in secure software development, not sophisticated attacks, drive most breaches. The consequence is not only financial, but it is a direct threat to patient safety.

Consequently, there is growing interest in leveraging artificial intelligence (AI) and machine learning to help identify and remediate software vulnerabilities before they can be exploited. Researchers are increasingly exploring AI-driven approaches to automate code analysis and vulnerability detection (Shaon & Akter, 2025), to augment traditional security audits and ultimately strengthening the resilience of healthcare software systems. Therefore, healthcare application software should be developed to eliminate zero-day vulnerabilities by ensuring that the software code is thoroughly free of errors that can then be exploited (Roumani, 2021). There is growing evidence of a general lack of security awareness among developers of healthcare software. These and other factors like insufficient security rules, tight budgets and timelines, and a lack of security testing of code before deployment contributes to the lack of resilience in developed software (Aljedaani et al, 2020, 2021). Additionally, most software developers significantly rely on a variety of software development kits (SDK), and these increase the dangers of data linkage, detectability, and re-identification of users in such ecosystems (Solomos, et al., 2019).

Thus, it is important to provide a solution to the privacy and security challenges faced by developers, data owners and healthcare providers caused by defects in software code. A way to achieve this is by identifying and addressing data privacy and security vulnerabilities in existing healthcare systems. By integrating "resilience and privacy by design" principles within

[1] https://www.nationalhealthexecutive.com/articles/wannacry-cyber-attack-cost-nhs-ps92m-after-19000-appointments-were-cancelled

[2] https://www.nao.org.uk/reports/investigation-wannacry-cyber-attack-and-the-nhs/

the software development lifecycle (SDLC), stakeholders can engineer more secure and private healthcare applications. This approach is the overarching goal of this research project.

Cyber resiliency, the ability of systems to anticipate, withstand, recover from, and adapt to cyberattacks (Ross, et al., 2021), is important in the healthcare sector, where sensitive data is constantly processed. Cyberattacks is the unauthorised access with the malicious intent to compromise, disable, or steal assets within an information system, pose a significant threat to healthcare. Consequences include loss of patient privacy (confidentiality), compromised treatment data (integrity), disrupted care (availability), financial losses, reputational damage, and even loss of life. The healthcare environment handles various data types, public, private, and confidential (Finn, Wright, & Friedewald, 2013), each with specific protection needs. Privacy, a cornerstone of the patient-provider relationship, gives patients control over their health data, while security involves protecting that data. Healthcare information systems (HIS), composed of hardware and software, are subject to regulations like the UK General Data Protection Regulation (GDPR) and Data protection Act (DPA) 2018. The challenge lies in the complexity of healthcare technology which leaves room for software weaknesses, a vast attack surface, and human error that cybercriminals can exploit. While both requirements apply to how the data of identifiable people are processed, the common law duty of confidentiality governs the disclosure of confidential information belonging to the patient and service-user, by protecting the identification of both living and deceased people. Resilience-by-design is a foundational prerequisite for the other dimensions of the framework. When a system is not resilient in its code, it cannot withstand or recover from targeted exploits. Thus, a framework that reduces the attack surface is necessary for system wide resilience.

To avoid ambiguity, the scope of 'cyber resilience' in this thesis is stated explicitly here. Ross et al.'s (2021) framework (Figure 1-1) comprises four dimensions: Anticipate, Withstand, Recover, and Adapt. This research addresses only the Anticipate and Withstand dimensions, and only at the pre-deployment, software development stage. It does not address Recover (restoring operations after an incident) or Adapt (evolving defences based on experience), both of which are operational, post-deployment concerns outside this thesis's scope. Accordingly, unless explicitly qualified otherwise, every subsequent reference to 'cyber resilience' in this thesis refers specifically to pre-deployment *resilience* achieved through

security and privacy by design during software development and not the complete cyber-resilience lifecycle.

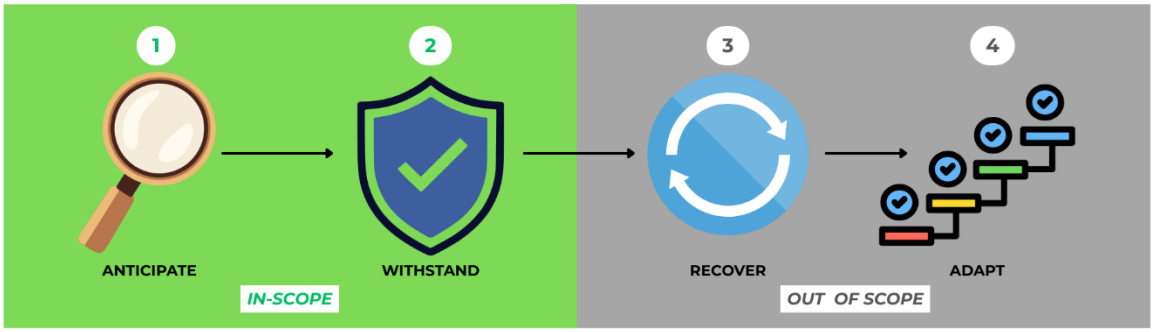


Figure 1-1: The four dimensions of cyber resilience (Ross et al., 2021).

1.1 Motivation

This research is motivated by a specific gap rather than the general threat landscape alone: current cybersecurity approaches treat privacy requirements and technical vulnerability assessment as separate domains, even though in healthcare the two are functionally inseparable. Traditional security methodologies and static code analysis tools lack the contextual understanding to identify healthcare-specific privacy threats, while privacy threat modelling methodologies operate independently of software vulnerability assessment frameworks. This fragmentation leaves interconnected privacy and security risks unidentified during development.

Addressing vulnerabilities at their source is more effective and cost-efficient than post-deployment remediation, but healthcare organisations face constraints such as regulatory validation requirements, operational continuity needs, and patient safety considerations, that make proactive, integrated assessment during development especially important. The incorporation of natural language processing (NLP) and machine learning into this assessment can increase detection accuracy, yet these approaches have not been systematically applied to healthcare-specific security and privacy requirements together.

1.2 The Significance of this Research

This thesis addresses the important intersection of healthcare cybersecurity, privacy protection, and software development security at a time when healthcare is faced with

unprecedented digital transformational challenges and cyber threats at an enormous scale. The significance of this work extends across multiple dimensions, including the immediate practical impact on healthcare security, theoretical advancement in digital security methodologies, and long-term contribution to patient privacy, safety and healthcare delivery resilience. Accordingly, as literature has revealed significant limitations in current healthcare software development cyber security approaches, this research effort directly addresses this.

The systematic review by Kruse et al. (2017) highlights the unique vulnerability profile of the healthcare sector, which differentiates it from other industries because of the sector-specific threats, including medical device compromise and patient data exploitation. This research addresses these limitations directly by developing an integrated framework that connects privacy threat modelling with NLP-enhanced vulnerability detection, purpose-built for the regulatory complexity and patient safety constraints that distinguish healthcare software development from other application domains.

There are systematic patterns of vulnerabilities in healthcare software and medical devices, yet existing security assessment approaches lack comprehensive frameworks for addressing both technical vulnerabilities and privacy threats simultaneously (Mejía-Granda et al., 2024). The focus of this research on mapping LINDDUN privacy threats to CWE vulnerabilities provides an unprecedented integration of privacy-specific threat analysis with traditional software security assessment. This is illustrated in Figure 1-2.

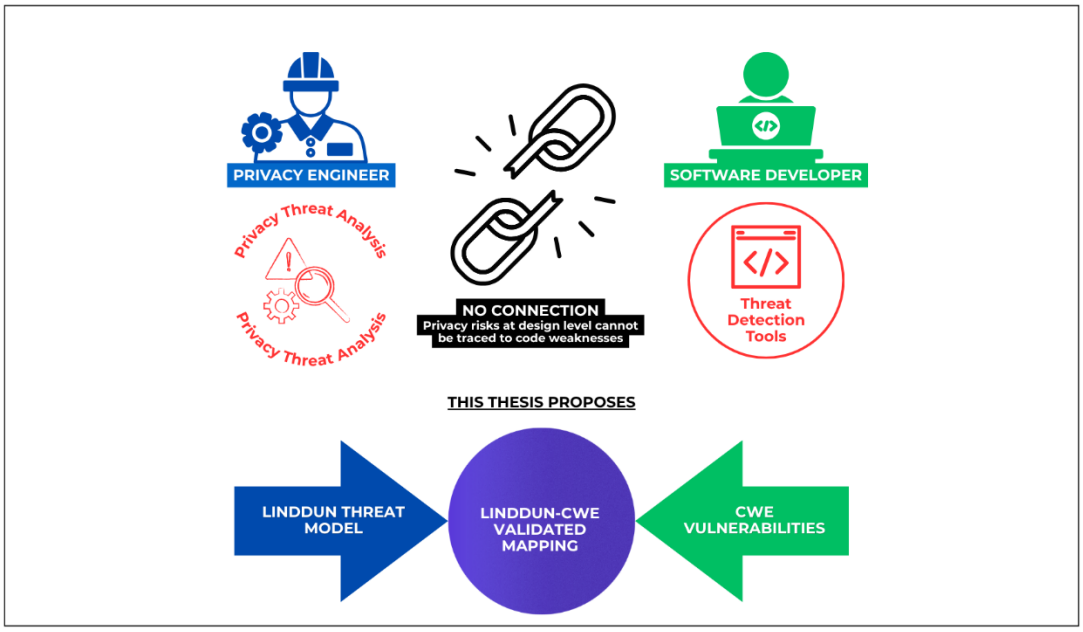


Figure 1-2: The core problem — privacy threat analysis (LINDDUN) and software vulnerability detection (CWE) operate as disconnected disciplines in healthcare software development.

Theoretically, the contribution of this research extends beyond immediate practical applications to advance academic understanding of healthcare cybersecurity as a distinct discipline. While LINDDUN (Deng et al., 2011) and CWE each represent mature frameworks within their respective domains, no prior work has systematically integrated them into a unified assessment approach applicable to healthcare software development and this is the methodological gap this thesis directly addresses.

Consequently, the integration of privacy threat modelling with software vulnerability assessment by this research endeavour represents a novel methodological approach that bridges two traditionally separate domains. The development of a healthcare-specific NLP dataset for cybersecurity applications contributes to the emerging field of domain-specific cybersecurity research. Unlike general cybersecurity applications, healthcare environments require an understanding of medical terminology, regulatory context, and patient safety implications that existing general-purpose models cannot adequately address. This thesis lays emphasis on NLP-enhanced static code analysis and addresses the limitations identified in current automated vulnerability detection research. Moreover, the integration of privacy threat modelling into automated code analysis represents a significant advancement beyond current static analysis approaches.

The product of this research is focused on implementing privacy and security during software development to address vulnerabilities at their source, potentially preventing costly post-deployment security incidents as documented in healthcare cybersecurity literature. The development of real-time vulnerability detection during the SDLC represents a fundamental shift from reactive to proactive security assessment, identified as critical by multiple systematic reviews. As a result, the integration of privacy threat modelling into software development lifecycles provides healthcare organisations with a systematic approach to address regulatory compliance requirements during development.

Finally, this research contributes to the growing field of privacy-preserving healthcare technologies identified in recent academic literature.

1.3 Problem Statement

Healthcare information systems are faced in security and privacy dilemma, which is the lack of unified security assessment methodologies that integrate the evaluation of technical vulnerabilities with privacy threat assessments during software development. This non-existent symphony leaves healthcare systems exposed to cyber threats that can compromise patient data, disrupt critical care services, and violate comprehensive regulatory frameworks such as HIPAA, GDPR, and FDA guidelines. The interconnected nature of healthcare systems means that privacy breaches can also manifest as technical vulnerabilities, and vice versa. Nonetheless, the existing assessment tools and methodologies fail to capture this synergy, leading to undetected risks. Hence, severe consequences, including cyberattacks, result in not only financial losses but can also directly endanger patient lives by tampering with medical devices or interrupting essential services. Furthermore, the niche and strict constraints in healthcare, including the need for constant availability and adherence to multiple regulatory standards, render traditional security assessment practices inadequate. The current reliance on separate, siloed evaluations of technical and privacy risks creates coordination challenges and scalability issues, particularly as healthcare organisations strive to accelerate software development cycles. Reduced to its precise form, this fragmentation persists because there is no validated, reproducible mechanism for translating design-level privacy threats that is identified through methodologies such as LINDDUN, into the code-level weaknesses

catalogued by taxonomies such as the Common Weakness Enumeration (CWE). Privacy threat modelling and software vulnerability assessment therefore remain methodologically disconnected: one operates at the level of data flows and design abstractions, the other at the level of implementation defects, with no systematic, expert-validated bridge between them. This is the specific gap this thesis addresses.

This gap exists partly because LINDDUN and CWE were developed by different research communities for different purposes, using different units of analysis. LINDDUN describes privacy threats (Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, and Non-compliance) as properties of data flows within a design-level data flow diagram (DFD), applied before a system is built. CWE, by contrast, catalogues concrete weaknesses in source code, such as improper input validation, insufficient logging, or missing access control checks, identified after code exists. No shared vocabulary connects a privacy property named at design time to the specific coding weaknesses that could realise it. Where mappings between the two have been proposed in prior work, they have typically been produced by a single author's judgement rather than validated through structured, multi-rater agreement, so their reliability and reproducibility have not been established.

This matters because the two forms of analysis are causally connected in practice even though they are methodologically separated. A privacy threat identified at design, for instance, a Linkability risk arising from how patient identifiers flow between system components, is ultimately realised, or not, by specific decisions made in code: how identifiers are stored, whether they are logged, how they are compared or joined across datasets. Without a validated mapping from the LINDDUN category to the corresponding CWE weaknesses, a privacy threat flagged during design analysis cannot be traced forward to the code constructs responsible for it, and a vulnerability found in code cannot be traced back to the privacy property it violates.

The consequence is a broken chain of accountability across the development lifecycle. Privacy threats identified at design remain abstract recommendations that never reach the developer writing the code, while automated code-level vulnerability detection, however accurate, has no mechanism for recognising when a detected weakness is privacy-relevant rather than purely a security defect. This also blocks a specific practical capability: building a

dataset or detection system that is privacy-aware at the code level first requires a validated mapping to know which CWE-labelled code patterns correspond to which privacy threats. Establishing that mapping and validating it empirically rather than asserting it by expert opinion alone, is the first contribution this thesis makes (Section 1.6, Contribution 1).

To address the identified problem of fragmented security and privacy assessment in healthcare software development, this research sought to answer the following questions:

RQ1: How can privacy threat modelling methodologies (specifically LINDDUN) be systematically integrated with software vulnerability taxonomies (specifically CWE) to create a unified framework for assessing both privacy threats and security vulnerabilities in healthcare software systems?

RQ2: What dataset characteristics and annotation strategies are required to enable effective AI-based vulnerability detection specifically tailored to the privacy and security requirements of healthcare software?

RQ3a: Which machine learning paradigm, traditional machine learning, graph neural networks, or transformer-based natural language processing, provides the most reliable and practical method for static code vulnerability detection in C/C++ healthcare software?

RQ3b: Does integrating the highest-performing paradigm identified in RQ3a within a hybrid symbolic–statistical architecture improve robustness over pure machine learning deployment, particularly for known vulnerability patterns and code where classifier confidence is reduced?

The following operational definitions apply to both RQ3a and RQ3b. For the purposes of this research, "reliable" is operationalised as detection performance measured by F1-score, recall, false negative rate, and AUC-ROC on a held-out test set, with particular weight given to recall and false negative rate given the patient safety implications of missed vulnerabilities in healthcare contexts. "Practical" is operationalised across four dimensions: (i) inference latency which is the time from code submission to result, assessed against a 500-millisecond threshold compatible with developer workflow integration; (ii) training feasibility, that is whether training can be completed on hardware representative of healthcare IT infrastructure, defined here as a single consumer-grade GPU within a 24-hour window; (iii)

infrastructure dependency, is whether the model requires GPU acceleration at inference time or can operate on CPU-only hardware, which is relevant to resource-constrained healthcare environments; and (iv) interpretability, if the model produces outputs that a developer can act upon without specialist ML expertise, including confidence scores and CWE-specific fix recommendations. Where these dimensions conflict, inference latency and infrastructure dependency are treated as threshold constraints that must be met, while F1-score governs selection among models that satisfy those constraints.

RQ4: Can an integrated vulnerability detection system be developed that achieves sufficient accuracy for practical deployment in healthcare software development environments, provides actionable corrective guidance, and meets the performance requirements for integration into development workflows?

1.3.1 Research Hypotheses

Based on the literature review and preliminary analysis, this research posits the following hypotheses:

H1: The seven LINDDUN privacy threat categories can be systematically and reliably mapped to Common Weakness Enumeration (CWE) entries, with expert validation achieving substantial inter-rater agreement, thereby creating an integrated privacy-security assessment framework applicable to healthcare software development.

H2: Transformer-based natural language processing models, which treat source code as a specialised form of language, will outperform both traditional machine learning approaches and graph neural networks for C/C++ vulnerability detection, achieving significantly higher F1-scores on healthcare-relevant vulnerability datasets.

H3: Building on the paradigm identified in RQ3a, and directly addressing RQ3b, a hybrid classification architecture combining symbolic pattern matching with statistical machine learning will achieve superior robustness compared to pure machine learning deployment, particularly for known vulnerability patterns and obfuscated code samples where ML classifiers exhibit reduced confidence, thereby justifying a hybrid rather than pure-ML implementation in the detection prototype.

H4: An integrated vulnerability detection prototype incorporating multi-modal feature extraction, hybrid classification, and CWE-specific fix suggestions can achieve detection accuracy exceeding 95% with a cold-path response time of 264 milliseconds, meeting the practical requirements for deployment in healthcare software development workflows. This accuracy threshold is evaluated on a held-out test set under the 1:3 class ratio used in training and should be interpreted as an upper-bound estimate pending deployment validation under realistic class distributions.

1.4 Aim and Objectives

The aim of this research is to enhance the pre-deployment cyber resilience and privacy protection of healthcare information systems by developing an integrated framework that combines privacy threat modelling with automated vulnerability detection during software development. This framework seeks to enable the identification and remediation of security vulnerabilities and privacy threats in healthcare software before deployment, thereby reducing the risk of data breaches, ensuring regulatory compliance, and ultimately safeguarding patient safety.

To achieve the stated aim, the following objectives have been defined:

Objective 1: To demonstrate data flow patterns in typical healthcare information systems and systematically map privacy and vulnerability misuse cases by integrating the LINDDUN privacy threat methodology with the Common Weakness Enumeration (CWE) taxonomy, creating a unified framework for privacy-security assessment.

Objective 2: To create a novel healthcare cybersecurity-focused dataset for NLP, comprising annotated C/C++ code samples with vulnerability labels and CWE classifications relevant to healthcare software security requirements.

Objective 3: To develop and evaluate a reliable method for static software code security testing using natural language processing techniques, comparing traditional machine learning, graph neural networks, and transformer-based approaches to identify the most effective paradigm for detecting vulnerable code in healthcare software.

Objective 4: To develop a prototype vulnerability detection solution capable of identifying security weaknesses in code during healthcare software development, providing actionable corrective recommendations, and to evaluate this solution through user acceptance testing with industry practitioners and academic researchers.

By answering the research questions and meeting the research objectives, a novel privacy focused cybersecurity dataset that is healthcare specific will be available for further NLP research. This is adequately captured in Figure 1-3 where the exact point of knowledge addition is shown within the solution space of the research methodology. Additionally, a state-of-the-art method of detecting weaknesses in software code is proposed.

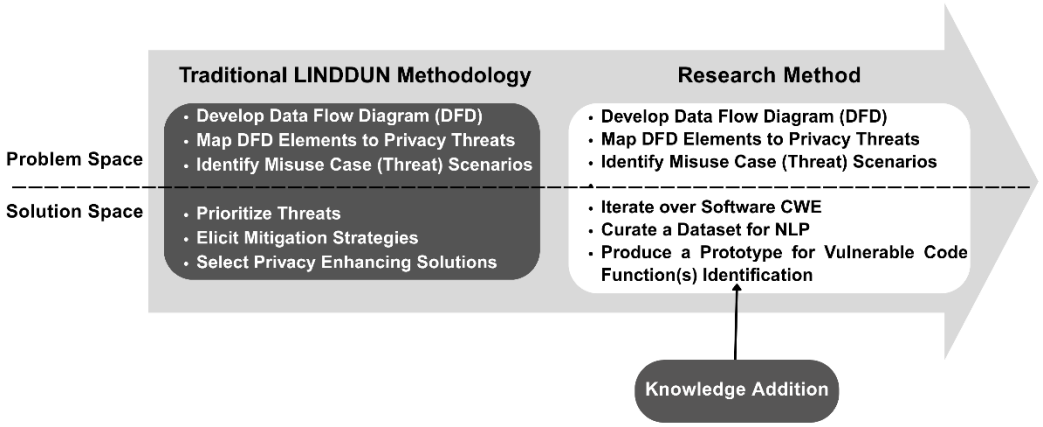


Figure 1-3: Research Objectives: Enhanced LINDDUN Methodology

1.5 Contributions

By addressing the research objectives, this study makes the following contributions to knowledge in the fields of healthcare cybersecurity, privacy engineering, and automated vulnerability detection. This is illustrated in Figure 1-4:

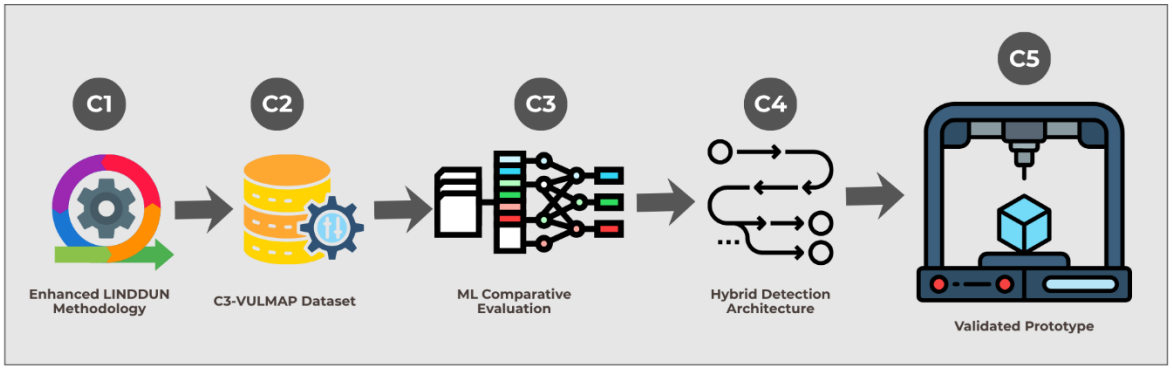


Figure 1-4: The five research contributions as a connected pipeline.

Contribution 1 — Enhanced LINDDUN Privacy Threat Methodology: This research proposes a novel extension to the LINDDUN privacy threat modelling methodology by systematically mapping the seven LINDDUN threat categories to the Common Weakness Enumeration (CWE) taxonomy. This integration creates a unified framework that connects high-level privacy threat analysis with concrete code-level vulnerability assessment, operationalising the gap identified in the Problem Statement as a concrete methodological contribution.

Contribution 2 — Healthcare-Aligned Vulnerability Dataset (C3-VULMAP): This research contributes a large-scale, openly published vulnerability dataset comprising 6.3 million annotated C/C++ code samples with binary vulnerability labels, CWE classifications, and LINDDUN privacy threat annotations. The dataset achieves healthcare alignment through two complementary mechanisms: a privacy-relevance filter derived from the LINDDUN-to-CWE mapping developed in Contribution 1, which retains only vulnerability instances corresponding to CWE categories that enable the seven LINDDUN privacy threat categories relevant to healthcare regulatory contexts; and a domain-prioritisation layer applied to repository-level metadata, which prioritises code from projects with explicit healthcare functionality, including medical imaging, clinical decision support, and health information exchange, drawn from the ReposVul source corpus. It is noted that most of the constituent code originates from general-purpose open-source C/C++ repositories rather than exclusively from healthcare codebases; the healthcare specificity therefore resides principally in the annotation scheme and vulnerability selection criteria rather than uniformly in code provenance. This distinction is acknowledged as a scope boundary of the contribution and is addressed in the dataset limitations discussion in Chapter 4. The dataset is the first publicly

documented vulnerability corpus to combine CWE classification with LINDDUN privacy threat labelling, enabling the development of models that associate code-level weaknesses with their privacy threat implications in healthcare software contexts.

Contribution 3 — Comparative Evaluation of ML Paradigms for Vulnerability Detection: This research gives a comprehensive comparative evaluation of machine learning approaches for C/C++ vulnerability detection, assessing models across three paradigms: traditional machine learning, graph neural networks, and transformer-based natural language processing. This evaluation also establishes empirical evidence regarding the most effective paradigm for static vulnerability detection in healthcare software contexts.

Contribution 4 — Hybrid Detection Architecture: This research presents a novel hybrid classification architecture combining symbolic pattern matching with statistical machine learning for vulnerability detection. This architecture aims to address the limitations of pure machine learning approaches by providing deterministic classification for known vulnerability patterns with graceful degradation for novel code, implementing a security-first design principle appropriate for healthcare contexts.

Contribution 5 — A Validated Prototype Vulnerability Detection System: This research delivers a deployment-feasible vulnerability detection system deployed via representational state transfer application programming interface (REST API) and web frontend for practical use, with CWE-specific remediation guidance enabling developers to address vulnerabilities before healthcare systems go live. The system is validated through user acceptance testing with industry practitioners and academic researchers to establish its practical utility for healthcare software development workflows.

The research underpinning this thesis has generated four publications at progressive stages of the work. These are listed in the preliminary pages of this thesis.

Collectively, these publications demonstrate that the core contributions of this thesis have been subjected to independent peer review across two journals, a workshop proceedings volume, and a flagship software engineering venue.

1.6 Thesis Scope

This thesis is concerned with the security and privacy of healthcare information systems at the software development stage, with particular focus on C and C++ codebases given their prevalence in medical device firmware, clinical infrastructure, and embedded health systems. The scope encompasses two complementary dimensions of privacy threat modelling at the architectural design level, and automated vulnerability detection at the source code level. On the privacy side, the thesis is scoped to the LINDDUN threat modelling framework and its application to healthcare system data flow diagrams, stopping short of runtime privacy enforcement or legal compliance assessment. On the vulnerability detection side, the scope is bounded to static analysis of source code, specifically function-level binary classification of vulnerable versus non-vulnerable C/C++ functions and multi-class CWE-based vulnerability matching. However, dynamic analysis, fuzzing, and binary-only analysis are outside the scope of this work. The machine learning evaluation is confined to supervised learning paradigms across traditional, graph neural network, and transformer-based model families, evaluated on the C3-VULMAP dataset constructed as part of this research. The hybrid detection system is scoped to a REST API deployment targeting real-time, CPU-feasible inference in resource-constrained healthcare development environments and does not extend to automated vulnerability remediation or patch generation. While the regulatory context of healthcare including HIPAA, GDPR, and NHS Data Security Standards informs the framing and design decisions throughout, the thesis does not conduct legal or compliance analysis. Geographically and institutionally, the work draws on publicly available open-source repositories and is not restricted to any single healthcare system or national context.

1.7 Thesis Structure

This thesis is structured into six chapters, each logically building on the one before providing a coherent narrative and an aggregated response to the research objectives. The structure reflects the progression from foundational context and theoretical frameworks to methodology development, empirical evaluation, and reflective synthesis of findings. Each chapter contributes distinctively toward the development of a privacy-aware, machine learning–based vulnerability detection framework tailored for healthcare software systems.

Chapter 1: Introduction presents the background and rationale for the study, articulating the significance of secure software development in healthcare contexts where privacy violations may result in severe ethical, legal, and clinical consequences. The chapter establishes the problem statement, research questions, and hypotheses before outlining the research aim and objectives. Expected contributions to knowledge are presented, and the chapter concludes with an overview of the thesis structure.

Chapter 2: Literature Review provides a critical examination of foundational topics relevant to the research. It surveys the secure software development lifecycle and relevant standards, best practices, and privacy-by-design frameworks. The review explores key themes in privacy engineering, application security testing, and the Common Weakness Enumeration (CWE) framework. It extends to the application of natural language processing (NLP) and machine learning techniques for vulnerability detection in source code, identifying gaps that justify the research direction.

Chapter 3: Enhanced LINDDUN Privacy Threat Methodology for Healthcare Software Systems introduces the enhanced privacy threat modelling methodology developed for the healthcare domain. The chapter details the systematic mapping of LINDDUN privacy threat categories to CWE entries, creating explicit linkages between high-level privacy concerns and code-level vulnerabilities. Expert validation of the mappings is presented, and practical application is demonstrated through pilot analysis of healthcare information system components.

Chapter 4: Dataset Preparation and Classification Approach address the creation of a novel healthcare cybersecurity-focused dataset and the comprehensive evaluation of machine learning approaches for vulnerability detection. The chapter documents the dataset construction process and presents experimental results comparing fourteen models across three paradigms: traditional machine learning, graph neural networks, and transformer-based architectures. A comparative analysis justifies the selection of models for prototype integration.

Chapter 5: Vulnerability Detection Prototype System demonstrates the practical application of the research through the design, implementation, and evaluation of a

deployment-feasible vulnerability detection system. The chapter details the system architecture, multi-modal feature extraction pipeline, hybrid classification approach, and safety rule mechanisms. User acceptance testing methodology and results are presented, validating the practical utility of the system with industry practitioners and academic researchers.

Chapter 6: Conclusion and Future Work summarise and presents the research findings, revisiting each research objective and evaluating hypothesis outcomes. The chapter articulates the achieved contributions to knowledge, discusses implications for theory and practice, acknowledges limitations, and provides recommendations for healthcare organisations, security practitioners, and researchers. Directions for future work are outlined and concluding remarks reflect on the significance and impact of the research.

2 Chapter 2: Literature Review

2.1 Background

The convergence of software development practices with healthcare data protection requirements represents a critical area of scholarly inquiry, and one that has received growing academic attention in the period since 2020. Traditional software security approaches have proven inadequate for healthcare environments, where vulnerabilities can directly compromise patient safety and regulatory compliance simultaneously, a dual consequence that distinguishes this sector from most other application domains (Ewoh & Vartiainen, 2024). The unique operational constraints of healthcare, including continuous availability requirements, legacy system integration, complex interoperability standards, and stringent regulatory obligations under frameworks such as HIPAA, GDPR, and NHS Data Security Standards, necessitate specialised security development methodologies that existing general-purpose approaches do not provide. Despite the increasing adoption of DevSecOps practices and shift-left security integration in software development more broadly, implementation within healthcare settings remains constrained by risk-averse organisational cultures, regulatory validation requirements, and the absence of comprehensive threat modelling approaches that address both technical vulnerabilities and privacy threats within a single coherent framework.

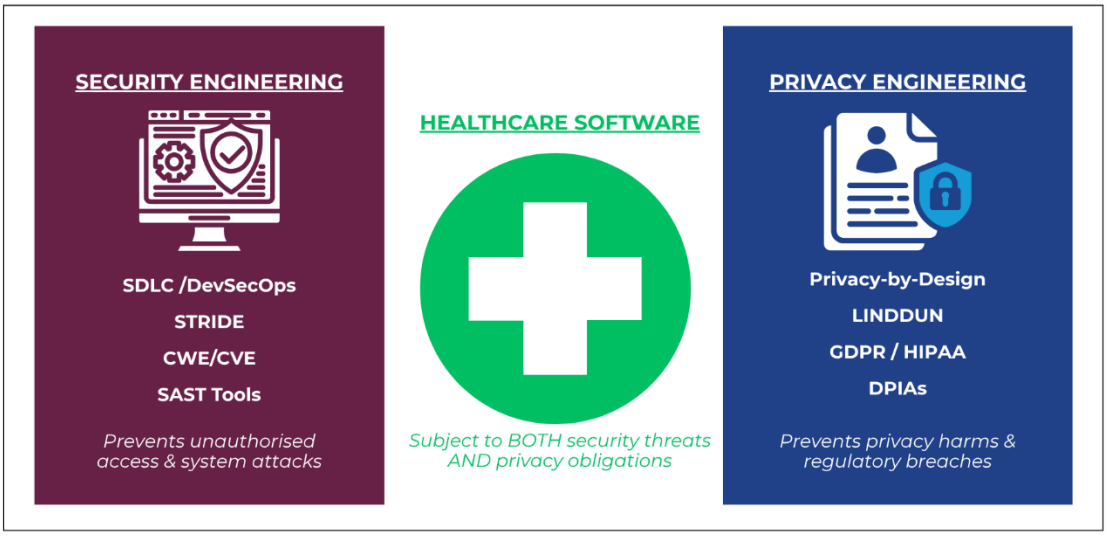


Figure 2-1: The two disciplines governing healthcare software protection operate independently, creating a gap that leaves privacy risks unconnected to code-level vulnerabilities.

Privacy threat modelling has emerged as an important discipline for healthcare software development, with the LINDDUN methodology providing a structured, privacy-focused approach to identifying threat categories spanning linkability, identifiability, non-repudiation, detectability, disclosure, unawareness, and non-compliance, each directly relevant to the data-intensive nature of clinical software systems (Sion et al., 2018). The integration of LINDDUN with software vulnerability taxonomies such as CWE represents a research opportunity of particular significance for healthcare, as discussed in detail in Section 2.5. Furthermore, the application of natural language processing and machine learning to static code analysis offers the prospect of detection capabilities that transcend the limitations of traditional rule-based tools, particularly in their ability to identify contextual vulnerability patterns relevant to healthcare-specific regulatory requirements. Building on the research rationale established in Chapter 1, this chapter critically examines the body of literature across these intersecting domains (Figure 2-1): secure software development, privacy engineering, threat modelling, application security testing, and machine learning-based vulnerability detection, identifying the specific gaps that motivate and bound this research.

This chapter presents a structured critical review of the literature across the domains that bound this research: secure software development, privacy engineering, threat modelling, application security testing, and machine learning-based vulnerability detection. Given the breadth of these intersecting domains, the review adopts a critical rather than fully systematic methodology across most sections, synthesising and evaluating literature identified through iterative, citation-chaining search rather than a single predefined protocol (section 2.12), however, reports findings from a fully systematic literature review conducted as part of this research (Ameh et al., 2025), for which the complete PRISMA-compliant search methodology is provided.

2.2 An Overview of Privacy and Security for Information Systems in Healthcare

Academic literature on healthcare information security has evolved significantly since 2020, driven by escalating cyber threats, regulatory changes, and technological advancements. This section examines existing research across key domains relevant to pre-

deployment cyber resilience and privacy in healthcare information systems, with particular emphasis on methodological approaches to software development security.

Several systematic literature reviews have documented the rapid evolution of healthcare security and privacy research. Jalali et al. (2019) conducted a comprehensive bibliometric analysis of 472 articles, revealing that technology-focused research predominated (over 50% of clusters) while managerial aspects received less attention (32%). Their study demonstrated significant growth in healthcare cybersecurity research, with particular emphasis on technical solutions over organisational and human factors. Similarly, Argaw et al. (2020) analysed healthcare digital security challenges during the COVID-19 pandemic, identifying nine main cybersecurity challenges and eleven key solutions adopted during crisis response, highlighting both the acceleration of digital health adoption and corresponding security vulnerabilities introduced during emergency implementations.

Research on security vulnerabilities in medical devices reveals systematic weaknesses across the healthcare infrastructure. Mejía-Granda et al. (2023) conducted a comprehensive analysis of security vulnerabilities in healthcare systems, examining both medical devices and software systems. Their study identified hard-coded credentials, authorisation problems, and input validation failures as the most prevalent security issues in healthcare environments. Additionally, Kumar et al. (2023) provided a critical cybersecurity analysis of Internet of Things applications in healthcare, identifying specific vulnerabilities including weak authentication mechanisms, insufficient encryption, and inadequate lifecycle management.

Threat modelling methods have become increasingly essential in systems development and maintenance. The LINDDUN methodology has emerged as a leading privacy-focused threat modelling approach, empirically validated across multiple studies for its correctness, completeness, and reliability in identifying privacy-specific threats that traditional security assessment approaches overlook (Wuyts et al., 2014; Wuyts et al., 2020). A full treatment of LINDDUN, including its application to healthcare system data flow diagrams and its relationship to the CWE taxonomy, is provided in Section 2.5.

The application of machine learning to software vulnerability detection has gained significant academic attention. Harzevili et al. (2024) conducted a systematic literature review

on automated software vulnerability detection, analysing trends and methodologies in the field. Their comprehensive review identified accelerated publication growth between 2020 and 2024, with a cumulative focus on deep learning approaches achieving 93–98% accuracy in vulnerability detection tasks. Importantly, their review also identified a persistent domain gap where most evaluated studies were trained and evaluated on general-purpose open-source repositories, with no studies in their corpus incorporating healthcare-domain provenance, LINDDUN privacy annotations, or vulnerability profiles representative of clinical software. This absence constitutes a material limitation for practitioners seeking to deploy vulnerability detection in regulated healthcare environments.

Natural language processing in cybersecurity has advanced significantly with transformer-based models. Aghaei et al. (2022) developed SecureBERT, a domain-specific language model demonstrating enhanced performance in Cyber Threat Intelligence analysis compared to general-purpose models through domain-specific tokenisation and pre-training on cybersecurity corpora. Similarly, CodeBERT has shown strong performance in vulnerability detection tasks through multi-head attention mechanisms that capture long-range dependencies in code (Feng et al., 2020). Hybrid approaches combining multiple neural architectures have demonstrated superior performance in adjacent domains; Mei et al. (2023) reported that CNN-BiLSTM hybrid architectures provide effective parallel processing for improved accuracy in IoT vulnerability detection, particularly relevant given the proliferation of connected medical devices. Furthermore, He et al. (2025) developed an improved vulnerability detection method combining multi-feature screening and integrated sampling models, while Bilgin et al. (2020) demonstrated that combining Abstract Syntax Tree representation with feature selection enhances detection of complex vulnerability patterns.

Research has identified unique challenges in distinguishing healthcare cybersecurity from other industries. Coventry and Branley (2018) conducted a systematic review of modern threats and trends, identifying the vulnerability of the sector to both traditional cyberattacks and healthcare-specific threats, including medical device compromise and patient data exploitation. Ewoh and Vartiainen (2024) emphasised the importance of addressing both technical vulnerabilities and human factors in cybersecurity implementations through

sociotechnical solutions, noting that healthcare information systems must be treated as sociotechnical systems rather than purely technical ones.

Research on privacy-preserving technologies in healthcare has augmented substantially, with federated learning emerging as a leading approach. Pati et al. (2024) examined privacy preservation for federated learning in healthcare, demonstrating how collaborative analytics can be conducted without data sharing. Blockchain applications have also received substantial academic attention; Moulahi et al. (2023) developed a blockchain-based federated learning mechanism for privacy preservation of healthcare IoT data.

This analysis of literature reveals a cluster of interconnected gaps in healthcare cybersecurity research. The most significant of these is the absence of a systematic mechanism for integrating privacy threat modelling with software vulnerability assessment. While LINDDUN and CWE frameworks exist and are independently validated within their respective domains, academic research lacks a structured, empirically verified approach for mapping privacy threats to software vulnerabilities in healthcare contexts. Honkaranta et al. (2021), in the most directly relevant prior attempt, sought to create a practical mapping between STRIDE threat categories and the CWE database and concluded that the exercise was incomplete, produced ambiguous results, and required expert judgement that the framework did not systematically guide and demonstrating that the mapping challenge is genuine rather than trivially solvable. Crucially, their work used STRIDE rather than LINDDUN, leaving the privacy-specific mapping space entirely unaddressed. The present research directly targets this gap.

A second gap concerns the development of healthcare-specific NLP datasets for cybersecurity applications. Existing vulnerability datasets such as Devign, Big-Vul, DiverseVul, and the Juliet test suite are constructed from general-purpose open-source repositories (Chen et al., 2023; Fan et al., 2020; Zhou et al., 2019). None incorporate healthcare-domain provenance, LINDDUN privacy annotations, or vulnerability profiles representative of clinical software environments. This creates a generalisability problem for models intended to operate on healthcare codebases, where the vulnerability distribution, regulatory context, and consequence of misclassification differ materially from general software engineering contexts.

A third gap concerns the near-total absence of real-time security analysis during healthcare software development. Most published research focuses on post-deployment assessment or benchmark evaluation, rather than on tools integrated into development workflows and validated under conditions representative of healthcare software engineering practice. These interconnected gaps motivate the three primary empirical contributions of this thesis, which are developed in Chapters 3, 4, and 5 respectively.

2.3 Secured Software Development Lifecycle

Security, which is used for analysing the quality of the final product, is a very important phase in software development (Olorunshola & Ogwueleka, 2021). The concept of software security refers to the assurance of integrity, confidentiality and availability of the code, data and services during software design and development. Often, security is only included after the development effort rather than as a priority by being integrated into the SDLC (Papageorgiou, 2018; Khan, et al., 2021). Furthermore, the drive to be the first to market technological innovations to achieve organisational objectives such as stock price gains or profit income has led both established businesses and startups to deploy new products containing vulnerable software, and only to provide security patches as continuous updates post-deployment (Niazi et al., 2020). This is a faulty approach because of the likelihood that end users may fail to apply the patches for different and sometimes legitimate reasons. Likewise, threat actors are likely to exploit so-called ‘zero day’ vulnerabilities (Al-Matouq et al., 2020). Some traditional security methods of securing software include safeguarding the network using intrusion detection and prevention systems, using firewalls, encryption techniques, and antivirus software, among others (Núñez et al., 2020). Nonetheless, despite the successes of these traditional methods, they may also possess inherent weaknesses since they are also software-based (Guo, 2023).

Early SDLC models, such as the classic waterfall model (Royce, 1970) and its successors, focused primarily on functionality, cost, and time, with security and privacy considerations largely treated as afterthoughts. Traditional SDLC approaches typically segmented development into phases such as requirements, design, implementation, testing, and deployment, and did not mandate explicit security activities in each of these phases.

Consequently, security was often addressed reactively, if at all, toward the end of the development effort or post-deployment (Siponen & Baskerville, 2001). Even so, researchers like Siponen and Baskerville (2001) identified this gap and argued that mainstream development methods lacked integrated security mechanisms and proposed new paradigms to embed security into the process. In practice, this omission of early-phase security considerations led to a "penetrate-and-patch" mentality, where vulnerabilities were identified post-release and patched in costly update cycles (Howard & Lipner, 2006). This approach often resulted in systems with fundamental security weaknesses and high remediation costs, since patching design-stage security flaws late in the lifecycle is exponentially more expensive (Odera et al., 2021). Additionally, privacy was seldom considered in traditional SDLC, since user data protection was generally limited to access controls or compliance checkboxes, without a comprehensive privacy-by-design strategy (Saeed et al., 2025). Overall, the historical narrative reveals that while traditional SDLC models brought discipline to software engineering, they left a critical gap in security and privacy assurance, prompting calls for a paradigm shift in development practices (Fonseca & Vieira, 2013).

However, by the early 2000s, the escalating pervasiveness of software vulnerabilities and cyberattacks emphasised the need to integrate security throughout the software development process. In response, both industry and academia developed secure SDLC frameworks that augmented traditional models with security practices at every stage. Notably, Microsoft introduced its Security Development Lifecycle (SDL) in the mid-2000s, thus institutionalising security activities such as threat modelling, secure coding standards, and security testing across the development and product teams (Howard & Lipner, 2006). Around the same time, the Open Web Application Security Project (OWASP) proposed CLASP (Comprehensive, Lightweight Application Security Process), and security expert Gary McGraw advanced the idea of software security touchpoints, a set of best practices (like code review, penetration testing, and risk analysis) to be applied within any SDLC model (McGraw, 2006). In the survey by Fonseca and Vieira (2014), they highlighted the following as seminal secure development lifecycle models: Microsoft SDL, OWASP CLASP, and Touchpoints by McGraw. According to them, the three listed methods offered comprehensive methodologies for

building security into the process, starting from requirements to release. Each of these frameworks introduced specific techniques. For instance, CLASP advocated developing "misuse cases" alongside use cases to capture security requirements, and SDL mandated activities like threat modelling early in design and rigorous security testing before release. These approaches marked a shift from ad-hoc security fixes to systematic security by design, emphasising preventive controls rather than after-the-fact detection.

Beyond full lifecycle methodologies, there also emerged complementary practices and maturity models aimed at helping organisations to incrementally improve software security. For instance, OpenSAMM (Software Assurance Maturity Model) by OWASP and the industry-driven BSIMM (Building Security in Maturity Model) provided frameworks for assessing and enhancing security practices of an organisation in development (Fonseca & Vieira, 2014). These resources acknowledged that even though organisations might be unable to replace their development methodology immediately, teams could gradually embed security controls into existing processes (Fonseca & Vieira, 2014).

Interestingly, newer development paradigms like agile methods, which iterate rapidly, handle security by integrating security posed challenges, leading to concepts like Agile Security and later DevSecOps (Myrbakken & Colomo-Palacios, 2017). While these modern approaches are state-of-the-art in theory, studies reveal that they are not widely adopted. Sinan et al. (2025) reported that many organisations did not widely adopt formal secure SDLC frameworks because of poor or limited awareness of their existence and capabilities, while some other organisations considered the perceived overhead. Similarly, Assal and Chiasson (2019) noted that developers often postpone or skip security practices because of organisational issues such as absent security requirements, insufficient expertise, and time pressure. This suggests that, despite the availability of mature secure development models, a cultural shift and management support are crucial to actualise security-by-design in practice.

Further, Secured SDLC (SSDLC) is a process flow used for software development in a timely fashion, at an optimal budget and to a high quality. The product of this process is usually a properly tested and ready-to-use software which has undergone the typical phases of requirement, design, coding, testing, development, and maintenance (Olorunshola & Ogwueleka, 2021). Examples of SSDLC include the Microsoft Trustworthy Computing Security

Development Lifecycle (MS-TCSDL), which introduces security practices at all stages of the SDLC process (Lipner, 2004). For instance, a risk assessment is carried out during the requirements phase, threat modelling and design review during the design phase, static analysis during the development phase, security testing and code review during the testing phase, etc. The MS-TCSDL has been consolidated with the secure software development framework (SSDF) by the National Institute of Standards and Technology (NIST) (Souppaya, et al., 2022). Other examples of secure software development include TSP Secure (Team Software Process for Secure Software Development) (Gupta, Faisal, & Husain, 2012), Secure Software Development Process Model (S2D-ProM) (Essafi, Jilani, & Ghezala, 2007), Requirements Engineering Security Maturity Model (RESMM) (Niazi, Saeed, Mahmood, & Zafar, 2020),and the Comprehensive and Lightweight Application Security Process (CLASP) (OWASP Foundation, 2021).

The mentioned methods of secured software development as seen in Figure 2-2 ensure that high-quality application software, which is resilient to cyberattacks and ensures data privacy, is released. These methods introduce threat modelling and code security testing activity at the requirements stage up to the testing phase of the SDLC. However, the research by Nina et al. (2021), which mapped the literature on secured software development, concluded that most approaches to secured SDLC are inadequate or impractical. Thus, there is a need for a more practical method that is easily adoptable by developers.

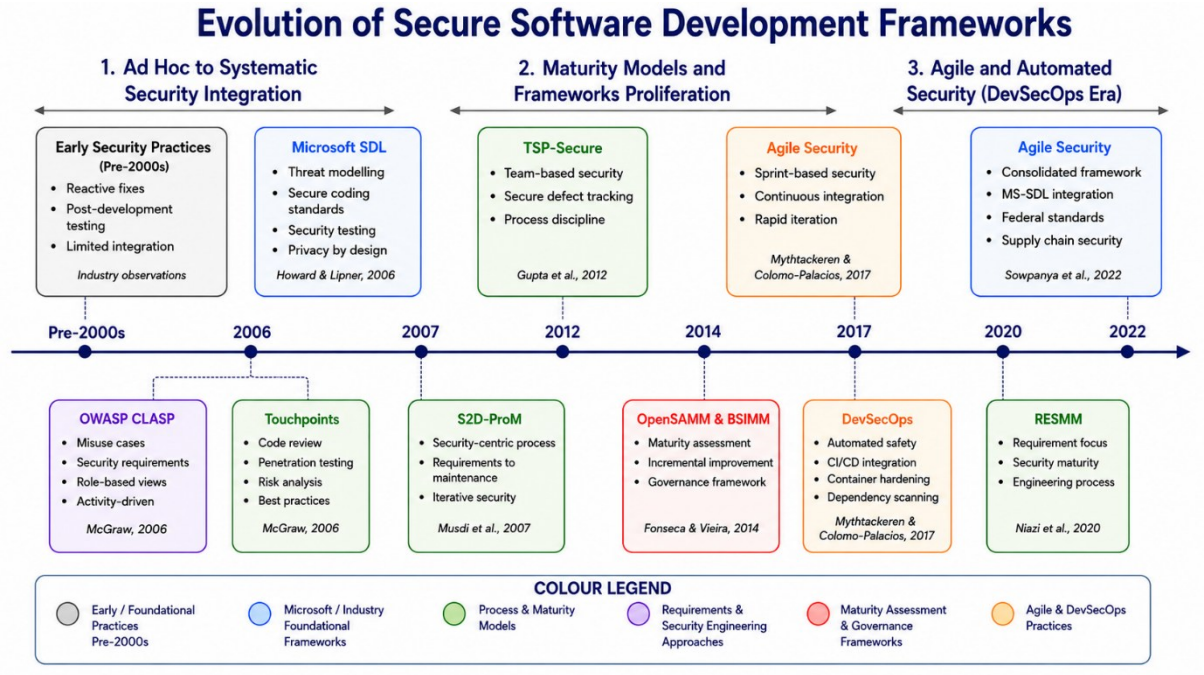


Figure 2-2: Evolution of Secured SDLC Frameworks

One of the cornerstones of the SSDLC is the principle of "shift-left" security, which basically means integrating risk analysis and mitigation measures early in the requirements and design phases (Mohammed et al., 2025). Threat modelling has emerged as a key practice to achieve this shift. Threat modelling is a structured technique in which teams identify assets, analyse potential threats and attack vectors, and devise mitigations before writing major portions of code. By anticipating how an attacker might target the system, developers can architect security controls from the outset. SDL by Microsoft made threat modelling a required activity during software design (Howard & Lipner, 2006), and methodologies like CLASP similarly recommended early "abuse case" development (a method of using a "black hat" or threat actor perspective to capture what the system should not do). Academic literature has reported the value of this approach. Lohmann et al. (2023) observed that threat modelling has gained significant traction as software threats have grown more complex, allowing organisations to proactively identify vulnerabilities and reduce risks in the design stage. Additionally, in a systematic review by Xiong and Lagerström (2019), which included over 50 articles, concluded that threat modelling methodologies can indeed be powerful but currently "lack common ground" because there are many competing methods including Spoofing, Tampering, Repudiation, Information, Denial of Service, Elevation of Privilege –

STRIDE (Das et al., 2024), Damage Potential, Reproducibility, Exploitability, Affected Users, Discoverability – DREAD (Kim et al., 2022), LINDDUN for privacy, Attack trees, etc., with inconsistent definitions and usage in practice. This fragmentation means that development teams sometimes struggle to adopt threat modelling effectively, despite understanding its importance. Nonetheless, the consensus in recent literature is that incorporating threat modelling (or similar risk analysis techniques) in early SDLC phases is critical for building secure systems (Priya & Arya, 2016). By doing so, teams can address design-level weaknesses (such as insecure architectures or data flows) before they manifest as hard-to-fix vulnerabilities in deployed software. In domains like healthcare, where threat modelling can identify potential privacy breaches or safety risks in health IT systems, this early integration is highly desirable. However, a challenge noted by industry studies is the shortage of skilled personnel because many developers have limited training in security analysis, and security specialists are not always involved at the early stages (Maher et al., 2019). Accordingly, an acute part of advancing state-of-the-art SDLC is not only developing threat modelling tools and frameworks but also training developers to adopt a security mindset and providing organisational support, like putting in place dedicated security champions in development teams.

Similarly, as security needs to be engineered from the beginning, privacy must be considered too. The concept of Privacy by Design (PbD), popularised by Ann Cavoukian in the 1990s and 2000s, encapsulates the idea that privacy should be embedded into system design "by default" rather than added on afterwards (Cavoukian, 2010). In the context of SDLC, Privacy by Design means that during requirements engineering and design, developers explicitly consider how personal data is collected, stored, and used, and implement principles like data minimisation, purpose limitation, and user consent throughout the development process (Cavoukian, 2012). Historically, most legacy systems in sectors like healthcare or finance were not built with such privacy considerations upfront, leading to systems that had to be retrofitted for compliance with laws such as HIPAA or GDPR. This reactive approach often left gaps where sensitive data could leak or be misused. In contrast, modern frameworks encourage treating privacy requirements at the same level as functional requirements from the start of any project. For example, methodologies now include

activities like Privacy Impact Assessments (PIAs) during the design phase and the use of privacy design strategies such as aggregation, encryption, and user-centric transparency features during implementation. Recently, comparative research by Del-Real et al. (2024) found that Privacy by Design as a concept is relatively well-defined (anchored in seven principles prescribed by Cavoukian and from a rights-based legal perspective), whereas Security by Design (SbD) has been less uniformly articulated.

Nevertheless, both concepts share a common mandate that, for effectiveness, PbD and SbD should be implemented early in the development process and throughout all lifecycle stages (Del-Real et al., 2024). In practice, this means that during requirements elicitation, teams should gather privacy requirements, including what data needs to be confidential, what user control is required, alongside functional specifications. Furthermore, during design, privacy-preserving architectural patterns should be applied; during coding, secure coding and data handling guidelines should be followed; and during testing, include privacy test cases such as verifying that data retention policies are correctly implemented.

The healthcare domain illustrates both the need and challenges of Privacy by Design. Health information systems handle extremely sensitive personal data, and regulations like the EU GDPR (enforced in 2018) explicitly require data protection by design and by default. Yet integrating these principles has been anything but trivial. In a recent study on electronic health records, it was noted that many solutions “do not extensively use Privacy by Design mechanisms” and thus remain vulnerable to data leaks (Arizon-Peretz et al., 2022). Additionally, when there is a lack of privacy consideration during system design and development, the risk of privacy violations such as data breaches or unauthorised access to patient data increases significantly (Casola et al., 2020). On the positive side, there was a proliferation of privacy engineering frameworks in the 2010s, including LINDDUN for privacy threat modelling, privacy engineering objectives by NIST, and various academic PbD frameworks, which have given developers tools to systematically address privacy. The literature critically notes, however, that simply having PbD principles on paper is not enough; there is often a gap between principle and practice (Ebad, 2022; Venkataramanan & Shriram, 2016; Van Rest et al., 2014; Cavoukian, 2009). Del-Real et al. (2024) also pointed out that PbD is comprehensive in theory (covering technical and organisational measures and tying into

recognised human rights), but its successful implementation requires an interdisciplinary effort from developers, UX designers, legal and compliance teams all need to collaborate to ensure privacy is truly embedded. In many organisations, this collaboration is still developing, and privacy considerations can be overshadowed by pressing delivery deadlines or unclear ownership. Therefore, as data protection enforcement increases, it is expected that PbD will become a standard part of the SDLC vocabulary, much like SbD. However, to achieve this, more standardised frameworks and education may be required, as well as demonstrating that investing in privacy upfront prevents costly breaches and maintains user trust (Ewoh & Vartiainen, 2024).

Despite the evolution of SDLC models to now explicitly include security and privacy activities, real-world adoption of these practices is faced with challenges. One major issue is organisational and cultural resistance. For instance, development teams often view security processes as burdensome or outside their primary responsibilities, especially when management does not clearly mandate or support these efforts (Assal & Chiasson, 2019). Likewise, Maher et al. (2020) report that a lack of clear vision from top management and inadequate guidelines on how to incorporate security practices are key barriers to implementing secure SDLC in organisations. This suggests that simply publishing a secure development standard is insufficient, as organisations must also invest in creating supportive policies, training developers, and aligning security efforts with business goals. Along the same lines, resource constraints are another problem, especially in situations where smaller organisations or startups may not have dedicated security experts or the budget to perform extensive security activities at each SDLC phase. Even in larger enterprises, there may be a shortage of skilled personnel, as noted in a survey by Acar et al. (2017a), developers often need human-centred guidance and better security tooling to successfully build security in Acar et al. (2017b).

Additionally, another gap lies in the tooling and integration of security into modern development workflows. While Agile and DevOps practices emphasise speed and continuous delivery, these could conflict with traditional security gate checks (Masud et al., 2022). The rise of DevSecOps aims to reconcile this by automating security tests and embedding security staff in agile teams, but literature reviews (Myrbakken & Colomo-Palacios, 2017; Sanchez-

Gordon & Colomo-Palacios, 2019) indicate that many organisations are still in early stages of adopting DevSecOps practices. For example, integrating threat modelling into an agile sprint-based process requires adjusting the practice to be lightweight and iterative, an approach that is still alien to most security teams. Moreover, measurements of success and maturity models (like BSIMM) show that while most organisations perform some security testing (dynamic analysis, penetration testing) before release, far fewer consistently perform early-phase tasks like architectural risk analysis or code review with security in mind (Sonnekalb et al., 2022). Therefore, it can be inferred that because of this uneven implementation, vulnerabilities can still slip through the cracks if early lifecycle defences are not thorough.

Even when privacy-by-design is recognised in principle, a persistent tension exists between rich data functionality and data minimisation, requiring strong governance to reconcile competing organisational priorities. As the privacy engineering literature discusses in detail in Section 2.5, the implementation gap between PbD principles and practice remains a persistent challenge despite the availability of frameworks such as ISO/IEC 27034 and the NIST Cybersecurity Framework (Markov et al., 2023).

An important, and often underemphasized, area of secure SDLC is the human factor. Secure processes and tools are only effective if the people using them (developers, testers, and project managers) embrace and properly execute them. As Kirlappos & Sasse (2015) noted, a strictly top-down security rule can fail. Rather, empowering developers to tailor and take ownership of security in their tasks leads to better compliance. This aligns with the idea of a “security culture” within development teams. Encouragingly, there is evidence that awareness is improving. Many computer science and software engineering programs now include courses on secure coding or privacy engineering. Organisations are creating “security champion” roles within dev teams. Over the last decade, breaches in sensitive sectors (e.g., massive healthcare data breaches, IoT device hacks impacting patient safety) have served as wake-up calls. For instance, a systematic review by Ewoh and Vartiainen (2024) on healthcare cybersecurity recommends secure design processes and phasing out insecure legacy systems as top interventions to prevent breaches. They argue that health information systems must be treated as sociotechnical systems by integrating technology, people, and processes to avoid what Gómez & Heeks (2016) calls “security design–reality gaps”. In other words, purely

technical fixes are not enough; one must consider workflow, user behaviour, and organisational context when implementing secure and privacy-centric designs.

2.4 Secure Software Development Standards, Guidelines, Best Practices and Procedures

In modern software engineering, adherence to robust security standards, guidelines, and best practices is important for ensuring that software systems are secure and resilient from the outset. This section critically evaluates frameworks and guidelines across international, national (UK), and healthcare contexts, examining their contributions to secure software development. It highlights how these standards and procedures align with SbD and PbD principles, thereby integrating security and privacy considerations into the earliest stages of the development lifecycle. Through this analysis, the section highlights the important role of formalised best practices and procedures in fostering resilient and privacy-focused software systems.

2.4.1 International Standards for Secure Development

The development of secure software does not occur in a regulatory vacuum. Across international, national, and sector-specific situations, a range of formal standards and guidelines has emerged to codify the principles of security and privacy by design into actionable requirements for software development organisations. These frameworks vary considerably in scope, specificity, and enforcement mechanism. Understanding the landscape of these standards is essential not only for positioning the technical contributions of this thesis within their governance, but also for appreciating why, despite the availability of mature and well-documented frameworks, significant security vulnerabilities continue to be introduced during software development. The following sections critically examine the most relevant international, national, and sector-specific standards, evaluating their comprehensiveness, practical usability, and alignment with the SbD and PbD principles that underpin this research.

2.4.1.1 ISO/IEC 27001 – Information Security Management Systems (ISMS)

The ISO 27001 is an internationally adopted standard that provides requirements for establishing a comprehensive information security management system. Although this standard is broad in scope, it includes controls for secure development processes as part of the overall risk management perspective. For example, it requires secure development policy and technical controls in the SDLC (Weir et al., 2021). The popularity of the standard is evidenced by its widespread global adoption in over 71,000 organizations obtaining ISO 27001 certifications by 2022[3]. This extensive acceptance may be because of its role in demonstrating top-management commitment to security and how it systematically addresses risks (Jøsang, 2024). However, ISO 27001 is a management framework rather than a technical guide, so its direct guidance on software coding practices is limited. Its effectiveness relies on organizations translating high-level controls into actionable secure development procedures. Studies have noted that achieving certification can significantly improve the security posture of an organization but also point out implementation challenges, especially for smaller firms, due to the required documentation, training, and audits (Folorunso et al., 2024). Overall, ISO 27001 sets an important security-by-design tone at the organizational level, but it must be complemented by more development-centric standards to directly impact software security during development.

2.4.1.2 ISO/IEC 27034 – Application Security Controls:

ISO 27034 is a lesser-known companion to 27001, that is focused specifically on integrating security into application development and operations. It provides an Application Security Management Process and defines Application Security Controls (ASC) to address vulnerabilities. In essence, ISO 27034 offers a detailed framework for specifying and validating security controls throughout the SDLC, aligning closely with related standards such as ISO 27005 for risk management. In practice, full adoption of ISO 27034 has been limited probably

[3] International Organization for Standardization. (2023, September). *The ISO survey of management system standard certifications – 2022 – explanatory note* [PDF]. https://www.iso.org/the-iso-survey.html

because the framework is extensive and may appear complex to implement for many organizations (Ahmad et al., 2024). Even so, its concepts, such as maintaining an organizational "application security normative framework" of approved controls and best practices, can guide firms in building their own secure development programs. For example, the early compliance by Microsoft with ISO 27034-1 in 2013 demonstrated its potential as a competitive differentiator for software suppliers targeting security-conscious industries. The comprehensive nature of the standard is a double-edged sword as it is very thorough (hence comprehensive in coverage) but can be seen as cumbersome or too theoretical for developers, impacting usability. Thus, while ISO 27034 embodies the ideals of building security in, organizations often cherry-pick its relevant parts or use it as a high-level reference rather than strictly certifying against it (Markov et al., 2023). The lack of widespread certification or regulatory push means there is little external incentive for vendors to adopt it fully, a point echoed by industry experts who observe that without customer or regulatory pressure, many software suppliers have historically lacked motivation to "build secure software from the start" (Assal & Chiasson, 2019).

2.4.1.3 NIST SP 800-53 and SP 800-64 – U.S. Federal Security Controls:

In the United States, the NIST has published influential guidelines for secure systems development. NIST Special Publication 800-53 (Rev. 5, 2020) is a comprehensive catalogue of security and privacy controls for federal information systems, many of which pertain to secure software development. For example, the controls for software testing, code review, and vulnerability management are covered in this publication. These controls are organized into families and baselines, providing a rigorous checklist to ensure security considerations are addressed. The breadth of the NIST SP 800-53 makes it quite comprehensive but also challenging to implement in full because organizations often must tailor the controls to their specific context and risk profile. The real-world uptake of the framework has been significant in government and regulated sectors including finance and healthcare industries, and more often so as a mandatory baseline. However, its complexity can reduce usability for developers, compliance tends to be driven by security teams or auditors, with developers receiving distilled requirements. In a secure development context, SP 800-53 stresses the

need for practices like threat modelling, least privilege, secure coding, and continuous monitoring, reinforcing SbD principles at a control level (Kurii & Opirskyy, 2022).

Earlier, NIST SP 800-64 (initially in 2004, Rev.2 in 2008) specifically addressed security considerations in the SDLC. It was one of the foundational guidelines urging that security activities be integrated into each phase of development, from initiation through disposition. NIST SP 800-64 advocated practices such as including security requirements in system design, conducting risk assessments and code reviews during development, and planning for secure deployment and maintenance. This guidance was visionary for its time, cementing the concept that security is "built in, not bolt on." In effect, it prefigured modern DevSecOps by decades. Although now outdated, the core message of SP 800-64 remains influential and has been carried forward by more recent frameworks (Kissel, Stine & Scholl, 2008).

2.4.1.4 NIST Secure Software Development Framework (SSDF) – SP 800-218 (2022):

Recognizing the need for concise developer-focused guidance, NIST released the SSDF to distil fundamental secure development practices that can be integrated into any SDLC. The SSDF (SP 800-218) recommends a core set of high-level practices grouped into categories such as Prepare the Organization like training, toolchain security, Protect the Software like secure coding, component management, Produce Well-Secured Software like threat modelling, testing, and Respond to Vulnerabilities like patching processes (NIST, 2022). Its creation was partly motivated by Executive Order 14028 in the U.S., which highlighted software supply chain security and required guidance for software producers. The SSDF is less prescriptive than a checklist like SP 800-53. It focuses on outcomes and best practices rather than detailed controls, which arguably improves its usability for practitioners. Early industry reception has been positive, with organizations using the SSDF as a baseline to evaluate DevSecOps pipelines and secure development policies (Kudriavtseva & Gadyatskaya, 2022). By aligning with practices already advocated in industry such as those by OWASP and SAFECode[4]

[4] https://safecode.org/publication/SAFECode_Principles_for_Software_Assurance_Assessment.pdf#:~:text=the%20SAFECode%20framework%20is%20that%20a%20software,of%20a%20product's%20intended%20operating%20environment.%20%E2%80%9D

(Software Assurance Forum for Excellence in Code), the SSDF has achieved real-world credibility. Nevertheless, being a voluntary framework, its uptake depends on organizational commitment. The new software vendor guidance in the United Kingdom explicitly mirrors practices of the NIST SSDF, indicating an emerging international consensus (Gov.UK, 2025; NIST, 2022). A challenge in SSDF implementation is ensuring that each high-level practice is translated into day-to-day development team activities, and this often requires cultural changes and tool support. Nonetheless, the SSDF represents an important step in codifying SbD practices in a flexible, adoptable manner, and it exemplifies how government standards bodies are promoting secure development as a norm.

2.4.2 National and Sector-Specific Security Frameworks

Standards are also tailored to national environments and specific industries, particularly for critical sectors like healthcare, to address unique regulatory and threat landscapes. In the UK, the National Cyber Security Centre (NCSC) and government bodies have issued guidance to imbue software development with strong security principles, while healthcare regulations such as HIPAA in the US and NHS standards in Britain enforce security and privacy controls in clinical software. The following sections provide a review of these standards, principles and guidelines.

2.4.2.1 UK NCSC Secure Development Principles

The NCSC has identified eight principles for secure development and deployment, intended to help organizations "understand the security implications of modern development practices". These principles, released around 2018–2020, emphasize a security-focused culture and sound engineering habits in an era of cloud services and continuous delivery. In summary, the 8 principles of the NCSC are: *(*1) Security is the concern of everyone, promoting a culture of shared responsibility; (2) Maintain sharp security knowledge, through training and awareness; (3) Produce clean, maintainable code, since simplicity and clarity reduce vulnerabilities; (4) Secure your development environment, to protect the integrity of the build process; (5) Protect code repositories, guarding against unauthorized changes; (6) Secure the build and deployment pipeline, ensuring Continuous Integration/Continuous Delivery (CI/CD) tools themselves are hardened; (7) Continually test your security, via automated and manual

testing without hindering rapid releases; and (8) Plan for security flaws, accepting that bugs happen and establishing processes to handle and learn from them (Barclay, 2021). These guidelines are high-level and non-prescriptive by design. Their strength lies in raising the security baseline of development teams. In terms of usability, these principles are written in developer-friendly language and avoid strict mandates, which arguably makes them easier to adopt as best practices (Shukla, 2024). The trade-off is that, unlike a standard, they are not directly measurable or enforceable but rather rely on organizational commitment to follow. Real-world uptake of these principles has been notable in UK government and public sector projects, where NCSC guidance often sets expected practice, and they have influenced procurement language such as requiring suppliers to follow these principles (Montasari, 2023). However, the voluntary nature means uptake in the broader industry depends on the security culture of the organization. Still, by highlighting SbD ideals, the NCSC principles serve as a link between abstract standards and practical team behaviours (Montasari, 2023).

2.4.2.2 UK Software Security "Code of Practice" (2025):

Very recently, the Department for Science, Innovation and Technology (DSIT) UK, in partnership with NCSC, launched a Software Security Code of Practice as a voluntary framework for software vendors. This code of practice defines 14 principles grouped under four themes: secure design and development, build environment security, secure deployment and maintenance, and communication with customers (DSIT, 2025). For example, under secure design and development, vendors are expected to ensure security throughout the lifecycle, including managing risks from third-party components. Under communication, vendors should be transparent about product lifecycle, like the end-of-support for legacy versions and disclose incidents or vulnerabilities that could affect customers. Notably, the code calls for accountability at the senior leadership level of software organizations, aiming to cultivate a top-down security-by-design culture (for instance making a board-level executive responsible for software security). The Code of Practice is explicitly aligned with international best practices, as it references the NIST SSDF and anticipates upcoming EU regulations (the Cyber Resilience Act) to ease cross-border compliance (DSIT, 2025). In terms of comprehensiveness, it covers not just technical measures but also processes (like maintaining an inventory of critical components for supply chain security) and people aspects

(ensuring developers have training in secure coding). A critical evaluation point is that this code remains voluntary (FSP Law, 2025). If widely adopted, the UK Code of Practice could materially improve baseline software security (through stronger secure-by-default postures in products), but measuring its impact will take time.

2.4.2.3 Healthcare Sector Frameworks

Healthcare information systems handle sensitive personal data and thus face stringent security and privacy requirements. Two cornerstone frameworks in this domain are the U.S. Health Insurance Portability and Accountability Act (HIPAA) and the UK NHS Data Security standards, both of which compel software development practices that safeguard patient data.

2.4.2.3.1 HIPAA (Health Insurance Portability and Accountability Act):

In the United States, HIPAA (1996) and its subsequent rules (Privacy Rule of 2000, Security Rule of 2003) establish national standards for protecting electronic Protected Health Information (ePHI). The HIPAA Security Rule in particular mandates healthcare providers and their software vendors to implement administrative, physical, and technical safeguards. While HIPAA is a law and not a software development standard per se, it has profound influence on software engineering in healthcare. Compliance requires that software systems incorporate controls for data confidentiality, integrity, access control, user authentication, audit logging, and transmission security (Elkourdi et al., 2024). In practice, this means developers of EHR (Electronic Health Record) systems, health apps, or any software using patient data must follow secure development practices (like encryption of data at rest and in transit, role-based access, input validation to protect data integrity, etc.) to meet HIPAA requirements. Ensuring HIPAA compliance throughout the SDLC poses critical challenges for engineering teams, as highlighted by a recent scoping review of the literature (Elkourdi et al., 2024). These challenges include translating high-level regulatory requirements into specific security requirements early in the requirements phase; integrating privacy and security risk assessments into design; implementing the required controls without degrading usability for clinicians; and rigorous testing for security and compliance (including documentation for audits). Elkourdi et al. (2024) note that addressing these issues often requires a

multidisciplinary approach, involving not just software developers but also compliance officers, security experts, and healthcare domain specialists. A positive consequence of HIPAA is that it has effectively enforced a form of PbD in healthcare. The organizations are acutely aware that neglecting security can lead to breaches, fines, and patient harm. However, a critique is that some developers treat HIPAA as a checklist, potentially meeting the letter of the law, for instance implementing required features, without fully embracing the spirit of robust software security. Real-world uptake of secure development practices in healthcare varies, for instance large healthcare organizations and established medical software vendors tend to have formal security programs (often aligning with ISO 27001 or NIST), whereas smaller mobile health app developers have sometimes struggled, leading to notable gaps and breaches (Aljedaani & Babar, 2021).

2.4.2.3.2 NHS Data Security and Protection (UK)

Following high-profile incidents like the 2017 there on the NHS, the UK health sector reinforced its data security framework (National Audit Office, 2017). The National Data Guardian (NDG) for Health and Care issued 10 Data Security Standards (NHS Digital, 2023) that all health and social care organizations must follow. These standards, while organization-focused, include several that cascade down to software development and suppliers. Notably, Standard 10: “Accountable Suppliers” mandates that IT suppliers are held accountable via contracts for protecting patient data and meeting the NDG standards. In practice, this means NHS trusts and hospitals now demand assurances from their software vendors that software products are built and tested securely. Suppliers may be required to demonstrate adherence to secure development practices (for instance, by conforming to standards like ISO 27001, undergoing independent security testing, or showing Cyber Essentials certification) as part of contract due diligence (NHS Digital, 2024). The NHS also introduced the Data Security and Protection Toolkit (DSP Toolkit), an annual self-assessment that organizations (and some suppliers) must complete to demonstrate compliance with these 10 standards. For software developers, this environment incentivizes SbD, as insecure products could lead to failing these assessments or breach of contract. The NDG standards emphasize leadership obligation, training, minimum necessary access, prompt software patching, and incident response – all of which influence how software is built and maintained (National Data Guardian, 2017). For

example, the expectation of staying on supported software and applying security patches (Standard 5) means developers must design systems to allow timely updates and avoid obsolete components, aligning with secure DevOps practices. A sector-specific challenge here is that healthcare software often integrates with legacy systems or medical devices, which can complicate the application of modern secure development practices[5]. Nonetheless, the push from NHS England has led to improved awareness because security requirements are now frequently included from the requirements phase when NHS trusts commission new systems, and there is a higher demand for techniques like penetration testing, code review, and threat modelling in the healthcare software procurement process. The practical uptake of these standards is closely monitored, and NHS organizations risk penalties or loss of connection privileges if they do not meet the toolkit requirements. Thus, unlike the voluntary nature of some industry frameworks, the healthcare sector in the UK has effectively operationalized these guidelines through mandatory compliance processes, driving adoption.

2.4.3 Secure Coding Standards and Practices

While high-level standards and regulations set the overall agenda, day-to-day software security hinges on the practices and guidelines developers follow when writing code. Secure coding standards provide language-specific or technology-specific rules and recommendations to prevent common vulnerabilities, and community-driven best-practice guides like those from OWASP translate security principles into concrete requirements. These resources are essential to achieving the goals of the broader frameworks at the implementation level.

2.4.3.1 OWASP Application Security Verification Standard (ASVS)

The OWASP ASVS is a widely respected community standard that acts as a comprehensive checklist of security requirements for web (and increasingly, mobile or API) applications (Elder et al., 2021). Now in version 4.0 (released 2019/2020), ASVS defines sets of verifiable security controls organized into categories (authentication, access control, input validation,

[5] https://www.simbo.ai/blog/addressing-legacy-systems-in-healthcare-the-risks-and-challenges-of-outdated-technology-in-cybersecurity-3763802/#:~:text=The%20Cybersecurity%20Risks%20of%20Outdated,need%20to%20address%20potential%20vulnerabilities.

cryptography, error handling, etc.) and three ascending levels of rigor (Level 1 basic, up to Level 3 advanced/high assurance). It essentially provides developers and security testers with a clear list of "requirements for secure development", from simple items like using strong password policies to advanced measures like implementing segmentation to limit malicious impact (Wen and Katt, 2023). The ASVS is highly usable for practitioners, since it is written in clear, testable statements, making it straightforward to integrate into development checklists, security test plans, or as acceptance criteria for secure user stories. Many organizations have adopted ASVS (or a bespoke subset of it) as the basis for their internal secure coding standards. The comprehensive nature of the ASVS (with over 200 requirements) is an indication of strength in coverage, but this can be overwhelming for smaller teams if approached naively. Another challenge is keeping the verification process efficient[6]; manual verification of all ASVS items could be labour-intensive, so organizations increasingly use automated tools for dependency checking, code scanning, and other processes to cover some requirements and rely on targeted manual code review for others.

2.4.3.2 CERT Secure Coding Standards

The Computer Emergency Response Team (CERT) Division of Carnegie Mellon's Software Engineering Institute has published secure coding standards for various programming languages (C, C++, Java, Perl, Android, etc.) (SEI, 2019). These are rule sets focused on avoiding coding constructs that can lead to security bugs, such as buffer overflows, integer wraparounds, race conditions, and other vulnerability patterns. For example, the CERT C Secure Coding Standard (SEI, 2016) contains rules like "MSC32-C: Properly seed pseudo-random number generators" and "STR31-C: Guarantee that storage for strings has sufficient space for character data and the null terminator". Each rule is accompanied by examples of non-compliant code and compliant solutions. The comprehensiveness of CERT standards is notable as they codify decades of vulnerability knowledge into actionable guidelines. Many of these rules can be checked by static analysis tools, and indeed tool vendors often align their checkers with CERT rules (increasing the real-world usability as developers get

[6] https://www.nccgroup.com/improving-your-application-security-with-the-owasp-asvs/#:~:text=Customizing%20ASVS&text=The%20levels%20range%20from%20L1,to%20your%20needs%2C%20sometimes%20heavily.

automated feedback on violations). Adopting CERT coding standards can significantly reduce certain classes of bugs; for instance, one study found that teams following the CERT C rules had measurably fewer security defects in production (Tlili et al., 2009). However, implementation challenges include the learning curve for developers (because the standards documents are extensive, and developers need training to internalize the rules) and the potential impact on development velocity in cases where strict adherence might flag many issues that need fixing) (Hellwig et al., 2016). Organizations must balance these by focusing on the most critical rules and integrating the checks into continuous integration so that enforcement is consistent but not overly burdensome (Madnick, Li & Choucri, 2009).

2.4.3.3 Other Coding Best Practices

Beyond formal standards like ASVS and CERT, the industry benefits from numerous guidelines and checklists that capture best practices. The *OWASP Top 10* (most common web vulnerabilities) and *CWE Top 25* (common weakness enumeration of most dangerous software errors) are widely referenced in developer training and used as baseline objectives (Bach-Nutman, 2020; Howard, 2009). These lists are not comprehensive standards, but they serve as an entry point to secure coding by raising awareness of frequent mistakes like SQL injection, cross-site scripting, and so forth. Many organizations use the OWASP Top 10 as a minimal compliance target. For more specific domains, there are guidelines like Secure coding for iOS/Android, Cloud security configuration benchmarks, which complement general standards. Overall, secure coding practices have seen improved uptake thanks to an increase in awareness and the availability of better tools. However, studies and surveys continue to find gaps in developer knowledge. For instance, a 2024 Open Source Security Foundation (OpenSSF) survey reported that 28% of software professionals are unfamiliar with secure development practices, and paradoxically the least experienced developers reported the highest lack of familiarity (75% of those with <1 year experience) (OpenSSF, 2024). This indicates that while standards and guides exist, a key challenge is educating and training developers to use them effectively. When developers do internalize secure coding principles, they become empowered to act as a first line of defence, catching and fixing issues before they ever become incidents. Thus, secure coding standards and practices are a critical pillar

of software security, translating high-level “build secure software” directives into concrete, day-to-day coding decisions that collectively implement SbD.

2.4.4 DevSecOps and Continuous Integration and Continuous Delivery/Deployment (CI/CD) Integration

Modern software development practices have continued to move towards agility and continuous delivery in a process called DevOps, and this has raised the question of how to integrate security without slowing down the pace of new software releases. DevSecOps considered an emerging philosophy, is the practice of blending security into DevOps such that security checks and controls become a seamless part of the development pipeline (Sánchez-Gordón & Colomo-Palacios, 2020). This approach directly supports SbD by ensuring every code change is evaluated for security from the start and throughout its journey to production.

A hallmark of DevSecOps is automating security in the Continuous Integration and Continuous Delivery/Deployment (CI/CD) pipeline. This typically involves embedding tools for Static Application Security Testing (SAST), dynamic testing (DAST), software composition analysis (SCA for dependencies), container image scanning, and infrastructure-as-code scanning into the build and deployment process. For example, when a developer commits code, automated workflows can immediately run static analysis to catch common coding errors (like the use of dangerous functions or known insecure patterns). Before an application is deployed, DAST tools or interactive application security testing can exercise the running code to find issues like XSS or access control weaknesses. Open-source dependency scanners check for known vulnerabilities in libraries. If any critical issues are discovered, the pipeline can alert developers or even halt the build, preventing insecure code from progressing. This automation addresses scale and speed which are the classic challenges in secure development (Zhao et al., 2024). In traditional approaches, security review might happen infrequently, like conducting a penetration test just before product launch, which is both too late and too slow for rapid iterations that occur today. By contrast, DevSecOps aims to provide near-instant feedback to developers, aligning with the “shift-left” principle of finding and fixing vulnerabilities early when they are cheapest to fix. Research confirms that organizations that integrate security into CI/CD report improved remediation times and fewer security fire-

drills late in the cycle (Zhao et al., 2024). Importantly, DevSecOps is not only about tools, but also about culture and processes (Prates & Pereira, 2025). It requires breaking down silos between development, security, and operations teams.

Despite the rapid adoption of SecDevOps, there are implementation challenges frequently discussed in literature. One major challenge is the knowledge and skill gap among developers and DevOps engineers regarding security (Nalluri & Kaur, 2024; Assal et al., 2025). If developers are not familiar in secure coding or interpreting security tool results, the value of automation can be lost as tools might produce findings that get ignored or improperly fixed. Another challenge is tool overload and integration (Rajapakse, et al., 2021). Large organizations have reported using numerous security tools, sometimes with overlapping functionality, causing complexity in pipeline integration and result management (Rajapakse, et al., 2021). This can be mitigated by selecting tools that integrate well with existing development platforms, or by using orchestration platforms that consolidate findings. The false positive rate of automated scanners is also a concern; if a tool inundates developers with dozens of alerts that turn out not to be real issues, it can breed alert fatigue.

A further issue is aligning DevSecOps with compliance requirements (Solanke, 2022). Highly regulated industries (healthcare, finance, government) often require certain testing or approval steps that do not naturally fit into a fully automated pipeline. DevSecOps practitioners have addressed this by generating audit artifacts from pipelines (for example, keeping logs of security scans, producing reports, even automating compliance checks) so that even compliance auditing becomes a mostly automated, continuous activity rather than a periodic manual one (Solanke, 2022).

Studies like Weir et al. (2021) have shown that in organizations with mature DevSecOps, security activities like automated testing and library patching are adopted very early and widely by teams, reflecting an embedded security culture. It is important to note that while DevSecOps and CI/CD integration provide the broader development ecosystem context within which shift-left security tooling operates, the scope of this research is limited to the static analysis and development-time vulnerability detection component of that ecosystem. The research does not extend to runtime analysis, CI/CD pipeline integration, or automated deployment security, these remain complementary activities that would be required

alongside the proposed methodology in a full DevSecOps implementation. The DevSecOps literature reviewed in this section therefore serves as contextual motivation for shift-left security tooling rather than as a description of the contribution territory of this thesis.

2.5 Privacy Engineering and Privacy-by-Design Methods

In tandem with security, privacy has become an overriding concern in system design, especially with stricter data protection regulations worldwide. Privacy engineering focuses on systematically identifying and addressing privacy risks in software systems. This has given rise to specialized frameworks and techniques to ensure privacy-by-design. One prominent example is LINDDUN, a privacy threat modelling methodology, alongside related practices like data protection impact assessments and privacy design patterns.

2.5.1 LINDDUN Privacy Threat Modelling

LINDDUN is a framework created by researchers in Leuven (Deng, et al., 2011) to systematically elicit and mitigate privacy threats in software architectures. The name LINDDUN is an acronym for a taxonomy of privacy threat types: Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, and Non-compliance. It parallels the well-known STRIDE model for security threats but focuses on aspects that could lead to violations of privacy principles. In applying LINDDUN, analysts go through the data flows of the system and consider, for each component and data interaction, which of the LINDDUN threat categories might be relevant. For each identified threat, LINDDUN provides suggested mitigation strategies or privacy-enhancing technologies (PETs) that could be applied. For example, if a threat of identifiability (re-identifying an individual from pseudonymized data) is noted, a mitigation might be to apply stronger anonymization techniques or aggregation. If non-compliance with privacy regulations is a threat (such as collecting data without consent), the mitigation might involve process changes or technical enforcement of consent checks.

One of its strengths is providing a privacy knowledge base that goes beyond generic security. It accounts for things like the distinction between confidentiality (a security concept) and privacy concerns such as unlinkability or plausible deniability, which are not typically covered by security threat models. By doing so, it fills a gap, traditional security engineering

might secure data from hackers, but privacy engineering ensures the system also protects the expectations of privacy from the user. For example, minimizing data, giving users control, and transparency. A 2025 study by Sion et al. expanded LINDDUN's threat knowledge to make it more robust and reusable, which indicates an active effort in the community to keep this approach practical and effective (Sion et al., 2025).

In terms of usability and challenges, early reviews of LINDDUN noted it could be time-consuming, as threat modelling inherently requires thoughtful analysis and possibly multidisciplinary input (security, legal, business) (Van Landuyt et al., 2020). Even so, there is also a level of expertise needed to translate legal privacy requirements into technical threats and controls (Robles-González et al., 2020). The abstraction of the method is a way to help software teams think about privacy from the design phase, which is exactly what PbD mandates.

Beyond LINDDUN, PbD as a broader concept, originally articulated by Ann Cavoukian in the 1990s, is now enshrined in laws like the EU GDPR (Art. 25) which requires data protection by design and by default (Von Grafenstein et al., 2022). To operationalize these principles, organizations are adopting practices such as Data Protection Impact Assessments *(DPIAs)* early in projects (Demetzou, 2019). A DPIA is essentially a structured risk assessment focusing on personal data handling, it overlaps with security risk assessments but includes privacy criteria such as necessity, proportionality, user expectations, etc. The output of a DPIA often includes requirements for the software like data minimization features and stronger user consent flows, thus feeding into the development process (Calvi, 2019). Additionally, catalogues of privacy design patterns have been published to guide developers in implementing solutions for common privacy requirements (Calvi, 2019).

A noteworthy development is the standardization of PbD by ISO 31700 (ISO, 2023) which was introduced as the first international standard explicitly on PbD for consumer products and services. It provides high-level requirements and guidance for embedding privacy controls throughout the product lifecycle, from conception to disposal. ISO 31700 echoes many of the points that a privacy engineer would consider (like identifying privacy impacts, protecting data with appropriate techniques, and ensuring user rights can be exercised). The

emergence of ISO 31700 indicates a maturation of privacy engineering, giving organizations a template to benchmark their practices and perhaps certify against in the future (ISO, 2023).

One key challenge in privacy engineering is balancing privacy with functionality and analytical needs (Carvalho et al., 2023). For example, applying too strong anonymization might limit useful analysis of data, whereas too weak undermines privacy. Techniques like differential privacy, homomorphic encryption, and federated learning (Myakala et al., 2024) are advanced privacy enhancing techniques (PET) that try to square this circle, but they are complex and not widely understood among average developers (ICO, 2024). Therefore, part of PbD is also educating engineers and product teams about such techniques and the value of privacy. Another challenge is the often-qualitative nature of privacy risk versus the more quantitative nature of security risk (you can estimate probability of a hack, but how do you quantify risk of “user feels they are being watched and loses trust”?) (Cheng, Lee and Choi, 2019). Frameworks like LINDDUN help by providing a systematic approach, but ultimately some privacy decisions require ethical and legal judgment in addition to technical judgment.

Privacy-by-design has continued to become more relevant in procurement and product development, especially post-GDPR. A 2021 survey of European companies found that the majority had taken some steps to integrate privacy considerations in their SDLC, but only a minority had a formal methodology for it (Andrade et al., 2022). This suggests that while awareness is high, practice is still catching up. We also see privacy engineering teams being established in larger tech companies, a practice that is distinct from security teams, to focus on these issues, which is a positive sign of specialization. Privacy engineering, as a discipline, complements security engineering because while security protects against unauthorized access, PbD often guides authorized uses to be within acceptable bounds. Both are needed for a robust trustworthy system. All told, methods like LINDDUN and DPIAs operationalize PbD by injecting privacy risk thinking into early design, much as threat modelling and secure design reviews do for security-by-design (Mollaeefar et al., 2025; Obiokafor et al., 2025; Hills et al., 2025). Both face adoption hurdles, but the trajectory is that they will become as standard as security testing soon, driven by both regulatory pressure and user demand for privacy.

2.5.2 Other Threat Modelling Frameworks

As enterprises continue to grow in complexity, the cyberattack surface continues to increase and to address these threats, attack trees or graphs can be used (Xlong, Legrand, Aberg, & Lagerstrom, 2022). The established way of systematic identification and categorization of all potential security issues in a technological project is called threat modelling (Aufner, 2020). Several mature frameworks are available in the field of software development.

STRIDE, an acronym for Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege, was developed at Microsoft by Kohnfelder and Garg (1999) as part of the Security Development Lifecycle and remains, according to Aufner (2020), the most widely cited threat modelling framework, principally because of its structural simplicity. STRIDE is applied systematically against a data flow diagram: each element (process, data store, data flow, or external entity) is evaluated in turn against the six threat categories, producing a comprehensive per-element threat inventory. This DFD-driven, per-element structure is what makes STRIDE directly comparable to LINDDUN, and is discussed further below. It has been adopted widely beyond its original Microsoft context, including mobile application development and software-defined networks. However, exhaustive per-element application is time-consuming at system scale, and because the six categories were derived from security properties rather than an independent taxonomy, the method may not surface threats that fall outside conventional confidentiality, integrity, and availability concerns.

The academia also frequently cites CORAS (Shokry,et al., 2022; Gritzalis, et al., 2020; and Sahay, et al., 2023). The seven steps in CORAS are focused on meeting with clients and analysts, and while it is a risk assessment framework for threat scenarios and corresponding countermeasures, it remains subjective in the assignment of weights (Sahay, et al., 2023). Quantitative threat modeling (QTM) is another highly researched threat modelling approach (Swathi & Yogish, 2021; Singh, et al., 2022; and Cremers, et al., 2022). QTM employs mathematical models for risk assessment based on the likelihood and impact of threats. The fuzzy logic threat modelling approach uses fuzzy sets and reasoning to manage uncertaininty during the threat assessment (Malik, et al., 2020; Harirchian & Lahmer, 2020; Prasojo, et al.,

2020; Peiris & De Silva, 2021; and Fan, et al., 2022). Fuzzy logic is similar to QTM and can be used when the avaliable information is not precise while providing flexible risk scores. However, Fuzzy logic like CORAS is subjective, requires expert knowledge like QTM and intepretation of results is subjective. T-Map is another mature threat modelling tool that functions by visualizing attack paths alongside their consequences (Hu, et al., 2022). T-Map is intuititive and easy to use, it facilitates collaboration and identifies critical assets and vulnerabilities but it is not suitable for complex systems and requires complex training to understand the tool (Gao, et al., 2023).

A further widely adopted framework, though structurally distinct from the design-time methodologies above, is the MITRE ATT&CK framework (MITRE Corporation, 2013). ATT&CK is a knowledge base of adversary tactics and techniques compiled from empirically observed, real-world attack campaigns, organised as a matrix mapping attacker goals (tactics, such as Initial Access, Privilege Escalation, or Exfiltration) to the specific methods (techniques) used to achieve them. Its principal strength is that it is grounded in documented adversary behaviour rather than hypothetical threat categories, and it has become the de facto standard for threat intelligence, detection engineering, and red-team exercises, supported by extensive industry tooling and continuous community-contributed updates. ATT&CK is not suitable as the privacy threat foundation for this research for two related reasons. First, it is post-compromise and adversary-behaviour-centric because it models what an attacker does once a system has already been breached or engaged, rather than eliciting design-time risks arising from a system's own legitimate, authorised operation. This distinction matters directly for healthcare privacy, since a clinician viewing a patient record for an unauthorised purpose, or a system linking data across contexts in a way a patient did not anticipate, involves no adversary in the ATT&CK sense at all, since no unauthorised access has occurred. Second, ATT&CK provides no privacy-specific taxonomy comparable to LINDDUN's seven categories; its tactics and techniques classify attacker actions, not the categories of harm, such as linkability, identifiability, or unawareness, that privacy regulation is designed to prevent. ATT&CK is consequently a strong fit for operational security monitoring but does not offer the DFD-driven, design-time elicitation structure this research requires to connect privacy threats

to code-level weaknesses. The summary of these risk assement frameworks is contained in Table 2-1.

2.5.3 Framework Selection Rationale for Healthcare Contexts

The comparative analysis in Table 2-1 reveals that all seven threat modelling frameworks share certain practical limitations, most notably the requirement for expert judgement and the time investment required for rigorous application. Selecting among these cannot be based on the absence of limitations rather must be based on conceptual fitness for the specific problem domain. For this research, the selection criterion is explicit: the framework must be capable of identifying the categories of harm that healthcare regulatory frameworks, GDPR, HIPAA, and NHS Data Security Standards in particular, are designed to prevent, and it must operate at a level of abstraction that connects design-time analysis to the code-level vulnerability assessment that follows in the research methodology.

Applied against this criterion, STRIDE and LINDDUN represent the two most credible candidates among the frameworks reviewed. STRIDE the most widely adopted threat modelling methodology, benefits from extensive tooling support including the Microsoft Threat Modelling Tool and is well understood across the software engineering community (Das et al., 2024). However, the threat categories of STRIDE, only map to security properties rather than privacy properties. Information Disclosure is the closest STRIDE category to a privacy concern, but it addresses unauthorised access by external actors and does not capture the privacy harms that are most consequential in healthcare contexts: a clinician accessing a patient record for an unauthorised purpose, a system linking data across contexts in ways the patient did not anticipate, a process collecting data beyond what is necessary for clinical care, or a user remaining unaware that their data is being shared with third parties.

Table 2-1: Comparative Summary of Threat Modelling Frameworks: Key Features, Strengths, and Weaknesses

Framework	**Full Name / Basis**	**Key Features**	**Strengths**	**Weaknesses**
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of	Categorises threats across six attack types; applied against system components via data flow diagrams	Well-structured and easy to understand; widely adopted across domains including mobile	Time-consuming to apply comprehensively; may not identify all threat types, particularly privacy-

Framework	Full Name / Basis	Key Features	Strengths	Weaknesses
	Service, Elevation of Privilege		and software-defined networks; extensive tooling support (e.g., Microsoft Threat Modelling Tool)	specific threats; limited quantitative risk assessment capability
MITRE ATT&CK	Adversarial Tactics, Techniques, and Common Knowledge	Matrix of empirically observed attacker tactics and techniques compiled from real-world campaigns; used for threat intelligence and detection engineering	Grounded in documented real-world adversary behaviour; extensive industry tooling and continuous community updates; de facto standard for SOC/detection use	Post-compromise, adversary-behaviour-centric rather than design-time; no privacy-specific taxonomy; does not model risks arising from legitimate, authorised system operation
LINDDUN	Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of Information, Unawareness, Non-compliance	Privacy-focused threat taxonomy applied systematically to data flow diagrams; maps threats to privacy-enhancing technology mitigations	Explicitly addresses privacy threats overlooked by security-oriented frameworks; well-suited to regulated domains such as healthcare; empirically validated	Time-consuming; requires multidisciplinary expertise spanning technical, legal, and organisational domains; less mature tooling ecosystem than STRIDE
CORAS	Construct a platform for Risk Analysis of Security-critical Systems	Seven-step risk assessment process built around client-analyst collaboration; produces threat scenarios with corresponding countermeasures	Structured collaborative process; produces detailed threat scenarios; well-documented methodology	Subjective in the assignment of risk weights; resource-intensive; relies heavily on expert judgement, limiting reproducibility
QTM	Quantitative Threat Modelling	Employs mathematical models to assess risk based on quantified likelihood and impact values	Objective and reproducible risk scores; supports prioritisation decisions with numerical evidence; reduces reliance on	Requires significant domain expertise to parameterise models accurately; obtaining reliable probability estimates for threat likelihood can be difficult in practice

Framework	Full Name / Basis	Key Features	Strengths	Weaknesses
			subjective judgement	
Fuzzy Logic Threat Modelling	Fuzzy sets and reasoning applied to threat assessment	Manages uncertainty in threat assessment by assigning flexible, graduated risk scores rather than binary classifications	Useful when available information is imprecise or incomplete; accommodates uncertainty more naturally than crisp quantitative models	Subjective in the definition of membership functions; requires expert knowledge to construct fuzzy rules; interpretation of results can be inconsistent across analysts
T-Map	Threat Map	Visualises attack paths alongside their consequences; identifies critical assets and potential vulnerabilities through graphical mapping	Intuitive and visually accessible; facilitates team collaboration; effective for identifying critical assets and attack paths in moderately complex systems	Not suitable for highly complex systems; requires specialised training to use effectively; limited quantitative risk assessment capability

These harms are not failures of security, but they may occur within fully authenticated, authorised, technically compliant systems. However, they are precisely the harms that GDPR Article 5 data minimisation principles, HIPAA minimum necessary standards, and the NHS Caldicott Principles are designed to prevent.

The seven threat categories of LINDDUN map directly to these regulatory obligations. Linkability corresponds to GDPR's purpose limitation principle. Identifiability corresponds to anonymisation and pseudonymisation requirements under both GDPR and HIPAA. Unawareness corresponds to transparency and consent obligations. Non-compliance provides an explicit regulatory alignment category that has no equivalent in STRIDE. This is not a superficial correspondence: Wuyts and Joosen (2015) demonstrate empirically that LINDDUN-based analysis surfaces privacy threats that STRIDE-based analysis systematically misses on the same system architecture, precisely because the threat categories are designed for different harm models.

The selection of LINDDUN over STRIDE is therefore not a choice between two frameworks that address the same problem with different efficiency characteristics. It is a choice between

a framework that addresses security threats and a framework that addresses privacy threats. Given that the regulatory obligations governing healthcare software development are primarily privacy obligations and given that the research problem identified in Chapter 1 concerns the integration of privacy threat analysis with software vulnerability assessment, LINDDUN is the only candidate framework from Table 1 that is conceptually fit for purpose. STRIDE, CORAS, QTM, Fuzzy Logic, and T-Map are each appropriate for security-oriented threat analysis within their respective scope boundaries, but none provide the privacy threat taxonomy that the enhanced methodology developed in this thesis requires as its foundation.

It should be noted that LINDDUN and STRIDE are not mutually exclusive in practice, organisations conducting comprehensive threat modelling may apply both frameworks to capture the full spectrum of security and privacy risks. The present research applies LINDDUN exclusively because its scope is the privacy-security integration gap identified in Chapter 1, and because the LINDDUN-to-CWE mapping developed in Chapter 3 requires a privacy threat taxonomy as its input. Extending the methodology to incorporate STRIDE-originated threats alongside LINDDUN-originated threats is identified as a direction for future work in Chapter 6.

2.5.4 Software Assurance Maturity Models (SAMM and BSIMM)

While specific standards and practices focus on what to do for security, software assurance maturity models address how well an organization implements those practices and provide roadmaps for improvement. They are essentially frameworks for assessing the depth and consistency of security activities across software development processes of an organization. Two influential models in this space are OWASP Software Assurance Maturity Model (SAMM) and the Building Security in Maturity Model (BSIMM).

2.5.4.1 OWASP Software Assurance Maturity Model (SAMM):

SAMM is an open framework that helps organizations assess and improve their software security posture in a structured way. It defines a set of security practice areas grouped into five business functions namely: Governance, Design, Implementation, Verification, and Deployment in SAMM v2.0 (Fucci et al., 2024). Here each practice area has up to three maturity levels. For instance, under Implementation, one practice is Secure Build Process; at

maturity Level 1, an organization might have basic version control and build scripts, while at Level 3, they have fully integrated security scanning and signed builds with an auditable trail. The model is prescriptive. It suggests that organizations mature by progressing through levels of each practice, and it provides guidance on what measures characterize each level. The usability of SAMM comes from its self-assessment model where organizations can evaluate themselves by using questionnaires or scorecards provided by the model to determine their current maturity, set target maturity levels based on their risk profile, and then follow the SAMM guidance to reach those targets (Gutfleisch et al., 2023). For example, a company might discover that its Training practice is at Level 1 (ad-hoc training) and decide to improve to Level 2 (role-specific security training) within a year, with concrete steps outlined by SAMM. The model also emphasizes that not every organization needs to reach the highest maturity in all practices; the goal is to be "good enough" in each area according to organizational risk and business needs (Fucci et al., 2024).

The comprehensiveness of SAMM is evident as it covers everything from policy and metrics at the governance level to technical activities like threat assessment and security testing. This breadth ensures that organizations consider a holistic program, not just piecemeal practices. Implementing SAMM can be challenging initially, as it requires honest introspection and cross-department collaboration to gather evidence of practices. There can be a tendency to overestimate maturity without objective data. To counter this, OWASP provides an assessment toolkit and even a Benchmark project where companies (anonymously) contribute their SAMM scores for comparison. One of the challenges noted by Gutfleisch et al. (2023) is sustaining management support for the improvement journey. The initial assessments and goal setting are useful, but achieving higher maturity often requires investment (time, budget for tools, training) and organizational change, which must be driven by leadership. If leadership loses interest, initiatives like assigning security champions or refining secure SDLC processes can stall.

2.5.4.2 Building Security in Maturity Model (BSIMM):

In contrast to the prescriptive approach of SAMM, BSIMM takes a descriptive, data-driven approach. BSIMM is often described as "the world's largest software security study", as it is based on observations of many organizations' security activities (Fucci et al., 2024). The

creators of BSIMM (originally from Cigital, later Synopsys) have assessed dozens of large firms starting around 2008, cataloguing what security practices those firms have in place. The BSIMM framework, currently in its 11th/12th iteration, consists of a set of 12 practice areas grouped into four domains (Governance, Intelligence, SSDL Touchpoints, and Deployment) (Weir et al., 2022). Example practice areas include Compliance and Policy, Architecture Analysis, Code Review, Penetration Testing, Configuration Management, etc. Within each practice, BSIMM describes specific activities and notes how many of the studied organizations perform them, thereby indicating what is common or uncommon.

A major finding from BSIMM data is that there is a set of foundational activities that most organizations tend to adopt first and do most frequently. Weir et al. (2021) analysed 12 years of BSIMM data and identified that about a dozen activities are *"*adopted early, together, and notably more often than any others*"* by development teams. These often include things like security requirements identification, code review (with security in mind), penetration testing, and security training for developers. These can be seen as the baseline of software security initiative. Other activities like creating a software bill of materials or building custom fuzz testing harnesses might come later as the program matures. This insight is valuable for planning roadmaps: it suggests focusing on those core practices first to build a solid foundation.

In terms of usability, BSIMM is less of a step-by-step guide and more of a mirror. It doesn't tell you how to improve per se, rather it tells you what others are doing (Jaatun, 2017). However, that information is gold for security managers making the business case for resources. For instance, if a CISO can show that 80% of business peers or competitors have a dedicated software security group and the company does not have one or in another scenario that most similar sized companies do threat modelling on key apps and the business currently does not, these can justify the business case for investment. Although BSIMM is comprehensive, it is not exhaustive as it deliberately focuses only on activities observed in practice. If no one in the BSIMM community is doing something (like, say, a very novel practice), it will be unavailable in BSIMM, which some argue might cause overlooking emerging practices. On the other hand, BSIMM is updated regularly as new practices gain

traction (for instance, activities related to cloud security and containerization have been added over time as those became prevalent).

One limitation is that BSIMM data skews towards larger organizations with established security programs, it was reported that a significant portion of BSIMM firms is in the Forbes Global 2000. Therefore, the "average" BSIMM practice might be aspirational for small and medium businesses. BSIMM authors acknowledge it's not a one-size-fits-all, but rather a reflection of "what the big players do." For smaller organizations or those just starting in software security, BSIMM can still be instructive, but they might reach a plateau that is suitable for them but below BSIMM average on some activities due to scale.

Interestingly, when comparing SAMM and BSIMM, it can be observed that these two models are not competitors but are often seen as complementary. SAMM can be used to plan and execute improvements (a roadmap), while BSIMM can be used to measure and benchmark (a yardstick). Indeed, a case study could use SAMM to structure their security program rollout and then use BSIMM to see how they stand relative to others at intervals.

One challenge with maturity models is the risk of a *"*check-the-box*"* mindset, focusing on achieving a maturity level or matching an industry norm rather than focusing on security outcomes (Heiland et al., 2013). To mitigate that, organizations should remember that the goal is risk reduction; maturity scores are just a means to gauge if processes likely lead to risk reduction. When used wisely, maturity models help ensure that as organizations grow or technology evolves, the security practices keep pace in a structured way.

2.5.5 Alignment with Security-by-Design and Privacy-by-Design Principles

All the discussed standards, practices, and models have a unifying theme which is the emphasis on SbD and PbD. These principles, broadly defined, mean that security and privacy are not reactive add-ons, but proactive considerations baked into the system from the earliest stages and maintained throughout the lifecycle (Cavoukian, 2010). The literature and frameworks reviewed demonstrate a clear trend that from international standards to DevSecOps pipelines, the goal is to shift security and privacy leftward, by embedding them in culture, design, code, and processes by default.

However, aligning with these principles often means addressing usability vs. security trade-offs head-on during design. A system that is secure by default might enforce strong passwords or multifactor authentication, which could affect user convenience which then tasks designers to look for creative solutions to maintain usability (like single sign-on to reduce login fatigue). Privacy by design might mean anonymizing data used for analytics, which could reduce data utility, and here engineers then explore advanced techniques to still extract insights without raw personal data. The core idea is that these considerations are part of the design criteria, just like performance or cost.

Additionally, another challenge is that SbD and PbD are sometimes perceived as abstract principles. Nevertheless, the work in standards and frameworks has been able to translate them into concrete checkpoints and activities. For instance, threat modelling at the design stage, which is a concrete activity, is a manifestation of security by design, because it influences the design to address threats. Similarly, having a privacy review as part of the architecture sign-off is a concrete way to implement privacy by design. Maturity models ensure that these activities happen consistently, for example, at higher maturity, no project goes forward without a threat model and security sign-off, which institutionalizes the practice.

2.6 Healthcare Information Systems: Architecture, Privacy, and Security

Healthcare information systems are made up of a complex architecture of electronic health records (EHRs), clinical applications, medical devices, and communication networks. The key components include:

Electronic Health Records (EHRs): These are the central repositories of patient data containing information such as demographics, medical history, diagnoses, imaging, etc., and it is often linked to practice management or hospital information systems. EHR databases store highly sensitive protected health information (PHI), making confidentiality and integrity paramount (Oh et al., 2021). Typically, access to EHR data is via clinical applications used by healthcare staff.

Interoperability Interfaces: Standards like HL7 v2/v3 and Fast Healthcare Interoperability Resources (FHIR) enable data exchange between systems. For instance, laboratory systems,

pharmacies, and regional health information exchanges (HIEs), all synchronise data with each other to function. These interfaces orchestrate patient data flow across disparate applications. However, older HL7 implementations often lack built-in security but rather rely on network-level safeguards, and modern FHIR APIs can introduce new risks if improperly secured (Gazzarata et al., 2024; Ewok & Vartiainen, 2024). Alhammad et al. (2024) found that many third-party health apps and aggregators implementing FHIR had authentication/authorisation vulnerabilities, exposing millions of patient records. Notably, while the FHIR standard itself is not flawed, misconfigured FHIR API endpoints are a weak link, especially once data moved to apps outside the control of the providers (Knight, 2021). This exemplifies the third-party API risk, which is when patient data that is being shared with external apps falls outside HIPAA protection and becomes vulnerable if those apps lack rigorous security (Alhammad et al., 2024).

Medical Imaging and PACS: The Picture Archiving and Communication Systems (PACS) store radiology and imaging data (such as X-rays, MRI scans, etc.) using the Digital Imaging and Communications in Medicine (DICOM) format. Here, the PACS servers' interface with the EHRs and imaging modalities to make images available to clinicians. Privacy concerns here include the large volume of identifiable images and metadata (Rempe et al., 2025). In practice, misconfigurations have led to massive breaches. Likewise, studies between 2019 and 2020 discovered that hundreds of PACS servers worldwide were exposed to the internet without any authentication protocol, collectively leaking millions of images and patient records (Eichelberg et al., 2020). Hence, attackers who exploit DICOM/PACS vulnerabilities can retrieve or even manipulate imaging data. Furthermore, the availability of PACS is also critical because ransomware attacks targeting PACS can paralyze diagnostic services.

Connected Medical Devices and IoMT: Monitors, infusion pumps and wearables are some networked devices that generate or transmit patient data which modern healthcare relies on. These so called “Internet of Medical Things” (IoMT) devices feed data into EHRs or cloud platforms for remote monitoring and telehealth. They broaden the attack surface of the healthcare system, as each device and wireless link could be a point of compromise (Svandova & Smutny, 2024; Rasool et al., 2022). Over the past decade, at least 1.5 million medical devices were compromised due to software vulnerabilities and connectivity flaws (Oh et al., 2021).

Additionally, many devices run legacy or unpatched software, so an attacker gaining access could potentially alter readings or pivot into hospital networks (Harikrishnan, 2025).

Telehealth Platforms and Mobile Health Apps: Telemedicine systems including those that are used for video consultations, remote diagnostics, etc., and patient-facing mobile apps became mainstream, especially during the COVID-19 pandemic (Shaver, 2022). These platforms extend data flows beyond traditional clinical settings. For instance, streaming health data from the home of a patient to healthcare providers in remote locations. This shift offers convenience but raises new privacy concerns because the data traverses the public internet, patient-held devices may lack enterprise-grade security, and session data (such as videos, chat logs) must be protected (Ritunga et al., 2025; Houser et al., 2023). The rapid adoption of telehealth was accompanied by a spike in cyberattacks. The FBI reported a ~400% increase in attacks in early 2020 as healthcare moved online (Yeo & Banfield, 2022). Thus, securing telehealth involves not only encryption of communication but also robust identity verification to prevent eavesdropping or fraud.

A systematic review by Aljedaani et al. (2020) revealed that over half of the studies identified limited security knowledge of the developers and the lack of clear guidelines as primary contributors to vulnerabilities in mHealth software. Such weaknesses are concerning because the compromise of a single insecure application could jeopardize both its own data and that of interconnected systems, for instance, the theft of app credentials could grant unauthorized access to cloud-based patient records. Furthermore, numerous mobile health applications operate outside traditional regulatory frameworks (for example, not qualifying as covered entities under HIPAA), thereby placing the onus of privacy protection predominantly on the diligence of app developers. Recent assessments have found that a substantial proportion of mobile health applications either lack privacy policies altogether or share user data ambiguously (Tangari et al., 2021).

Relating to all these are cloud-based services and data repositories which healthcare increasingly utilizes. Cloud storage of health records, cloud-hosted patient portals, and Software-as-a-Service for hospital operations are common. While cloud providers offer scalability and advanced security features, misconfigurations or improper use can expose massive amounts of data (Mehrtak et al., 2021). For example, a misconfigured cloud database

or storage bucket can be left publicly accessible, as has occurred in incidents where lab results or medical images were accidentally exposed online. The integration of cloud APIs and third-party services also introduces supply chain risks, such as an attack on a less-secure partner system can cascade into a healthcare network. In 2023, over half of the individuals affected by healthcare data breaches were impacted via attacks on third-party business associates rather than directly on providers (American Hospital Association, 2023). This demonstrates that the healthcare attack surface is not confined to the perimeter of one organisation, but it involves vendors, contractors, and interconnected systems.

Therefore, the modern healthcare environment is a vast, interconnected digital landscape, where numerous points of entry exist for adversaries. Hospitals and clinics must now defend not just a single network, but a constellation of EHR systems, medical devices, telehealth platforms, mobile apps, and cloud services. Each component, if not adequately secured, can be the weak link that attackers exploit. The complexity and breadth of this ecosystem amplify the challenge of protecting patient data, as security teams must address everything from encryption of medical imaging data to secure coding of health APIs, to firmware updates on surgical robots. This expanding attack surface underscores the need for comprehensive and proactive security strategies tailored to unique context of healthcare (Ewoh & Vartiainen, 2024). It also reinforces why privacy risks can no longer be seen as isolated technical issues because they are systemic, spanning multiple technologies and stakeholders across the healthcare continuum.

2.6.1 Privacy Implications of Healthcare Components

Each of the above-described component carries specific privacy challenges. EHR databases for instance, are rich targets for unauthorized access and data breaches due to the concentration of PHI. Furthermore, interoperability APIs, if breached, can leak longitudinal patient records at scale. While imaging data can inadvertently reveal user identity from faces in medical images or the unique anatomy and must be de-identified before secondary use. Even the IoMT devices raise concerns of linkability, when an attacker can correlate device data with patient identity, and detectability which is the mere observation and deduction that since a person uses a certain medical device, they have a known health status. Additionally,

telehealth now blurs the boundaries of data control, requiring strong consent and secure data handling across jurisdictions.

To systematically address these concerns, privacy threat modelling frameworks such as LINDDUN have been applied in healthcare system design. LINDDUN provides a structured method to analyse data flows for privacy threats (Wuyts & Joosen, 2015). Unlike traditional security threat models emphasize integrity/availability threats (spoofing, tampering, etc.), LINDDUN focuses explicitly on privacy and compliance issues (Badreddine et al., 2025). Therefore, by examining the data flow diagram (DFD) of a system through these lenses, analysts can uncover nuanced privacy weaknesses often overlooked by general security analysis (Ameh et al., 2025).

Take for instance, an EHR module in a DFD of a typical hospital. Using LINDDUN, one identifies that patient data is shared among departments and perhaps with external labs and/or insurers. This raises 'Linkability' concerns because the data from disparate sources could be linked to build a fuller profile of a patient than intended. Which begs to questions, if the identities of patient are pseudonymized for research, could an adversary still correlate records and re-identify individuals? The 'Identifiability' aspect of LINDDUN analysis addresses this by examining anonymization techniques. Indeed, an analysis by Ameh et al. (2025) demonstrated how anonymized health data could be vulnerable to re-identification through correlation attack. They also considered 'Disclosure of information' threats which are the unauthorized access to EHR data through various vectors such as SQL injection, compromised credentials, etc, leading to confidentiality breaches. In a comprehensive threat model, these findings are tabulated as privacy "misuse cases" tied to specific system elements.

Of significance, the LINDDUN-based threat analysis can be mapped to known software vulnerabilities to connect high-level privacy risks and low-level technical flaws. In the above EHR example, a Linkability threat (linking patient records across systems) might correlate with implementation weaknesses like insufficient access control or cleartext identifiers. In fact, Ameh et al. (2025) mapped identified software related privacy threats from the healthcare information system DFD threats to Common Weakness Enumeration (CWE) for software. This meticulous LINDDUN-to-CWE mapping creates a taxonomy of privacy weaknesses, showing how an abstract threat (like re-identification of patient data) could stem from concrete coding

issues (for instance, the use of predictable record identifiers, inadequate data masking). Thus, by establishing such links, researchers can leverage the extensive CWE database to find and fix the root causes of privacy threats in software (Ameh et al., 2025). In essence, LINDDUN guides the what-can-go-wrong analysis for privacy, and CWE enumerates the how it can go wrong in code. This represents the state-of-the-art in privacy-aware secure design by embedding privacy threat modelling into the software development lifecycle and connecting it to vulnerability mitigation.

Despite such advances, significant challenges and gaps remain in protecting healthcare information systems. One challenge is the insider threat. Healthcare data breaches are frequently caused by insiders, and this is often unintended. Insider misuse can range from the curious staff probing on records to negligent handling of data. An exploratory analysis of U.S. healthcare breaches (2015–2020) found that a vast majority of compromised records (73% of ~141 million records) were due to unintentional human factors including carelessness, lost devices, misdirected emails, rather than external attacks (Yeo & Banfield, 2022). For example, employees emailing patient files to the wrong recipient or losing an unencrypted laptop can leak tens of thousands of records. Such errors accounted for more incidents than overt malicious insiders in that study (ibid.). Nonetheless, malicious insiders remain a serious concern to because the same study documented 217 insider wrongdoing incidents such as staff deliberately accessing PHI without need, affecting over 55 million records (Yeo & Banfield, 2022). Classic security controls (such as firewalls, intrusion detection) offer little or no defence if an authorized user abuses their access (ibid). This underscores a gap in current practice where technical measures must be paired with administrative safeguards like strict access policies, monitoring of user activity, and staff training. Indeed, many healthcare breaches could be prevented by improving user authentication, enforcing least privilege, and fostering a privacy-centric culture among employees (Basil et al., 2022).

Another persistent threat is ransomware, which has reached epidemic levels in healthcare. Hospitals have been hit by ransomware attacks that encrypt EHR databases and PACS archives, crippling operations until a ransom is paid. Ransomware not only threatens availability of care but often leads to privacy breaches as attackers usually exfiltrate data before encryption. Basil et al. (2022) observe that the impact of ransomware in healthcare

goes beyond financial loss; patient safety can be directly jeopardized. Sobering statistics shows that 15% of all ransomware incidents globally in 2017 targeted healthcare, and half of hospital cybersecurity incidents that year involved ransomware (Eichelberg et al., 2020). This trend exposes a gap in preparedness as many healthcare providers lag in basic cyber hygiene including patching, backups, and incident response drills, making them "low-hanging fruit" for ransomware gangs. Efforts like network segmentation, up-to-date offline backups, and continuous staff phishing awareness training are essential mitigations but are usually not uniformly implemented.

Furthermore, the API and third-party risk are a newer challenge arising from health data interoperability initiatives. As regulations push for patient data portability, such as the U.S. ONC rules under the 21st Century Cures Act (Gordon & Mandl, 2020), healthcare systems are opening APIs for apps to connect. In theory, this empowers patients but in practice, it expands the threat landscape. Third-party apps may not uphold the same security standards as a hospital. As mentioned, by independent research[7], aggregators bridging EHR data for thousands of providers had critical 'auth' flaws, allowing one account to illicitly fetch millions of records. Furthermore, when patients transmit their data to a consumer app, that data may fall outside the scope of HIPAA, which means the app is not legally required to safeguard it to the same degree (Kim & Sittig, 2020). This regulatory gap creates confusion about accountability. Healthcare organizations must thus vet third-party integrations carefully and educate patients on choosing reputable apps. Nevertheless, achieving end-to-end privacy assurance in an ecosystem of interconnected services remains an open problem.

From a regulatory perspective, laws like HIPAA and the GDPR provide important privacy baselines. HIPAA mandates administrative, technical, and physical safeguards for PHI, and requires healthcare providers to notify breaches, facing heavy fines for non-compliance. (Since the HIPAA Privacy Rule took effect, U.S. regulators have imposed over $130 million in

[7] https://www.healthdatamanagement.com/articles/hl7-responds-to-concerns-about-security-concerns-about-fhir

penalties for data breaches and violations[8]. GDPR, with its broad protections for personal data, likewise enforces principles of data minimization, consent, and the "right to be forgotten" backed by fines up to 4% of global turnover. These regulations have driven improvements in healthcare system security by compelling encryption of data at rest and in transit, audit trails for access, and privacy impact assessments for new systems (Oh et al., 2021). They also motivate adoption of privacy by design, thus, integrating privacy considerations from the early stages of software development. However, compliance does not equal security because many breaches occur in ostensibly compliant organizations, due to sophisticated attacks or human error. A critical view in recent literature is that regulations provide a framework, but healthcare entities often treat compliance as a checklist rather than actively evolving their security posture (Basil et al., 2022). For instance, simply encrypting a database, which is a HIPAA addressable safeguard, may not be enough if an adversary can phish the credentials of a system administrator to access it. Thus, regulatory pressure must be coupled with continuous risk management.

2.7 Application Security Testing

Introducing security practices at the design phase of the SDLC not only enhances the overall quality of the final software product but it also ensures that security and privacy requirements are achieved at every stage (Núñez, et al., 2020). In addition, it reduces the cost and time wasted in frequently applying security patches and updates because even if it is impossible to account for zero-day attacks, resilience would have been built into an extent during the SDLC (Mohino, Higuera, et al., 2019).

Considering how important it is to resolve the issue of software vulnerability and discovery; several academic and industry experts have proposed different solutions. These solutions are not holistic (either a lack of soundness or completeness, or both) (Ghaffarian &

[8] https://ahisp.ahima.org/Page/human-factors-in-electronic-health-records-cybersecurity-breach-an-exploratory-analysis#:~:text=consequences%20included%20penalties%20ranging%20from,compliance.%5E%7B16

Shahriari, 2017). During the development process, static and dynamic code analysis are popular.

Application security testing in software development traditionally relies on two complementary approaches. Static Application Security Testing (SAST) involves scanning source code (or binaries) for known vulnerability patterns without executing the program (Livshits & Lam, 2005; Jovanovic, Kruegel & Kirda, 2006). This approach enables early detection of issues during development. For example, flagging dangerous API calls or insecure data flows before the application runs (Nguyen et al., 2024). In contrast, Dynamic Application Security Testing (DAST) evaluates an application in an executing state, typically by probing a running system or web application with simulated attacks and inputs to find exploitable conditions (Singh et al., 2024; Kaixuan et al., 2023). Dynamic analysis can uncover runtime vulnerabilities that static analysis might not detect, such as misconfigurations or authentication logic flaws, by observing actual system behaviour under attack scenarios (Singh et al., 2015; Nguyen et al., 2024).

However, each method has inherent strengths and weaknesses. While static analysis is useful in detecting errors early in the development process, but may miss runtime errors, dynamic testing on the other hand provides real time verification but may introduce additional workloads leading to disruptions in system operations if not carefully managed (Nguyen et al., 2024). Interestingly, current best practice recognizes that a combination of both SAST and DAST yields the most comprehensive coverage of security issues (Nunes et al., 2025). Indeed, hybrid approaches and interactive application security testing frameworks now attempt to integrate static and dynamic techniques to capitalize on the advantages of each while mitigating their individual limitations (Seth et al., 2023; Aggarwal & Jalote, 2006). A useful instance of such an integration is when static code analysers are used to guide fuzz testing by identifying critical inputs or paths, and instrumentation during dynamic tests can feed information back to refine static analysis models in an interplay that research shows can improve vulnerability discovery rates and reduce false positives (Barakat et al., 2024; Guan et al., 2017). Largely, both approaches are essential and complementary in modern secure software development lifecycles, aligning with the defence-in-depth principle of catching issues at multiple stages.

Over time, static analysis for software security has evolved over the decades from basic lint-like tools to sophisticated analysers employing data flow analysis, abstract interpretation, and even machine learning (Pargaonkar, 2023). Early work demonstrated the use of static code analysis for finding security bugs in tools that could detect SQL injection or cross-site scripting flaws in web applications by tracing tainted data flows (Jovanovic et al., 2006). Interestingly, these academic prototypes and early commercial tools, such as Fortify SCA and FindBugs in the 2000s, established that many common vulnerabilities could be automatically identified in source code (Ayewah & Pugh, 2008). Over time, static analysers have improved in precision and scale by incorporating context-sensitive analysis, interprocedural control flow, and frameworks for different languages, yet have historically suffered from high false-positive rates and coverage gaps (Johnson et al., 2013). However, recent research has aimed to push static analysis to the next level by leveraging artificial intelligence. For example, machine learning and deep learning models have been trained on code to detect vulnerability patterns or predict risky code segments (Shen & Chen, 2020; Li, Zhao & Zhang, 2018). These data-driven approaches can augment traditional static analysis by potentially reducing noise or catching more subtle flaws, though their effectiveness is still an active research area. State-of-the-art static analysis tools now often integrate into development environments and CI/CD pipelines, providing real-time feedback to developers (Luo et al., 2021). This approach of embedding SAST during the early stages of the development process, is a cornerstone of DevSecOps practices (Akbar et al., 2022; Morales et al., 2020). Nevertheless, despite technical advancements, static analysis remains an imperfect art that attempts to balance thoroughness with precision, and critical design-level vulnerabilities remain elusive to purely static techniques.

Dynamic analysis techniques have likewise advanced. Traditionally, dynamic testing was associated with manual penetration testing. This was because security experts actively probed an application using fuzz testing, in a process that was first introduced in the 1990s to feed random or malformed inputs to programs (Miller et al., 1990). Early dynamic tools, such as web vulnerability scanners, automated some of this process by crawling web applications and injecting common attack patterns including SQL injections, XSS payloads, etc. Over the past decade, dynamic application security testing has been bolstered by significant

innovations like coverage-guided fuzzing, which intelligently evolves inputs to maximize code coverage and uncover deeper flaws (Li et al., 2019). Modern fuzzers and dynamic analysers often integrate instrumentation like sanitizers or monitors, to catch memory errors, concurrency issues, and other runtime anomalies that static analysis cannot easily infer (Yang et al., 2025; Kong et al., 2024). Moreover, according to Sherin et al. (2023), dynamic web testing tools have grown more sophisticated in handling modern web technologies. For instance, dealing with single-page applications, authentication, and stateful sessions (Alazmi & De Leon, 2022). The state-of-the-art in DAST includes both open-source tools (like OWASP ZAP, which is widely used in industry pipelines) and commercial suites that automate complex attack scenarios (Alazmi & De Leon, 2022). Hybrid approaches are also emerging as state-of-the-art. One such hybrid approach is Interactive Application Security Testing (IAST) tools which hook into a running application (often in a test environment) to analyse executed code paths in real-time, by essentially combining static code knowledge with dynamic monitoring (Pan, 2019). This hybrid method can observe an application from "within," potentially yielding higher accuracy (fewer false alarms) by validating findings at runtime, and better coverage by analysing code that static methods might flag but confirming them through execution (Pan, 2019). Another frontier is the use of runtime application self-protection (RASP), where instrumentation lives in production to detect and block attacks in real time, thereby blurring the line between testing and active defence (Seth et al., 2025).

Dynamic application testing has moved toward greater automation and intelligence in security testing, demonstrated by continuous fuzzing platforms (for example OSS-Fuzz) that have discovered thousands of real-world vulnerabilities (Böhme et al., 2020). Despite these advances, dynamic testing techniques still face challenges in scaling to large, complex systems and in reliably triggering certain classes of vulnerabilities (Li et al., 2018).

2.8 Machine Learning in Static Application Security Testing (SAST)

Traditional SAST tools are largely rule-based, relying on manually crafted patterns and heuristics (regexes or data-flow rules) to catch known security issues (Chess & McGraw, 2004). This approach can be effective for well-understood vulnerability patterns, but it has notable limitations (Croft et al., 2021). In practice, rule-based scanners often produce many

false positives (flagging safe code as vulnerable) or false negatives (missing real flaws) when their rules are not perfectly tuned (Kang et al., 2022). Maintaining these rules requires a lot of time and effort, and any gap in the rule set may leave zero-day vulnerabilities undetected. Machine learning (ML) based SAST has emerged to address these gaps by learning vulnerability patterns from data instead of relying solely on human-defined rules. The promise is that ML can potentially analyse code semantics more flexibly and reduce the noise of irrelevant findings. However, adopting ML also brings challenges in model training, explainability, and integration into development workflows, which this subsection examines critically (Hüther et al., 2023).

2.8.1 Rule-Based vs. ML-Based SAST Approaches

Traditional static analysers such as Fortify, Checkmarx, SonarQube, use a database of rules or patterns to detect insecure code (Ayewah & Pugh, 2010). They excel at finding well-known issues including the use of dangerous APIs or coding errors that match a *CWE* pattern and provide explicit reasoning or the rule for each alert. However, these tools notoriously report a high volume of alerts that turn out to be benign in practice by producing high false positive rates. The fundamental problem is that without actual execution, the tool often lacks context to know if a suspicious pattern is truly exploitable or not. Conversely, purely rule-based tools may also miss subtle vulnerabilities leading to false negatives, if no rule captures the specific vulnerability pattern. A typical case may be when rule might flag any use of 'strcpy' as dangerous (to catch buffer overflows), but a particular usage might be in a context where it is actually safe, yielding a false alarm. Conversely, an unsafe code pattern that does not match any known rule will slip by undetected. As Yang et al. (2019) puts it, SAST results require extensive tuning and validation to be useful.

Machine learning offers a data-driven alternative where models learn to classify code as vulnerable or not based on examples, potentially discovering complex patterns that manual rules might not cover (Mechri et al., 2025). Rather than relying on explicit rules for each vulnerability, an ML model like a neural network, can infer patterns from large datasets of code labelled with known vulnerabilities. This can make detection more adaptive, for instance, catching variants of an unsafe coding pattern even if they do not exactly match a predefined rule. Further, ML-based approaches have shown the ability to identify

vulnerabilities that signature-based tools missed (Bridges et al., 2023). Moreover, as Muneer et al. (2024) reported, this approach can prioritize the most risk-prone code by learning from past vulnerability data, possibly reducing the burden of triaging countless static analysis warnings. On the downside, ML models act as black boxes since their predictions lack the transparent rationale that rule-based tools provide, which can affect developer trust and usability (Wiggerthale & Reich, 2024; Sarker et al., 2024). Additionally, the accuracy of an ML model heavily depends on its training data, so in a situation where the data is skewed or limited, the model may still produce false positives/negatives (Mohamed et al., 2025). In fact, a well-known challenge is that ML might pick up on spurious patterns in the training set such as particular variable names or coding styles, that do not generalize to other code (Li et al., 2023). Thus, while ML-based SAST can be more powerful, it must be developed and used carefully to truly surpass the traditional methods.

2.8.2 Evolution of ML Techniques Applied to SAST

A landmark study by Li et al. (2018) introduced VulDeePecker, one of the first deep learning–based SAST systems. VulDeePecker learns vulnerability patterns from code examples rather than using human-crafted rules. It represents code as 'code gadgets' (snippets of code around potentially vulnerable operations) and uses a bi-directional LSTM neural network to classify those gadgets as vulnerable or safe. Notably, VulDeePecker was able to detect multiple types of vulnerabilities simultaneously and achieved higher detection rates than traditional static analysers (which were limited to rule sets). In a comparative evaluation, it significantly outperformed both manual-rule tools and prior clone-based detectors, even discovering several real-world vulnerabilities that had been silently patched but not listed in vulnerability databases. This was a striking proof-of-concept that ML could catch issues that elude conventional methods.

Following this, the ML-in-SAST field rapidly expanded. In 2019, Zhou et al. proposed Devign, which applied Graph Neural Networks (GNNs) to vulnerability detection. Here, Devign converts code into rich graph representations (capturing code syntax and data flows as a code property graph) and then learns to predict vulnerable functions via GNN message passing. By leveraging the structural semantics of code, Devign achieved substantial improvements, reaching10% higher accuracy than earlier approaches on average. This demonstrated the

value of graph-based deep learning in capturing complex vulnerability patterns such as those involving multiple interacting code statements, that sequence-based models might miss. Around the same time, researchers also explored convolutional neural networks (CNNs) and other deep networks on code. For instance, some approaches encoded abstract syntax tree tokens or code n-grams as vectors and fed them into CNN models to automatically extract features. Russell et al. (2018) and others showed that even relatively simple deep models (CNNs or LSTMs) could outperform older statistical models in predicting vulnerable code segments by learning contextual code features. These early studies set the stage for increasingly sophisticated ML techniques in SAST.

Transformer models were then used for code analysis. Researchers treated source code as a form of language and fine-tuned transformer-based language models for vulnerability detection. For example, Ziems and Wu (2021) modelled vulnerability detection as an NLP task by applying a BERT-based model to tokenized code, with transfer learning from natural-language corpora, and reported over 93% accuracy in identifying vulnerable C code. The bidirectional context understanding of models like BERT proved advantageous, outperforming earlier RNN models like Bidirectional long-short term memory network (BiLSTM) on vulnerability tasks. In one comparative evaluation, BERT-based classifiers detected vulnerabilities more accurately than LSTM-based ones, underscoring the benefit of the deeper semantic grasp of transformers. By 2020, specialized transformer models for code appeared, notably Another interesting model is CodeBERT (Feng et al., 2020), a model pre-trained on source code and natural language documentation. Akshar et al. (2024) reported that fine-tuning CodeBERT for vulnerability detection yielded state-of-the-art results with approximately 90% accuracy using a CodeBERT-based model, outperforming models using simpler embeddings like Word2Vec. Transformers not only capture syntax but also learn subtler semantics (like data flow through variables) when pre-trained on large code corpora.

Another notable development is the emergence of security-specific language models. Aghaei et al. (2022) introduced SecureBERT, a BERT variant pre-trained on cybersecurity text (like threat intelligence reports) to better understand security-related language. While SecureBERT focuses on natural language (descriptions of the vulnerability), its creation reflects a broader trend of tailoring ML models to the security domain. Similarly, VulBERTa

(Hanif & Maffeis, 2022) is a pre-trained model specialized for source code vulnerabilities, which simplifies the BERT model training on code by focusing on vulnerability-relevant tokens. These domain-tuned models have reported improved performance on security tasks compared to general-purpose models, as they incorporate knowledge of common vulnerability patterns and security terminology.

2.8.3 ML Techniques for Code Vulnerability Detection

Machine learning approaches in SAST can be categorized by the type of model and input representation they use. Key techniques include using neural networks on code sequences, application of transformer language models, graph neural networks and classical machine learning ensembles.

Early applications of ML in code vulnerability detection treated code as a sequence of tokens (like text) and applied deep neural networks to classify the sequence (Liang et al., 2025). Here, recurrent neural networks (RNNs), especially LSTMs and bidirectional LSTMs, were used to capture long-range dependencies in code. For instance, the BiLSTM by VulDeePecker could learn patterns across multiple lines of code comprising a vulnerability (Li et al., 2018). Convolutional neural networks *(CNNs)* have also been used on token sequences or token embeddings. Here, they can automatically learn local patterns, such as a particular combination of function calls and arguments that signals a bug. Another instance is when Wang et al. (2019) embedded abstract syntax tree tokens and fed them to a deep belief network or CNN to extract code semantics for classification. These sequence-based models established a baseline but often struggled with modelling complex program structures.

Additionally, with transformers like BERT, and its variants CodeBERT, SecureBERT, etc., represent a more advanced sequence model that can capture context in a bidirectional manner. In code analysis, transformer models are usually fine-tuned on code labelled with vulnerabilities. They excel at understanding the context around potentially vulnerable code, such as data dependencies and control flow, by attending to all parts of the input sequence. Researchers have found that BERT-based models outperform earlier CNN/RNN approaches for vulnerability detection (Tiwari, 2025; Jean et al., 2024). Domain-specific transformers also exist such as CodeBERT (Feng et al., 2020) which is trained on GitHub code in multiple

languages, making it adept at general code understanding, while SecureBERT (Aghaei et al., 2022) is trained on security texts to imbue security-relevant context. Fine-tuning these pre-trained models on vulnerability datasets has led to top-tier accuracy of often 85–90%+ on benchmark datasets (Mim et al., 2024; Haurogné et al., 2024). These models effectively leverage both the syntax and some semantics of code.

Also graph neural networks have been applied on code for vulnerability detection because code has an inherent graph structure including abstract syntax trees (AST), control-flow graphs, and data-flow graphs, GNN-based techniques explicitly leverage these structures. A GNN can propagate information along edges representing “uses” or “dependencies” in code, which is very useful for certain vulnerability types. Take for instance those involving a flow from a water source to a sink. Devign (2019) demonstrated the power of this approach by learning from code property graphs that combine multiple code relationships. Many subsequent works used graph embeddings of code to for example, represent a function such that nodes are statements and edges denote data flows, then predict if the function is vulnerable (Saimbhi et al., 2025; Shao et al., 2025; Zhang et al., 2024; Lin et al., 2023; Wu & Zou, 2022). GNNs have achieved strong results, especially on vulnerabilities that require understanding the link between different parts of a program (Chu et al., 2024), especially in a situation where a variable is not validated in one function and later used dangerously in another function (Wen et al., 2023). In fact, a recent systematic review noted that graph-based models are the single most prevalent ML approach in the vulnerability detection literature (Shimmi et al., 2025). This popularity stems from their ability to encode rich semantic information beyond flat token sequences.

Not surprisingly, not all ML in SAST involves deep learning. Some studies use more traditional classifiers like Random Forests (Resende & Drummond, 2018), Support Vector Machines (Blinowski et al., 2021), or XGBoost (Wen et al. 2023), usually in combination with hand-engineered features. Software metrics, for example, such as the complexity, function length, number of risky API calls, can be extracted from code, and a Random Forest model can be trained to predict vulnerable files or functions based on those features. While these classical ML models are generally faster to train and easier to interpret than deep neural networks (Shendkar et al., 2024; Si-Ahmed et al, 2024), their accuracy is often subpar deep

learning as the task become more complex (Arp et al, 2024). Nonetheless, there are some niche use-cases, for instance, integrating with static analysis outputs (Sayoud, 2024; Santos et al., 2021). A practical example is a tool that runs a static analyser and then uses a machine learning filter (like an ensemble classifier) to predict which of the warnings from the analyser are true positives versus likely false positives (Tsiazhkorob & Ignatyev, 2024). This hybrid approach combines the extensive knowledge encoded in static rules with the ability of the ML model to weigh the context, thereby reducing noise.

2.8.4 Challenges in the State-of-the-Art of ML in SAST

Interestingly, the literature shows that static and dynamic analysis tend to find different, yet overlapping sets of vulnerabilities, meaning neither approach alone is sufficient for comprehensive security assurance (Nunes, et al., 2025; Khanum, Qadir & Jehan, 2023).

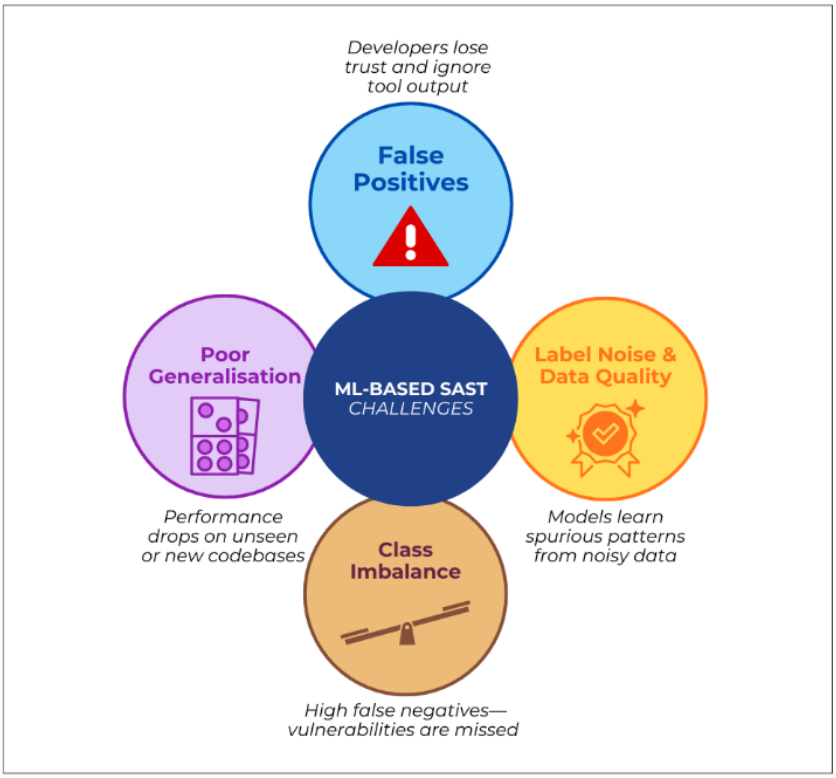


Figure 2-3: The four principal challenges in ML-based static application security testing.

Even so, with the present substantial progress, significant challenges remain in application security testing. One well-documented issue with static analysis is the prevalence of false positives, warnings about code that appears vulnerable but is not actually exploitable

(Wadhams et al., 2024; Leokhin et al., 2024; Guo et al., 2023; Ehsan et al., 2024; Aydos et al., 2022). However, this has a direct impact on developer adoption based on studies which found that excessive false alarms lead developers to lose trust in the tool and even ignore its output (Leokhin et al., 2024). In a recent literature review, over two-thirds of surveyed papers identified false positives as a primary pain point for SAST users (Wadhams et al., 2024). Such false alerts considered time wasters by the developers because they spend resources investigating non-issues and can lead to frustration and reduced productivity (Wadhams et al., 2024). Researchers are actively exploring ways to mitigate this; for instance, Guo et al. (2023) discuss strategies to filter or prioritize static analysis warnings using machine learning and contextual information to improve relevance. Another challenge is poor output usability since many static tools present results in formats or volumes (for example large XML/JSON reports with thousands of entries) that are overwhelming, making it hard for developers to triage findings (Wadhams et al., 2024). Improving the presentation of analysis results (such as richer IDE integration, dashboards, or automated fix suggestions) has been suggested to increase the practical usefulness of SAST (Odermatt et al., 2022). Moreover, setting up and tuning static analysis for a given project can be time-consuming, with activities such as configuring the correct rules, suppressing known safe findings, and integrating the tool into build pipelines often requires considerable effort, which some teams perceive as a barrier (Wadhams et al., 2024). This is compounded in environments with polyglot codebases or rapid release cycles, where static tools must be continuously managed. Figure 2-3 gives an overview of the challenges.

Dynamic analysis faces its own set of hurdles. By nature, DAST (including fuzzing and automated scanning) is resource-intensive and can have slower turnaround. Running a full dynamic scan or extensive fuzz campaign can take hours or days, which does not always fit smoothly into agile development workflows. Li et al. (2018) note that dynamic analysis often suffers from impacted speed, low efficiency, inadequate scalability, sometimes requiring significant manual effort and expertise to set up the test harness or tailor test cases. For example, a web application scanner might need custom scripts to handle authentication or workflows, failing which it may miss large portions of the attack surface of the application (Alazmi & De Leon, 2022). In addition, dynamic testing can have blind spots because it can

only find vulnerabilities in the portions of code it executes. Reaching deep or rarely used functionality may require creative input generation, and certain logic flaws might not be apparent without very specific scenarios that automated tools struggle to infer. Consequently, dynamic analysis tends to have a higher false-negative rate (undetected vulnerabilities) compared to static analysis, which theoretically examines all code paths (Li et al., 2018). Another gap is that many classes of security weaknesses, such as flaws in application design or business logic, remain difficult for both SAST and DAST (Metin et al., 2025; Felmetsger et al., 2010). These might include cases where the code is functionally correct, but the security requirement is not met (take for instance an application allowing an action that should have been forbidden by business rules). Such issues often require human insight or contextual understanding beyond what automated tools currently possess. Privacy concerns add another dimension as it is quite germane to ensure that software handles personal data correctly (preventing leaks or misuse), but few application security testing tools explicitly address privacy requirements. Static taint analysis can be extended to trace personal data flows and identify potential privacy violations, but aligning these findings with legal/privacy requirements is non-trivial (Ferrara et al., 2021).

Finally, there are evaluation and coverage gaps in the current state of practice. It is challenging to measure how secure an application is even after using multiple tools considering that each tool may report different findings, and there is no easy way to know if combined all critical issues have been caught. Benchmark suites and competitions (like the CWE/SARD benchmarks) attempt to evaluate tool effectiveness but translating these results to practical assurance remains difficult. Recent systematic evaluations have shown that using only one approach can leave significant risk uncovered: for instance, one analysis recommends DAST as essential for catching Broken Access Control bugs that SAST missed, while conversely SAST was better for certain issues like detecting hardcoded credentials (Khanum et al., 2023). This highlights that security testing results from various tools need to be correlated and prioritized to give developers a clear action plan, a gap in integrated workflows. Moreover, the human factor cannot be ignored, because if developers are overwhelmed or alienated by security tools, the tools will not yield practical security improvements (Thomas et al., 2018). Usability studies (Wadhams et al., 2024) emphasize that

addressing the developer experience of security testing tools is as important as improving detection capabilities. Moving forward, research in application security testing should not only focus on technical enhancements (like better analysis algorithms and AI-driven vulnerability discovery) but also on these socio-technical aspects, such as but not limited to, bridging the gap between automated security analysis and developer action in the software development lifecycle.

Considering these challenges, this research positions SAST as a foundational pillar in addressing the persistent gaps in application security assurance, particularly within the highly regulated healthcare domain. While both SAST and DAST contribute uniquely to vulnerability detection, the high false-negative rates and runtime constraints of dynamic approaches underscore the necessity of robust static analysis early in the software development lifecycle. By enhancing SAST with context-aware analysis, automated prioritization, and seamless integration into development workflows, this work aims to mitigate false positives, improve detection precision, and ensure that critical security and privacy flaws, especially those impacting sensitive patient data, are identified before deployment. In doing so, it connects theoretical detection capability and practical developer adoption, aligning with SbD and PbD principles, and providing a scalable, proactive strategy for safeguarding modern healthcare software systems while supporting compliance with regulatory frameworks such as HIPAA and the UK Data Security Standards. Building on this foundation, the next section examines how machine learning techniques are being leveraged to augment SAST capabilities, enabling more adaptive, precise, and context-aware vulnerability detection that addresses the limitations of traditional rule-based static analysis.

2.9 Security-Relevant Features and Vulnerabilities in C/C++

This section examines the security characteristics of C/C++ software in healthcare contexts. Section 2.9.1 establishes why C/C++ remains the dominant language in safety-critical healthcare systems despite its known security challenges. Section 2.9.2 surveys the most prevalent vulnerability types in C/C++ code. Section 2.9.3 discusses the clinical and regulatory impact of these vulnerabilities and reviews existing datasets that reflect their distribution. Section 2.9.4 examines how NLP and SAST methods have been applied to C/C++

vulnerability detection in healthcare contexts. Figure 2-4 provides an overview of the argument across all four subsections.

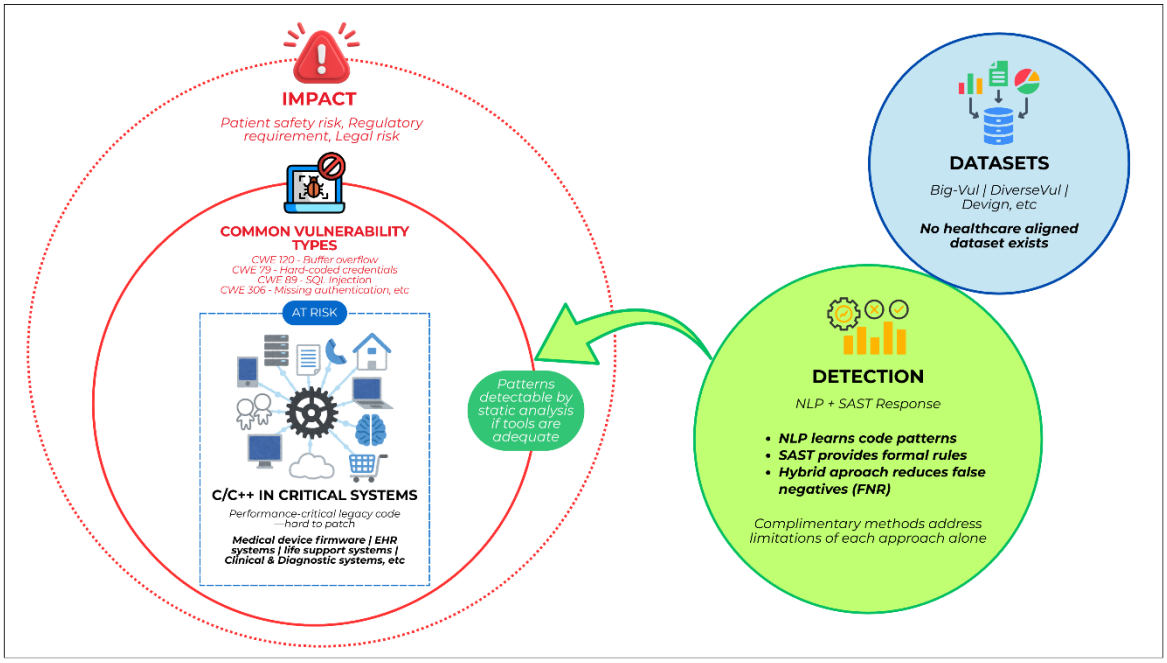


Figure 2-4: Overview of Section 2.9: the case for applying NLP and SAST to C/C++ healthcare software. C/C++ dominates safety-critical systems, carries well-documented vulnerability patterns with serious clinical consequences, and lacks domain-specific datasets — collectively motivating the NLP and SAST hybrid approach and the C3-VULMAP dataset contribution.

2.9.1 C/C++ in Critical Systems and Legacy Software

C/C++ have long been foundational for system-level and performance-critical software (Rassokhin, 2020). However, even with the advent of safer languages, they remain prevalent in domains like embedded systems, high-performance computing, and even healthcare device firmware, because of their efficiency and low-level hardware control (Yang et al., 2024). Legacy codebases and modern performance-sensitive applications continue to rely on C/C++ for exactly these reasons, even though these languages lack built-in memory safety mechanisms (Yang et al., 2024). In safety-critical contexts such as in medical devices and industrial control, this reliance is double-edged in aspects of the flexibility and speed of C/C++ enable advanced functionality, but the absence of strict safety guarantees introduces significant security risks (Williams & Woodward, 2015; Ng, 2023). Recent studies underscore that much of the recent technological infrastructure still “heavily relies on C/C++ projects,” making them critical targets for security exploits (Ahmed et al., 2025). Vulnerabilities in such

low-level software can have widespread downstream impacts, as demonstrated by incidents in healthcare where insecure medical device software exposed sensitive data or even allowed remote control of devices (Bracciale et al., 2023). In an analysis of medical devices across 36 countries, Bracciale et al. (2023) identified 661 distinct software vulnerabilities in devices used by national health services, over half classified as high severity, including flaws that enabled unauthorized access to infusion pumps and remote manipulation of insulin dosage with potentially fatal outcomes (Bracciale et al., 2023). These examples highlight why C/C++ code in critical domains must be scrutinized for security weaknesses.

Historically, many of the earliest catastrophic software exploits (from the Morris worm to modern malware) have originated from coding errors in C/C++ (Orman, 2003; DeLozier, 2024). Far from being solved problems, such vulnerabilities persist in current code. The healthcare industry, for instance, continues to grapple with latent flaws in its C/C++-based software and even a recent survey urges "significant improvements in protecting medical devices from cyberattacks" by adopting more secure coding practices (Mejía-Granda et al., 2024). Thus, understanding the security-relevant aspects of C/C++ is crucial, as these languages remain deeply entrenched in both legacy and modern critical software.

2.9.2 Common Vulnerability Types in C/C++ Software

Memory safety issues are the hallmark vulnerabilities of C/C++ (DeLozier, 2024). The languages allow direct memory management (via pointers, manual allocation, etc.) without runtime bounds checking or garbage collection, leading to well-known bug classes. Buffer overflows (writing beyond allocated buffer bounds) are a classic example; they have been exploited for decades and remain "a common and dangerous vulnerability in C/C++" (ibid.). Closely related are off-by-one errors and stack smashing issues, which similarly involve writing to memory out of the intended range. Another prevalent flaw is use-after-free, where a pointer to freed memory is dereferenced. This temporal memory error can corrupt program state or allow attackers to execute arbitrary code by reallocating the freed memory (Yang et al., 2024). Double-free and dangling pointer bugs are similar C/C++ issues that attackers can leverage for heap corruption. Memory leaks, while often viewed as reliability issues, can also constitute security vulnerabilities, for example, in long-running systems or those with limited memory, leaks might be exploited to cause denial-of-service or to gradually manipulate

program behaviour. All these vulnerabilities stem from the lack of automatic memory management: C/C++ put the onus on developers to allocate, use, and free memory correctly, and any mistake can introduce undefined behaviour with security implications (Chen et al., 2021).

2.9.3 Impact of C/C++ Vulnerabilities and Datasets Reflecting Their Prevalence

The security impact of C/C++ vulnerabilities is evident in both real-world incident reports and curated datasets of bugs. Many high-profile attacks (example given, Heartbleed in OpenSSL, WannaCry exploitation, etc.) ultimately traced back to C/C++ coding flaws such as buffer mismanagement. Recognizing this, the research community has compiled multiple vulnerability datasets centred on C/C++ code. For instance, Devign, introduced by Zhou et al. (2019), provides a collection of vulnerable and fixed C/C++ functions from open-source projects to facilitate machine learning on vulnerability identification. The importance of rich code representations (graphs of program semantics) for capturing C/C++ vulnerability patterns are highlighted by Zhou et al (2019). Another influential corpus is Big-Vul, a large-scale dataset of C/C++ vulnerabilities mined from GitHub commits and annotated with Common Vulnerabilities and Exposures (CVE) data (Fan et al., 2020). Big-Vul contains thousands of real vulnerability examples (3,754 vulnerabilities from 348 projects, spanning 91 CWE types) and has been widely used to train and benchmark vulnerability detection models (Fan et al., 2020). The prominence of C/C++ in these datasets is no coincidence, it reflects that most known software vulnerabilities (tracked in CVEs) come from projects written in C or C++ (Ahmed et al., 2025; Fan et al., 2020). Recent benchmarks like DiverseVul (Chen et al., 2023) and MegaVul (Ni et al., 2024) similarly focus on C/C++ code, and include rich metadata such as CWE categories and patches, to aid analysis. They underscore common weaknesses in C/C++ such as memory corruptions and input validation errors. Importantly, these datasets also reveal gaps in current detection capabilities. Capabilities where models often achieve high precision on simpler patterns but miss vulnerabilities requiring understanding of pointer aliasing or cross-function interactions (Risse & Böhme, 2024).

2.9.4 Focusing NLP and SAST Methods on C/C++ in Healthcare Contexts

Given the above, there is strong justification for focusing advanced analysis techniques, including Natural Language Processing (NLP)-inspired code analysis and improved Static Application Security Testing (SAST), on C/C++ code in critical domains like healthcare. First, the risk profile is high as vulnerabilities in medical software can directly impact patient safety (Mejía-Granda et al., 2024; Bracciale et al., 2023). Unlike web or enterprise software where a bug might leak data, a flaw in a C++-based pacemaker controller or infusion pump can threaten human life. This elevates the need for proactive vulnerability detection. However, as discussed, conventional static analysers struggle with C/C++ and may be insufficient to catch complex flaws in medical device code (Chakraborty et al., 2022). Here, NLP and machine-learning-based methods offer a promising complementary approach. Recent work has shown that large language models and deep learning can be trained to recognize vulnerability patterns in code, sometimes generalizing beyond the rules of traditional analysers (Zhou et al., 2024). By leveraging learning on big code datasets (like Big-Vul) and treating code as another form of text, NLP-based techniques can detect subtle issues such as the misuse of APIs, anomalous pointer usage sequences, or missing checks which are issues that might not be explicitly enumerated in rule-based scanners. For instance, emerging research by Zhou et al. (2024) demonstrates how prompting large code models (CodeLLMs) can help surface security issues that static tools overlook, by reasoning about code logic in a more human-like way. These techniques can be integrated with SAST, thereby making an NLP model to filter or prioritize static analysis warnings, or flag suspicious code regions for deeper analysis, thereby addressing the false-positive problem and context-sensitivity gaps of traditional tools (Shiri Harzevili et al., 2024).

Focusing on C/C++ code in healthcare software with such hybrid methods is further justified by the observation that many healthcare exploits involve known weakness patterns (hard-coded credentials, buffer overruns, etc.) that are detectable with improved analysis. The healthcare context also brings regulatory pressure such as the FDA premarket cybersecurity guidelines, to bear, which increasingly demand rigorous code security reviews. NLP-assisted static analysis could help meet these requirements by scaling security auditing across the large C/C++ codebases found in medical devices and hospital systems. In summary,

because C/C++ remains ubiquitous in performance-critical healthcare applications, and because the cost of their vulnerabilities is extraordinarily high, it is vital to apply state-of-the-art vulnerability detection techniques (both improved static analysis and machine-learning/NLP-based analysis) to this context. This combined approach aims to overcome the shortcomings of each method alone. Thus, using NLP to handle the ambiguous or complex patterns in C/C++ code, and using static analysis to provide formal assurances and catch straightforward issues, thereby yielding more effective and comprehensive vulnerability detection for healthcare software written in C/C++. This direction aligns with recent calls in the research community for multi-faceted security analysis, ensuring that the next generation of tools can better secure C/C++ systems that we entrust with human lives (Zhou et al., 2024; Mejía-Granda et al., 2024).

2.10 Common Weakness Enumeration (CWE) and Software Vulnerability Taxonomies

The Common Weakness Enumeration (CWE) is a comprehensive, community-developed taxonomy of software and hardware weakness types (Mejía-Granda et al., 2024). It provides a unified language for describing the root causes of software vulnerabilities, enabling researchers and practitioners to classify security flaws consistently across tools and databases (Howard, 2009; Kanakogi et al., 2021). CWE was introduced in the mid-2000s as part of broader efforts to catalogue software security errors, building on prior vulnerability taxonomies. Previous taxonomies include, Tsipenyuk, Chess, and McGraw (2005) proposed the “Seven Pernicious Kingdoms” classification to help developers understand common coding mistakes leading to security bugs. While such early taxonomies defined broad categories of errors (e.g. input validation, access control) to guide secure coding (Tsipenyuk et al., 2005). However, CWE extended this approach by defining hundreds of fine-grained weakness types in a hierarchical structure (Martin & Barnum, 2008). The CWE project organizes weaknesses into abstract classes, base-level weaknesses, and variant-specific instances, forming a detailed ontology of software flaw types. As of the latest reports, CWE encompasses over 1,000 distinct weakness entries as part of its catalogue (Mejía-Granda et al., 2024). These range from high-level classes (for instance, Improper Input Validation) to specific issues (such as Buffer Overflow via strcpy) with unique identifiers (for example CWE-

20, CWE-119). Each CWE entry includes descriptions, examples, potential mitigations, and references to related vulnerabilities, serving as a rich knowledge base for secure development (ibid.). The structured nature of CWE makes it a foundational standard for software assurance since it allows security teams to discuss "root cause" weakness types rather than just symptoms or exploits (Meyers & Meneely, 2021). In essence, CWE has become the de facto vulnerability classification scheme, supported by organizations like MITRE and NIST as part of the Security Content Automation Protocol (SCAP) for vulnerability management (Mejía-Granda et al., 2024). This widespread adoption aims to improve software security by focusing attention on eliminating known categories of dangerous errors. Notably, Howard (2009) advocated that removing the most prevalent CWEs (the "Top 25" list) from software would significantly enhance overall security, underscoring the role of the CWE as a measuring stick for secure coding priorities (ibid.). However, despite its strengths, the expansiveness of the CWE also presents challenges such as the sheer number of categories and occasional overlaps which can complicate its use, and also ensuring consistent classification of new vulnerabilities into CWE remains an ongoing task (Honkaranta et al., 2021). CWE provides a structured and evolving vocabulary for the software security community, enabling both historical analysis of flaws and the proactive mitigation of known weakness types.

2.10.1 CWE Usage Across Domains and Common Weaknesses in Healthcare Software

The extensive CWE taxonomy is applied across diverse software domains to understand and mitigate vulnerabilities. In general IT and web applications, CWE categories such as buffer errors, injection flaws, and cross-site scripting (XSS) are frequently referenced by security tools and standards. For instance, the CWE Top 25 "Most Dangerous Software Weaknesses" (an annually updated list) often highlights issues like Out-of-Bounds Write (CWE-787) or SQL Injection (CWE-89) that are pervasive in typical software systems[9]. Notwithstanding, many secure development practices incorporate CWE by designating these top-ranked weaknesses as critical issues to avoid or remediate first (Howard, 2009). Static analysis and vulnerability

[9]https://www.armis.com/blog/the-threat-within-analyzing-mitres-list-of-the-most-dangerous-software-weaknesses/#:~:text=The%20Threat%20Within%3A%20Analyzing%20MITRE%27s,Its

scanning tools commonly label detected flaws with CWE identifiers, which helps developers link tool findings to known weakness definitions and remediation guidance (Kanakogi et al., 2021). This practice has effectively made CWE a cross-domain “common denominator” for discussing software security defects.

Importantly, different industry sectors tend to exhibit distinct weakness prevalence profiles, and CWE is used to analyse those domain-specific trends. In the healthcare software domain, which involves electronic health record systems, medical device firmware, and healthcare IoT, studies have identified several recurring weakness patterns. A recent large-scale analysis of health IT vulnerabilities from 2001–2022 (based on NIST National Vulnerability Database) found that improper access control and poor authentication mechanisms are among the most common weaknesses in healthcare systems (Mejía-Granda et al., 2024). Specifically, critical issues such as missing or weak user authentication, poor credential management, and hard-coded credentials have been frequently observed in medical software (ibid.). These weaknesses correspond to CWE categories like Missing Authentication (CWE-306) or Use of Hard-coded Password (CWE-798) and can lead to unauthorized access to sensitive patient data. Another prevalent category in healthcare is insecure communication and API usage, reflecting weaknesses in protecting data in transit or exposing unsafe interfaces (example given, a lack of encryption or input validation on medical device APIs). For example, Mejía-Granda et al. (2024) reported that many vulnerabilities in medical devices are rooted in configuration mistakes and broken access control, which align with CWE entries on security misconfiguration and improper authorization. Moreover, healthcare software has been plagued by classic injection flaws (example give, SQL injection, OS command injection) and buffer overflows, which are also well-categorized by CWE-89, CWE-120, etc. These technical weaknesses often manifest in healthcare due to legacy systems and insufficient secure coding practices, reinforcing the need for CWE-based mitigation checklists in this sector (Mejía-Granda et al., 2024). Other domains show different CWE patterns. For instance, in industrial control systems (ICS) and IoT devices, memory safety issues and cryptographic weaknesses may dominate, whereas in financial or enterprise applications, issues like improper input validation and logic errors are more prominent. The breadth of CWE allows it to cover all these domains, and researchers have used it to compare

vulnerability profiles. For instance, Dang and François (2018) utilized CWE classifications to study vulnerabilities in software-defined networking (SDN/NFV) systems, identifying domain-specific weakness trends. Likewise, the CWE community has special interest groups working to refine the taxonomy for domains like healthcare and operational technology, ensuring the categories remain relevant (The MITRE Corporation, 2025).

2.10.2 Role of CWE in Secure Software Development and Vulnerability Detection

Integrating CWE into the secure software development lifecycle (SSDLC) has proven beneficial for both identifying vulnerabilities early and standardizing remediation efforts. Developers and security engineers leverage CWE in multiple ways during development and testing. First, CWE serves as an educational checklist of what not to do in coding. Many organizations train developers using the CWE list or the CWE/SANS Top 25 to illustrate common mistake patterns (Howard, 2009). By understanding CWE-defined weaknesses, developers can more readily recognize risky coding practices and implement safer alternatives. During design and code implementation, static analysis tools often detect potential security bugs and label them with CWE IDs (Williams et al., 2016). This mapping helps developers quickly grasp the nature and severity of an issue by referring to the CWE description and mitigation guidance. The CWE documentation provides examples and mitigations for each weakness, which developers can consult to fix the specific problem in code (The MITRE Corporation, 2025). This direct linkage of code scanning results to CWE knowledge has been shown to facilitate more consistent and effective fixes (Williams et al., 2016). Furthermore, organizations use CWE coverage as a metric for security testing completeness. A case in point is the organizations ensuring that their test suites or code reviews consider the most critical CWE categories relevant to their software[10]. On the vulnerability detection side, CWE plays a key role in vulnerability databases and automated tools. The NIST National Vulnerability Database (NVD) assigns a CWE tag to most reported

[10] https://www.opsmx.com/blog/risk-based-prioritization-how-to-use-cwe-effectively-in-application-security/#:~:text=CWE%20facilitates%20systematic%20analysis%20by,help%20speed%20up%20remediation%20efforts.

CVEs (Common Vulnerabilities and Exposures records) to indicate the underlying weakness that led to the vulnerability (Kanakogi et al., 2021). This tagging not only standardizes vulnerability reports but also enables trend analysis (example give, tracking which categories of weaknesses are rising or falling in prevalence). Over time, the percentage of CVEs with an associated CWE has grown, and since around 2008, it has become common to link CVEs to root-cause CWE categories (ibid.). However, it is noteworthy that not all CVE entries have CWE attribution. As of 2021 roughly 28% of CVEs lacked a CWE linkage, often labelled as "NVD-CWE-Other" when the weakness is unclear (ibid.). This gap indicates ongoing challenges in consistently identifying root causes for every vulnerability. Even so, most published vulnerabilities can be analysed through the CWE lens, which has enabled advanced methods like machine learning to prioritize and predict vulnerabilities by type. For instance, researchers have trained models to detect specific CWE-classified weaknesses in code (rather than generic "vulnerability" labels), yielding better accuracy by focusing on the unique patterns of each weakness class (Meyers & Meneely, 2021). The diversity of CWE categories means models and tools must account for a wide range of vulnerability patterns, from memory corruption to cryptographic issues (Atiiq et al., 2024). Nonetheless, by treating CWE identifiers as a form of ground truth for what the flaw is, these tools can provide more interpretable results, like telling a developer that a finding is "CWE-79 (XSS)" with clear remediation steps, rather than a nebulous warning.

2.10.3 Relationship of CWE with CVE and CAPEC

The CWE is one pillar of a trio of security enumeration standards maintained largely by MITRE: CVE (Common Vulnerabilities and Exposures), CWE (Common Weakness Enumeration), and CAPEC (Common Attack Pattern Enumeration and Classification). Each serves a different purpose, and they interrelate to provide a fuller picture of software security issues. The CVE is an indexed list of specific vulnerability instances, essentially records of real-world security bugs in software products. Each CVE entry describes a particular flaw in a product version (with an identifier like CVE-2025-12345), but CVEs themselves do not generalize beyond the instance. On the other hand, CWE, as discussed, abstracts from individual incidents to the underlying weakness type. The relationship is such that many CVE entries include a field mapping the vulnerability to one or more CWE categories that represent

the root cause (Kanakogi et al., 2021). For example, a CVE about a buffer overflow in a medical device might be tagged with CWE-120 (Buffer Copy without Checking Size of Input). This CWE link helps stakeholders understand why the vulnerability occurred and what other contexts might suffer similar flaws. Studies have shown that broad CWEs (like CWE-20: Improper Input Validation) tend to be associated with many CVEs, sometimes as a catch-all, which can complicate using CWE for precise vulnerability tracing (ibid.). Nonetheless, the CVE–CWE linkage is fundamental: it enables counting how many vulnerabilities of each type exist, identifying trends (for instance, "injections are rising this year"), and prioritizing defence efforts by weakness prevalence.

Meanwhile, CAPEC catalogues attacker techniques and attack patterns. Each CAPEC entry describes a method an adversary might use (like SQL injection attack, Phishing attack, etc.), including the steps and mechanisms of the attack. The relationship between CAPEC and CWE is that CAPEC attack patterns often exploit certain CWE-class weaknesses[11]. In the documentation for CAPEC, each attack pattern is typically linked to the relevant CWE IDs that it targets. For instance, the CAPEC entry for an SQL injection attack will reference CWE-89 (SQL Injection) as the weakness it exploits. Conversely, a given CWE (like Cross-Site Scripting, CWE-79) may list related CAPEC patterns that illustrate how an attacker could leverage that weakness. This triad relationship can be visualized in Figure 2-5. An attacker uses a CAPEC method to exploit a CWE weakness present in a system, which then gets reported as a CVE if discovered. In practice, researchers and threat analysts use these links to trace from an observed vulnerability to possible attack vectors, and vice versa. The benefit of correlating CWE with CVE and CAPEC is a more holistic understanding of security threats. This synergy is being leveraged in advanced cybersecurity analytics. For instance, Shi et al. (2024) built a threat knowledge graph linking CVEs, CWEs, and CPEs (Common Platform Enumerations for product names) to uncover hidden relationships and propagate security knowledge across

11 https://capec.mitre.org/#:~:text=Understanding%20how%20the%20adversary%20operates,community%20understanding%20and%20enhance%20defenses

these datasets. Such approaches can identify clusters of vulnerabilities that share weaknesses and attack patterns, informing defensive strategies (Shi et al., 2024).



Figure 2-5: Relationship between CAPEC, CWE and CVE

2.10.4 Mapping Threats and Misuse Cases to CWE Types: Efforts and Challenges

Although CWE provides a powerful way to classify technical software weaknesses, it does not directly connect to the higher-level threats or misuse scenarios typically identified during architectural threat modelling (Honkaranta et al., 2021). Frameworks such as STRIDE or misuse case modelling operate at a broader, conceptual level. For example, describing a scenario like “an attacker gains access to patient data through a software flaw.” In contrast, CWE focuses on specific implementation issues, such as SQL injection or missing encryption. To strengthen early-stage security design, researchers and practitioners have therefore worked on methods to link these abstract, design-level threats to concrete CWE-defined weaknesses (Albanese et al., 2024. Establishing such mappings helps architects ensure that each identified threat is associated with the relevant underlying weaknesses and that these weaknesses can be systematically mitigated during development. This identified gap is one of the objectives of this research.

One influential line of research in this area is the work of Williams et al. (2015), who proposed a systematic method for generating abuse cases via structured scenarios illustrating how attackers could exploit system functions. Their approach combines Microsoft’s threat-modelling methodology with MITRE’s CAPEC attack patterns and CWE software weakness taxonomy. In practice, once a potential threat scenario is identified, the method

maps it to corresponding CAPEC attack patterns and then derives the CWE-classified weaknesses that would enable those attacks (Williams et al., 2016). This produces detailed and actionable abuse case narratives, such as: *"If an attacker attempts an XSS attack [CAPEC], the system must not contain CWE-79 (XSS vulnerability)."* By establishing these explicit links between misuse cases and CWE categories, the method helps development teams prioritise security requirements and focus remediation efforts on the specific weaknesses most likely to facilitate high-impact threats.

Industry and government initiatives have made similar efforts to harmonize high-level threat scenarios and concrete software weaknesses. A notable example is the Software Engineering Institute's Malware-Driven Overlooked Requirements (MORE) project. This platform enriches exploited vulnerability reports with associated misuse cases and overlooked security requirements, each annotated with CWE identifiers (Mead et al., 2016). By tagging malware and CVE reports with their underlying CWE weakness types, MORE enables requirements engineers to search for recurring software flaws and derive corresponding security requirements early in the development process. This approach integrates real-world incident knowledge directly into design-time decision-making, guiding architects and developers in identifying and avoiding the weakness patterns that have historically led to significant security exposures (Mead et al., 2016).

Despite these efforts, mapping threats to CWE weaknesses is non-trivial and presents conceptual challenges (Honkaranta et al. (2021). Threat modelling frameworks like STRIDE use abstract threat categories (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), which do not correspond one-to-one to specific CWEs. Honkaranta et al. (2021) attempted to create a practical mapping between STRIDE threat categories and the CWE database. Their study tried multiple approaches including using intermediary mappings via OWASP Top 10, using the CWE Top 25 list, and leveraging attributes of CWE entries (such as "Technical Impact" fields) to align with notions of impact by STRIDE (Honkaranta et al., 2021). They found that the most promising approach was to map STRIDE threats to CWE based on the Technical Impact and Scope attributes defined in CWE (which describe the consequence of a weakness, like confidentiality loss or privilege escalation). For example, a STRIDE threat of "Information Disclosure" could be mapped to

CWE entries whose impact is leakage of information. Even so, the researchers concluded that significant gaps and ambiguities remain. The conceptual gap between architectural threats and low-level weaknesses means that expert judgment is often needed to do the mapping. Honkaranta et al. (2021) note that differences in terminology and granularity make automated mapping difficult, and that many mappings had to be done manually by domain experts. These findings highlight that while CWE is very useful in implementation and testing phases, using it directly in early design (threat modelling) is still a maturing area.

2.10.5 From Privacy Threat to Privacy Harm: A Conceptual Model

The preceding discussion establishes that CWE, CVE, and CAPEC are related but distinct constructs, and that mapping design-level threats to code-level weaknesses remains methodologically underdeveloped. This raises a point that must be made explicit before this thesis proceeds to construct its own LINDDUN-to-CWE mapping in Chapter 3: the presence of a CWE-classified weakness in code does not, by itself, constitute evidence of a privacy breach, a regulatory violation, or actual harm to a data subject. A weakness is a necessary precondition for harm, not a sufficient one. Figure 2-6 makes this causal chain explicit.

The model traces six stages from a design-level privacy concern to an actual privacy-harming outcome: a LINDDUN threat identified during data flow analysis may, under the condition that a plausible attacker scenario can be constructed, be formalised as a misuse case. A misuse case may, under the condition that it can be mapped to a concrete implementation defect, correspond to a CWE weakness, this mapping is the specific contribution developed in Chapter 3. A CWE weakness may, under the condition that it is present in shipped code and is subsequently discovered and disclosed, be registered as a CVE. A CVE may, under the condition that an attacker possesses the capability, access, and motive to act, result in exploitation. Exploitation may, under the condition that the exploited data is sensitive, attributable to an individual, and used to that individual's detriment, result in privacy harm.

Each arrow in this chain is conditional, not deterministic. This has two direct implications for how the contributions of this thesis should be interpreted. First, the LINDDUN-to-CWE mapping presented in Chapter 3 operationalises only the second link in this chain, the

transition from misuse case to CWE weakness, and expert validation of that mapping (Section 3.2.6) establishes the plausibility of this link, not the certainty of downstream harm. Second, the detection architecture presented in Chapters 4 and 5, which identifies CWE-classified weaknesses in code, should correspondingly be understood as identifying *conditions under which* privacy harm becomes possible, not as detecting privacy breaches that have occurred or are guaranteed to occur. This distinction is revisited in the discussion of the system's outputs in Chapter 5 and in the scope, limitations set out in Chapter 6.

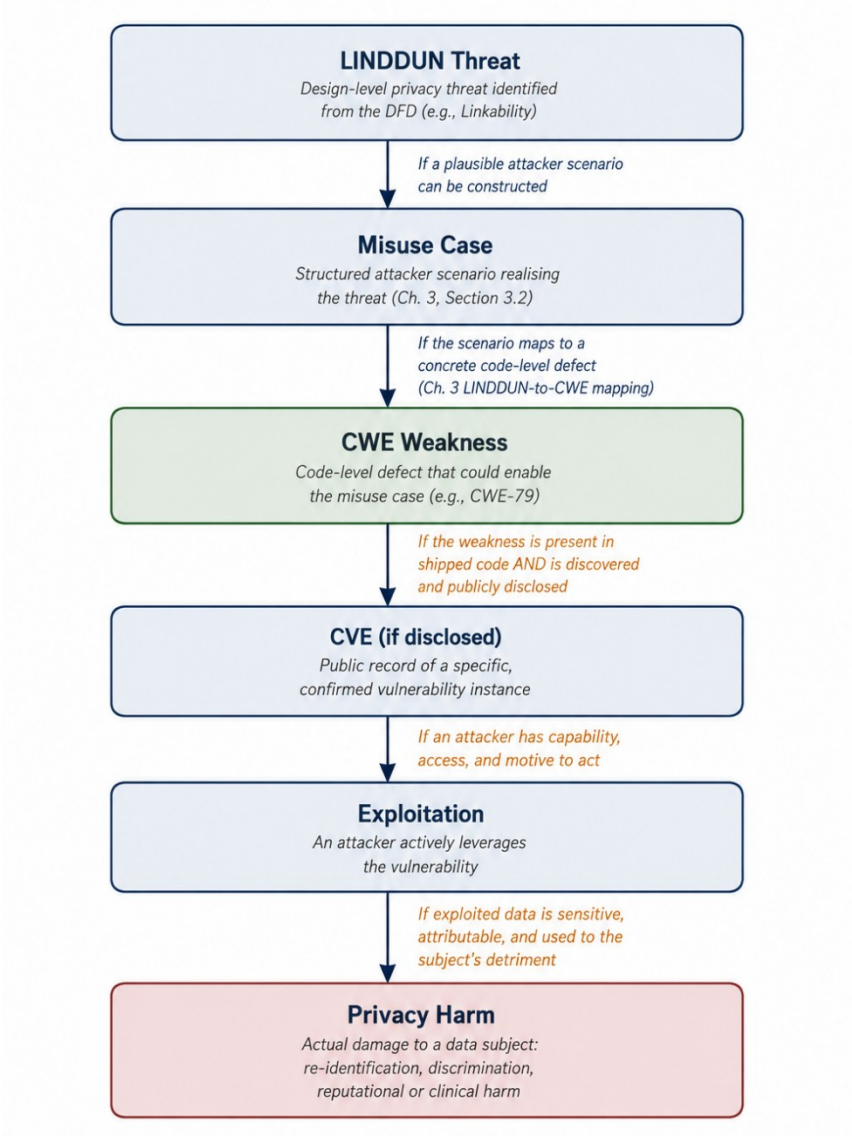


Figure 2-6: Conceptual model of the causal chain from a design-level LINDDUN privacy threat to actual privacy harm

2.11 NLP for Software Vulnerability Detection

Early applications of machine learning to software vulnerability detection approached source code predominantly as textual data, drawing heavily on natural-language-processing techniques. Initial models relied on basic tokenisation, n-gram representations, and manually engineered features; however, these methods frequently failed to capture deeper semantic or structural relationships within code, limiting their effectiveness in detecting complex vulnerabilities. VulDeePecker (Li et al., 2018) marked a significant early advancement, demonstrating that learned representations could reduce dependence on handcrafted rules, as discussed in Section 2.8.3.

Parallel research also investigated convolutional neural networks (CNNs) and broader recurrent neural network (RNN) architectures, demonstrating that these models could identify specific vulnerability patterns when trained on sufficiently large and representative datasets (Harzevili et al., 2024; Wu & Zou, 2022). Collectively, these early deep-learning approaches provided important evidence that data-driven models could surpass traditional static analysis techniques, particularly by reducing false negatives through improved recognition of recurring vulnerability structures. However, their reliance on sequential, token-based representations introduced significant limitations. These models struggled to capture long-range dependencies, cross-function relationships, and unseen structural variations in code, thereby reinforcing the need for more advanced techniques capable of modelling richer program context and semantics.

The introduction of “code as data” and the naturalness hypothesis further spurred the evolution of NLP in software engineering. Researchers observed that source code, despite its formal syntax, exhibits repetitive and predictable patterns analogous to natural language corpora (Allamanis et al., 2018). Large codebases (so-called “big code”) enabled statistical language models to learn the regularities of how programmers write code (ibid.). This led to models treating code like natural language, moving from fragile rule-based analysis to data-driven NLP techniques for software artifacts. By the late 2010s, attention-based models and Transformer architectures began to dominate (Schneider, 2024). Sequence-to-sequence models and attention mechanisms were applied to tasks like code summarization, defect

prediction, and clone detection, laying groundwork for code-focused Transformers as seen today. The evolution culminated in pre-trained code models around 2020, which leverage massive code corpora and learned representations to transfer across many software engineering tasks (Feng et al., 2020). These advances marked a shift from treating code as a bag of tokens toward capturing its structure and context more effectively.

2.11.1 Natural Language vs Programming Language Processing

While programming languages share some characteristics with natural language (for instance, both have vocabularies and grammars), there are fundamental differences that necessitate specialized treatment in NLP for code. Programming languages are formal and unambiguous, with strict syntax and well-defined semantics enforced by compilers, whereas natural language is often ambiguous and context dependent. This means that an NLP model for code can in theory rely on clear structural cues such as punctuation, keywords, and grammar, but must also handle unique challenges like long dependency chains across function calls or files (Pei et al., 2023). Code tokens (identifiers, literals, operators) follow different frequency distributions than words in English; code is highly repetitive and exhibits a property of naturalness (high predictability) due to developers reusing common patterns (Allamanis et al., 2018). Moreover, source code often contains informative naming and comments, natural language elements embedded within code, which can provide semantic hints. Modern approaches attempt to exploit this bimodality. For example, CodeBERT was explicitly trained on paired natural language such as documentation or queries, and source code, treating both as two sides of the same coin (Feng et al., 2020). Such models learn joint embeddings for code and English, enabling tasks like code search (finding code using a natural language query) and code documentation generation.

Another distinction is that reasoning on code requires understanding executable semantics which is what the code is intended to do, an attribute that is not required in typical NLP on plain text. A natural language sentence can often be interpreted in isolation, but a snippet of a code meaning may depend on external context like variable types, library APIs, or data flow through the program. This has led researchers to incorporate program analysis into NLP pipelines (Hai et al., 2025; Hill et al., 2009; Pollock et al., 2007). Unlike plain text, source code can be parsed into ASTs and graphs capturing control or data flow. These

structured representations are often combined with NLP models to link formal code structure and textual representation (Guo et al., 2021).

2.11.2 Transformer-Based Code Models: CodeBERT, GraphCodeBERT, CodeT5

Transformer architecture has revolutionized NLP for code in recent years, giving rise to large, pre-trained language models (PLMs) for source code (Husein et al, 2025; Xiao et al., 2025; Xu and Zhu, 2022). CodeBERT, GraphCodeBERT, and CodeT5 are prominent examples of such models, each illustrating the capabilities and limitations of transformer-based code analysis. CodeBERT (Feng et al., 2020) is an encoder-only Transformer model trained on millions of source code and natural language pairs. It learns a joint embedding space for code and documentation and achieved state-of-the-art results on tasks like code search and code documentation generation (ibid.). For vulnerability detection, CodeBERT can be fine-tuned as a classifier to predict if a piece of code is vulnerable (Xiong & Dong, 2024). Its strength lies in capturing lexical and syntactic patterns in code across diverse programming languages. However, being an encoder-only model, CodeBERT is not designed for code generation or repair tasks, and it also has a limited input length (usually a few hundred tokens) (Khan et al., 2025), which means it may truncate or miss context in large functions or multi-file vulnerabilities (Fan et al., 2024).

GraphCodeBERT (Guo et al., 2021) builds on CodeBERT by incorporating explicit data-flow information during pre-training. Rather than treating source code solely as a linear sequence of tokens, GraphCodeBERT leverages semantic graphs that capture data dependencies. Specifically, the “where-the-value-comes-from” relationships between variables. This enables the model to encode aspects of program semantics, such as whether a variable originates from user input or propagates into a security-critical API. Through structure-aware pre-training tasks, including predicting edges in data-flow graphs, GraphCodeBERT achieves improved performance on code understanding tasks such as clone detection and code refinement (Guo et al., 2021). The integration of code-graph information enhances its capacity to reason about relationships within code, which is particularly important for vulnerability detection, for example when analysing tainted data flows. However, GraphCodeBERT also inherits several limitations as it introduces additional model complexity and computational overhead, and it retains a fixed input context window. Consequently,

although it captures intra-procedural data flows effectively, it may struggle to represent cross-function or cross-file interactions when these exceed its input window or were not sufficiently represented during pre-training.

CodeT5 (Wang et al., 2021), by contrast, employs an encoder–decoder Transformer architecture to support both code understanding and code generation tasks. A key feature of CodeT5 is its identifier-aware design, which places specific emphasis on variable and function names. Wang et al. (2021) introduced a specialised pre-training objective that encourages the model to distinguish and accurately reconstruct identifier tokens when masked, thereby improving semantic comprehension derived from naming conventions. Additionally, CodeT5 incorporates a dual training objective that learns to generate code from comments and comments from code, strengthening alignment between natural language descriptions and program logic. Owing to its encoder–decoder architecture, CodeT5 is applicable not only to classification tasks such as defect or vulnerability detection, but also to code generation and automated repair. Empirical evaluations show that CodeT5 significantly outperforms earlier models across tasks including defect detection, code clone identification, and code summarisation (Wang et al., 2021). Its primary limitation, however, lies in its scale: with hundreds of millions of parameters, particularly in newer “CodeT5+” variants, the model requires substantial computational resources for both fine-tuning and inference. Moreover, although it captures richer semantics than encoder-only models, CodeT5 still processes code as a sequence of tokens and does not inherently perform dynamic analysis. As a result, it may face challenges in detecting vulnerabilities that depend on runtime behaviour or complex pointer interactions.

Overall, CodeBERT, GraphCodeBERT, and CodeT5 have become widely adopted baseline models in contemporary vulnerability detection research (Sheng et al., 2024). Nevertheless, many studies have extended these architectures with additional mechanisms, such as graph-based neural components, enhanced context-fusion strategies, or domain-specific fine-tuning, to better address the complexity and diversity of real-world software vulnerabilities.

Beyond these academic models, a new generation of large-scale proprietary code-oriented language models has emerged, including OpenAI Codex (Chen et al., 2021) and

Meta's Code-Llama (Rozière et al., 2023). With billions of parameters and training on massive code corpora, these models demonstrate the ability to detect certain vulnerabilities through zero-shot or few-shot prompting. However, because they are trained as general-purpose code assistants rather than security-focused models, they frequently overlook domain-specific or subtle security flaws. In response, specialised variants of BERT/GPT optimised for security tasks have been developed. These include VulBERTa, a RoBERTa-based model fine-tuned explicitly for vulnerability detection (Hanif & Maffeis, 2022). While these specialised models incorporate security-relevant knowledge, they often remain constrained by the same limitations observed across the field, particularly the scarcity and uneven quality of labelled vulnerability data, as discussed later in this chapter.

2.11.3 Key Challenges in Learning-Based Vulnerability Detection

Despite the progress in applying NLP to code, significant challenges remain in detecting software vulnerabilities through these methods. One major challenge is achieving true semantic understanding of code. Modern code models mostly learn statistical patterns, and they may flag code that "looks like" known vulnerable code without understanding the underlying program logic. This can lead to false positives when a pattern appears dangerous but is handled safely by subtle logic. Conversely, purely pattern-based models might miss vulnerabilities that require reasoning about uncommon or complex conditions. Long dependency chains and context length pose another critical difficulty. Vulnerabilities often arise from interactions across multiple parts of a program. For instance, a value might be read from an untrusted source in one function, passed through several layers, and finally used in a dangerous operation elsewhere. Capturing these long-range dependencies exceeds the capacity of models that only consider a single function or file at a time. Current Transformer models have a limited context window (often a few hundred or thousand tokens), which is insufficient for large codebases where the relevant clue for a vulnerability might reside in a different file or far-away code block (Liu et al., 2024). As a result, cross-file or system-level vulnerabilities remain very challenging for NLP-based detectors (Sheng et al., 2024). Recent surveys highlight that handling cross-function control flow and data flow is an open problem because models struggle when the vulnerability is not self-contained in the input snippet used for analysis (ibid.). Researchers are exploring ways to mitigate this, such as splitting code and

using retrieval or hierarchical models to stitch together information, but no perfect solution exists yet.

Another challenge is the noise and quality issues in real-world code data. Real-world code is messy with the inclusion of dead code (Chen et al., 2024), debugging fragments, and idiosyncratic styles that can confuse models (Santa et al., 2025). More importantly, real-world vulnerability datasets often contain noisy labels and imbalanced samples. Many publicly available vulnerability datasets have been found to contain mislabelled examples (Chen et al., 2023). For instance, a recent study revealed that a widely used dataset (Devign/CodeXGLUE) had only about 24% of its "vulnerable" functions truly vulnerable, while the rest were mislabelled or not security-relevant, due to the automated collection process (Ding et al., 2024). Similarly, the big "Big-Vul" dataset collected from CVE patches had label accuracy around 25–50% for vulnerable samples, meaning a large portion of functions marked as vulnerable in training data were in fact false positives (Chen et al., 2023; Ding et al., 2024). Training on such noisy data can mislead models, causing them to learn spurious patterns and inflate their evaluation scores by inadvertently "cheating" on overlapping or mislabelled samples. Additionally, data leakage is another pitfall where code datasets often contain duplicate or near-duplicate code across train and test splits (Joeres et al., 2025; Lopez et al., 2025). This risk is directly relevant to the dataset construction and experimental design of this thesis and is addressed explicitly as a limitation in Section 6.8.

Additionally, vulnerabilities are rare events relative to all code, leading to extreme class imbalance. A model might see very few examples of a buffer overflow, amidst thousands of safe functions. Without special handling, the learning algorithm can be biased towards the majority (non-vulnerable) class, resulting in high false negative rates (Mao et al., 2025). Mao et al. (2025) report that deep models often exhibit high false negatives and false positives due to the scarcity and imbalance of vulnerability data. This has driven work on data augmentation and synthesis for vulnerable code. For example, synthetic generation of vulnerable code snippets or augmenting minority classes has shown some success in improving detection rates (Mao et al., 2025). However, synthetically generated data may not capture the full complexity of real vulnerabilities, and models might be overfit to artificial patterns if not careful.

Finally, generalizability issues limit the deployment of NLP-based vulnerability detectors. A model trained on one dataset or programming language often fails to generalize to others. Many learned models are overly specialized to the patterns in the training data; when faced with a novel codebase or a new vulnerability type (so-called zero-day vulnerabilities), their performance degrades sharply (Sheng et al., 2024). Studies have shown that accuracy can dramatically drop in real-world settings that diverge even slightly from the training distribution (Xie et al., 2024). This is a critical concern for security, because attackers are constantly finding new exploitation techniques and a robust detector must handle vulnerabilities it has never seen before. Achieving that may require models to incorporate deeper program analysis or integrate domain knowledge (such as common weakness enumerations, API specifications) rather than purely learning from data. There is also the challenge of explainability and false alarms. Here, security auditors are often wary of "black-box" predictions. If an NLP model flags a snippet as vulnerable, it should ideally explain why by pointing to the insecure data flow or the misuse of an API. Currently, most ML models for code are not interpretable, which hampers trust in their results (Liang et al., 2025). High false positive rates can erode developer confidence in automated tools. While some transformer models (like large GPT-based systems) can generate explanations or fixes, they may still hallucinate or be inconsistent, so their reliability is in question (Huang et al., 2023).

Although, the state-of-the-art models can detect many common vulnerabilities and even approach human-like performance on curated benchmarks, but significant challenges remain. Key barriers include capturing the true semantics of code (not just surface patterns), handling the long-range and multi-module nature of real vulnerabilities, improving data quality and quantity, and ensuring models generalize across different contexts and remain interpretable. These challenges motivate ongoing research and the need for novel approaches such as models that combine NLP with static analysis, or domain-specific training data that cover critical application areas like healthcare software.

2.11.4 Datasets and Generalizability in Software Security Tasks

The development of datasets has played a pivotal role in advancing NLP methods for vulnerability detection, yet current benchmarks have notable gaps affecting generalizability.

Early works relied on small, synthetic datasets such as the Juliet test suite[12] (with short code examples for known CWE vulnerabilities) or on handcrafted examples (Alqaradaghi et al., 2021). As the field progressed, researchers compiled larger datasets from open-source repositories including Devign (Zhou et al., 2019) collected real vulnerability fixes from C/C++ projects; Big-Vul (Fan et al., 2020) aggregated thousands of CVE-related code changes; D2A and Draper VDISC provided labelled vulnerable versus safe functions from real software. These datasets enabled training of data-driven models, but they often suffer from the noise issues described above.

Further, a notable gap is the lack of domain-specific vulnerability datasets. Most public datasets draw from open-source systems or synthetic examples, which may not reflect security challenges in specialized domains like healthcare, industrial control, or IoT firmware. This motivates the creation of targeted datasets such as a healthcare cybersecurity code dataset which might include hospital information system software, medical device firmware code, or healthcare-specific protocols, which are not present in current corpora. By focusing on a particular domain, one can capture domain-specific vulnerability patterns (such as misuse of healthcare data standards or medical device interface flaws) that general models might overlook. Indeed, researchers have pointed out the need for benchmarks that simulate real-world scenarios more closely, including repository-level and system-level code analysis (Sheng et al., 2024). The goal is to push models to analyse code in context (entire projects, with realistic dependencies) rather than isolated snippets. Until such datasets are widely available, models fine-tuned on synthetic examples might not generalize to enterprise-scale software. Cross-project generalization remains an open problem and in a recent survey, it was found that many ML-based detectors show significant performance drops on software code outside of their training data, underscoring weak generalizability (Liang et al., 2025). Achieving robust generalization likely requires both better data (including larger and more varied training sets, covering multiple languages and domains) and better modelling, by using

[12] https://www.nist.gov/publications/juliet-11-cc-and-java-test-suite#:~:text=Abstract,code%20to%20test%20tool%20discrimination.

techniques for adapting models to new projects, or training on abstract representations that capture vulnerability concepts beyond surface code (Cao et al., 2025; Hu et al., 2023).

Another issue connected to datasets is the evaluation metric and task definition. Different works have framed vulnerability detection differently. While some do so as a binary classification of functions (Harzevili et al., 2024), others as a multi-class classification into specific CWE types (McLaughlin & Lu, 2024), and others as a code summarization or repair task (where the model must identify and even fix the vulnerable line) (Zhou et al., 2024). This makes it hard to compare approaches directly and to understand their limits.

Consequently, Xie et al. (2024) constructed a benchmark with five tasks of increasing difficulty (from simply detecting if any vulnerability exists, up to pinpointing the exact vulnerable line) and tested a range of large language models. They found that while many models can handle the simplest classification, their performance deteriorates on more specific tasks like exact location or explaining the vulnerability. This indicates current models do not fully comprehend the vulnerability because if while even being able to detect, are unable to show the exact what or where. For the field, this means datasets are needed that not only tell if code is vulnerable but also provide rich annotations (like the vulnerable lines, the CWE category, and probably a description) to train and test models on deeper understanding.

In conclusion, dataset limitations are a major bottleneck for NLP-based software vulnerability detection. High-quality, domain-relevant, and diverse data is essential for training models that generalize and perform reliably. The research community is actively addressing this by curating better datasets and benchmarks. The motivation for a novel healthcare-oriented NLP dataset for security, as in this thesis, is well-grounded because it will fill a crucial gap by providing focused data on a critical domain, enabling the development of models that can detect vulnerabilities in healthcare software with greater accuracy. Coupled with a robust machine learning method for static code analysis, such contributions will help move the field toward practical, trustworthy vulnerability detection tools that can be deployed in real-world software development workflows.

2.12 Systematic Review Methodology (Ameh et al., 2025)

The findings synthesised in this section, including the study counts referenced throughout the gap analysis below, derive from a fully systematic literature review conducted as part of this research and reported in full elsewhere (Ameh et al., 2025). This sub-section summarises methodology of the review for transparency; the complete protocol, search strings, and included-study list are provided in that appendices of the publication.

The review followed the structured eight-step framework proposed by Tingelhoff et al. (2024) for software engineering systematic reviews, rather than the PRISMA clinical-research framework, on the grounds that Tingelhoff et al.'s framework is better suited to the study designs and evaluation methods characteristic of software engineering research. The eight steps including scope definition, search strategy development, eligibility criteria, systematic search execution, screening and selection, data extraction, quality assessment, and synthesis, were each explicitly documented (Ameh et al., 2025, Table 2).

Databases and search strategy. An automated search was conducted across four databases: the ACM Digital Library, IEEE Xplore, SCOPUS, and Web of Science, selected for their complementary coverage of computing and multidisciplinary security/systems literature (Valente et al., 2022). Search strings were developed around four research objectives (vulnerability detection tasks, ML methodologies, datasets/tools, and challenges) and iteratively refined through pilot testing following Collins et al. (2021). The search was restricted to studies published between 1 January 2019 and 31 December 2024, in English, and available as peer-reviewed conference or journal articles.

Screening and selection. Screening proceeded in three stages: (1) title/abstract screening by the lead author; (2) independent application of inclusion/exclusion criteria by two further authors, with disagreements resolved by discussion; (3) full-text review of the remaining pool by the lead author to confirm eligibility. Inter-rater reliability was assessed via a pilot screening of 50 randomly selected papers, achieving a Cohen's kappa of 0.84.

Inclusion criteria: peer-reviewed conference or journal articles, published 2019–2024, in English, explicitly addressing software vulnerability detection in source code using ML/deep learning techniques, with methodology sections specifying algorithms, datasets, and evaluation metrics.

Exclusion criteria: insufficient technical detail to assess the ML approach; vulnerability detection outside source-code analysis (e.g., binary-only or network-traffic analysis); non-peer-reviewed materials (book abstracts, editorials, workshop posters); duplicate publications (most comprehensive version retained); studies not addressing any of the four review objectives.

Results. *An initial pool of approximately 202,952 records was retrieved across the four databases (ACM: 189,016; IEEE: 1,034; SCOPUS: 67; Web of Science: 12,835). Following title/abstract screening, independent eligibility assessment, and full-text review, 83 studies were retained for synthesis (ACM: 33; IEEE: 9; SCOPUS: 11; Web of Science: 30). The full PRISMA-style flow is shown in*

Figure 2-7.

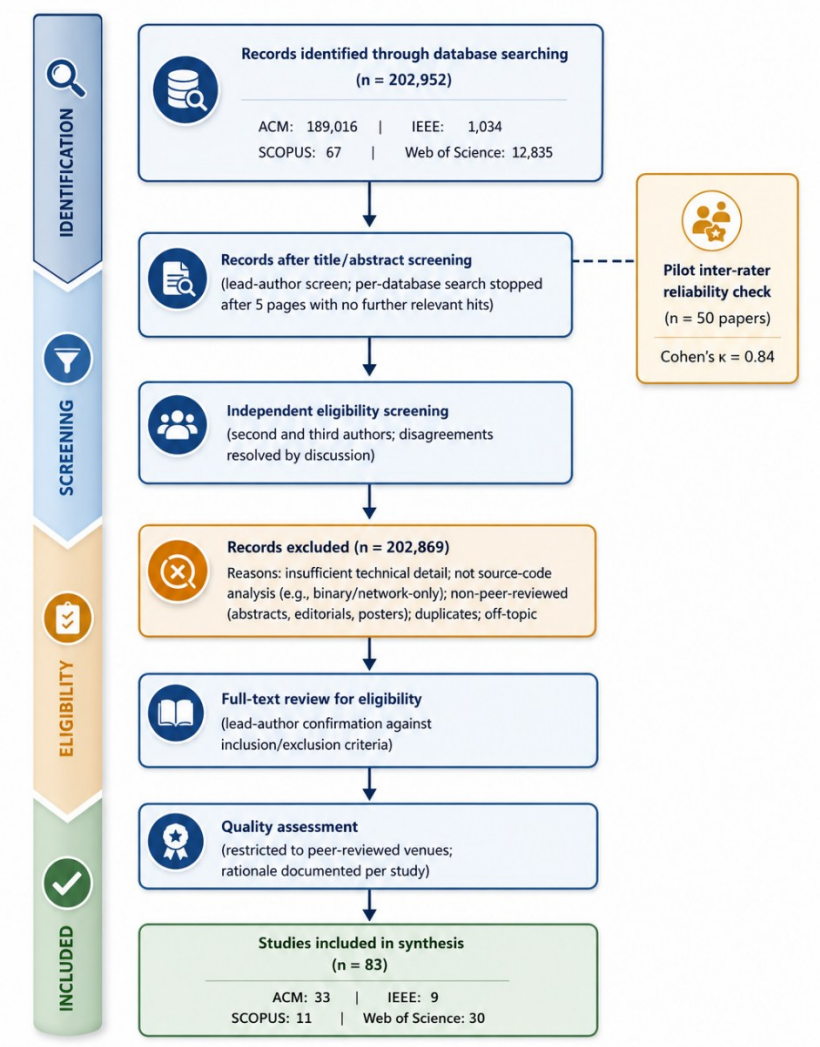


Figure 2-7: PRISMA-style flow diagram of the study selection process for the systematic review underpinning the gap analysis in this section

2.13 Summary of Identified Gaps

The preceding review of literature across the domains of secure software development, privacy engineering, threat modelling, application security testing, and machine learning-

based vulnerability detection reveals a constellation of interconnected research gaps. Although these have been noted individually within their respective thematic sections, this concluding section draws them together systematically to articulate the precise research space that this thesis occupies.

The gaps identified below are drawn from the literature examined across the preceding sections of this chapter (Sections 2.2–2.11), which reviewed publications primarily from 2019 to 2025 across secure software development, privacy engineering, threat modelling, and machine learning-based vulnerability detection. Where a gap references specific study counts, this refers to the 83-study systematic review reported in Section 2.12 (Ameh et al., 2025), whose full database and date-range scope is documented there. Claims below of the form 'the literature reviewed did not identify' or 'no study was found' refer to this defined scope and are not claims about the entirety of unpublished, non-English, or non-indexed work.

Gap 1: The absence of a validated bridge between privacy threat modelling and code-level vulnerability detection. Privacy threat modelling frameworks such as LINDDUN operate at the architectural and data flow level, identifying categories of privacy risk in system design. Vulnerability detection tools and taxonomies such as CWE operate at the code level, classifying technical weaknesses in implementation. Despite the conceptual relatedness of these two disciplines, the literature reviewed in this chapter did not identify a systematic, empirically validated mechanism for translating design-time privacy threats into the specific code-level weaknesses through which those threats are realised. Honkaranta et al. (2021) confirm that existing mapping efforts between threat frameworks and CWE remain incomplete and largely manual, requiring expert judgement and producing inconsistent results. This gap means that even when LINDDUN analysis correctly identifies a privacy threat during design, development teams have no structured means of ensuring that the corresponding CWE-classified weaknesses are targeted during static analysis. The absence of this bridge represents the central motivating gap of this thesis.

Gap 2: The lack of a large-scale, healthcare-specific vulnerability dataset. Most publicly available vulnerability datasets including Devign, Big-Vul, DiverseVul, and the Juliet test suite, are drawn from general open-source repositories with no domain-specific curation. As the

literature review establishes, and as the systematic review by Ameh et al. (2025) quantifies, 50 of 83 reviewed studies rely exclusively on C/C++ code but none of these 83 studies incorporate healthcare-domain provenance, LINDDUN privacy annotations, or vulnerability profiles representative of clinical software. This creates a significant generalisability problem: models trained on general-purpose datasets cannot be expected to perform reliably on healthcare software, which exhibits distinct vulnerability profiles characterised by hard-coded credentials, access control failures, and insecure medical device interface patterns. The absence of a healthcare-specific, privacy-annotated dataset is identified as a critical bottleneck for the development of domain-grounded detection tools.

Gap 3: The disconnect between privacy-by-design principles and their practical operationalisation in healthcare software development. The literature documents broad awareness of privacy-by-design and security-by-design principles across regulatory frameworks including GDPR, HIPAA, and NHS Data Security Standards, as well as methodologies such as LINDDUN. However, a persistent implementation gap exists between these principles and their consistent application in practice. Del-Real et al. (2024) note that PbD is well-defined in theory, but its implementation requires interdisciplinary collaboration that most development organisations have yet to achieve. Ebad (2022) and Van Rest et al. (2014) similarly observe a gap between principle and practice. In healthcare specifically, Arizon-Peretz et al. (2022) found that many EHR solutions do not extensively employ PbD mechanisms despite regulatory mandates. This gap is compounded by the failure of the reviewed literature to identify tooling that connects privacy engineering activities at design time to code-level security testing, leaving shift-left security objectives unfulfilled.

Gap 4: The high false positive rate and limited adaptability of rule-based static analysis tools represent a barrier to adoption that is structurally more severe in healthcare software development than in general software engineering contexts, for three reasons specific to the domain. First, healthcare development teams typically lack dedicated security engineering staff, meaning that vulnerability alerts are triaged by developers with limited security expertise alongside their primary development responsibilities; excessive false positives in this context do not merely reduce efficiency but increase the probability that genuine vulnerabilities are dismissed as noise alongside spurious ones (Wadhams et al.,

2024). Second, regulatory validation requirements under FDA premarket cybersecurity guidance and NHS Data Security Standards mean that security review findings must be documented and justified, a process that is disproportionately costly when a large proportion of findings are false positives requiring documented dismissal. Third, the operational continuity requirements of healthcare systems create organisational pressure to minimise disruption from security tooling, making teams more likely to disable or ignore tools with high false positive rates rather than invest in tuning them. Despite these heightened consequences, existing static analysis tools are evaluated almost exclusively on general-purpose benchmarks where false positive costs are treated as uniform across contexts. Within the literature reviewed, no published work was identified that evaluates false positive impact specifically under healthcare development workflow conditions or that proposes domain-adapted thresholds accounting for the asymmetric cost of false alarms in this sector.

Gap 5: The under exploration of computationally efficient and interpretable machine learning architectures for vulnerability detection is a gap with direct consequences for healthcare deployment that the literature reviewed here has not addressed. Two healthcare-specific requirements heighten this above a general computational efficiency concern. First, interpretability in healthcare AI is not merely a usability preference but is increasingly a regulatory obligation. Both the UK Medicines and Healthcare products Regulatory Agency and the US FDA are developing frameworks for qualifying AI-based software as medical device components or as components of software-as-a-medical-device, both of which require that decision pathways be auditable and explainable to clinical governance bodies (MHRA, 2024; FDA, 2023). A vulnerability detection system whose predictions cannot be explained cannot be certified under these frameworks regardless of its accuracy. Second, many healthcare software development organisations, in particular smaller medical device manufacturers and NHS trust in-house development teams, do not operate GPU infrastructure. The systematic review by Ameh et al. (2025) found that 70 of the 83 reviewed vulnerability detection studies rely exclusively on deep learning architectures requiring GPU acceleration for practical inference, meaning that within this reviewed body of work, most published models are architecturally inaccessible to the organisations with the greatest need for automated security tooling. Traditional machine learning and hybrid

approaches were comparatively rare within these 83 studies, appearing in only 6 and 5 studies respectively, leaving a significant gap in understanding whether simpler, more deployable approaches can achieve adequate performance for healthcare contexts.

Gap 6: The lack of real-time, deployment-ready vulnerability detection systems with false-negative guarantees. The literature review reveals that the overwhelming majority of ML-based vulnerability detection research is conducted in academic benchmark settings with no pathway to production deployment. Systems are evaluated on held-out test sets but not deployed as callable services integrated into development workflows. Furthermore, no system was identified within the reviewed literature that provides hard guarantees on false negatives which is a requirement of particular importance in healthcare, where an undetected vulnerability in medical device software or an EHR system can have direct patient safety consequences. The combination of real-time feasibility, CPU-only inference, CWE-specific remediation guidance, and false-negative guarantees was not identified as an occupied position within the literature reviewed for this thesis.

Gap 7: Inconsistency in evaluation methodology across the vulnerability detection literature. The literature review identifies significant heterogeneity in how vulnerability detection models are evaluated: some studies frame the task as binary classification at function level, others as multi-class CWE classification, and others as line-level localisation. Most report precision, recall, and F1 but omit false positive rate, true negative rate, and PR-AUC, metrics of particular importance in highly imbalanced settings. This inconsistency makes direct comparison across studies unreliable and prevents the field from establishing clear performance baselines. The reviewed literature's lack of standardised evaluation protocols, particularly for multi-class CWE classification in domain-specific contexts, constitutes a methodological gap with implications for how progress in the field is measured and reported.

Taken together, these seven gaps define the research space that this thesis addresses. Gaps 1 and 3 motivate the LINDDUN-to-CWE mapping contribution. Gap 2 motivates the construction of the C3-VULMAP dataset. Gaps 4 and 5 motivate the comparative evaluation across 14 models and three ML paradigms. Gaps 6 and 7 motivate the hybrid detection system and its evaluation against a comprehensive metric suite. The subsequent chapters present each contribution in turn; with the specific gap it addresses making explicit the thread of

continuity between the literature reviewed here and the research conducted. Beyond the seven gaps enumerated above, a further observation from the systematic review warrants acknowledgement, though it falls outside the scope of the present research to address directly.

A further observation from the systematic review concerns the near-total absence of self-supervised and semi-supervised learning paradigms in vulnerability detection research, with only one of 83 reviewed studies employing such approaches (Cheng et al., 2022). This is significant given that class imbalance and labelled data scarcity are the dominant practical obstacles to deploying ML-based vulnerability detection in specialised domains. The present research does not directly address this gap, the supervised learning baseline established in Chapter 4 is a deliberate methodological choice that prioritises reproducibility and comparability with the existing literature, but the comprehensive supervised evaluation contributes an empirical reference point against which future self-supervised or semi-supervised approaches can be benchmarked. The conditions under which unsupervised paradigms might outperform the supervised baseline established here, and what improvement threshold would justify their additional complexity in healthcare deployment contexts, are identified as priority directions for future work in Chapter 6.

3 Chapter 3: Enhanced LINDDUN Privacy Threat Methodology for Healthcare Software Systems

3.1 Background

The preceding chapters established the threat landscape facing healthcare software systems and reviewed the body of literature on secure software development, privacy engineering, and machine learning-based vulnerability detection. Chapter 2 identified a persistent and consequential gap in current practice: privacy threat modelling frameworks and software vulnerability assessment tools operate as separate disciplines, with no systematic mechanism for translating design-level privacy risks into the code-level weaknesses through which those risks are realised. This chapter addresses that gap directly by developing and applying an enhanced privacy threat modelling methodology purpose-built for the healthcare software context.

While established security frameworks such as the CIA (confidentiality, integrity, availability) triad provide an indispensable foundation, they do not capture the full spectrum of privacy-specific vulnerabilities that arise in healthcare software development. Privacy encompasses dimensions frequently overlooked by classical security measures, including data collection practices, authorisation scopes, and patient agency over personal information. A system may achieve technical security against external threats and yet still compromise privacy through unnecessary data collection, inappropriate internal access, or inadequate transparency to patients about how their data is processed. In healthcare, where the sensitivity of clinical data and the directness of its connection to patient safety create heightened obligations, this distinction between security and privacy is not merely academic. It has direct consequences for regulatory compliance, patient trust, and the integrity of care delivery.

Privacy-by-Design, recognised in frameworks such as GDPR Article 25, addresses this by requiring privacy considerations to be embedded from the earliest stages of software development rather than applied retrospectively. Practically, this demands a threat modelling approach capable of identifying privacy-specific risks at the architectural level and tracing them through to concrete implementation guidance for developers. As established in Chapter

2 (Section 2.5), LINDDUN is the most appropriate framework for this purpose in healthcare settings since its seven privacy threat categories align closely with the data-intensive, regulation-constrained nature of clinical software, and its DFD-centric methodology provides the structured analytical foundation needed to connect design-level risks to code-level investigation. Building on that established justification, this chapter extends the standard LINDDUN solution space in two novel ways: by systematically mapping each identified privacy threat to relevant Common Weakness Enumeration entries, and by using those mappings as the basis for constructing a healthcare-specific vulnerability dataset for machine learning applications.

The methodology developed in this chapter and its application to healthcare information system components have been reported in a peer-reviewed publication: Ameh, J.E., Otebolaku, A., et al. (2025), "C3-VULMAP: A Dataset for Privacy-Aware Vulnerability Detection in Healthcare Systems," *Electronics*, 14(13), 2703. That publication constitutes a preliminary output of this research programme and presents the core findings of the LINDDUN-to-CWE mapping and the resulting dataset construction. The account provided in this chapter and in Chapter 4 represents the full and complete treatment of the methodology, of which the published paper is a condensed version. All validation data, traceability artefacts, and expert review results reported in this chapter are original to this thesis and were not fully reported in the published paper due to space constraints.

3.1.1 Privacy-by-Design and Traceable Privacy Risks

Given the significant privacy risks in healthcare software, privacy considerations must be central to design, implementation, and testing and thus surpassing traditional emphasis on generic cybersecurity measures. For instance, telehealth devices, even when securely protected against intrusion, can pose privacy risks by recording sensitive aspects of daily life or sharing health-related information beyond patient expectations (Mishkin et al., 2023).

In response, PbD has emerged as an essential principle within healthcare software engineering, emphasising early integration of privacy requirements throughout the software development lifecycle. This paradigm, recognised in regulations such as GDPR Article 25 ("Data Protection by Design and by Default"), urges developers to anticipate and embed

privacy safeguards from inception rather than relying on post-deployment remedies. Practically, this involves implementing data minimisation, purpose limitation, and clear consent mechanisms from the outset. Jawad (2024) argues that employing PbD in digital health systems significantly reduces privacy risks by curbing unnecessary exposure and aligning operational behaviour with patient expectations.

To systematically assess privacy risks, researchers have refined threat modelling methodologies to incorporate privacy-specific frameworks. A notable example is LINDDUN, which structures privacy threat analysis around seven categories. By mapping these categories to software design elements, LINDDUN enables identification of privacy weaknesses such as re-identification risks or insufficient user awareness, that conventional security reviews might miss. Wuyts et al. (2018) demonstrate that privacy-oriented threat modelling uncovers issues like excessive data aggregation and missing consent flows which may not register as traditional vulnerabilities.

However, identifying privacy threats at a high level is only the first step; the goal is translating these insights into concrete software defences. Traditional threat modelling outputs, such as "possibility of patient re-identification from anonymised records", can be too abstract for developers to act upon directly. The innovation in an enhanced LINDDUN approach is mapping each identified threat to tangible software vulnerabilities, creating a clear line of sight from design-phase concerns to code-level solutions. For example, if threat modelling flags the risk of linkability (an attacker linking patient records across systems), this can be tied to specific software weaknesses such as persistent unique identifiers. The actionable follow-up might be implementing tokenisation or encryption to break the link.

This research proposes systematic mapping of LINDDUN threats to Common Weakness Enumerations (CWEs), linking each high-level privacy threat to relevant coding vulnerabilities. For instance, a Disclosure threat might correspond to CWE-200 (Exposure of Sensitive Information) or CWE-522 (Insufficiently Protected Credentials). This mapping yields bidirectional traceability: given code or a known weakness, one can identify which privacy threat it enables; conversely, given a privacy threat, one can pinpoint which weakness classes to check in implementation. This approach bridges the traditional gap between design-level

risk assessment and code-level security testing, ensuring privacy considerations lead to concrete code hardening rather than remaining as abstract checkboxes (Ameh et al., 2025).

A further benefit of threat-to-weakness traceability is improved accountability. By creating a traceable matrix, engineers can track each identified threat to mitigations (design decisions or code fixes) and then to test cases validating those mitigations. This is analogous to maintaining a requirements traceability matrix but focused on privacy threats, enabling project teams to demonstrate for compliance or auditing purposes that each privacy risk has been addressed by specific measures. Threat modelling thus becomes not a one-time documentation exercise but a living guide informing development and testing tasks.

Integration of privacy considerations into testing is equally critical. While conventional security testing focuses on identifying exploitable flaws, privacy-focused testing examines data lifecycles, compliance, and user experience to verify alignment with privacy policies and regulations. Test cases should validate behaviours such as secure deletion of patient data and comprehensive audit logging. Development teams can deploy static code analysis, dynamic API assessments, and multidisciplinary review including legal, clinical, and engineering perspectives, to uncover less apparent privacy challenges such as metadata leaks or confusing consent interfaces. By embracing PbD with traceable implementation, healthcare organisations can ensure their software not only withstands external threats but defaults to privacy-preserving operation throughout the development lifecycle.

The traditional LINDDUN process comprises a problem space and a solution space (as seen in Figure 1-3). In the problem space, analysts first construct a DFD of the system, modelling data sources, processes, storages, and flows with trust boundaries. This DFD-centric modelling is standard in threat analysis (for example, the Microsoft threat modelling tool begins with a DFD) (Das et al., 2024). Once the DFD is established, the next step is threat mapping where for each DFD element and flow, the LINDDUN methodology enquires whether any of its seven privacy threat types applies. For instance, a database storing patient data might implicate Disclosure or Identifiability threats, whereas a data flow across a network might raise Linkability or Unawareness issues. In parallel, misuse cases (negative use cases) are created to identify specific attacker scenarios against privacy requirements. These misuse cases illustrate how an adversary or error could lead to a privacy violation (example given,

forging records, exploiting unencrypted logs). Together, the DFD-based threat enumeration and misuse case analysis populate the LINDDUN privacy threat model for the system. Recent literature defines these steps and threat taxonomies precisely (Das et al., 2024, Lund et al., 2010).

However, the solution space of LINDDUN involves prioritization and mitigation. After threats are identified, they must be prioritized (often using risk factors or stakeholder input) to focus effort on the highest-impact privacy violations. Then, mitigation strategies are planned. LINDDUN provides guidance by mapping threats to specific countermeasures or privacy-enhancing technologies. For instance, a Disclosure threat might be mitigated by encryption or anonymization while an Unawareness threat by enhancing transparency and consent mechanisms. The KU Leuven LINDDUN knowledge base and related literature catalogue such mitigations. Analysts select from available controls or privacy design patterns (such as de-identification, pseudonymization, data minimization techniques) that address each threat. These steps integrate privacy-by-design principles into system architecture. Although detailed automated guides exist (some mapping tools for STRIDE/DINDDUN have been developed), practitioners still largely perform this analysis manually, supported by checklists and mitigation catalogues.

Building on this foundation, the enhanced research method extends solution space of LINDDUN in two novel ways. First, it systematically maps LINDDUN threats to software CWEs. In other words, each high-level privacy threat is linked to relevant coding vulnerabilities or weakness classes. For example, a LINDDUN Disclosure threat may correspond to CWE-200 (Information Exposure) or CWE-202 (Exposure of Sensitive Information Through Data Queries). This mapping is derived by analysing how privacy issues manifest in code. By bridging from privacy threats to specific CWE entries, developers can trace an abstract privacy threat back to concrete coding patterns to check or fix. The second enhancement is creation of a healthcare-specific vulnerability dataset for NLP. This involves collecting and labelling code functions from software with both CWE and LINDDUN annotations. As reported by Ameh et al. (2025), the C3-VULMAP dataset contains 86,239 vulnerable and 6,222,075 million non-vulnerable C/C++ functions, each tagged with CWE categories and mapped to LINDDUN

privacy threat types. This healthcare-focused dataset enables machine learning and NLP techniques to detect privacy-relevant vulnerabilities.

Consequently, this LINDDUN-to-CWE mapping constitutes a novel contribution because it connects privacy analysis and code-level vulnerability assessment. Prior work has largely treated privacy threat modelling and security bug detection as separate tasks. However, by aligning the privacy concepts derived from LINDDUN with the CWE taxonomy, this method closes the gap as it allows automated scanners or ML models to predict privacy threat categories from code (and vice versa). As demonstrated by the experimental evaluation presented in Chapter 4 of this thesis (Ameh et al., 2025). In doing so, the dataset "fills a critical gap in privacy threat modelling and vulnerability detection". Hence, integrating CWE mapping and domain-specific data collection extends the standard LINDDUN solution space into implementation-level analysis, which is especially valuable for complex healthcare systems where traceability from requirements to code is essential.

3.1.2 Language and Implementation Context

The healthcare software domain is not only defined by high-level architectures, but also by the low-level implementation technologies used to build those systems. A notable characteristic is the prevalence of C and C++ in performance-critical healthcare applications. Many hospital systems and medical devices have legacy components written in C/C++, which were historically chosen for their efficiency and fine-grained control over hardware (important for real-time monitoring, imaging, and other resource-intensive tasks). For instance, imaging workstations that process MRI/CT scans, PACS servers that handle large image files, patient monitoring bedside devices, and laboratory analysers often rely on C/C++ codebases. Even as higher-level languages gain popularity for web interfaces and scripting, the backbone of healthcare IT, especially where speed and hardware interaction matter, frequently involves native code. A recent survey confirms that C/C++ remain dominant in medical device firmware and high-performance health applications (Zouev, 2019). The continued use of these languages comes at a cost as they introduce well-known memory safety vulnerabilities that can directly impact privacy if not managed. Unlike memory-safe languages (Java, C# or Python), C/C++ require manual memory management, making them prone to issues such as buffer overflows, heap/stack corruptions, use-after-free errors, and

memory leaks (Pereira et al., 2021). In a healthcare context, these vulnerabilities are not merely theoretical, they can lead to exposure of sensitive patient data. For example, a buffer overflow in a medical imaging software could allow an attacker to read memory containing patient images or reports (a violation of confidentiality). Similarly, a use-after-free bug in an EHR module might be exploited to retrieve fragments of prior patient data transactions from memory. Real-world incidents echo this concern. Yadav (2024) notes that the IoMT and modern healthcare tech expansions bring new vectors for software exploits, necessitating robust handling of such classic vulnerabilities in code. Given that memory corruption and related CWE are frequent in C/C++, tying them to privacy impacts is a key motivation for the approach used in this study. In the enhanced LINDDUN methodology, each privacy threat identified is considered in view of potential C/C++ weaknesses that could cause it. For example, a Disclosure of Information privacy threat might correspond to CWE-125 (out-of-bounds read) if an out-of-bounds memory access could read sensitive data; an Unawareness threat (user unaware of data handling) might map to CWE-359 (exposure of private record via debug code) if developers leave unintended access points. By integrating such mappings, it is ensured that the privacy analysis does not stop at the design level but drills down to the code constructs enabling (or preventing) those threats. To effectively analyse code for privacy weaknesses, focusing on the function-level granularity has proven advantageous. A function (or method) in source code is a natural unit of computation and encapsulates specific behaviours in healthcare software. A function might load a patient record, perform an image processing step, or transmit a message.

This research has adopted function-level vulnerability analysis to assess privacy risks in C/C++ code. This decision is supported by the empirical evaluation of He et al. (2025), which found function-level analysis superior to line-level or file-level for capturing vulnerability patterns in healthcare code. Each function often corresponds to a specific processing step (example given, encrypting a file, querying the database, parsing a message); thus, analysing weaknesses at this level aligns with how developers think about program logic and how code reviews are conducted in practice. Notably, functions also map well to threat scenarios: one can often say, “this function could lead to an Identifiability threat if it mishandles patient IDs,”

making it straightforward to link a function's CWE-tagged vulnerability to a LINDDUN threat category. The result is a traceable chain from privacy requirement to code implementation.

The intersection of design-level privacy analysis with code-level scrutiny is crucial for healthcare, where the stakes of a data breach are extremely high. The next section (3.2 Methods) builds on this foundation: it details how this integrated LINDDUN-CWE approach is carried out in practice, including the adaptation of LINDDUN steps to the healthcare DFD and the development of tools to systematically map and detect privacy-related weaknesses at the code level.

3.2 Methods

This section details the methodology developed to extend the LINDDUN privacy threat modelling framework for a healthcare software context. The overall workflow (illustrated in **Error! Reference source not found.**) starts with inputs including a domain-specific DFD of the healthcare system, a classification of data assets, systematically eliciting privacy threats in each LINDDUN category, then mapping each identified threat instance to candidate software weakness types using the CWE taxonomy. This end-to-end methodology aligns with best practices in PbD, which demands traceable links from identified threats down to implementation fixes (Sion et al., 2024). It also addresses a noted gap in traditional threat modelling, which is the lack of connection between abstract threat models and the concrete code-level vulnerabilities that enable those threats (Honkaranta et al., 2022).

3.2.1 Context

Alabi (also referred to as the patient) visits his GP, for a routine check-up. He explains his symptoms including shortness of breath, chest pain, and occasional wheezing. The GP reviews his medical history, performs a physical check, and orders blood tests, a chest X-ray, and a pulmonary function test (PFT). The GP then updates the patient's personal and health (symptoms and test requests, etc) details on the electronic health record (EHR) system. The GP decides that the condition requires treatment with medicine based on the results of the tests and the diagnosis. He adds information about the prescribed medication, such as dosage instructions and possible side effects, into the EHR system.

The GP and Alabi talk about how important it is to make lifestyle changes so that the treatment will have a better effect. Referrals and healthy living suggestions are entered into the EHR system, to ensure that care is coordinated.

The patient's prognosis shows that he requires specialized care, the GP refers him to consultants by initiating online referral letters via the EHR system and sends them securely to cardiologists for his atherosclerosis and pulmonologists for his long-term lung disease. The letters of referral have information about Alabi's health and personal life, as well as the appropriate test results and the reason for the referral. Routine follow up and preventive care visits are also set up for the patient by the GP. All these are set up on the EHR which is a part of the healthcare information system.

Overall, the integration of different technologies (see Healthcare Information System (HIS) High-level Data Flow Diagram (DFD) *(See* Figure 3-1*) like the EHR system, the e-prescribing system, the referral management system, secure messaging, remote monitoring, and automated reminders make it easier for Alabi's doctors to talk to each other, work together on his care, and handle his conditions in an efficient way. These tools help streamline workflows, improve patient safety, make data easier to access, and make sure that healthcare is centred on the patient.*

However, if the code of the underlying software of the components in this system is vulnerable to threats, the system is insecure and can lead to privacy breaches.

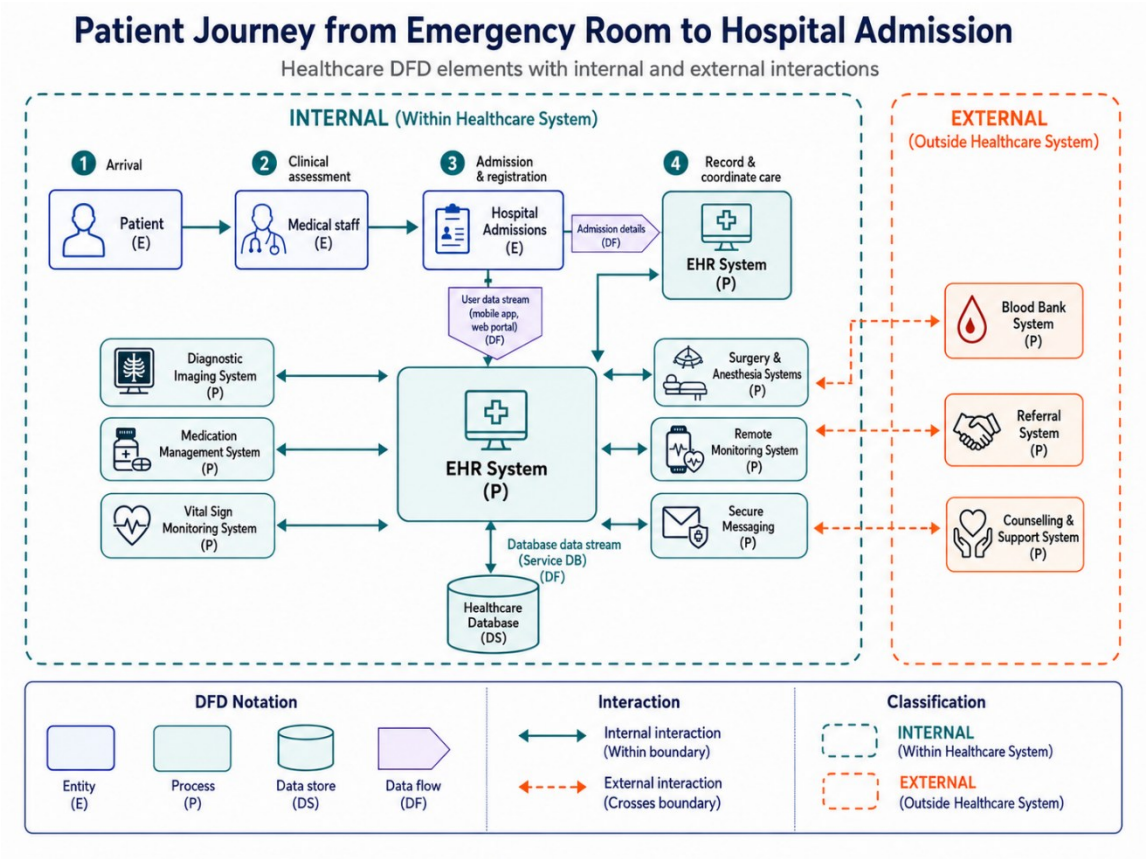


Figure 3-1: Healthcare Information System (HIS) High-level Data Flow Diagram (DFD)

3.2.1.1 Domain Data Flow Diagram as the Anchor

Privacy threat modelling is most effective when grounded in a clear understanding of the data flows and architecture of a system. Therefore, a healthcare-specific DFD is used as the anchor for privacy analysis, capturing how patient data moves through the system components and where it crosses trust boundaries. LINDDUN and similar methodologies explicitly start with DFD modelling of the system, because the DFD identifies all data stores, processes, and external entities, essentially mapping what data exists, where it flows, and who handles it. This forms the foundation for systematically analysing privacy threats. In the healthcare domain, a DFD can be structured around the typical patient journey through an information system of a hospital which starts from initial registration, through diagnostics and treatment, to discharge and follow-up. As seen in **Error! Reference source not found.**, each stage in this journey involves distinct subsystems and data assets that must be examined for privacy risks.

In a hospital scenario as presented in Section 3.2.1, the patient journey can be outlined in several steps, each supported by specific software subsystems:

Registration: The patient's personal data (demographics, contact, insurance details) is collected, often via a front-end registration module or patient portal, and stored in the core Electronic Health Record (EHR) system. The EHR serves as the central repository for patient identifiers and medical history. This step establishes a first point of data entry, where privacy concerns include proper notice/consent and secure handling of personally identifiable information.

Diagnostics: During diagnosis, various systems produce clinical data. For example, laboratory tests may be ordered and results received through a Laboratory Information System (LIS), and medical imaging studies (X-rays, MRIs) are managed by a Picture Archiving and Communication. The PACS stores sensitive imaging data and communicates with the EHR (often via HL7 or DICOM interfaces) to attach reports to the patient's record. Privacy threats at this stage involve unauthorized access to test results or images, and linkage of imaging data to patient identities (addressed by controls like access authentication and pseudonymization in research contexts).

Treatment: Treatment generates data on medications and interventions. A Medication/Pharmacy system handles e-prescriptions, medication administration records, and pharmacy dispensing information. For instance, when a doctor prescribes a drug in the EHR, the order may be processed by the pharmacy subsystem, which could be internal or an integrated external service. Also, bedside medical devices and IoMT (Internet of Medical Things) monitoring devices continuously collect vital signs and therapy data (example given, ICU monitors, insulin pumps). These devices feed data into a Patient Monitoring System in the hospital network. The IoMT extends healthcare beyond traditional on-premises boundaries with wearable or home devices enabling remote patient monitoring, blurring the lines of the network perimeter of the hospital. Privacy issues during treatment include ensuring that only authorized caregivers access patient vitals and medication orders, and that device data transmissions are encrypted to prevent eavesdropping.

Discharge: At discharge, a summary of the hospitalization and follow-up instructions are compiled. The EHR typically generates a discharge summary containing diagnoses, procedures, and recommendations. This summary may be shared with external healthcare providers (example given, the primary care physician or a specialist for referral) through a Referrals system or health information exchange. Here, data leaves the direct control of the hospital, crossing into external systems, a critical trust boundary. Ensuring privacy might involve secure messaging or document exchange protocols and obtaining patient consent for data transfer. Unauthorized disclosure of discharge summaries is a concern, as these documents contain comprehensive personal health information.

Follow-up: After discharge, patients often continue care via outpatient visits or telehealth. A Secure Messaging platform or patient portal enables communications such as appointment scheduling, follow-up questions, or remote consultations. Patients and providers exchange messages that may include personal health details (symptoms, medication updates). This subsystem must preserve confidentiality and integrity of communications (often through end-to-end encryption and strong authentication) to prevent privacy breaches. Additionally, any data from follow-up interactions might flow back into the EHR (including` notes or updates to the record).

3.2.2 Constructing the Domain Data Flow Diagram

The first step in the methodology was to construct a rigorous domain DFD capturing the architecture of the healthcare system and data movements. This domain DFD defines the scope of analysis, laying out all relevant entities (external actors and system components), processes, data stores, and data flows in the healthcare information system (HIS). Thereafter, key assumptions and granularity rules were established to keep the DFD both comprehensive and analysable. For instance, the model assumes a baseline trust relationship within a single hospital domain but treats external services (cloud storage, third-party labs, etc.) as separate entities with distinct trust boundaries. Granularity was chosen to be high-level enough to encompass entire subsystems (such as an Electronic Health Record system, a medical imaging PACS, or a patient monitoring IoT platform) yet detailed enough to distinguish critical data stores (example given, patient database vs. audit log repository) and flows between them. This approach follows recommended threat modelling practices wherein the DFD should

“accurately reflect the system” and delineate trust boundaries for security/privacy analysis (Scandariato et al., 2015).

To construct the DFD, a large body of healthcare domain literature including, guidelines, policies, procedures and protocols, were consulted to elicit all relevant entities, processes, stores, and flows associated with a typical patient journey. Using scenario walkthroughs (from patient registration to diagnosis and treatment, through discharge and follow-up), how data moves through various modules were identified. For example, personal data flows from a registration web portal to an EHR database, lab results flow from diagnostic devices to the EHR, and images flow from radiology devices to a PACS archive. Each data store (databases for patient records, image repositories, etc.) and each process (such as user authentication, data analysis algorithms, message dispatch services) was represented as a DFD element. Following, the trust boundaries were explicitly drawn, distinguishing, for instance, the on-premises network of the hospital from cloud-based telehealth services or from the device domain of the patient, since crossing these boundaries often introduces privacy threat opportunities. All sensitive data classes (personally identifiable information, medical images, prescriptions, etc.) were annotated on the flows, following the principle that understanding what data is in motion is crucial for uncovering privacy threats (Abomhara et al., 2015). The resulting DFD (shown previously in Figure 3) provides a unified view of the HIS ecosystem, including deployment and operational constraints as whether each component is on-premises, cloud-hosted, or a hybrid was captured. This is important because different deployments carry different privacy implications (for example, a cloud analytics service might entail data leaving the direct control of the hospital, raising confidentiality concerns).

Referring to the patient interaction with a HIS outlined in context section 3.2.1, Figure 3-1 lists each element of the DFD. Privacy and security are given high priority when designing the DFD, so every location where data is stored, processed, or transferred is carefully evaluated. This step produces a validated domain model of the data flows in the system. The domain DFD becomes the foundation for further analysis, ensuring threat identification is systematic and reflects actual system behaviour. It also guarantees comprehensive coverage, as each DFD component and data flow becomes a focus for privacy threat detection, consistent with LINDDUN recommendations for mapping threats to DFD elements (Wuyts & Joosen, 2015).

The privacy threat categories for each DFD element are identified as per Table 3-1 and then each intersection is marked by a Y to show a potential privacy threat at a possible point of compromise or DFD element (entity = E, process = P, dataflow = DF and data store = DS).

Table 3-1: Showing the DFD elements and mapping the LINDDUN components to the possible points of compromise.

DFD Element	Possible Point of Compromise	L	I	N	D	D	U	N
Entity (E)	Patient	Y	Y	-	-	-	Y	Y
	Medical staff	Y	Y	-	-	-	Y	Y
Process (P)	EHR System	Y	Y	Y	Y	Y	-	Y
	Diagnostic Imaging System	Y	Y	Y	Y	Y	-	Y
	Medication Management System	Y	Y	Y	Y	Y	-	Y
	Vital Sign Monitoring System	Y	Y	Y	Y	Y	-	Y
	Referral System	Y	Y	Y	Y	Y	-	Y
	Remote Monitoring System	Y	Y	Y	Y	Y	-	Y
	Secure Messaging	Y	Y	Y	Y	Y	-	Y
Data store (DS)	Healthcare Database	Y	Y	Y	Y	Y	-	Y
Data flow (DF)	User data stream (mobile app, web portal)	Y	Y	Y	Y	Y	-	Y
	Service data stream (Portal Service)	Y	Y	Y	Y	Y	-	Y
	Database data stream (Service DB)	Y	Y	Y	Y	Y	-	Y

3.2.3 Systematic Threat Elicitation per LINDDUN Class

Using the domain DFD derived from the scenario in Section 3.2.1 as a foundation, a systematic privacy threat elicitation was performed across all seven LINDDUN categories. LINDDUN provides structured threat trees and catalogues of example threats for each category (Wuyts & Joosen, 2015), which was adapted to the healthcare context. For each DFD element (each process, data store, and data flow), the question was asked, “what could go wrong” with respect to each LINDDUN privacy property, considering both typical misuse cases and healthcare-specific scenarios. This exhaustive per-class analysis helps ensure coverage as a recent empirical study showed that analysts using a structured method like LINDDUN are less likely to miss threat types that an unstructured approach might overlook (Sion et al., 2024). Below, the elicitation for each LINDDUN class are detailed, highlighting the key privacy threat scenarios identified in our healthcare domain:

Linkability: This is a privacy threat when seemingly unrelated information can be connected to the identity of an individual. This means that it takes

Equation (3-1)

$$y = (x_1, x_2)$$
$$\text{where } x_1, x_2 \in \{E, DF, DS, P\}, x_1 \neq x_2$$

for the threat to affect the DFD element.

The Linkability threat y is defined as an ordered pair (x_1, x_2), where each element independently ranges over the set of DFD components {E, DF, DS, P}, representing entities, data flows, data stores, and processes respectively. The condition $x_1 \neq x_2$ requires that the two elements be distinct, reflecting the fact that linkability arises from the correlation of data across separate components rather than within one. The threat is realised when an adversary can observe or infer a connection between x_1 and x_2 sufficient to identify or profile an individual. For example, correlating a data flow from a patient monitoring device (x_1) with a data store in the EHR system (x_2) through a shared quasi-identifier. The threat therefore exists at the intersection of the pair, not at either element in isolation.

It was investigated to find ways an unauthorized party could link two or more separate data records or transactions to the same person. In a healthcare system, cross-system correlation attacks are a prime concern (example given, linking the hospital record of patient with their pharmacy prescription data or wearable device data). Metadata and telemetry surfaces that might enable such linking were considered. For instance, if different subsystems use the same patient identifier or if timestamps and device IDs in medical IoT telemetry could correlate with hospital events, an adversary could aggregate those to infer health status of a user. Prior research has demonstrated how ostensibly anonymous data can be linked via shared quasi-identifiers, compromising privacy (Rocher, Hendrickx, & de Montjoye, 2019). In the threat elicitation phase, scenarios were identified such as an attacker linking lab results to a specific patient by correlating the time and rare test type with a public social media disclosure or linking a de-identified medical record with an identified billing record through consistent internal IDs. Each of such scenario was documented as a Linkability threat instance. Special attention was paid to digital exhaust such as log entries, error codes, or device metadata that could act as a “common thread” across systems. For example, if a medical device sends a telemetry tagged with a pseudonymous patient ID, and that same ID appears in pharmacy or billing systems, it creates a linkability risk. These analyses reflect known

patterns where metadata allows cross-context linking of identities (Wieringa et al., 2020; Garfinkel, 2015 & Miller, 2012). Each identified linkability threat was recorded alongside the DFD element(s) involved. For example, a data flow from an on-premises database to a cloud analytics service might introduce linkability if records are not properly anonymized before transfer.

Identifiability: This is a privacy threat affecting all the DFD elements. It is when an individual can be directly recognized by their data. It refers to

Equation (3-2)

$$y = (x, y_a),$$
$$\text{where } x \in \{E\}, y_a \in \{DF, DS, P\}$$

The Identifiability threat y is defined as an ordered pair (x, y_a), where x represents the identity-bearing entity, specifically the individual whose identity is at risk of exposure, typically a patient or clinical staff member, drawn from the entity set {E}, and y_a represents the attribute-bearing DFD element that carries, holds, or processes the particular data attribute through which the threat is realised, drawn from {DF, DS, P}. The threat is realised when the attribute y_a is sufficient, either alone or in combination with external knowledge, to identify the individual x. The separation of the entity and its attribute across two distinct DFD element types reflects the structural nature of identifiability risk, the threat does not reside in the entity or the attribute independently, but in the traceable relationship between them.

Here, the focus was on threats that an individual could be re-identified from supposedly anonymized or aggregate data. Healthcare data often gets de-identified for secondary use, but there are many re-identification vectors an attacker might exploit. Known quasi-identifiers were systematically considered, such as birth date, zip code, gender and combinations thereof that could uniquely identify patients if data sets are not sufficiently generalized or masked. Each data store in the DFD that holds de-identified or pseudonymized data were analysed to ask, “could an attacker with external knowledge re-link this to a person”? For instance, an adversary with some background information might re-identify a patient in a released dataset by matching attributes (a classic example being identifying the

record of a patient from their birthdate and postal code (Rocher et al., 2019)). Also, anonymization pitfalls were examined in system processes. Considering if a subsystem exports data for research, “does it properly remove or perturb all identifiers”? One concrete scenario identified was the risk of re-identifying patients from “anonymized” clinical notes if rare diagnoses or combinations of symptoms act as fingerprints. Another was the possibility of re-identifying individuals from aggregate statistics (example given, if a system provides an API that returns the number of patients with a rare condition in a small region). We leveraged guidance from privacy literature (Garfinkel, 2015) to evaluate the robustness of de-identification techniques in use. Each identifiability threat instance captures the conditions under which re-identification could occur, highlighting any CWE-relevant causes like use of weak pseudonyms or improper data randomization.

Non-repudiation & Detectability: While non-repudiation is the ability to prove someone performed an action or made a statement. It affects the data flow, data store and process DFD elements. Although non-repudiation is desirable for e-commerce transactions, in the context of patient privacy, individuals should be able to choose not to be identified after using the HIS. Although the entity (E) is the only element able to non-repudiate, the threat of being able to non-repudiate occurs at DF, DS and P. As with identifiability, each DFD elements occurs in

Equation 3-3

$$y = (x, y_a),$$
$$\text{where } x \in \{E\},\ y_a \in \{DF, DS, P\}$$

The Non-repudiation threat y is defined as an ordered pair (x, y_a), where x represents the non-repudiating subject, the entity whose action or transaction is incontrovertibly recorded, drawn from {E}, and y_a represents the DFD element at which the evidence of that action is captured or retained, drawn from {DF, DS, P}. Although only an entity can perform a non-repudiable action, the threat is realised through the data flows, stores, and processes that record and preserve that evidence irrevocably. The distinction between x and y_a clarifies that non-repudiation as a privacy threat is not about the act itself but about the capacity of the system to bind an identity to an action through persistent records.

However, detectability refers to the ability to track or monitor an individual through their data usage. This occurs at data flow, data store and process DFD elements. Here,

Equation 3-4

$$y = f(x), \text{ where } x \in \{DF, DS, P\}.$$

The Detectability threat *y* is defined as a function of a single DFD element *x*, drawn from the set {DF, DS, P}. Unlike Linkability, which requires a pair of elements, Detectability can be realised through a single data flow, data store, or process that inadvertently reveals the existence of an item of data or an event to an observer. The threat is realised when an adversary can infer the presence of a patient record, a clinical event, or a system state from observable characteristics of *x*, such as response time variation, differential error messages, or traffic volume, without requiring access to the content of the data itself.

In a privacy context, non-repudiation threats mean that actions by users (patients or staff) are incontrovertibly logged or exposed, removing the ability of the user to deny or plausibly hide an action. Detectability refers to the tendency of the system to inadvertently reveal the occurrence of an event or the existence of data belonging to a user. These components were handled together because they often stemmed from similar mechanisms (audit trails, system notifications, etc.) that both record and reveal sensitive actions. The audit trails and logging of the system were examined since healthcare systems must log access to patient records for security, but overly detailed logs including recording every user query with patient IDs in plaintext, could become a privacy liability if misused. One threat example is that a patient cannot hide the fact that they accessed certain information or used a service, because the system irrevocably associates their identity with that action (a non-repudiation issue), which might discourage them from seeking sensitive care. Further, the error channels and system messages were also considered for detectability issues. For instance, if an outsider can trigger an action (like entering a fake patient ID) and observe different error messages depending on whether that ID exists, they could detect the presence of that patient in the system (a privacy breach). Such side-channel detectability has been noted as a risk in web applications where timing or messages differ for existent vs. non-existent accounts (Troncoso et al., 2021). In context, an attacker was identified that could detect if a particular

person is a patient by attempting a password reset with their email leading to a subtle privacy threat if the system responds “email not found” for non-patients but sends a reset link for patients. Another detectability threat was observing network traffic volume or timing from a particular monitoring device in a hospital room. Here, a spike might indicate a critical event for the patient in that room. All these threats were documented and noted whether the system provided any observability risks such as, excessive transparency to admins that might violate patient privacy, or interfaces that inadvertently allow detectability of patient info. Then it was flagged where design trade-offs between accountability and privacy occur, recognizing that healthcare regulations require certain logs (HIPAA mandates auditing access to records), but from a threat-model perspective those same logs must be safeguarded to prevent misuse.

Disclosure of Information: Disclosure of Information can be described as the unintended exposure of personal information at the DFD elements to unauthorised individuals. Here,

Equation 3-5

$$y = f(x), \text{ where } x \in \{DF, DS, P\}.$$

The Disclosure of Information threat *y* is defined as a function of a single DFD element *x*, drawn from {DF, DS, P}. The threat is realised when *x* exposes personal data to an unauthorised actor, whether through a vulnerable data flow transmitting unencrypted content, a data store with insufficient access controls, or a process that leaks sensitive information through its outputs, logs, or error messages. Entities are excluded from *x* because disclosure is a property of the system components that handle data rather than of the subjects whose data is processed. Each realisation of the threat is therefore traceable to a specific component in the data flow diagram where the exposure occurs.

This category covers classic confidentiality breaches, the unauthorized release or exposure of personal data. Here, the systematic analysis asked, for each data store and flow, how information could leak or be improperly accessed. Consideration was made for inadvertent leaks via buffers, logs, and serialization processes in the software. For example, memory handling in systems were examined using native code (C/C++ components common in medical device firmware or high-performance imaging systems). Past studies

show that memory safety vulnerabilities like buffer overflows or use-after-free can cause sensitive data to be read by attackers. Indeed, scenarios were identified where a buffer overflow in a medical device might expose patient vitals or where a debug log file inadvertently recorded full patient records. Furthermore, analysis was made on, 'how data is serialized and transmitted between subsystems'. "Is sensitive data sent in cleartext (leading to CWE-319 Cleartext Transmission)"? "Do APIs accidentally expose more information than intended (similar to CWE-201 Exposure of Sensitive Data through Sent Data)"? Each potential weakness that could lead to Disclosure of Information was noted. As an example, one threat instance was an EHR database backup process that writes unencrypted patient data to a cloud storage, if the cloud bucket is misconfigured, being a direct information disclosure risk. Another example, is when an imaging system might keep cached images in a temporary folder without proper access controls, allowing other processes to read them. These are representative of the kinds of weaknesses known to plague healthcare systems and lead to breaches (Guo, Bettaieb, & Casino, 2024). Going forward, error-handling and exception messages were also considered for overly informative outputs such as a crash dump that includes patient data. Throughout, CWE examples for information leaks were referenced, such as CWE-200 (Exposure of Sensitive Information to an Unauthorized Actor) and CWE-209 (Information Exposure Through an Error Message), to ensure our elicitation aligned with known vulnerability modes. This produced a set of Disclosure threats ranging from technical low-level leaks (memory dumps, core dumps containing PHI) to higher-level process issues (employees emailing spreadsheets with unredacted patient information which, while procedural, stem from system design allowing such export. Each was tied back to specific DFD flows or components, ensuring traceability.

Unawareness: It refers to when the individual is unaware of the data collected about them and how it is being used. Here,

Equation 3-6

$$y = f(x), \text{ where } x \in \{E\}$$

The Unawareness threat *y* is defined as a function of an entity *x*, drawn exclusively from {E}. The threat is realised when the data subject such as a patient or clinical staff member,

lacks adequate knowledge of what data is being collected about them, how it is being used, or what privacy-impacting actions are occurring within the system. Unlike the preceding categories, which can be realised through data flows, stores, or processes independently of the awareness of the subject, Unawareness is inherently a subject-centred threat. Its realisation depends on the relationship between the entity and the transparency mechanisms of the system, or their absence, rather than on any specific technical component in the data flow diagram.

Unawareness threats occur when data subjects (patients or practitioners) are not adequately informed about how their data is used or what privacy-impacting actions are occurring. In a healthcare software setting, how consent and notice are implemented across the system was probed. Whether patients receive proper disclosures like a mobile health app informing them that data will be shared with a research database and whether there are any "invisible" data processing steps were considered. Here, key scenarios involved consent/notice propagation failures, for instance, if a patient consents to share data for treatment, but that data is also used for secondary research without an explicit additional consent, that constitutes an unawareness threat. Each data flow was inspected to see if it aligns with a consent given; any mismatch was flagged. Yet, another aspect is default settings and UI design, privacy-unfriendly defaults and UX debt can leave users unaware of data sharing. It was identified that some subsystems enabled broad data sharing with third parties by default (for example, a telehealth platform automatically sending usage stats to an analytics service unless the user opts out, which many may not realize how to do). Also considered was the complexity of privacy policies and whether the system design inherently prevents users from understanding them this is a form of unawareness threat, as per transparency requirements of the GDPR. One concrete threat instance was the finding that audit logs of patient access (meant for compliance) were not made visible to patients themselves, hence, the patients were unaware of who had viewed their records, contrary to the principle of transparency. While this is a compliance issue, it is also framed as a privacy threat. The lack of user awareness can undermine trust and lead to unchecked data misuse (Grundy et al., 2019). For each unawareness issue, the lacking design element or requirement was recorded.

Non-compliance: Finally, this occurs when there is a breach in regulatory or ethical obligations regarding data privacy. This affects the entire HIS. Here,

Equation 3-7

$$y = f(x), \text{ where } x \in \{E, DF, DS, P\}.$$

The Non-compliance threat *y* is defined as a function of any DFD element *x*, drawn from the full set {E, DF, DS, P}. This is the only LINDDUN category whose domain spans all four element types, reflecting the fact that regulatory and policy obligations apply to every component of the healthcare information system, entities who process data, data flows that transmit it, data stores that retain it, and processes that transform it. The threat is realised when *x* exhibits behaviour that violates a legal or policy requirement, whether through inadequate data retention controls, overly broad access privileges, absent encryption on a transmission, or an entity performing an action outside their authorised scope. The breadth of the domain underscores that non-compliance is a systemic threat requiring assessment across the entire data flow diagram rather than at specific architectural hotspots.

Where the system might violate legal or policy requirements, resulting in privacy threats due to policy–implementation drift was considered. This step involved mapping the regulatory and policy obligations (such as HIPAA rules, GDPR principles, hospital internal privacy policies) to what the system does. A threat was logged whenever a point was found where the behaviour of the software might lead to non-compliance, thus endangering privacy rights. Examples of regulatory control points checked include data retention and deletion ("does the system properly delete patient data after the retention period?" or "could old records linger in backups indefinitely, violating policies?"), access controls ("do users get the minimum necessary access?" or "is there an over-broad access somewhere that violates need-to-know?"), and breach notification readiness (if the system does not log certain accesses, it might fail requirements to detect/report breaches). Alongside, emerging regulations like the requirements of the GDPR for data subject rights were investigated such as right to be forgotten, right to data portability, etc because if the architecture cannot support these (say, data is siloed such that it is impractical to delete all the data of someone on request), that is a compliance gap and hence a privacy threat. Even so, the analysis turned

up issues like an archiving system that kept diagnostic images without a mechanism to delete on patient request, representing a potential GDPR violation. Another non-compliance threat was insufficient encryption on certain data flows, flouting a hospital policy that all patient data in transit must be encrypted. For each such issue, the threat entry notes the specific requirement breached and the component involved. This approach aligns with the idea that privacy threat modelling should incorporate legal requirements as first-class concerns (Iwaya et al., 2023), ensuring that the design of the system is evaluated against not just technical threats but also compliance obligations.

The definitions of these properties are covered by the ISO/IEC 27001:2022 (Nair & Greeshma, 2023). By exhaustively enumerating threats in all LINDDUN classes, a comprehensive threat list was generated for the HIS domain. This list is the basis for the next step (Section 3.2.4), where each threat is mapped to root causes in terms of software weaknesses. It is worth noting that the threat elicitation process was iterative which involved initial brainstorming using LINDDUN threat trees, then followed by validation against real incident examples and known healthcare breaches from the literature, to ensure realism. The result was a threat catalogue covering not only obvious issues (like data breaches) but also subtle privacy pitfalls (like inference via side channels, or institutional policy gaps) specific to healthcare scenarios. This thorough approach provides confidence that the subsequent mapping to CWEs addresses a wide range of privacy vulnerabilities that could manifest in the system.

For each applicable category, the LINDDUN threat trees (Appendix 8.1) were consulted to identify specific threat scenarios. These threat trees systematically decompose broad categories into concrete attack patterns. For example, the “Linkability” tree contains branches such as the use of unique identifiers to correlate records or the combination of location data to recognize individual users. Likewise, the “Identifiability” tree includes branches involving the linking of pseudonymized data through additional attributes. The trees are navigated with reference to the context provided by the DFD. For instance, if a data flow contains anonymized laboratory results, the Identifiability tree would be followed, focusing on branches where an attacker could potentially re-identify a patient by cross-referencing that flow with external datasets. These threat trees offer structured terminology for

articulating attack vectors, and their branches are customised to align with the specifics of the healthcare system under consideration. Through this process, high-level LINDDUN threats are translated into concrete, data-flow-specific risks.

Table 3-2: Formal Notation and DFD Element Definitions for LINDDUN Threat Category Representations

Threat Categories	**(x,y)**	**DFD ELEMENTS**			
		E	**DF**	**DS**	**P**
Linkability	∈ {E, DF, DS, P}	x	x	x	x
Identifiability	∈ {E, DF, DS, P}	x	x	x	x
Non-repudiation	∈ {DF, DS, P}	-	x	x	x
Detectability	∈ {DF, DS, P}	-	x	x	x
Information disclosure	∈ {DF, DS, P}	-	x	x	x
Content unawareness	∈ {E}	x	-	-	-
Policy/consent noncompliance	∈ {E, DF, DS, P}	x	x	x	x
DFD=Data Flow Diagram, E=Entity, DF=Data Flow, DS=Data Store, P=Process					

During this phase, relevant threat categories were systematically documented for each source-flow-destination combination. Where the mapping Table 3-2 indicated applicability, the corresponding threat tree was employed to develop potential threat descriptions. Additionally, any traversals of trust boundaries were identified, as these could have introduced novel threats (for example, an external entity forging data to connect identities). At this stage, mitigation measures were not yet applied, and all discovered threats were catalogued for subsequent analysis. This comprehensive mapping process, informed by the LINDDUN threat tree knowledge, yielded a list of candidate privacy threats anchored in the system's data flow diagram (DFD).

Subsequently, each identified threat scenario was formalized as a misuse case (also referred to as an “abuse case”). A misuse case constituted a use-case that delineated the actions of an adversary resulting in a privacy violation. For instance, if a threat involved linking patient lab results via reused patient IDs, the related misuse case would have portrayed an attacker intercepting lab data and associating it with a known patient ID. The documentation of threats as misuse cases necessitated clear articulation of the objectives, preconditions, and detailed actions of the attacker.

A concise misuse case narrative was prepared for each previously identified threat, specifying the attacker (like external intruder, malicious insider), the target (such as patient

records), the attack methods (which included data interception or metadata analysis), and the resulting privacy breach (such as identity disclosure). Misuse cases enhanced clarity and traceability by systematically linking each threat scenario to specific system behaviours. Furthermore, they provided a foundation for the subsequent step, wherein mapping to CWE incorporated the adversarial perspective encapsulated within these scenarios.

3.2.4 Semantics of the LINDDUN-to-CWE Mapping

Before presenting the systematic mapping process, it is necessary to state precisely what a LINDDUN-to-CWE mapping entry asserts, since the relationship between a design-level privacy threat and a code-level weakness is not uniform in strength or kind. This thesis distinguishes three relationship types:

- **Enables**: the CWE weakness, if present and triggered, is sufficient on its own to realise the privacy threat, with no further intervening condition required. For example, CWE-200 (Information Exposure) *enables* a Disclosure threat directly, if the weakness exists in an exploitable form, the disclosure follows.
- **Contributes to**: the CWE weakness is one of several necessary conditions for the threat to be realised, typically in combination with an additional weakness, external dataset, or contextual factor. For example, CWE-287 (Improper Authentication) *contributes to* an Identifiability threat, since re-identifying an ostensibly anonymised individual typically also requires an auxiliary correlation step beyond the authentication weakness itself.
- **Provides evidence of**: the CWE weakness is a signal that the threat *may* be present, without itself causing or guaranteeing the threat. This relationship applies where the weakness reflects an architectural or procedural pattern associated with the threat class, but confirming the threat requires investigation beyond the code, most often for Non-compliance and Unawareness threats (discussed below).

These relationship types are not distributed evenly across LINDDUN categories. Threats with a concrete technical mechanism, Linkability, Disclosure of Information, and Detectability, are more often connected to their CWEs through *enables* or *contributes to* relationships, consistent with the higher inter-rater agreement these categories achieved during expert

validation (Section 3.2.6). Non-compliance and Unawareness threats, by contrast, more often connect to their CWEs through provides evidence *of* relationships, reflecting the lower agreement levels and greater reliance on policy-bordering judgement reported for these categories in Section 3.2.6. This thesis defines these relationship types as an analytical framework applied at the category level throughout the discussion of the mapping; systematically tagging each of the 965 individual matrix entries with its specific relationship type is identified as a refinement for future work (Section 6.9).

A direct consequence of the *provides evidence of* relationship is that code remediation alone does not resolve every threat the mapping identifies. Many Non-compliance and Unawareness threats are, by their nature, only partially addressable at the code level. Consider a Non-compliance threat flagged via CWE-778 (Insufficient Logging): adding the missing log entry is a code-level fix, but whether the resulting the retention period of the log satisfies GDPR storage limitation requirements, or whether the logged data itself should have been collected at all, are policy and legal determinations that no code change resolves. Similarly, an Unawareness threat, a patient not being adequately informed that their data is being shared with a third party, may correspond to a missing consent-capture field in the software (a code-level gap the mapping can flag), but resolving the underlying threat requires an organisational decision about consent workflows, and in some cases a legal determination of what constitutes adequate notice under applicable regulation. The LINDDUN-to-CWE mapping therefore serves a diagnostic rather than a fully curative function for these categories: it identifies the code-level symptom and the point in the system where a policy, organisational, or legal safeguard is needed, but the corresponding remediation for Non-compliance and Unawareness threats routinely extends beyond what the detection prototype described in Chapters 4 and 5 can resolve. This distinction is revisited in the discussion of the prototype's scope in Chapter 5 and the limitations discussed in Section 6.8.

3.2.5 Systematic Mapping of LINDDUN Privacy Threats to Common Weakness Enumeration (CWE) Entries

With a comprehensive set of privacy threats elicited from the domain DFD (Section 3.2.3), the next step was to systematically map each identified threat instance to one or more relevant software weakness types from the Common Weakness Enumeration (CWE)

catalogue (seen in Appendix B: Threat to CWE Mapping). This mapping ties the high-level privacy threats defined by LINDDUN with concrete code-level vulnerabilities, providing a structured link that enhances interpretability. To accomplish this, a mapping rubric was established to guide the selection of candidate CWE entries for each threat scenario. Each LINDDUN threat which is a potential privacy breach point in the data flow diagram, was analysed to determine what specific weakness patterns in implementation could lead to that threat. This approach is in line with emerging evidence-based threat modelling techniques that leverage known vulnerability databases to inform threat identification (Özkan & Singelée, 2024). By correlating threats with CWE categories, the methodology ensures that abstract privacy concerns are grounded in well-defined software fault patterns, which is crucial since generic, expert-only threat lists risk being incomplete (Sangaroonsilp et al., 2023). The resulting LINDDUN/CWE correlation offers an explicit "threat-to-weakness" trace that stakeholders can readily interpret because it not only highlights what vulnerabilities exist but also explains their privacy implications. Evidence-driven mapping criteria. For each proposed threat–CWE pairing, the methodology required clear justification supported by evidence. In practice, this meant documenting a representative vulnerability pattern (a code scenario or design flaw) that exemplifies how the threat could manifest, the conditions or context that make the pattern exploitable, the likely privacy consequences if the weakness is exploited, and a plausible fix or mitigation strategy. Requiring these elements for every mapping ensured that the association was not speculative but grounded in a realistic example and aligned with known CWE definitions (which often include descriptions of causes, consequences, and mitigations). For instance, if the threat of Identifiability (re-identifying an anonymized person) was considered for a database module, a candidate CWE might be CWE-287: Improper Authentication, supported by a pattern such as weak anonymization logic combined with predictable record IDs; the condition enabling the weakness could be the lack of salted identifiers, the consequence would be patient re-identification, and the mitigation could involve stronger pseudonymization or authentication checks. By capturing this evidence, the mapping process became reproducible and reviewable, each link from threat to CWE was backed by a rationale that could be scrutinized by others. This approach mirrors best practices in secure design, where each identified risk is tied to concrete examples and

solutions (Ruohonen, 2025). It also reflects the CWE usage context, which encourages providing demonstrative scenarios and mitigations for each weakness (MITRE, 2023). Ensuring such rich context for each mapping helped maintain precision and avoid over-generalizing the impact of a threat. Iterative disambiguation and many-to-many relationships. Mapping privacy threats to weaknesses is not a one-to-one exercise; a single threat can originate from multiple distinct weaknesses, and conversely, one CWE type can be relevant to different privacy threats depending on context. This many-to-many relationship required an iterative and careful disambiguation process. The approach was first linking threats to a CWE group and then refining these links through manual verification against domain-specific security standards and expert knowledge. For example, a Linkability threat (correlation of identities across systems) was initially mapped to various weaknesses related to information exposure. Through refinement it became clear that CWE-200: Information Exposure was a primary match for Linkability in our context, capturing issues like unsanitized patient IDs in logs, while CWE-201: Information Exposure Through Sent Data was also considered for cases where metadata leakage could enable linking records. In another case, Identifiability threats (re-identifying individuals from ostensibly anonymous data) corresponded closely with CWE-203: Information Exposure Through Discrepancy, since subtle differences in system responses could reveal whether a record exists.

These examples illustrate that multiple CWE entries within the same family might relate to one threat category. The rubric therefore allowed multiple CWE mappings per threat when justified, but with each relationship explicitly documented. Conversely, some CWE classes (like CWE-120: Buffer Overflow) could implicate several LINDDUN threat categories, a buffer overflow might lead to Disclosure of Information if it exposes memory, or Non-compliance if it violates data handling policies. In such cases, the specific consequence dictated the mapping: if a buffer overflow led to unauthorized disclosure of patient data, it was mapped under the privacy Disclosure threat category; if it primarily caused a system crash affecting accountability, it might map under Detectability or Availability concerns rather than a privacy threat. Handling these ambiguities required expert judgment to ensure each mapping was contextually appropriate. The iterative review involved checking that each threat/CWE link is bidirectionally meaningful. Thus, starting from the threat, the CWE should represent a

credible cause; starting from the CWE, one should be able to see how it could give rise to the privacy threat. Where the linkage was not obvious in both directions, the mapping was either revised or dropped to avoid spurious associations. This rigorous process aligns with observations in recent research that adapting a privacy threat framework to code-level issues is non-trivial. Notably, Wang et al. (2023) observed that privacy threats often manifest across multiple components and are not isolated to single lines of code. This insight reinforced the importance of our many-to-many mapping approach: it ensured that the mapping did not overly constrain a threat to one weakness or vice versa, which would risk overlooking complex, compound vulnerability scenarios. Instead, the final mapping acknowledges the nuanced ways a privacy threat can emerge from a combination of weaknesses, while still giving clear traceability to specific CWE-defined issues. The outcome of this process was a validated set of threat–CWE category pairs forming the basis of a privacy-focused mapping matrix, which serves as a cornerstone for traceability and further analysis, as described next.

3.2.6 Traceability and Reproducibility Artifacts

All threat mapping decisions were captured in structured artifacts to ensure traceability (from high-level threat down to code example) and reproducibility of the analysis. The core artifact is a mapping matrix found in Appendix C, that tabulates each identified privacy threat alongside its associated CWE weaknesses and supporting details. This matrix is organized by DFD element × LINDDUN threat category × CWE, meaning each row corresponds to a specific combination of a system component (or data flow) and a privacy threat, and lists the CWE(s) that were mapped to that threat in that context. By structuring the results in this multi-dimensional way, the methodology maintains a clear lineage which demonstrates that one can start at a particular subsystem or data store in the healthcare system, see which privacy threat(s) were considered for it, and then see which CWE vulnerabilities are linked to those threats. This accomplishes the goal of making the privacy analysis actionable, stakeholders can trace a path from an architectural element (e.g., “Patient Database”) through a threat (“Linkability of patient records across systems”) to concrete weaknesses (e.g., CWE-200 Information Exposure via log files) that developers or auditors should examine. Such traceability is crucial for accountability and aligns with recommendations for privacy engineering in regulated domains, where one must demonstrate how identified risks tie to

mitigation measures (Nyst et al., 2019). Each entry in the matrix is accompanied by several reproducibility artifacts that document the reasoning and evidence behind the mapping. Notably, a unique identifier was assigned to every threat, that is CWE mapping entry, which can be used to reference it in discussions or in future work.

Each entry in the traceability matrix is accompanied by a short rationale of one or two sentences justifying why the particular CWE is relevant for the given threat in that context. For example, a matrix entry might read:

> *DFD Store > Imaging Archive; Threat > Disclosure of Information; CWE-502 (Deserialization of Untrusted Data), with a rationale explaining that unprotected image files could be deserialized by untrusted viewers, risking exposure of embedded patient data.*

Each mapping entry also includes references to mitigations or controls that address the weakness, drawing on established security guidelines such as those published by OWASP and NIST. By linking each CWE to known mitigations, the matrix not only flags a potential problem but also points toward solutions, reinforcing the actionability of the threat model outputs and aligning with privacy-by-design principles that require not just identifying problems but also considering remedies (Chereja et al., 2025).

The traceability matrix was designed to be comprehensive yet navigable. It was implemented as a structured spreadsheet with filtering and sorting capabilities, enabling analysts to filter by LINDDUN category to see all CWE mappings for a given threat type, or to sort by CWE to see all privacy threats linked to a specific weakness. This proved helpful in identifying systemic patterns, for instance, if multiple threats across different subsystems all mapped to CWE-89 (SQL Injection), it highlighted a broad vulnerability scenario impacting privacy across the system, such as a widespread risk of Disclosure of Information through injection flaws. Such patterns, once traceable, can inform priority areas for security improvement. The matrix format also supports versioning and change control, allowing entries to be updated with new version timestamps as the system or threat landscape evolves.

3.2.7 Expert Review and Validation Protocol

To ensure the quality and credibility of the LINDDUN-to-CWE mappings, a formal expert review and validation protocol was carried out. This step was critical given the innovative nature of bridging privacy threats to code-level weaknesses, making it necessary to assess, within a small purposively selected expert panel, whether the mapping was technically plausible and practically meaningful. The reviewer panel consisted of three domain experts selected for their complementary backgrounds: a senior privacy engineer with experience in healthcare software, an application security specialist familiar with CWE and static analysis, and an academic researcher in threat modelling methodology who had not been involved in the initial mapping work. All reviewers were briefed on the context and aims of the enhanced LINDDUN methodology but conducted their evaluations independently to avoid groupthink, recording their judgments on each threat–CWE mapping in isolation during the first round.

Each expert was provided with the complete set of traceability artifacts from Section 3.2.6 (the mapping matrix, rationales, and misuse case scenarios) together with structured evaluation forms (Appendix D). For each entry, reviewers were asked to rate whether a given threat–CWE link was "Valid", "Questionable", or "Invalid", and to flag any mappings they felt were missing. A mapping was automatically accepted if at least two of the three experts rated it "Valid" and none rated it "Invalid". Entries failing this threshold were set aside for adjudication. This acceptance criterion was chosen to balance rigour with practicality, ensuring that no strongly contested mapping would be accepted unexamined, while not rejecting mappings over minor uncertainty. Inter-rater agreement was calculated using Cohen's κ on a per-pair basis and Fleiss' κ for overall agreement across all three reviewers, yielding a substantial overall agreement of $\kappa \approx 0.78$, consistent with expert evaluation studies in comparable security research contexts (Özkan & Singelée, 2024).

The three-reviewer panel was constructed through purposive rather than random or exhaustive sampling, selected specifically to provide complementary disciplinary coverage rather than to be statistically representative of the population of qualified reviewers: a senior privacy engineer contributed healthcare-domain and data-protection expertise, an application security specialist contributed CWE and static-analysis expertise, and an academic threat-modelling researcher, deliberately uninvolved in the original mapping work,

contributed independent methodological scrutiny. This composition was chosen to expose the mappings to three distinct professional perspectives likely to disagree for different reasons, rather than to achieve a large or statistically powered sample. Whether three reviewers were sufficient to reach a stable, saturated judgement, meaning that a fourth or fifth reviewer would be unlikely to substantially alter the pattern of agreement, cannot be established from this panel alone. The tight clustering of the three pairwise Cohen's κ values (0.781, 0.781, 0.780, Section 3.2.6) is suggestive of stability, since high agreement was not driven by any single reviewer pair, but three reviewers is too small a panel to test saturation formally, and the result should not be read as evidence that additional reviewers would not identify further contested mappings, particularly within the Identifiability and Non-compliance categories, which already exhibited comparatively lower agreement. Accordingly, the results reported in this section should be understood as substantial agreement achieved within a small, purposively selected expert panel, providing informed preliminary evidence of the mapping's plausibility, rather than as an establishment of the mapping's validity across the discipline of healthcare software development. Broader validation, involving a larger and more diverse reviewer pool sampled to the point of demonstrated saturation, is identified as necessary future work (Section 6.9).

Following the independent review, a structured moderation workflow was executed to resolve disagreements. The three experts convened with a moderator to discuss each contested entry, referencing the traceability artifacts throughout. In several cases, disagreements arose not from a fundamental objection to a mapping but from ambiguity in the rationale text; these were resolved by updating the documentation rather than discarding the mapping. For example, a contested mapping of a Detectability threat to CWE-209 (Information Exposure Through Error Messages) was initially questioned on the grounds that this CWE addresses security rather than privacy. Adjudication clarified that verbose error messages in a login system could allow an observer to detect user existence, a privacy breach, and consensus was reached in favour of retaining the mapping. A second contested mapping involved a Non-compliance threat mapped to CWE-327 (Use of a Broken or Risky Cryptographic Algorithm). Reviewers initially questioned this mapping on the grounds that CWE-327 is a broad category encompassing many forms of cryptographic weakness and could plausibly apply across numerous unrelated components, raising concern that the mapping was too generic to be diagnostically useful for a specific Non-compliance threat instance. Adjudication resolved this by tying the mapping to a specific policy citation of a hospital or regulatory requirement that patient data be encrypted to a defined standard, narrowing the

mapping from a general cryptographic concern to a concrete, auditable compliance obligation. The mapping was retained following this clarification.

A third contested mapping involved an Identifiability threat mapped to CWE-359 (Exposure of Private Personal Information). Reviewers initially raised a similar breadth objection, questioning whether CWE-359 was too general to distinguish an Identifiability threat, re-identification of an ostensibly anonymised individual, from a more general Disclosure of Information threat. Adjudication resolved this by clarifying that the specific mapping concerned exposure of an attribute or combination of attributes that enabled re-identification, rather than direct unauthorised viewing of already-identified data, distinguishing the Identifiability mechanism from Disclosure. The mapping was retained following this clarification.

The acceptance criterion for the adjudication phase required either unanimous consensus or strong endorsement from at least two experts with no remaining objection. By the end of this process, 189 of the original 205 mappings (92%) were retained. This 92% figure and the 76% first-round figure reported above measure different things and should not be conflated. The 76% reflects independent agreement: three reviewers rating each mapping in isolation, with no discussion between them, before seeing one another's judgements. The 92% reflects consensus following moderated group discussion of the 40 contested mappings, in which reviewers were exposed to each other's reasoning and, in several cases, revised their position after hearing a colleague's rationale. Group discussion resolves disagreement, but resolution is not independent confirmation, a mapping retained after adjudication was found acceptable through discussion and social consensus, not through a second, independent expert arriving at the same judgement in isolation. The 92% retention rate should therefore be read as evidence that disagreements were resolvable through discussion, not as a stronger or more reliable validity signal than the 76% independent-agreement figure. In total, 16 of the 205 sampled mappings (8%) were rejected outright and removed from the matrix: 10 in the first round, where a majority of reviewers rated the mapping Invalid, and 6 during adjudication, where discussion failed to resolve the reviewers' objections.

As a final validation step, the methodology was piloted on two representative subsystems before being scaled to the full system and the machine learning dataset. The Electronic Health

Record module was selected for its rich data interactions and numerous user-facing processes, while the Medical Imaging subsystem was chosen for its performance-intensive, legacy C/C++ code, both representing boundary cases that would stress-test the methodology. The pilot confirmed that the threat elicitation checklist and mapping rubric could be applied consistently by practitioners other than the original authors, and that all identified threats could be mapped to an existing CWE entry in the matrix, affirming adequate coverage. One edge case surfaced during the pilot, an audit logging feature in the EHR that raised a Non-repudiation threat through inappropriate deletion of logs, which initially lacked a CWE mapping, as CWE focuses primarily on software weaknesses rather than procedural issues. Expert deliberation led to the addition of CWE-778 (Insufficient Logging) to the matrix, and this refinement was incorporated back into the methodology.

Through the combination of independent expert review, consensus-driven adjudication, and real-world subsystem piloting, the enhanced LINDDUN-to-CWE mapping achieved substantial agreement within the expert panel and was refined. This protocol mirrors quality assurance practices in security research where expert consensus is used to assess the plausibility of novel frameworks such as security ontologies and classification schemes (Koboyatshwene & Ayalew, 2025). The high retention rate of 92% suggests that the initial elicitation and mapping process was methodologically sound, with expert input primarily serving to increase confidence and precision rather than to correct fundamental errors. The final vetted mapping matrix and its associated artifacts thus provide a preliminary, expert-agreed foundation for the operational and machine learning applications described in the following section.

3.3 Outcomes of applying the enhanced LINDDUN-based methodology to the healthcare software system

This section presents the outcomes of applying the enhanced LINDDUN-based methodology to the healthcare software system. The results are organised into four parts: the coverage of threats across the data flow of the system (Section 3.3.1), illustrative mappings from identified threats to known software weaknesses (Section 3.3.2), the quantitative validation of those mappings through expert review (Section 3.3.3), and the

actionability of the results for integration into development processes (Section 3.3.4). Together these outcomes demonstrate how the methodology identifies privacy weaknesses and bridges the gap from high-level threats to concrete, code-level security measures.

3.3.1 Threat Coverage Across the DFD

The enhanced threat modelling exercise produced a broad set of privacy threats distributed across all major subsystems and trust boundaries of the healthcare system. Every element in the data flow diagram was associated with at least one identified privacy threat instance, confirming comprehensive coverage. Table 3-3 summarises the distribution of the sixty-three threat instances by subsystem and by the trust boundaries they traverse.

Although every subsystem contained privacy issues, certain components emerged as significant flashpoints owing to a higher concentration of threats. Data flows crossing external trust boundaries exhibited the highest threat densities. The electronic health record database, where the most sensitive personal data is stored, showed the greatest concentration of threats at twenty-two instances, with clusters of Identifiability and Unawareness threats at points where data is exported or accessed by third-party services. The interface between the patient mobile application and the cloud healthcare platform similarly contained a concentration of Linkability and Disclosure threats. These results indicate that the external communication points and central data repositories of the system are the richest in privacy risk.

The identification of these flashpoints confirms that certain architectural elements warrant particular attention. Over half of all documented threats were associated with a small number of critical data flows, chiefly those bridging distinct trust domains. This outcome supports the established observation that trust boundaries frequently coincide with elevated privacy risk (Wuyts et al., 2014). By applying each threat category systematically across the data flow diagram, the analysis ensured that no segment of the system was overlooked, and it identified the interfaces and components that should be prioritised for privacy-by-design mitigation.

Table 3-3: Distribution of Identified Privacy Threat Instances Across Healthcare Information System Components by LINDDUN Category

Subsystem / Component	Trust Boundary	LINDDUN Threat Instances							Total
		L	I	N	D	D*	U	Nc	
Data Stores									
Electronic Health Record (EHR) Database	Internal / External*	4	3	2	3	4	3	3	**22**
Diagnostic Imaging System (PACS)	Internal / External	2	2	0	2	2	1	2	**11**
Blood Bank System	Internal	0	2	0	0	0	0	1	**3**
Clinical Processes									
Surgery & Anaesthesia Systems	Internal	0	2	0	1	0	0	1	**4**
Medication Management System	Internal / External	2	0	1	0	1	0	0	**4**
Vital Sign Monitoring System (IoMT)	Internal / External	2	0	1	0	1	0	1	**5**
Counselling & Support (Telehealth)	External	0	1	0	2	0	1	0	**4**
Referral System	Internal / External	1	0	0	0	0	1	1	**3**
External Entities (Actors)									
Patient (Mobile App / Portal Interface)	External*	1	1	0	0	0	1	0	**3**
Healthcare Professional	Internal	0	0	1	1	0	0	0	**2**
Administrator	Internal	0	0	0	0	1	0	1	**2**
TOTAL		**12**	**11**	**5**	**9**	**9**	**7**	**10**	**63**

Notes: L = Linkability; I = Identifiability; N = Non-repudiation; D = Detectability; D* = Disclosure of Information; U = Unawareness; Nc = Non-compliance.

** External boundary denotes data flows crossing the organisational trust perimeter (e.g., patient-facing interfaces, third-party API integrations, cloud platform connections).*

Highlighted rows indicate components with highest threat concentrations or critical external trust boundary crossings. The EHR database subsystem exhibits the highest threat density (22 instances), with clusters of Identifiability and Unawareness threats at third-party service integration points.

3.3.2 Representative Mappings and Examples

A central outcome of the enhanced methodology is the systematic linkage of each identified privacy threat to one or more Common Weakness Enumeration (CWE) categories, supporting targeted remediation.

Table 3-4 summarises the complete mapping, which comprises nine hundred and sixty-five (965) screened CWE pairings distributed across the seven threat categories. It is important to clarify that this figure represents the total number of threat-to-weakness pairings rather than the count of distinct CWE identifiers, since the mapping is many-to-many and a single CWE may appear under more than one LINDDUN category where the context justifies it. The number of unique CWE identifiers operationalised during dataset construction is reported in Chapter 4. Three representative cases, one each for Linkability, Identifiability, and Disclosure, illustrate how abstract privacy threats translate into actionable technical terms.

Because this chapter and Chapter 4 report several distinct quantities that all relate to CWEs, this section states them explicitly to avoid ambiguity. Sixty-three (63) is the number of discrete privacy threat instances identified during DFD-based threat elicitation (Table 3-3), each instance representing one privacy concern located at a specific DFD element. Nine hundred and sixty-five (965) is the total number of candidate CWEs screened for relevance to the seven LINDDUN threat categories via the threat-tree analysis (Table 3-4), summed across categories; this is a candidate pool, not a count of unique CWEs or of validated relationships, and a given CWE may appear in the screened pool for more than one category. Two hundred and five (205) is the size of the stratified sample of threat–CWE mappings, drawn proportionally from this screened pool across all seven categories, submitted to three-reviewer expert validation (Section 3.2.6). Of these, one hundred and fifty-five (76%) were accepted in the first review round, and one hundred and eighty-nine (92%) were retained following adjudication of the flagged mappings. Seven hundred and seventy-three (773) is a separate quantity entirely: the number of unique CWE identifiers present as ground-truth labels in the C3-VULMAP dataset (Chapter 4), which is a large-scale vulnerability dataset built independently from public repositories (DiverseVul, ReposVul) and is not a subset or superset of the mapping matrix described in this chapter. Table 38 (Appendix B) additionally presents a small number of representative worked examples illustrating specific misuse cases and their associated CWEs; these are illustrative excerpts, not an independent count.

Linkability example. The analysis uncovered a Linkability threat involving the reuse of predictable patient record identifiers across different modules. An adversary able to observe or guess these identifiers could link a patient's data from the telehealth module to their records in the electronic health record database, compromising unlinkability. This scenario was mapped to CWE-200 (Exposure of Sensitive Information to an Unauthorised Actor), which captures the underlying problem of exposing identifiers that should remain confidential or randomised. The mapping directs developers towards mitigations such as cryptographically random identifiers or access tokens that prevent cross-system linkage.

Table 3-4: Privacy & Security Threat Mapping: 965 Candidate CWEs Screened Across Seven LINDDUN Threat Categories

LINDDUN Threat Category	Threat Tree	CWEs
Linkability	2 Cases 4 Misuse Cases	16 CWE Categories 180 CWEs
Identifiability	5 Cases 5 Misuse Cases	22 CWE Categories 276 CWEs
Non-repudiation	2 Cases 2 Misuse Cases	7 CWE Categories 108 CWEs
Detectability	3 Cases 3 Misuse Cases	14 CWE Categories 176 CWEs
Disclosure of Information	4 Cases 4 Misuse Cases	2 CWE Categories 68 CWEs
Unawareness	2 Cases 2 Misuse Cases	3 CWE Categories 116 CWEs
Non-compliance	2 Cases 2 Misuse Cases	4 CWE Categories 41 CWEs

Identifiability example. An Identifiability threat was found in the authentication interface. When an external system queried the interface with a patient identifier, the response behaviour differed according to whether the patient existed, for instance by returning a specific error only when an identifier was valid. This discrepancy could allow an attacker to confirm the presence of a particular patient record through probing. The threat was mapped to CWE-203 (Observable Discrepancy), which denotes the leakage of sensitive information through observable differences in behaviour. The mapping highlights the need for consistent, privacy-preserving error handling, leading to recommendations for generic error responses and rate limiting.

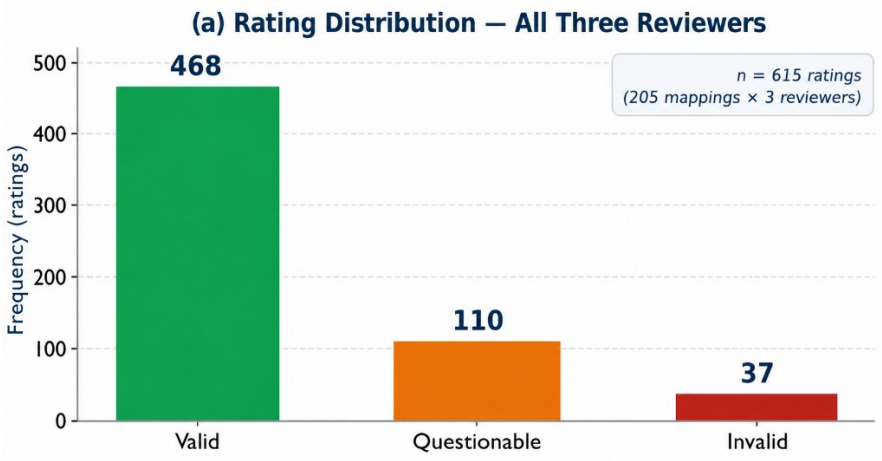
Disclosure example. A Disclosure of Information threat was identified in the data transmission layer of the telehealth platform, where sensitive health telemetry and personal details were transmitted from patient devices without encryption. This represents a clear information disclosure risk, since an eavesdropper could intercept and read the data. The weakness was mapped to CWE-319 (Cleartext Transmission of Sensitive Information), underscoring the need to enforce transport-layer encryption for all personal health data in transit.

Each exemplar demonstrates how the mapping step produces actionable insight. Rather than leaving privacy threats at an abstract level, the methodology connects them to the low-level

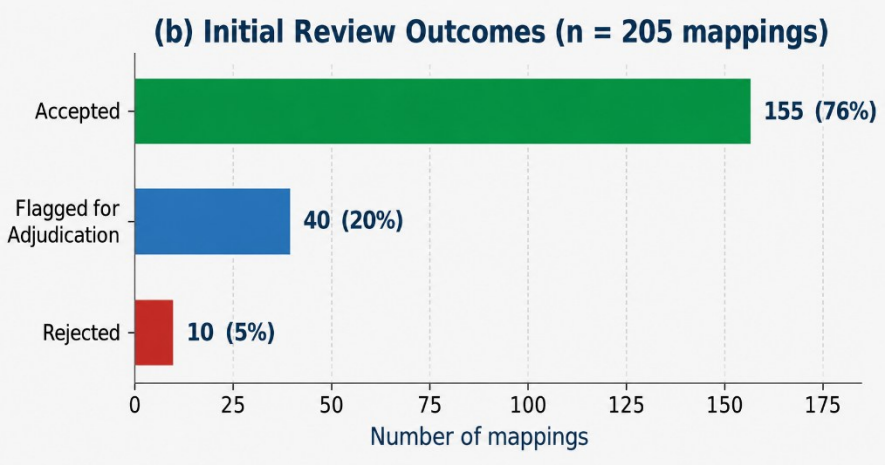
weaknesses with which developers and security teams are already familiar. This establishes a standardised framework that bridges privacy threat categories with concrete, code-level vulnerabilities (Sion et al., 2018), allowing established resources such as vulnerability databases, CWE documentation, and secure coding guidelines to be applied directly to the threats identified.

3.3.3 Quantitative Validation Outcomes

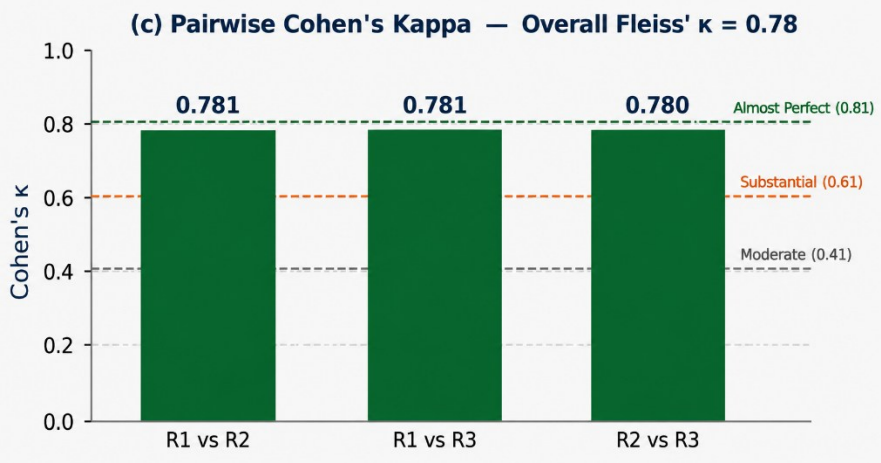
To assess inter-rater agreement on the mappings within a purposively selected expert panel, avalidation study was conducted on a stratified sample of two hundred and five mappings drawn proportionally from across the seven LINDDUN categories. Three independent reviewers, comprising a privacy engineer, an application security specialist, and an academic threat-modelling researcher, each rated every mapping in the sample as Valid, Questionable, or Invalid. Inter-rater agreement was then computed to measure the consistency of the analysis. Figure 3-2 presents the results across four panels.



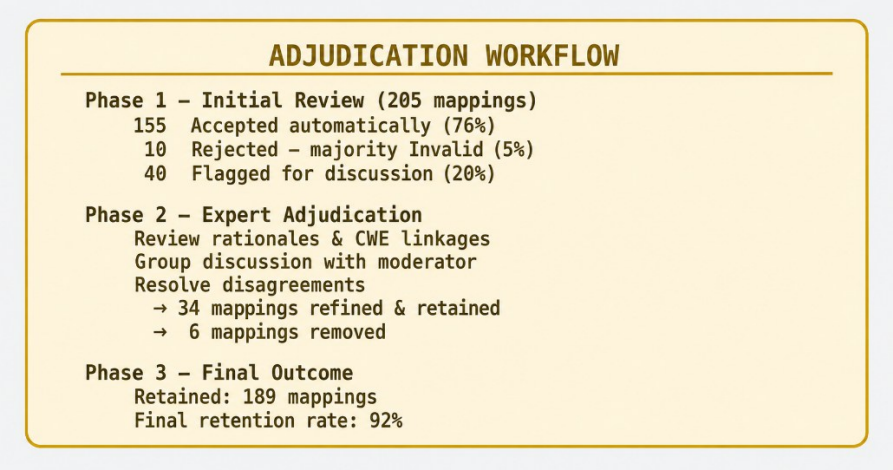
(a) the distribution of all six hundred and fifteen ratings;



(b) the initial review outcomes for the two hundred and five mappings;



(c) the pairwise Cohen's kappa between reviewer pairs against the Landis and Koch interpretive thresholds;



and (d) the three-phase adjudication workflow and final retention.

Figure 3-2 (a–c): Inter-rater agreement from the three-reviewer expert validation of the LINDDUN-to-CWE mappings.

Panel (a) shows the distribution of the six hundred and fifteen individual ratings issued across the sample. The substantial majority, four hundred and sixty-eight ratings, classified mappings as Valid, with one hundred and ten rated Questionable and only thirty-seven rated Invalid. The small proportion of Invalid ratings indicates that fundamental objections to the proposed mappings were uncommon, and that the principal source of uncertainty lay at the boundary between Valid and Questionable rather than in outright rejection.

Panel (b) summarises the initial review outcome for each of the two hundred and five mappings under the acceptance rule defined in Section 3.2.6, whereby a mapping is accepted when at least two of the three reviewers rate it Valid with none rating it Invalid. On this basis, one hundred and fifty-five mappings, or seventy-six per cent, were accepted in the first round without revision. Forty mappings, twenty per cent, were flagged for adjudication, and ten, five per cent, were rejected based on a majority Invalid rating.

Panel (c) reports the pairwise Cohen's kappa between each reviewer pair, plotted against the interpretive thresholds of Landis and Koch (1977). Agreement was consistent across all three pairs, with values of 0.781, 0.781, and 0.780, each falling within the substantial range. The aggregate agreement, computed as Fleiss' kappa across all three reviewers, was 0.78, likewise interpreted as substantial agreement. The close clustering of the pairwise values demonstrates that the high overall agreement was not driven by any single reviewer pair but reflected genuine consensus across the panel.

Panel (d) illustrates the three-phase adjudication workflow. Following the initial review, the forty flagged mappings proceeded to expert adjudication, in which reviewers re-examined the rationales and CWE linkages in moderated discussion. Of these, thirty-four were refined and retained while six were removed. Combined with the one hundred and fifty-five first-round acceptances, this yielded one hundred and eighty-nine retained mappings, a final retention rate of ninety-two per cent.

The level of agreement achieved is noteworthy given the interpretive judgement inherent in privacy threat analysis. It indicates that the structured methodology, with its defined threat categories and evidence-based mapping rule, produces results on which independent experts can substantially concur. This compares favourably with prior observations that privacy threat

modelling exercises can suffer from low inter-rater agreement owing to the difficulty of interpreting privacy concerns (Sion et al., 2018).

The validation also revealed which categories were most and least contentious. Linkability and Disclosure of Information achieved the highest agreement, consistent with their concrete and technically grounded nature, since a data linkage vulnerability or an unauthorised exposure maps closely to well-established weakness patterns. Identifiability and Non-compliance produced slightly lower agreement. For Identifiability, reviewers occasionally differed on the most appropriate CWE for threats that bordered on policy rather than purely technical concerns. For Non-compliance, the difficulty arose from the regulatory context-dependency of certain mappings, where the applicability of a given CWE varied with jurisdiction. In both cases the disagreements were resolved through the moderated adjudication described in Section 3.2.6.

Figure 3-3 illustrates the distribution of accepted mappings across the seven LINDDUN threat categories and the major CWE groups to which they were mapped. Examining acceptance rates by LINDDUN category first, Figure 3-4 shows that all seven categories achieved majority acceptance, though agreement varied slightly across them.

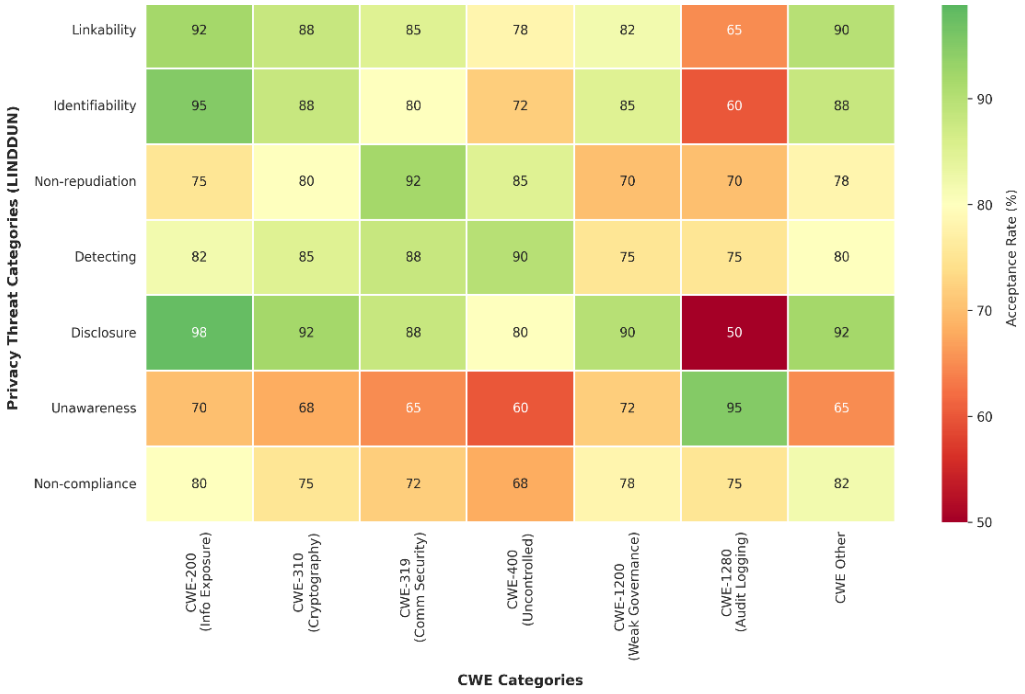


Figure 3-3: Heatmap of Expert Acceptance Rates Across LINDDUN Threat Categories and CWE Groups

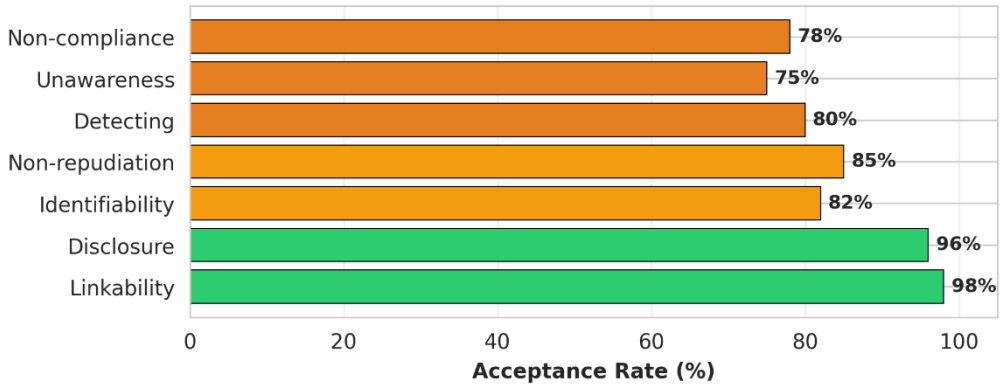


Figure 3-4: Expert Reviewer Acceptance Rate by LINDDUN Threat Category

Linkability and Disclosure of Information threats achieved the highest acceptance rates, with nearly all proposed mappings in both categories confirmed by the expert panel. This outcome is consistent with the relatively concrete and technically grounded nature of these threats, it is generally straightforward for security experts to agree on an appropriate CWE for a data linkage vulnerability or an unauthorised data exposure scenario, as both map closely to well-established software weakness patterns.

Identifiability and Non-compliance threats produced slightly lower agreement levels. For Identifiability, reviewers occasionally disagreed on the most appropriate CWE to represent threats that bordered on policy rather than purely technical issues. For Non-compliance, the challenge arose from the legal and regulatory context dependency of some mappings, where the severity and applicability of a given CWE varied depending on the jurisdiction or regulatory framework under consideration. In both cases, disagreements were resolved through the moderation process described in Section 3.2.7, and a consensus position was reached for each contested entry.

Turning to the CWE distribution, Figure 3-5 shows that weaknesses related to information exposure, primarily the CWE-200 series, and those addressing cryptographic and communication failures, such as CWE-319 (Cleartext Transmission of Sensitive Information), were the most frequently confirmed mappings, together accounting for approximately 50% of all accepted entries. This pattern directly reflects the threat flashpoints identified in the DFD analysis, where a large proportion of privacy threats involved either unauthorised access to patient data or the absence of adequate encryption during data transmission. CWEs

associated with audit logging and user transparency, which correspond most closely to Unawareness and Non-repudiation threats, appeared less frequently in the accepted mapping set, consistent with the comparatively lower incidence of those threat types observed during elicitation.

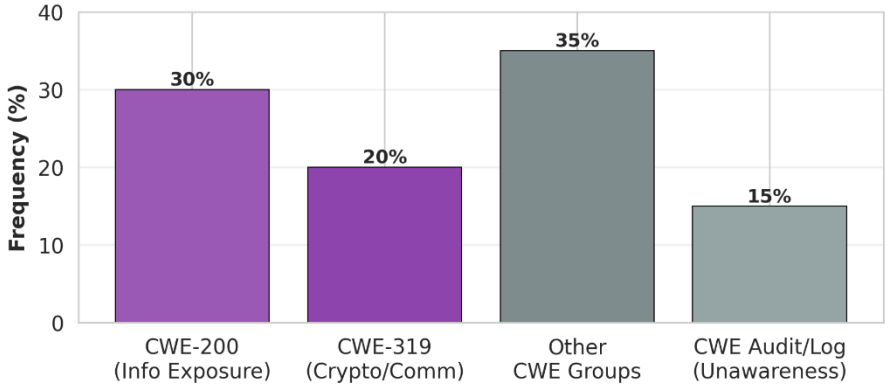


Figure 3-5: Distribution of Accepted Mappings Across CWE Groups in the Expert-Reviewed LINDDUN-to-CWE Mapping Matrix

3.3.4 Actionability and Integration Readiness

A key advantage of linking privacy threats to CWE weaknesses is the actionability of the results. Each identified threat instance from the analysis is not an isolated concern, rather it is connected via traceable links to specific code-level patterns and mitigation techniques. In practice, this means the outcome of threat modelling directly informs developers what to fix and how. For example, if a Disclosure threat was mapped to CWE-319 (cleartext sensitive data transmission), the trace link leads to the exact modules or functions where data is sent without encryption, and it points to the remedy (for example, “enforce TLS encryption on data transport” in that module). By providing these concrete trace links, the results ensure that the abstract findings of the threat analysis are immediately usable by implementation teams. This emphasis on traceability reflects an alignment with modern secure development life cycles, where privacy considerations are meant to be embedded from design through coding (Ameh et al., 2025).

Finally, to support ongoing integration and tool support, all the validated threats, mappings, and traceability information have been consolidated into a structured dataset. This dataset captures each threat instance with its LINDDUN category label, the corresponding CWE identifier, references to example code (where applicable), and the suggested mitigation

or requirement to resolve it. The creation of this dataset not only serves as documentation but also enables automation (e.g., it could be used to train detection tools or to track privacy requirements in project management systems). The schema and contents of the dataset are discussed in detail in Chapter 4, where it is used to develop and evaluate automated privacy vulnerability detection.

3.4 Discussion

The research presented in this chapter addressed two distinct but complementary objectives within the broader context of healthcare software security and privacy. The first objective sought to demonstrate data flow patterns in a typical healthcare information system and map privacy and vulnerability misuse cases using the LINDDUN privacy threat methodology to the Common Weakness Enumeration taxonomy. This objective has been fully addressed through a systematic and rigorous process that created explicit linkages between high level privacy concerns and concrete code level vulnerabilities. The second objective aimed to create a novel healthcare cybersecurity focused dataset for natural language processing. This objective has been partially addressed through the groundwork established by the methodology described in this chapter, with the resulting dataset now available for machine learning applications.

The importance of fully achieving the first objective cannot be understated. Historically, privacy threat modelling and software vulnerability assessment have been treated as separate activities within healthcare cybersecurity. As highlighted by Aijaz et al. (2023), most existing methodologies fail to integrate privacy analysis with vulnerability assessment, resulting in fragmented insights and limited understanding of how specific software weaknesses can compromise patient privacy. The methodology developed in this research closes this conceptual and operational gap by establishing a systematic linkage from each element of the data flow diagram, through relevant LINDDUN privacy threat categories, to the specific Common Weakness Enumeration entries capable of enabling those threats. This integrated process produces a coherent “privacy threat chain,” offering a clear traceable path from architectural risk to implementation flaw and ultimately to code-level controls.

3.4.1 Systematic Threat Elicitation Across LINDDUN Categories

The threat elicitation process conducted across all seven LINDDUN privacy categories demonstrate the comprehensiveness of the approach. The concentration of Linkability threats at data exchange points between the EHR, pharmacy, laboratory, and imaging subsystems reflects a structural characteristic of healthcare architectures: the clinical necessity of sharing patient data across multiple systems creates persistent cross-system correlation risks that are difficult to eliminate without compromising operational functionality. The prevalence of metadata and telemetry surfaces as linkability vectors, rather than direct identifier reuse, underscores that de-identification alone is insufficient as a mitigation strategy; the system architecture itself must be designed to limit the availability of quasi-identifiers across trust boundaries.

The identifiability analysis revealed that healthcare systems frequently attempt de-identification while retaining numerous re-identification pathways through combinations of residual attributes. This finding is consistent with established research demonstrating that ostensibly anonymous health data remains vulnerable to re-identification through modest auxiliary information (Rocher et al., 2019). The implication for developers is that de-identification must be treated as a system-wide design property rather than a data transformation step applied at a single point, since the function-level analysis identified re-identification risks arising from authentication logic, API response behaviour, and identifier management practices distributed across multiple subsystems.

The Non-repudiation and Detectability findings expose an inherent tension in healthcare software design: the audit trails mandated by HIPAA and similar regulatory frameworks for accountability purposes are simultaneously potential sources of privacy harm if their granularity or accessibility is not carefully controlled. The specific risk of detectability through differential system responses, where varying error messages or response timings allow an external observer to infer whether a patient record exists, illustrates that privacy threats can arise from design decisions that appear operationally neutral, reinforcing the argument that privacy analysis must be applied systematically to all system interfaces rather than only to data storage and transmission components.

The Disclosure findings demonstrate that information exposure risks in healthcare software arise from two distinct and complementary pathways: technical memory management vulnerabilities in C/C++ components, and configuration and transmission failures in higher-level system interfaces. The co-occurrence of these two categories within the same system analysis confirms that an effective disclosure mitigation strategy must address both the code-level memory safety practices of legacy medical device components and the transport-layer encryption and serialisation practices of modern API-based healthcare platforms. Treating these as separate concerns addressed by separate teams, risks leaving the gap between them unaddressed.

The Unawareness findings highlight that transparency failures in healthcare software are frequently embedded in default configurations and system design choices rather than in overt policy violations. The identification of subsystems that enable broad third-party data sharing by default, and of audit logs that are inaccessible to the patients whose records they document, illustrates that unawareness threats often arise not from malicious intent but from design cultures that prioritise functionality over transparency. This finding has direct implications for how PbD should be operationalised in healthcare development: transparency must be treated as a functional requirement with testable acceptance criteria, not as a post-hoc compliance consideration.

The Non-compliance findings revealed that regulatory obligations under HIPAA, GDPR, and organisational policy are frequently unenforceable not because of deliberate circumvention but because the system architecture does not support the mechanisms required to comply. The identification of archiving systems without patient-controlled deletion capability, and of data flows lacking the encryption mandated by hospital policy, illustrates that non-compliance threats often arise from gaps between what regulations require and what legacy or third-party system components can technically deliver. This architectural non-enforceability represents a category of privacy risk that purely technical security analysis would not surface, reinforcing the value of integrating regulatory compliance assessment within the threat modelling process.

3.4.2 LINDDUN to CWE Mapping Contributions

The systematic mapping of LINDDUN threat categories to Common Weakness Enumeration entries represents a methodological contribution that extends existing threat modelling practice. Previous research has established the value of LINDDUN in healthcare contexts (Ameh et al., 2025) and recognized the importance of CWE for vulnerability classification (MITRE, 2023). However, the explicit bidirectional mapping between privacy threat categories and software weakness types remains underdeveloped in published literature. The approach taken here derives CWE mappings through careful analysis of how specific code level weaknesses could enable privacy threats identified through LINDDUN analysis.

Hence, the many-to-many approach of the mappings reflects reality. Consequently, a single LINDDUN threat category may correspond to multiple distinct CWE entries depending on context and manifestation. A Disclosure of Information threat might arise from CWE 200: Exposure of Sensitive Information to an Unauthorized Actor if data leaks through logs or CWE 319: Cleartext Transmission of Sensitive Information if data flows unencrypted across networks or CWE 502: Deserialization of Untrusted Data if malicious code embedded in serialized patient records allows unauthorized data access (MITRE, 2023). The context determines which specific weaknesses are relevant.

Conversely, a single CWE entry may contribute to multiple privacy threats depending on the data it exposes or processes. Research by Zhou et al. (2025) in their systematic literature review of large language models for vulnerability detection demonstrated that a memory safety vulnerability such as CWE 125: Out of Bounds Read could result in Disclosure of Information if it reads sensitive data, Non repudiation threats if it compromises audit information, or Identifiability threats if it exposes identifying information that could enable re identification. The methodology handles this complexity through evidence-based mapping that documents the specific threat scenario enabling the threat CWE link.

The evidence-based approach required for each mapping ensures that associations are grounded in realistic scenarios rather than speculative connections. For each threat CWE pairing, the methodology captured representative vulnerability patterns, conditions enabling

exploitability, privacy consequences if exploited, and plausible mitigation strategies. Schneider et al. (2024) demonstrated in their empirical study of dataflow diagrams that the specificity of threat to vulnerability linkages significantly impacts the utility of threat models for developers. This documentation creates an audit trail demonstrating that the mapping is not arbitrary but derives from careful analysis of threat manifestation patterns in healthcare code contexts.

3.4.3 Expert Review and Validation Protocol Outcomes

The expert review and validation protocol achieved inter rater agreement of approximately 0.78 using Fleiss' kappa statistic (Figure 3-2). This level of agreement exceeds thresholds commonly considered substantial in research environments. The achievement of substantial agreement is noteworthy given that expert assessments of complex security scenarios often produce lower agreement levels. Shoufan and Damiani (2017) documented in their research on inter rater reliability percentages of information security experts that security professionals frequently demonstrate low agreement when assessing threat severity, with some studies showing agreements as low as 13.8 to 20.1 percent when using standard security frameworks. The substantially higher agreement achieved here suggests that the structured methodology provided by LINDDUN and the evidence-based mapping rubric created sufficient clarity to guide expert judgments toward consensus.

The adjudication process that followed initial independent reviews proved valuable for resolving disagreement. In several cases where experts initially disagreed, closer examination of traceability artifacts clarified the conceptual basis for mappings, leading to consensus after discussion. Van Landuyt et al. (2024) demonstrated in their comparative evaluation of threat modelling tools that structured artifacts significantly improve expert consensus when resolving disagreements in security assessments. In other cases where agreement could not be reached after adjudication, mappings were conservatively excluded from the final set. This conservative approach prioritized precision over recall, accepting that some valid mappings might be excluded rather than including potentially spurious associations.

The pilot application of the methodology to two subsystems including the Electronic Health Record module and the Medical Imaging subsystem provided preliminary support for

the approach's applicability in realistic contexts. All threats identified in these pilot subsystems could be mapped to existing Common Weakness Enumeration entries, suggesting that the mapping matrix achieved good coverage of privacy threat scenarios likely to arise in actual healthcare software systems. Furthermore, Van Landuyt (2024) in their evaluation of threat modelling tools emphasized that coverage validation through real system pilots significantly strengthens confidence in threat modelling approaches. This preliminary pilot evidence offers cautious support that the resulting artifacts can be practically applied in healthcare settings beyond the controlled analysis environment.

3.4.4 Bridging Design and Implementation

A core contribution of this research lies in creating a practical connection between design level privacy analysis and implementation level vulnerability assessment. Traditional threat modelling approaches such as LINDDUN operate primarily at architectural and design levels. Developers then implement systems following (or not following) the design specifications. Security testing and code analysis operate at the implementation level. However, explicit linkages between what designers identified as risks and what code scanners flag as vulnerabilities have been sparse in healthcare contexts, an observation supported by systematic reviews of healthcare threat modelling practices (Aijaz et al., 2023).

The function level granularity of analysis proved particularly appropriate for this connection. Functions represent units of computation that correspond to implementation design decisions. A healthcare software function that encrypts a file, queries a database, or parses a message encapsulates specific privacy relevant behaviours. By mapping LINDDUN threats to specific CWE weaknesses that could enable those threats, the methodology enables developers and security testers to understand not merely that a CWE 125 out of bounds read exists, but why it matters: specifically, what privacy threat it enables and what patient privacy impact it could cause. Huda et al. (2023) demonstrated in their machine learning approach to healthcare cyber threat analysis that function level granularity significantly improves both model interpretability and practical applicability for developer communities.

By integrating privacy considerations into vulnerability management, organisations transition from mere compliance to a security approach centred on safeguarding sensitive

information. When development teams comprehend that a buffer overflow within patient data handling software poses a Disclosure of Information risk, they assess its seriousness more effectively than when only presented with a generic CWE 120 classification. Contextualising vulnerabilities in terms of privacy impact drives more focused prioritisation and remediation efforts. Fu et al. (2024), through their tool development research, found that vulnerability detection systems incorporating privacy threat context notably enhance developer responsiveness and accelerate remediation when compared to standard vulnerability alerts.

3.4.5 Healthcare Dataset Creation for Machine Learning

The partial addressing of the second objective involving creation of a healthcare cybersecurity focused dataset for natural language processing represents important groundwork. By systematically annotating code functions with both CWE categories and LINDDUN privacy threat types, the resulting dataset creates machine learning training data that links vulnerability detection with privacy impact assessment. Ameh et al. (2025) in their presentation of the C3-VULMAP dataset demonstrated that this dual annotation structure enables development of models that not only detect vulnerable code but also predict privacy threat implications. This dual annotation capability addresses a significant limitation in existing vulnerability detection datasets.

The scale of the resulting dataset proves significant. With 6,308,314 functions (86,239 vulnerable, 6,222,075 non-vulnerable), the dataset provides substantial negative class representation addressing the class imbalance problems that have plagued earlier security-oriented code datasets. Yu et al. (2024) noted in their systematic literature review of large language models for vulnerability detection that class imbalance remains a persistent challenge in security research, frequently limiting model performance. The healthcare specificity of the annotation adds domain value that general software vulnerability datasets cannot provide. Models trained on this dataset can learn healthcare specific privacy threat patterns that may not manifest in general purpose code.

The function level granularity of dataset entries aligns with practical vulnerability detection workflows in healthcare organizations. Code review, static analysis tools, and

dynamic testing often operate at function or method level granularity. González-Manzano and Garcia-Alfaro (2025) demonstrated in their research on CNN based vulnerability detection that function level predictions produce outputs directly interpretable by developers. A function level prediction of vulnerability combined with prediction of which LINDDUN threat it enables provides actionable guidance for security improvement.

3.4.6 Practical Implications for Healthcare Organizations

The methodology and resulting artifacts provide several practical capabilities for healthcare organizations. First, security and privacy teams can apply the systematic threat elicitation process to map their own systems' architectures to privacy risks. By constructing data flow diagrams and applying the LINDDUN threat trees adapted for healthcare, organizations can identify privacy gaps in their systems. The healthcare specific DFD template provided through the patient journey example in this chapter offers a starting point that organizations can customize for their specific systems. Nagaraja et al. (2025) demonstrated in their cyber threat modelling work on LLM based healthcare systems that systematic DFD construction creates a practical foundation for threat identification in complex healthcare architectures.

Second, organizations can use the LINDDUN to CWE mapping matrix to prioritize code review and security testing activities. Rather than attempting to examine entire codebases for all possible Common Weakness Enumeration categories, development teams can focus on the specific weaknesses known to enable privacy threats identified in their threat models. This targeted approach increases the efficiency of security analysis while improving its effectiveness for privacy protection.

Third, the validated dataset provides a resource for training machine learning models to detect privacy relevant vulnerabilities in healthcare code. Organizations developing healthcare software can use or build upon this dataset to train models supporting their development processes. The models can provide automated detection of vulnerability patterns combined with privacy threat categorization, supporting both developers and security professionals in understanding vulnerability implications. Savanović et al. (2023) demonstrated in their work on healthcare 4.0 intrusion detection that machine learning

models trained on healthcare specific data substantially outperform general purpose security models when applied to healthcare contexts.

3.4.7 Methodological Limitations and Constraints

The research approach presented here encompasses several limitations that merit explicit discussion. The threat elicitation process, while systematic, relies on expert judgment in determining which LINDDUN threat categories apply to specific data flow diagram elements. Despite the structured approach and the use of threat trees to guide elicitation, elements of subjective interpretation remain. Aijaz et al. (2023) noted in their systematic evaluation of healthcare threat modelling methods that all existing approaches require expert judgment, creating potential for analysis variability across different healthcare organizations.

The Common Weakness Enumeration taxonomy continues to evolve, with new weaknesses added and existing ones refined. The mappings established at the time of analysis reflect current CWE structure, but future evolution of the taxonomy could create misalignment. MITRE (2023) documents ongoing CWE development processes including new weakness categories and existing entry refinements. Healthcare specific weaknesses that might warrant dedicated CWE entries remain unrecognized in the current taxonomy. The methodology provides a framework for updating mappings as the CWE taxonomy evolves, but ongoing maintenance represents a practical constraint.

The focus on C and C++ code reflects the prevalence of these languages in performance critical healthcare systems, but many contemporary healthcare systems incorporate Python, Java, C sharp, and JavaScript. Yadav (2024) noted that while C and C++ dominance persists in medical devices, modern healthcare platforms increasingly employ memory safe languages. The memory safety vulnerabilities that dominate C and C++ concern do not apply equally to memory safe languages, requiring distinct mapping approaches for different technology stacks. Organizations working with polyglot codebases must apply multiple analytical lenses.

The data flow diagram representation, while valuable for capturing system architecture, inherently abstracts away implementation details. Schneider et al. (2024) demonstrated empirically that DFD abstractions sometimes obscure security relevant details such as authentication mechanisms, encryption implementations, or access control policies. The DFD

provides a foundation for threat modelling but does not eliminate the need for implementation level security analysis.

3.4.8 Contribution to Healthcare Cybersecurity

The enhanced LINDDUN methodology advances healthcare cybersecurity practice across four dimensions.

First, it puts privacy-by-design principles into practice by creating a systematic process through which privacy requirements can be traced from architectural design through to code implementation. Rather than remaining as abstract architectural principles, privacy considerations are translated into concrete implementation guidance and inputs for automated analysis tools, giving development teams a clear and actionable path from design-time threat identification to code-level remediation.

Second, the methodology addresses an identified gap in healthcare-specific threat modelling. Systematic reviews have revealed that domain-specific applications of frameworks such as LINDDUN in healthcare contexts remain limited (Aijaz et al., 2023). The approach presented here demonstrates how LINDDUN can be adapted and extended to capture healthcare-specific privacy concerns, and how the resulting threat models can be connected directly to guidance that developers can act upon during implementation.

Third, the research establishes a foundation for machine learning-based privacy threat detection. By constructing a large-scale dataset annotated with both vulnerability classifications and LINDDUN privacy threat labels, the work enables the development of models that support automated, privacy-focused vulnerability discovery. This automation has the potential to help healthcare organisations of all sizes improve their privacy posture without requiring extensive in-house security expertise, an important consideration in resource-constrained clinical environments. Recent systematic reviews confirm that domain-specific training data significantly improves the effectiveness of machine learning models for healthcare cybersecurity applications (Huda et al., 2023).

Fourth, the expert-reviewed traceability artifacts, comprising the threat tree documentation, misuse case narratives, and LINDDUN-to-CWE mapping matrix, provide a reusable knowledge base that other healthcare organisations and researchers can build upon.

Sion et al. (2024) demonstrated that specialised privacy threat knowledge bases substantially improve the consistency and scalability of threat modelling analyses across organisations. By making these artifacts available, the methodology contributes not as a one-off analytical exercise but as an extensible resource that can accelerate threat modelling in other healthcare systems and contexts.

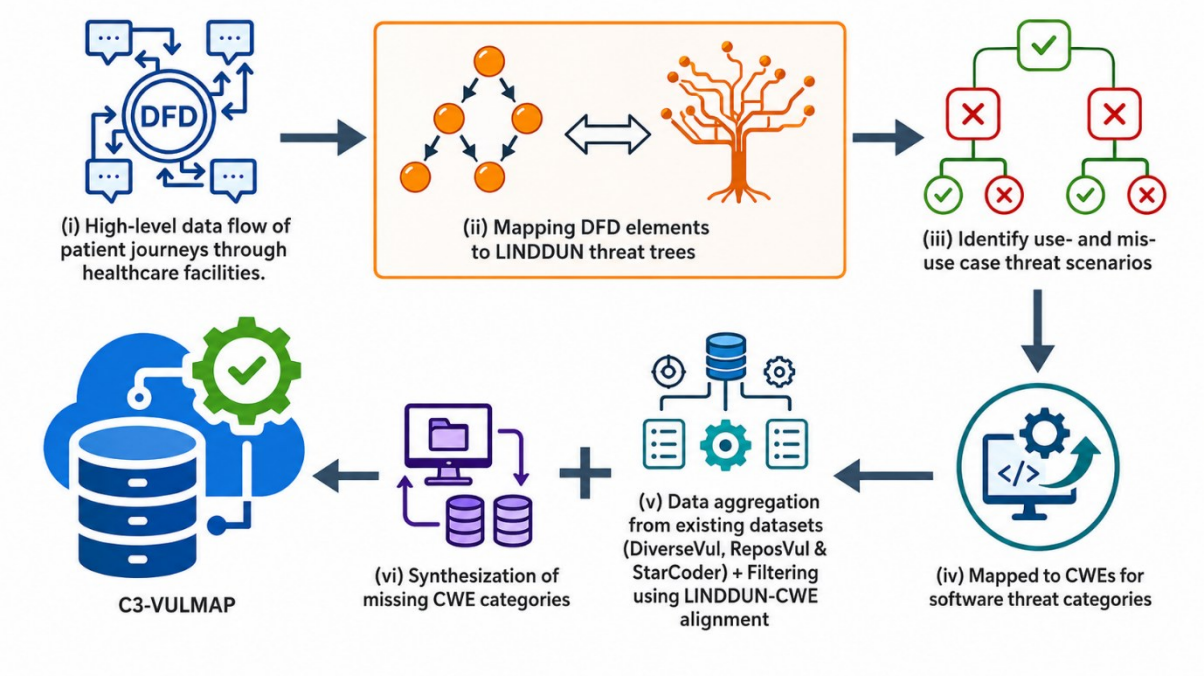


Figure 3-6: The Enhanced LINDDUN Privacy Threat Modelling Process for Healthcare Software Systems, Showing the Systematic Pathway from Data Flow Analysis to CWE Mapping (Stages i–iv) and its Role in Informing the C3-VULMAP Dataset Construction Pipeline (Stages v–vi, Detailed in Chapter 4)

Figure 3-6 presents the complete six-stage pipeline underpinning this research, with the first four stages constituting the enhanced LINDDUN methodology that is the primary focus of this chapter. The process begins with (i) the construction of a high-level data flow diagram capturing patient journeys through healthcare facilities, which establishes the architectural foundation for all subsequent privacy analysis. Building on this, (ii) each DFD element is systematically mapped to the corresponding LINDDUN threat trees, enabling structured enumeration of privacy threats across all system components and trust boundaries. From this enumeration, (iii) use and misuse case threat scenarios are identified, providing the narrative and contextual grounding through which each threat is made concrete and actionable. These scenarios are then (iv) mapped to their corresponding CWE entries, translating high-level

privacy threats into specific, code-level software weakness categories. Together, these four stages deliver the validated LINDDUN-to-CWE mapping matrix and associated traceability artifacts that are the principal contributions of this chapter. The final two stages, (v) aggregation and filtering of existing vulnerability datasets using the LINDDUN-to-CWE alignment, and (vi) synthesis of missing CWE categories, extend this foundation into the construction of the C3-VULMAP dataset and are described fully in Chapter 4.

4 Chapter 4: Dataset Preparation and Classification

4.1 Introduction

The LINDDUN-to-CWE mapping developed and validated in Chapter 3, which achieved substantial agreement within a small expert panel, established a structured correspondence between design-level privacy threats and the code-level weaknesses through which those threats are realised in healthcare software. That mapping now serves as the foundational layer for the two contributions presented in this chapter: the construction of a large-scale, healthcare-specific vulnerability dataset, and a comprehensive comparative evaluation of machine learning approaches for static code vulnerability detection.

This chapter fully addresses Objective 2 and partially advances Objective 3 of this research. Objective 2 requires the creation of a novel healthcare cybersecurity-focused dataset for natural language processing, comprising annotated C/C++ code samples with vulnerability labels and CWE classifications relevant to healthcare software security requirements. Objective 3 requires the development and evaluation of a reliable method for static software code security testing using NLP techniques, comparing traditional machine learning, graph neural networks, and transformer-based approaches to identify the most effective paradigm for detecting vulnerable code in healthcare software.

Most existing vulnerability corpora are constructed with a general-purpose scope and lack the healthcare specificity required for domain-sensitive applications such as electronic health record systems, medical device firmware, and patient data processing platforms. The labelling of such vulnerabilities with corresponding privacy threats is largely absent in available datasets, limiting their usefulness for training models that must understand both the technical and regulatory dimensions of healthcare software security. This chapter addresses these limitations through a dataset construction methodology that combines multiple state-of-the-art source corpora, a privacy-filtering strategy grounded in the LINDDUN-to-CWE alignment from Chapter 3 and controlled synthetic augmentation for underrepresented vulnerability types. The resulting dataset provides the empirical foundation for the machine learning evaluation that follows, in which fourteen models spanning three paradigms are evaluated under a consistent protocol, with findings directly

informing the model selection decisions for the vulnerability detection prototype presented in Chapter 5.

4.1.1 Motivation for a Healthcare Cybersecurity Natural Language Processing Dataset

Existing vulnerability databases such as the National Vulnerability Database (NVD)[13] and the Common Vulnerabilities and Exposures (CVE)[14] repository provide extensive information on security flaws, but they are not tailored to the needs of healthcare information systems. They do not capture the regulatory and privacy contexts specific to healthcare applications, nor do they provide annotated code samples suitable for machine learning-based detection tasks (Mejia-Granda, et al., 2024). Vulnerabilities in healthcare software systems often expose sensitive personal and medical data, creating privacy implications that extend beyond technical fault descriptions (Ewoh & Vartiainen, 2024).

The general-purpose nature of datasets like CVE and NVD limits their applicability in modelling domain-specific threat patterns. These datasets lack semantic enrichment, contextual labelling, and threat-model alignment that would be required to train models to detect privacy vulnerabilities specific to healthcare systems. Generic datasets do not incorporate privacy threat taxonomies such as LINDDUN, which are necessary for evaluating privacy risks during software development (Wuyts, Scandariato, & Joosen, 2014). They also do not provide examples of how such vulnerabilities manifest in code, particularly in languages common to healthcare systems like C and C++.

To address these limitations, there is a clear need for a domain-specific dataset that captures the intersection of privacy, software vulnerabilities, and healthcare development. Such a dataset should contain labelled code samples, enriched with both CWE identifiers and LINDDUN threat categories, allowing machine learning models to learn patterns that contribute to privacy risks. The mapping of LINDDUN threats to CWE identifiers, developed in

---

[13] https://www.nist.gov/itl/nvd#:~:text=Secure%20.gov%20websites%20use%20HTTPS,of%20the%20nation's%20cybersecurity%20infrastructure.

[14] https://github.com/password123456/nvd-cve-database

Chapter 3, provides the necessary framework for encoding privacy semantics in code vulnerability labels, enabling privacy-aware detection strategies using natural language processing and static code analysis.

4.1.2 Gap in Healthcare-Relevant Code Datasets

Several vulnerability datasets have been created to support machine learning research in secure software development. Examples include the Devign, Big-Vul, ReVeal, and CodeXGLUE. Each of these contributes significantly to the field of automated vulnerability detection, but none of them sufficiently address the healthcare domain or privacy threat modelling.

The Devign dataset, for instance, consists of labelled vulnerable and non-vulnerable C functions, extracted from real-world projects (Zhou et al., 2019). While useful for binary classification tasks, it does not support multi-label classification, nor does it provide privacy-related annotations. Similarly, Big-Vul, a large-scale dataset built from open-source repositories, offers access to vulnerability-fix pairs across multiple programming languages, but lacks domain-specific context and does not reflect healthcare code bases or medical information flows (Fan, et al., 2021).

CodeXGLUE, a benchmark suite for code intelligence tasks, incorporates datasets like Devign but offers no augmentation for regulatory or privacy-oriented tasks (Lu et al., 2021). Furthermore, these datasets focus primarily on software weaknesses from a performance and stability standpoint, ignoring the broader socio-technical implications in safety-critical domains such as healthcare. They also lack integration with threat modelling methodologies such as LINDDUN, which is essential for identifying and mitigating privacy risks systematically.

Another major limitation is the underrepresentation of low-level, memory-safe programming practices essential for C and C++, which dominate embedded and real-time medical systems (Mejia-Granda et al., 2024). Most existing datasets either mix programming languages without distinction or prioritize higher-level languages like Python and Java. Consequently, the unique vulnerability patterns found in C/C++, including pointer misuse, buffer overflows, and manual memory management errors, remain under-explored in publicly available corpora.

The absence of datasets that incorporate LINDDUN multi-label hierarchies, combined with a lack of healthcare-specific scenarios and privacy annotations, underscores the necessity of developing a novel corpus. This corpus should offer labelled code examples from real or representative healthcare systems, tagged with both CWE and LINDDUN identifiers. Such a resource would significantly advance research in privacy-respecting software engineering by enabling the training and evaluation of models that understand the link between low-level software weaknesses and high-level privacy threats.

4.1.3 Challenges in Code Vulnerability Classification

The task of classifying code vulnerabilities in a privacy-aware context introduces a series of challenges that affect both dataset construction and model performance. One of the most prominent issues is class imbalance. Vulnerabilities do not occur uniformly across software systems, and some CWEs appear far more frequently than others (Chakraborty et al., 2022). For instance, buffer overflows, which are common in low-level programming, may significantly outnumber logic or design-related flaws, leading to skewed data distributions. This imbalance can bias classifiers towards majority classes and diminish their sensitivity to rare but critical vulnerabilities (Leevy et al., 2018).

Another complication is found in the context sensitivity of code. Identical or similar code snippets may be vulnerable in one context but safe in another, depending on variable scope, function calls, and user inputs. This variability requires models to account not just for syntax but also for program semantics and control flow dependencies (Leevy et al., 2018). Furthermore, many vulnerabilities span multiple lines or involve interprocedural relationships that challenge simple token-based or sequential learning models (Peng et al., 2024).

The complexity of software systems further compounds these challenges. Code may involve intricate library dependencies, macro expansions, and deeply nested conditional logic, making static analysis and vulnerability localization difficult. In healthcare systems, this complexity is magnified because of the integration of heterogeneous systems, such as imaging devices, patient record software, and hospital network infrastructure, each with distinct data access and protection needs (Mejia-Granda et al., 2024).

In addition to structural complexity, language-specific issues also play a significant role. The prevalence of C and C++ in healthcare-related embedded systems necessitates the handling of memory management constructs that are absent in higher-level languages. Vulnerabilities such as use-after-free, double-free, and buffer overflows require fine-grained modelling of heap, stack, and pointer behaviour, which remains a difficult task for most traditional learning algorithms (Zhou et al., 2019).

Finally, the reliability of any classification model is limited by the quality of its training data. Label noise, introduced through human annotation errors, ambiguous vulnerability descriptions, or incomplete documentation, can distort model learning. This is particularly problematic in the case of evolving vulnerability definitions and taxonomies, where different annotators may disagree on whether a given code segment constitutes a privacy threat or what CWE best describes its weakness (Fan et al., 2020). The justification for this dataset is captured in the Figure 4-1 below.

Addressing these challenges requires a combination of thoughtful dataset engineering, sophisticated feature extraction, and the application of models that are sensitive to context, structure, and uncertainty. As this chapter demonstrates, these elements have been incorporated into the construction of a new dataset that supports the multi-label classification of healthcare-related code vulnerabilities aligned with privacy threats.

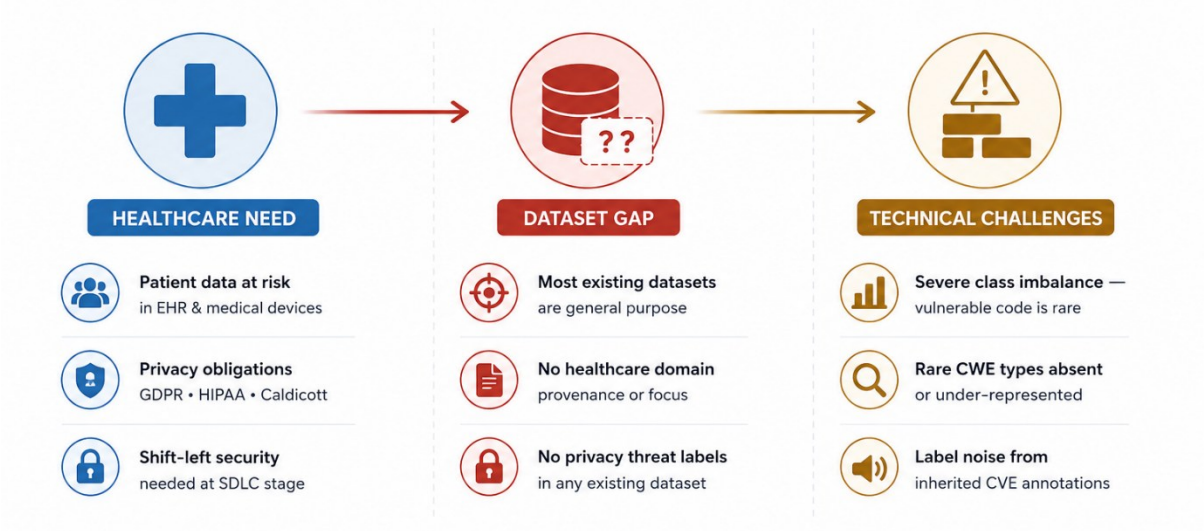


Figure 4-1: The motivation for a healthcare-aligned vulnerability dataset.

4.2 Dataset Construction

The dataset construction in this research represents a connection between privacy threat modelling in healthcare and the development of automated vulnerability detection methods for software code. The second research objective is addressed by developing a cybersecurity dataset tailored for healthcare-related natural language processing tasks. This also advances the third objective by generating labelled data suitable for static code security classification. Recent systematic reviews indicate that advancements in machine learning-based vulnerability detection are hindered by existing dataset limitations, such as narrow domain coverage, incomplete weakness taxonomies, limited annotation depth, and inconsistent coding context (Shiri Harzevili et al., 2024; Ameh et al., 2025; Hanif et al., 2021). To address these challenges, the dataset was constructed using a rigorous methodology combining multiple state-of-the-art sources, a filtering strategy prioritising healthcare privacy, and meticulously controlled synthetic augmentation.

4.2.1 Source datasets and selection rationale

The data construction process began with a survey of existing software vulnerability datasets and large-scale code corpora. Prior work has shown that deep learning models for vulnerability detection benefit from datasets that combine large volume, diverse project origins and fine-grained labels such as Common Weakness Enumeration identifiers (Chen et al., 2023; Shiri Harzevili et al., 2024). However, many earlier datasets have suffered from limited project diversity, noisy labels and outdated or tangled patches, which restrict their usefulness for training and evaluating modern models (Ghaffarian & Shahriari, 2017; Hanif et al., 2021).

Two recent datasets directly addressed several of these concerns and therefore formed the backbone of the construction process. The DiverseVul dataset provides 18,945 real vulnerable C and C++ functions that span 150 CWE types, together with 330,492 non-vulnerable functions extracted from 7,514 vulnerability fixing commits in 797 open-source projects (Chen et al., 2023). DiverseVul draws on security issue reports and their associated commits, and it applies rigorous filtering to improve label accuracy when compared with prior datasets that were derived from the National Vulnerability Database alone (Chen et al., 2023).

The projects represented in DiverseVul include systems and library code from several domains, such as software infrastructure, network services and applications where performance and safety are critical. This diversity introduces a wide range of coding styles and idioms, which is important for reducing model overfitting to narrow patterns.

The ReposVul dataset contributed a complementary perspective to the design and construction process of the dataset. ReposVul is a repository level high quality vulnerability dataset that integrates 6,134 Common Vulnerabilities and Exposures (CVE) entries, covering 236 CWE types across 1,491 open-source projects written in C, C plus plus, Java and Python (Wang et al., 2024). It was created through an automated framework that untangles vulnerability fixing commits, extracts multi granularity dependency information and filters outdated patches. As a result, ReposVul provides repository, file, function and line level views of each vulnerability and improves the alignment between code changes and security fixes (Wang et al., 2024). For data construction in this research, the focus was on function level representations and associated CWE labels.

A further inclusion was the StarCoder dataset, which aggregates a very large collection of source code and related artefacts from The Stack. StarCoder and StarCoderBase were trained on approximately one trillion tokens from more than eighty programming languages, GitHub issues, commit messages and Jupyter notebooks (Li et al., 2023). Although StarCoder is not a vulnerability dataset, its breadth provides broad exposure to coding idioms and language constructs that underpin many vulnerability patterns. Analysis of this corpus informed the design of prompts for synthetic code generation and supported decisions about realistic naming conventions, control structures and error handling practices for healthcare related software components.

The selection of these sources followed three main criteria. First, the datasets needed to provide granular function level code with explicit vulnerability labels, ideally grounded in widely accepted taxonomies such as CWE. Second, the sources had to exhibit diversity in projects, contributors and application domains to minimise coder bias and project specific overrepresentation. Third, at least some sources had to offer contextual metadata at repository or commit level to allow subsequent mapping to privacy threat models in healthcare. The combination of DiverseVul and ReposVul satisfied the first two criteria, and

ReposVul particularly satisfied the third. StarCoder (the Stack) contributed complementary large scale contextual information about code and developer practices across many domains. Table 4-1summarises the characteristics of DiverseVul, ReposVul and StarCoder (the Stack) and highlighted their suitability as building blocks for a privacy-oriented healthcare dataset.

Table 4-1 - Summary of Constituted Datasets

Aspect	DiverseVul	ReposVul	StarCoder (the Stack)
Size	18,945 vulnerable functions	6,134 CVE entries	~250 billion tokens
Total Functions	330,492 non-vulnerable	262,216 functions	N/A
Languages	C/C++	C, C++, Java, Python	86 languages
Projects	797	1,491	GitHub repositories
CWE Coverage	150 types	236 types	N/A (pattern-focused)
Granularity	Function-level	Repository/file/function/line	Code/context-level
Sources	Security issues/commits	Mend/Git commits	GitHub issues/notebooks

4.2.2 Integration into a healthcare privacy-oriented dataset

The integration stage as seen in Figure 4-2, combined the selected sources into a unified dataset capable of supporting both privacy analysis and binary vulnerability classification. To streamline this process, a simplified representation was adopted, introducing a single column named “label” that assigns each function a binary classification value. A value of one (1) indicates that the function contains a vulnerability, whereas a value of zero (0) denotes that the function is non-vulnerable. This approach aligns with the research objective of designing a reliable method for supervised classification of defective code segments, and it ensures a consistent representation across all integrated samples. Subsequently, the integration process extracted only C/C++ functions from the selected datasets. Although DiverseVul and ReposVul contain different metadata structures and varying levels of granularity, only function-level code was retained for the integrated dataset. As a result, every entry represents a coherent unit of program logic that can be reasonably analysed by a static code security model.

Following extraction, a harmonisation procedure was implemented. Given that the source datasets store code snippets in varying formats and encodings, the consolidation process included several cleaning operations to standardise the textual form of the code. This

involved normalising whitespace, removing unresolved encoding artefacts, and ensuring that special characters and indentation structures were consistent across all extracted functions. Additionally, the harmonisation stage eliminated duplicate samples by comparing the full function body and examining any remaining structural markers that indicated the same location in the originating repository. Whenever duplicates were identified, only a single representative instance was retained, which ensured balanced training data and prevented the overemphasis of specific code fragments during classification.

This deduplication operated at the level of exact, literal string matching on the full function body and therefore removed identical or near-identical duplicate entries arising from the same code appearing in multiple source repositories or dataset versions. It does not detect semantic or near-duplicate code clones, functions that are logically equivalent but differ in variable names, formatting, statement ordering, or minor refactoring, a distinction discussed as a limitation of the experimental design in Section 6.8.

Importantly, the integration process was characterised by the adoption of a binary vulnerability orientation. This decision was motivated by recent findings suggesting that binary classification provides a strong foundation for early-stage automated vulnerability detection systems, especially in domains where data privacy is critical and where the primary requirement is to flag potentially unsafe code prior to deployment (Shiri Harzevili et al., 2024). By simplifying the learning problem and removing the complexity associated with hundreds of possible CWE types, many of which appear infrequently in real-world code, the binary label approach also enhances model stability when dealing with highly imbalanced datasets, as observed in existing large-scale vulnerability corpora.

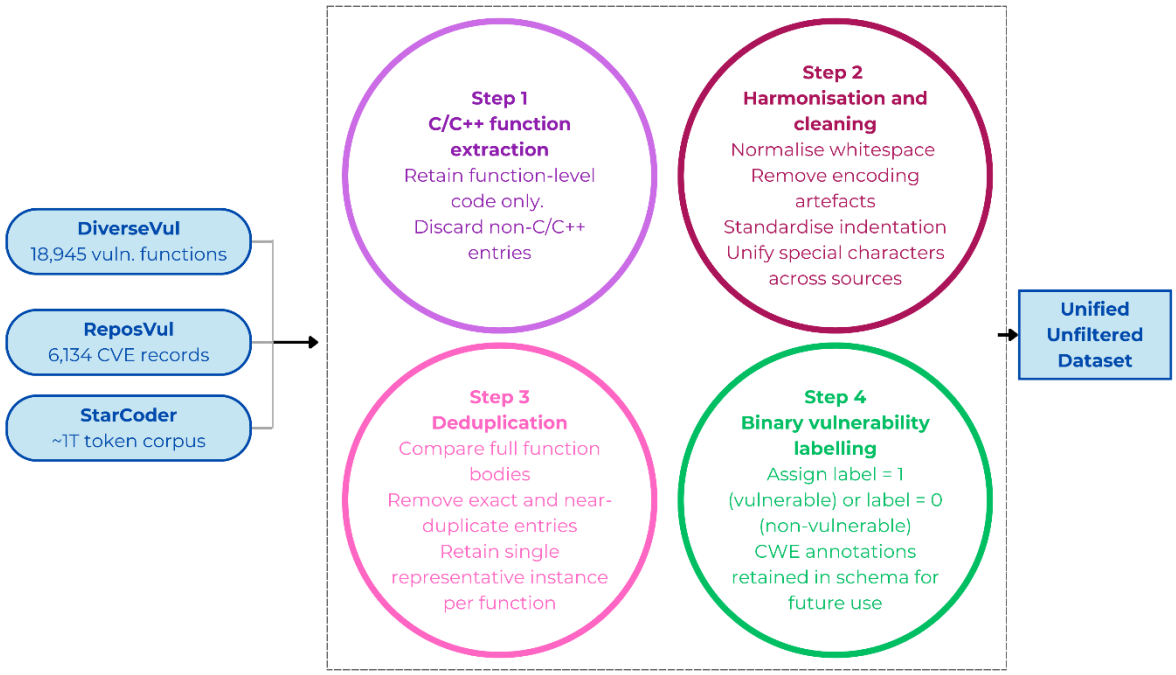


Figure 4-2: Dataset Integration Pipeline — Source Combination, Harmonisation, and Binary Labelling

4.2.3 Privacy filtering and healthcare domain alignment

Although DiverseVul and ReposVul provide rich vulnerability data, they are general purpose and do not explicitly target privacy threats or healthcare systems. To satisfy the privacy and healthcare focus of this thesis, an additional filtering and alignment process was required (Figure 4-3). The final dataset needed to capture vulnerabilities that impact confidentiality, integrity and other privacy properties of patient data in electronic health records, telehealth services and medical device software.

The first filtering step applied a mapping between privacy threat categories in the LINDDUN framework and the specific CWEs. Using an alignment that was developed in Chapter three, functions whose CWE labels fell into privacy relevant categories were retained, while functions associated with purely availability or performance issues that had no plausible privacy impact were removed. This ensured that every vulnerable instance in the resulting dataset had a direct or indirect relationship with privacy harm.

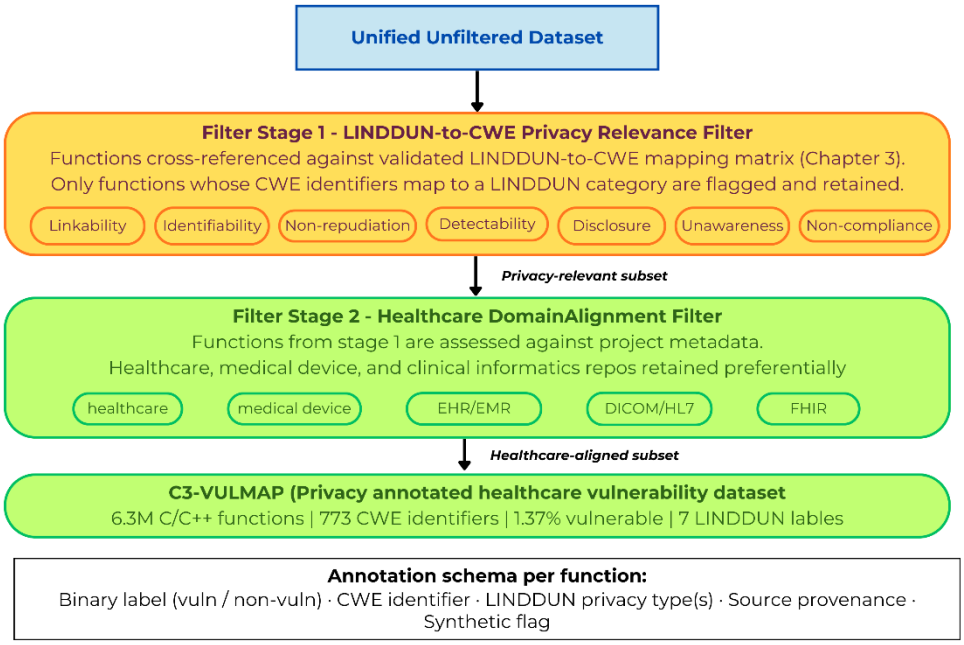


Figure 4-3: Two-Stage Privacy Filtering and Healthcare Domain Alignment Pipeline

The second filtering step strengthened the connection to healthcare software. In ReposVul it is possible to trace vulnerabilities back to their repositories and to inspect project level descriptions. This information was used to identify projects that were directly related to health information systems, medical imaging, clinical decision support, telemedicine or device control. When a project had clear health related functionality, its C/C++ vulnerability instances were prioritised for inclusion. In other cases, where the code was more general purpose, but the vulnerability type had strong relevance to healthcare privacy, the instances were retained to preserve coverage of critical CWE categories.

This selective prioritisation produced a dataset that is healthcare-aligned rather than healthcare-specific. Here, the alignment lies in the privacy-relevant labelling scheme applied to every function, not in the origin of every function's source code. A function retained because its CWE is privacy-relevant does not thereby become healthcare code as it remains general-purpose code carrying a healthcare-relevant annotation. Approximately 55% of the functions in the merged dataset originate from repositories identified during construction as verifiably associated with healthcare functionality, such as electronic health record systems, medical imaging, clinical decision support, telemedicine, or device control, based on inspection of ReposVul project-level metadata. The remaining approximately 45% originate

from general-purpose repositories retained specifically because their associated CWE fell within a privacy-relevant category under the LINDDUN-to-CWE mapping (Chapter 3), independent of any verified healthcare association. This proportion is approximate, reconstructed from construction-time records rather than a per-function provenance field retained in the final dataset schema (Table 4-4). The exact per-function source-repository labels were not preserved through the harmonisation stage, which precludes a detailed comparison of CWE distribution and function characteristics between the two groups at this time.

During extraction, all original labels that distinguished between multiple weakness types or contextual project-level fields were removed, to prevent inconsistencies between sources from propagating into the merged dataset. Consequently, the classification process could focus solely on distinguishing defective functions from safe ones.

The emerging dataset further enriched the privacy focus by extending the annotation scheme. Each vulnerable function received not only a CWE identifier but also one or more LINDDUN privacy types, assigned through a combination of rule based mapping and manual review as described in Chapter 3.

4.2.4 Synthetic augmentation and coverage of rare weaknesses

Synthetic data generation was performed in two stages, both using OpenAI's gpt-3.5-turbo model via the OpenAI API, targeting different gaps in the merged real-world dataset.

Stage One targeted CWE categories identified as critically underrepresented or entirely absent from the merged DiverseVul/ReposVul corpus relative to the LINDDUN-to-CWE mapping developed in Chapter 3 (the categories listed in Table 4-2). For each targeted CWE, the model was prompted with a system role of "You are a security expert generating code vulnerability examples in C/C++" and a user prompt supplying the CWE identifier and description, requesting a paired vulnerable code example, a corresponding secure/fixed version, and a brief explanation, at temperature 0.7 with a maximum of 1,000 output tokens per call. This stage generated 19,600 candidate samples across 96 CWE categories (200 per category for 94 categories, 400 for two: CWE-190, CWE-200).

Stage Two addressed a separate concern which was several CWE categories already present in the merged real-world corpus were unevenly represented, risking the multi-class equivalent of the class-imbalance problem discussed in Section 2.11.3, where a small number of dominant CWE categories could disproportionately influence model training. This stage used the same underlying model with a distinct, more constrained prompt strategy, generating a uniform target of 20,408 candidate samples for each of 49 CWE categories (999,992 candidate samples total).

Candidate samples from both stages were subsequently passed through the same manual inspection and deduplication process applied to the wider merged corpus (Section 4.2.2), rather than being incorporated in full. This step removed candidates judged to be low-quality, non-compilable, unrepresentative of the target CWE, or duplicative of existing samples in the merged corpus or of one another. As with the exact-match deduplication described in Section 4.2.2, this process was not logged in a way that preserves the per-CWE candidate-to-retained mapping, so the exact number of synthetic samples from each stage that were ultimately retained in the released C3-VULMAP dataset cannot be reported separately from the dataset's overall composition (Table 4-4). The generation methodology, prompts, and per-category generation targets for both stages are fully specified above. What cannot be independently verified from the artefacts available is the precise number of synthetic samples, by CWE category, present in the final released dataset. Retaining an unambiguous synthetic-sample identifier through every stage of dataset construction, from generation through manual review to final merge, is identified as a requirement for any future release or extension of C3-VULMAP (Section 6.9).

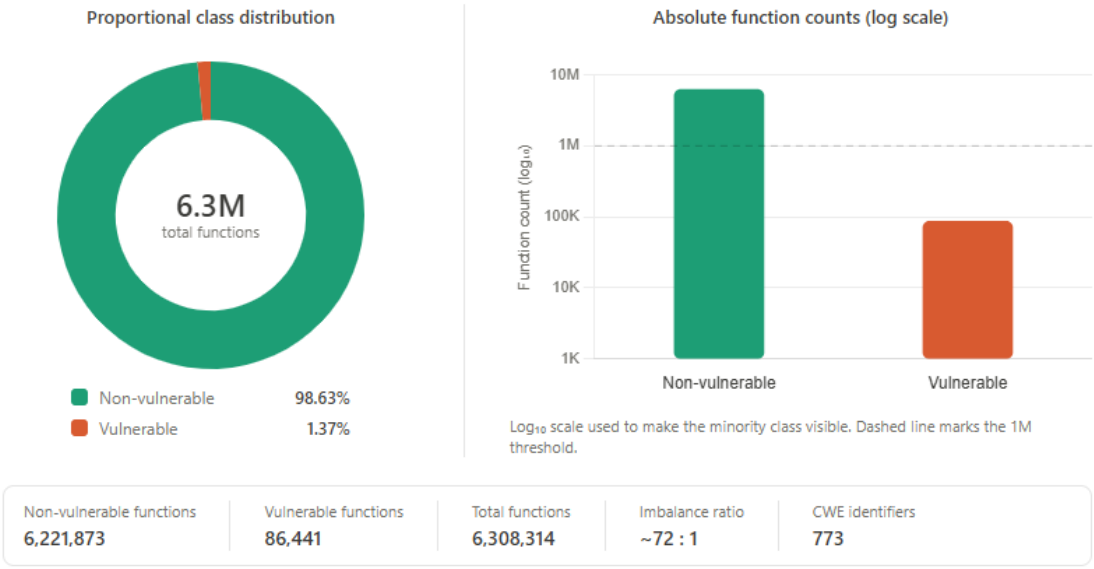


Figure 4-4: Class Imbalance Distribution in C3-VULMAP — Proportional Split (left) and Absolute Counts on Log Scale (right)

Table 4-2: CWEs with Missing or Few Representations in the Pre-Augmentation Merged Dataset

CWE ID	Name
CWE-322	Key Exchange without Entity Authentication
CWE-639	Authorization Bypass Through User-Controlled Key
CWE-640	Weak Password Recovery Mechanism for Forgotten Password
CWE-425	Direct Request (‘Forced Browsing’)
CWE-324	Use of a Key Past its Expiration Date
CWE-694	Use of Multiple Resources with Duplicate Identifier
CWE-841	Improper Enforcement of Behavioural Workflow
CWE-1046	Creation of Immutable Text Using Untrusted Input
CWE-628	Function Call with Incorrectly Specified Arguments
CWE-360	Trust of System Event Data
CWE-1235	Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations
CWE-1043	Data Element Aggregating an Excessively Large Number of Non-Primitive Elements
CWE-1048	Invokable Control Element with Large Number of Outgoing Calls
CWE-1092	Use of Same Variable for Multiple Purposes
CWE-1071	Empty Code Block
CWE-586	Explicit Call to Finalize()
CWE-1097	Duplicate Key in Associative List (e.g., Dictionary or Map)
CWE-1067	Excessive Execution Time for Operation
CWE-1104	Use of Unmaintained Third-Party Components
CWE-346	Origin Validation Error
CWE-1044	Architecture with Number of Horizontal Layers Outside of Expected Range
CWE-1049	Excessive Data Query Operations
CWE-117	Improper Output Neutralization for Logs (Log Injection)
CWE-654	Reliance on a Single Factor in a Security Decision

CWE ID	Name
CWE-924	Improper Enforcement of Message Integrity During Transmission in a Communication Channel

These CWE categories were identified as absent or critically underrepresented following the integration of DiverseVul and ReposVul sources and prior to the synthetic augmentation stage described in Section 4.2.4. They were the specific targets of the GPT-3.5 Turbo augmentation process.

The scale of synthetic augmentation applied to the target CWE categories was substantial. The CWE categories listed in Table 4-2 were absent or represented by fewer than ten functions in the pre-augmentation merged dataset; following augmentation, several of these categories including CWE-322, CWE-639, CWE-425, CWE-694, CWE-324, CWE-640, and CWE-841, became the most frequently represented vulnerability types in the final dataset, as shown in **Error! Reference source not found.**. This outcome reflects the targeted nature of the augmentation strategy, which was designed to elevate underrepresented privacy-critical CWE categories to a level sufficient for reliable model training rather than to preserve the natural frequency distribution of vulnerabilities in real-world codebases. The intermediate pre-augmentation counts were not separately archived during dataset construction, and a precise before/after comparison at the individual CWE level is therefore not available. What can be stated is that the top-ranked entries in Table 4-3 are primarily or substantially synthetic in origin, and this should be considered when interpreting model performance on these specific CWE categories. The potential for models to learn patterns characteristic of synthetically generated code rather than authentic vulnerability manifestations for these categories is acknowledged as a dataset quality limitation and is addressed in the discussion of limitations in Section 6.8, where ablation studies comparing model performance with and without synthetic samples are identified as a priority for future work.

Table 4-3: Top Twenty CWE Identifiers by Label Occurrence Count (spanning vulnerable and patched non-vulnerable functions)

Rank	CWE ID	Count	Percentage
1	CWE 322	73,992	1.17
2	CWE 639	60,341	0.96
3	CWE 425	54,057	0.86
4	CWE 694	53,988	0.86
5	CWE 324	53,483	0.85
6	CWE 640	53,361	0.85
7	CWE 841	53,197	0.84
8	CWE 1043	46,988	0.74

Rank	CWE ID	Count	Percentage
9	CWE 1085	40,753	0.65
10	CWE 1117	40,731	0.65
11	CWE 549	40,718	0.65
12	CWE 360	40,686	0.64
13	CWE 1235	40,628	0.64
14	CWE 628	40,621	0.64
15	CWE 1127	40,600	0.64
16	CWE 605	40,587	0.64
17	CWE 1087	40,570	0.64
18	CWE 1070	40,565	0.64
19	CWE 563	40,535	0.64
20	CWE 1114	40,525	0.64

4.2.5 Resulting Dataset Characteristics and Justification

The resulting dataset presents a large scale and privacy-oriented resource for vulnerability detection in healthcare software development. Its construction reflects an intentional balance between representativeness, granularity and suitability for supervised learning in static security analysis. An examination of the dataset structure, distributions and categorical patterns reveals several characteristics that justify its use in privacy aware code analysis.

The final dataset contains 6,308,314 C/C++ functions, each associated with a binary label indicating whether the function is vulnerable or non-vulnerable. As shown in Table 4-4, this representation preserves only the essential information required for classification tasks, including the function body, the assigned binary label and a reduced set of metadata fields that support privacy alignment. The dataset is therefore consistent with the design goal of enabling scalable supervised learning, while discarding the heterogeneous annotation structures inherited from the source datasets. This harmonised representation ensures uniformity across all samples and supports the development of robust models capable of learning vulnerability related patterns within code.

Two properties of the cwe_id field affect how CWE-level counts in this chapter should be interpreted. First, a patched, non-vulnerable function (label = 0) retains the CWE identifier of the vulnerability it resolves, rather than being left unlabelled. CWE counts therefore span both vulnerable and non-vulnerable functions, not vulnerable functions alone. Second, a

single function may carry more than one CWE identifier where multiple weaknesses were identified in the same function. As a result, per-CWE counts reported in Table 4-3 count CWE label occurrences across the full 6,308,314-function dataset, not instances of the 86,239 vulnerable functions specifically, and the sum of per-CWE counts exceeds both the vulnerable-function count and, for the same reason, cannot be directly compared to it. This distinction is maintained consistently in the discussion of CWE prevalence below.

Table 4-4: Core Dataset Schema Showing Harmonised Fields

Column Name	Description	dtypes
code	Extracted C or C plus plus function	object
label	Binary vulnerability indicator (1 = vulnerable, 0 = safe)	integer
cwe_id	CWE identifier(s) associated with the function; stored as a comma-separated list where a function maps to more than one CWE. Populated for both vulnerable functions (label = 1) and their corresponding patched, non-vulnerable functions (label = 0), since a patched function retains the CWE identifier of the vulnerability it resolves.	string
cwe_description	Brief CWE textual description	string
CWE Name	CWE canonical name	string
CWE Category	Parent CWE category code	object
CWE Category Name	Parent CWE category name	string
Privacy Threat Types	Mapped LINDDUN privacy threats	string
programming_language	Language identifier (entirely C or C plus plus)	object

A key characteristic of the dataset is its extreme class imbalance. As shown in Table 4-5, 98.63% of functions are labelled as non-vulnerable, while only 1.37% represent vulnerable instances. This distribution mirrors real world code bases in which vulnerable functions are vastly outnumbered by safe ones. Table 4-5 reports this binary label distribution only and it is independent of the CWE-level counts in Table 4-3, which are not restricted to the 86,239 vulnerable functions and should not be summed against or compared directly to the figures in this table. While this imbalance introduces methodological challenges, it also strengthens ecological validity because machine learning methods deployed in healthcare settings must operate under similar conditions. The imbalance further emphasises the necessity of integrating advanced resampling techniques and cost sensitive learning strategies at later stages of the classification pipeline.

Table 4-5: Label Distribution Across All 6.3 Million Functions

Label	Count	Percentage
0	6,222,075	98.63 percent
1	86,239	1.37 percent

The dataset consists solely of C/C++ code, reflecting their prevalence in medical devices and healthcare computing. Focusing on these languages helps eliminate confounding variables and supports privacy and security research by ensuring detected patterns relate directly to vulnerabilities. The dataset includes 773 unique CWE identifiers, covering a wide variety of vulnerability types. Table 4-3 lists the top twenty CWEs, which are mostly related to session management, access control, authentication, communication, and memory issues, key risks to health data confidentiality and integrity.

The frequency of the top 20 CWE identifiers within this healthcare-aligned dataset reveals patterns that directly correspond to the security and privacy requirements inherent in clinical information systems. These frequencies count CWE label occurrences across both vulnerable functions and their patched, non-vulnerable counterparts, and should be read as an indication of which weakness types are most represented in the dataset's labelling, not as a count of distinct vulnerable functions. These weaknesses are unevenly distributed across functional domains; moreover, many demonstrate direct associations with confidentiality violations, improper access control mechanisms, or inadequate data handling practices. Each of these vulnerabilities possesses the potential to disrupt clinical workflows and erode patient trust. Consequently, a detailed examination of high-frequency CWEs is essential to illuminate their particular significance within healthcare environments.

CWE 322: Key Exchange Without Entity Authentication

CWE 322 is the most frequently occurring CWE label within the dataset, spanning both vulnerable functions and the patched functions that resolve them. This vulnerability is introduced when cryptographic key exchanges occur without verifying the identities of communicating entities. In healthcare contexts, where sensitive medical data traverses distributed networks through telehealth platforms, imaging systems, and electronic health

records, the failure to authenticate during key exchange creates substantial risk. Specifically, adversaries can impersonate clinical devices, intercept diagnostic imagery, or inject falsified results, consequences that pose acute threats to both data confidentiality and the integrity of clinical decision-making.

CWE 639: Authorization Bypass Through User-Controlled Keys

CWE 639 encompasses vulnerabilities that permit privilege escalation via manipulated user inputs. Healthcare systems typically feature multiple user roles including clinicians, laboratory staff, administrators, and external contractors. Inasmuch as access control mechanisms frequently rely upon user-supplied identifiers, attackers may successfully impersonate higher-level accounts or access restricted patient records. Consequently, such breaches jeopardize privacy obligations and fundamentally undermine system integrity.

CWE 425: Direct Request and Forced Browsing

CWE 425 arises from insufficient restrictions on access to internal resources. In healthcare settings, this vulnerability can permit unauthorized parties to directly reach endpoints containing patient data, administrative dashboards, or diagnostic tools, systems that should be safeguarded by robust authentication layers. Such attacks risk exposing entire patient rosters or medical documents; therefore, the consequences directly breach confidentiality mandates and violate established legal protections for health data.

CWE 694: Use of Multiple Resources with Duplicate Identifier

CWE 694 arises when a system uses multiple resources that can share the same identifier in a context where unique identifiers are required. In healthcare systems, this can occur where patient records, session tokens, or device identifiers are not guaranteed unique, allowing the system to inadvertently operate on the wrong resource. Where an attacker can force two resources to share an identifier, they may bypass access controls that assume uniqueness, potentially retrieving or modifying a different patient's data than intended.

CWE 324: Use of a Key Past its Expiration Date

CWE 324 concerns the continued use of a cryptographic key after its defined expiration date. In healthcare contexts, where cryptographic keys protect data in transit between clinical

devices, telehealth platforms, and record systems, continued use of an expired key undermines the security guarantees the expiration policy was intended to enforce, potentially allowing decryption or impersonation using a key that should no longer be trusted.

CWE 640: Weak Password Recovery Mechanisms

CWE 640 indicates deficient account recovery processes for forgotten passwords. Healthcare institutions serve diverse user populations with varying levels of digital proficiency; consequently, substandard recovery procedures facilitate account hijacking through social engineering or brute-force methods. Compromise of such accounts carries serious ramifications, given their extensive access to confidential medical records, consent documentation, and clinical orders.

CWE 841: Improper Enforcement of Behavioural Workflows

CWE 841 relates to failures in mandating correct operational sequences within clinical applications. Standardized workflows guide safe clinical procedures and ensure appropriate clinical governance. Inadequate enforcement permits circumvention of validation steps, unauthorized order entry, or unintended data access. Therefore, such vulnerabilities endanger both patient safety and privacy simultaneously.

CWE 1043: Data Exposure Through Incomplete Sanitization

CWE 1043 encompasses cases where sensitive data is only partially removed or anonymized. Given the frequent processing of identifying details, including names, biometric information, and clinical observations, insufficient sanitization can leave residual traces in logs, exports, or temporary files. Consequently, such exposures enable patient reidentification and result in regulatory non-compliance.

CWE 1085: Invokable Control Element with Excessive Volume of Commented-out Code

CWE 1085 concerns functions or methods containing an excessive volume of commented-out code, which can obscure the current logic, complicate review, and in some cases leave disabled but reachable insecure code paths in place. In healthcare software undergoing frequent regulatory-driven revision, this weakness increases the risk that reviewers overlook residual insecure logic left commented out rather than fully removed.

CWE 1087: Class with Virtual Method without a Virtual Destructor

CWE 1087 arises in C++ code when a class defines virtual methods but lacks a virtual destructor, which can result in undefined behaviour, including resource leaks or incomplete object destruction, when an object is deleted through a base-class pointer. In healthcare device or record-management software using polymorphic class hierarchies, this can result in improper cleanup of sensitive in-memory patient data on object destruction.

CWE 1117: Callable with Insufficient Behavioral Summary

CWE 1117 concerns functions or methods that lack sufficient documentation of their behaviour, side effects, or preconditions, increasing the risk that a caller misuses the function in a way that introduces a vulnerability. In collaboratively developed healthcare software, insufficiently documented functions handling patient data increase the risk of privacy-relevant misuse by developers unfamiliar with a function's full behaviour.

CWE 1114: Inappropriate Whitespace Style

CWE 1114 concerns inconsistent or non-standard whitespace formatting within source code. While not a direct security weakness, inconsistent formatting can obscure logic during code review, increasing the likelihood that reviewers overlook substantive issues, including privacy-relevant ones, in poorly formatted sections of code.

CWE 549: Missing Password Masking

CWE 549 addresses interface flaws that reveal password characters during entry. While seemingly minor compared to cryptographic vulnerabilities, this issue presents considerable risk in shared clinical spaces where visible passwords can enable unauthorized access and undermine patient privacy safeguards. Given the prevalence of shared workstations in clinical environments, this vulnerability warrants serious consideration.

CWE 360: Trust of System Event Data

CWE 360 arises when a system trusts event or log data as an accurate record without verifying that it has not been manipulated. In healthcare systems, audit logs and system event records are often relied upon for compliance and incident investigation; if event data can be

forged or altered without detection, both regulatory audit trails and clinical incident investigations become unreliable.

CWE 1235: Incorrect Use of Autoboxing and Unboxing for Performance Critical Operations

CWE 1235 concerns the performance and correctness risks introduced by automatic conversion between primitive types and their object wrapper equivalents in performance-critical code paths. While not a direct confidentiality or access-control weakness, its presence in performance-sensitive healthcare processing pipelines, for example, real-time vital sign monitoring or high-throughput record processing, can degrade system responsiveness in ways that affect the timeliness and reliability of clinical data handling.

CWE 628: Function Calls with Incorrectly Specified Arguments

CWE 628 arises when a function is called with arguments that are incorrect in number, type, or order relative to what the function expects, whether due to a coding error or a mismatch introduced during refactoring or interface changes. Unlike a general variable-handling error, this weakness is specific to the call-site interface contract between a function and its caller. In healthcare software, where functions handling patient identifiers, dosage values, or clinical codes are frequently reused across modules, an incorrectly specified argument, for example, passing a patient identifier into a parameter position expecting a record type or a numeric dosage, can result in the wrong data being processed, retrieved, or disclosed without triggering an immediately visible failure.

CWE 1127: Exposure of Sensitive System Information

CWE 1127 reflects disclosure of system details that facilitate further exploitation. Within healthcare, such exposures may reveal integration points with laboratories, authentication services, or external health exchanges, thereby aiding attackers in mapping and targeting critical infrastructure.

CWE 605: Multiple Binds to the Same Port

CWE 605 arises when an operating system or application permits more than one socket to bind to the same network port, typically through options such as SO_REUSEADDR or SO_REUSEPORT. Where this is permitted without adequate safeguards, a second, potentially

malicious process can bind to a port already in use by a legitimate service and intercept traffic intended for it. In healthcare systems, where telehealth platforms, medical device interfaces, and record-system APIs communicate over fixed, well-known ports, this weakness could allow an unauthorised process on the same host to intercept patient data in transit between clinical components before it reaches its intended destination.

CWE 1070: Serializable Data Element Containing non-Serializable Item Elements

CWE 1070 arises when a data element marked as serializable contains one or more members that are not themselves serializable, which can cause serialization to fail or behave unpredictably at runtime. In healthcare systems that serialise patient records for storage, transmission, or interoperability exchange (for example, HL7 or FHIR payloads), this can result in incomplete or corrupted data being transmitted or persisted.

CWE 563: Assignment to Variable without Use

CWE 563 concerns variables that are assigned a value that is never subsequently used, often indicating dead code, an incomplete implementation, or a logic error. In privacy-sensitive code, an unused assignment involving patient data can indicate a data-handling path that was intended to sanitise, redact, or check the value but was never completed, leaving the original assignment as a latent risk.

Collectively, these CWEs represent a comprehensive landscape where security and privacy threats begin from weaknesses in access controls, cryptographic design, workflow enforcement, memory safety, and secure communications. Their prevalence mirrors longstanding vulnerabilities documented in healthcare platforms, electronic health record systems, and medical devices. Furthermore, the patterns identified herein suggest that effective vulnerability detection requires models capable of recognizing both technical manifestations and contextual implications specific to clinical environments. Understanding these top CWEs thus forms a robust theoretical and empirical foundation for developing machine learning models aimed at detecting vulnerabilities with direct consequences for patient confidentiality, clinical safety, and regulatory adherence. Such models must balance the need for high detection accuracy with the interpretability requirements inherent in

healthcare deployment contexts, where clinical stakeholders require transparent reasoning for security decisions.

The distribution of CWE parent categories reveals similar spread. The dataset includes 32 distinct CWE categories, with the largest category (Category 1006) accounting for 37.10**%** of all samples. This category alone contains 578 unique CWEs, indicating that the dataset captures a wide spectrum of sub types under major vulnerability families. As displayed in Table 4-6, several categories have more than three hundred unique CWEs, demonstrating significant diversity and complexity. Such diversity strengthens the ability of the dataset to support generalisable vulnerability detection models that can operate across multiple weakness types commonly exploited in health software ecosystems.

Table 4-6: Distribution of The Largest CWE Categories and Their Unique Subtypes

Category Code	Count	Percentage	Unique CWEs
1006	2,340,321	37.10	578
1211	324,151	5.14	344
255	303,641	4.81	327
417	296,910	4.71	332
3	273,527	4.34	326
8	267,291	4.24	328
19	259,014	4.11	320
137	258,294	4.09	326
1212	242,731	3.85	324
199	217,271	3.44	316

Another noteworthy characteristic is the distribution of LINDDUN privacy relevance categories assigned across the dataset. Every function, whether vulnerable or non-vulnerable, is labelled with the LINDDUN category associated with its CWE identifier under the mapping developed in Chapter 3. This label indicates that the function's associated CWE belongs to a privacy-relevant weakness category, not that the function itself has been observed to exhibit a privacy threat. A non-vulnerable, patched function inherits the LINDDUN label of the CWE it resolves, in the same way it inherits the CWE identifier itself (Section 4.2.4): the label describes the privacy relevance of the function's weakness category and does not assert that a threat is present or active in that function's current, patched state. Seven LINDDUN categories are represented: Non-compliance, Detectability, Identifiability, Linkability, Non-repudiation, Unawareness and Disclosure of Information. Their frequencies are relatively

balanced, ranging between 14.11% and 14.51% of the dataset, as shown in Table 4-7. This balanced distribution indicates that the dataset's privacy-relevant CWEs are spread evenly across the seven LINDDUN categories and confirms that the filtering and alignment processes described in Section 4.2.3 successfully isolated CWEs with privacy relevance across all seven categories, rather than concentrating on a narrow subset. It should not be read as evidence that a comparably large number of functions in the dataset actively exhibit privacy threats.

Table 4-7: Distribution of Mapped LINDDUN Privacy Threat Types

LINDDUN Privacy Relevance Category	**Count**	**Percentage**
Non-compliance	915,608	14.51
Detecting	913,335	14.48
Identifiability	907,726	14.39
Linkability	896,134	14.21
Non-repudiation	894,299	14.18
Unawareness	890,997	14.12
Disclosure of Information	890,215	14.11

Note: a LINDDUN category label indicates that the function's associated CWE falls within that category's privacy-relevant weakness taxonomy (Chapter 3, Section 3.2.5), not that the function has been observed to exhibit the corresponding privacy threat. This applies equally to vulnerable and non-vulnerable functions; a patched, non-vulnerable function retains the LINDDUN label of the CWE it resolves and should not be described as exhibiting a privacy threat.

Taken together, the dataset characteristics demonstrate a resource that is extensive in scale, rich in categorisation and aligned with real world healthcare privacy concerns. Its breadth of CWE coverage, balanced representation across LINDDUN privacy relevance categories, and inclusion of millions of real code samples establish a strong empirical foundation for vulnerability detection. These characteristics justify the central role of the dataset in the thesis. It supports objective two by providing a novel, healthcare specific cybersecurity dataset for NLP based privacy and vulnerability research. It also supports objective three by supplying a large and well-structured set of labelled functions on which to develop, train and evaluate static code vulnerability classifiers capable of assisting secure healthcare software development.

4.3 Dataset Preparation and Sampling Strategy

This study uses a dataset containing around 6.3 million C/C++ code samples, each labelled with one of 773 distinct Common Weakness Enumeration (CWE) classifications. This figure is independent of the 965 candidate CWEs screened during the LINDDUN threat-tree analysis in Chapter 3 (Section 3.2.5); the two datasets were constructed through separate processes and the overlap between them is not assumed. As one of the largest vulnerability detection datasets ever compiled for traditional machine learning, it far surpasses commonly referenced benchmarks like Draper VDISC and the SARD repository (Fan et al., 2020). Its large sample size allows for strong statistical analysis, and the wide range of CWE categories ensures thorough coverage of vulnerabilities found in healthcare software systems.

To address the class imbalance while preserving the integrity of vulnerable samples, a simple random undersampling approach was applied to the non-vulnerable class rather than synthetic oversampling techniques such as SMOTE. Specifically, all 86,239 vulnerable samples were retained in their entirety, and 258,717 non-vulnerable instances were selected at random using a fixed random seed to ensure reproducibility, establishing a 1:3 ratio between vulnerable and non-vulnerable classes. This sampling strategy produced a final training set of approximately 345,000 samples, sufficiently large for robust model training while substantially reducing the computational burden associated with processing the full 6.3 million sample corpus. The decision to undersample rather than oversample was motivated by concerns that synthetic sample generation may introduce artefacts that do not reflect genuine vulnerability patterns, particularly for security-critical applications where false positives carry significant operational costs (Feng et al., 2021; Zhao et al., 2023).

4.4 Experimental Setup

All experiments reported in this chapter were conducted on a dedicated research workstation configured specifically for the computational requirements of the multi-paradigm machine learning evaluation. The following subsections document the hardware platform, software environment, and dataset configuration in sufficient detail to support full reproducibility of all reported results.

4.4.1 Hardware Configuration

The hardware platform used across all experiments is summarised in Table 4-8. All traditional machine learning experiments were completed on the CPU. Graph neural network and transformer-based model training utilised the NVIDIA GeForce RTX 4070 Laptop GPU with 8 GB dedicated GDDR6 VRAM and CUDA 12.1 for accelerated computation. Mixed-precision training (FP16) was enabled for all transformer models via PyTorch Automatic Mixed Precision to reduce VRAM consumption and accelerate computation, with gradient scaling applied to prevent underflow in low-precision gradients. The 64 GB system RAM was essential for loading and processing the 6.3-million-function dataset corpus during preprocessing and the stratified sampling pipeline.

Table 4-8: Hardware Specification of the Experimental Platform

Component	**Specification**
Device Name	Enenche_SHU-NLP
Processor	13th Gen Intel Core i9-13900H @ 2.60 GHz (24 cores: 8P + 16E)
Installed RAM	64.0 GB DDR5 (63.6 GB usable)
GPU Model	NVIDIA GeForce RTX 4070 Laptop GPU
GPU Driver Version	32.0.15.7371
Dedicated GPU Memory	8 GB GDDR6 VRAM
Shared GPU Memory	39.8 GB (system-shared)
CUDA Version	12.1
Operating System	Windows 11 Pro, 64-bit, x64-based processor
Storage	NVMe SSD (local dataset and model storage)

4.4.2 Software Environment

The software stack was selected to ensure compatibility across all three experimental model families, traditional machine learning, graph neural networks, and transformer-based models, while maintaining reproducibility through fixed library versions. Table 4-9 documents all libraries and frameworks used. All random seeds were fixed at 42 across NumPy, PyTorch, and scikit-learn to ensure deterministic results were supported by the underlying hardware and operations. Experiments were managed through dedicated Python virtual environments to prevent dependency conflicts across the three families.

Table 4-9: Software Library Versions Used Across All Experimental Paradigms

Library / Framework	Version
Python	3.10.12
PyTorch	2.1.0+cu121
Hugging Face Transformers	4.35.2
Hugging Face Datasets	2.14.6
Tokenizers	0.15.0
PyTorch Geometric	2.4.0
tree-sitter (C/C++ parser)	0.20.4
XGBoost	2.0.1
scikit-learn	1.3.2
imbalanced-learn (SMOTE)	0.11.0
NumPy	1.26.1
pandas	2.1.2
SHAP	0.43.0
matplotlib	3.8.1
seaborn	0.13.0

4.4.3 Training Configuration Summary

Training hyperparameters for each experimental paradigm were designed to ensure fair cross-paradigm comparison through consistent data splitting and evaluation protocols. The stratified 72/8/20 train-validation-test split was applied identically across all fourteen models, with stratified sampling preserving class distributions across all partitions. Model-specific training configurations, including optimiser settings, learning rate schedules, early stopping criteria, batch sizes, and gradient management strategies, are documented in full within the relevant evaluation subsections: Section 4.5.1 for traditional machine learning, Section 4.5.2 for graph neural networks, and Section 4.5.3 for transformer-based models. All random seeds were fixed at 42 across NumPy, PyTorch, and scikit-learn to ensure deterministic results were supported by the underlying operations.

4.5 Dataset Evaluation

4.5.1 Traditional Machine Learning Approaches for Healthcare Code Vulnerability Detection

Traditional machine learning methods are still useful for finding vulnerabilities in source code, especially in healthcare, because they are easy to understand, efficient, and work well with healthcare regulations. These models, like random forests and support vector machines, provide clear decision-making processes, which is important for transparency and

compliance. In contrast, deep learning often acts as a “black box”, making it harder for security teams to explain results.

The selection of traditional machine learning approaches as baseline methods for vulnerability classification is justified by several considerations specific to the healthcare cybersecurity domain. First, the healthcare domain demands that selected models generate not only accurate predictions but also actionable insights into vulnerability characteristics. Traditional ML algorithms facilitate feature importance analysis, revealing which code metrics and patterns contribute most significantly to detection. This transparency supports refinement of secure coding practices and developer training programmes within healthcare software teams (Hou et al., 2023).

Second, traditional machine learning algorithms achieve competitive performance on datasets of moderate size because they require fewer parameters and less computational complexity during training. While the 345,000-sample training set represents a substantial corpus, it remains modest relative to the millions of samples typically required for effective deep learning model training. The interpretable nature of traditional approaches additionally facilitates debugging and error analysis, enabling practitioners to understand why specific code samples are misclassified and to refine feature engineering accordingly (Zagane et al., 2020).

Third, the comparative analysis establishes baseline performance metrics that subsequent deep learning and graph-based approaches must exceed to justify their increased computational complexity and reduced interpretability. Healthcare organisations selecting vulnerability detection approaches must balance detection accuracy against interpretability requirements, computational resource availability, and integration constraints within existing development workflows. Traditional machine learning approaches provide pragmatic solutions that satisfy these competing constraints while enabling ongoing refinement as healthcare-specific vulnerability datasets expand through continued secure development initiatives.

Healthcare systems, such as electronic health records and patient management platforms, need quick vulnerability checks during code deployment. Traditional ML models

are faster and use less memory, making them practical for environments with limited resources and no specialised hardware.

4.5.1.1 Feature Engineering

Feature engineering plays a vital role in conventional machine learning pipelines, as the relevance and quality of the features extracted can significantly impact model effectiveness. For detecting vulnerabilities in source code, a hybrid feature approach that merges Term Frequency-Inverse Document Frequency (TF-IDF) vectorization with metrics tailored to programming code was implemented. This combination allows for capturing both lexical cues, such as patterns commonly found in vulnerable code, and structural characteristics connected to security issues (Kalouptsoglou et al., 2022). This vectorization process was configured to pull 5,000 features utilizing unigrams and bigrams (n-grams ranging from 1 to 2), with a minimum document frequency of 5 and a maximum of 0.95. These settings were chosen to ensure broad vocabulary representation while minimizing noise, so exceedingly rare tokens or common stop words do not dominate the feature set. The resulting sparse matrix highlights lexical patterns like risky function invocations, sequences of memory allocation, and input handling constructs that often appear in vulnerable code. TF-IDF has proven effective in code analysis by weighting tokens based on their distinguishing value across datasets, emphasizing code elements that separate vulnerable examples from secure ones (Kalouptsoglou et al., 2022).

Beyond lexical data, 15 code metrics reflecting various aspects of source code structure and style were derived. These metrics fall into three main groups. First, structural metrics (e.g., line count, character count, token count, average and maximum line lengths) help describe the complexity and scale of code samples. Second, style-related metrics (example given, ratios of blank lines and comments, estimated brace depth, semicolon counts) offer insights into coding practices and organization. Third, security-focused metrics for C/C++ code track pointer use, malloc/free calls, strcpy, sprintf, and gets function usage—features frequently linked to vulnerability categories like buffer overflows (CWE-120), use-after-free bugs (CWE-416), and format string flaws (CWE-134) (Croft et al., 2023). Eeven so, to standardize input for further processing, all numeric features underwent scaling using StandardScaler, ensuring fair contribution across distance-based algorithms and gradient

descent methods. Finally, the TF-IDF sparse matrix was concatenated with the standardized metrics, creating a combined feature set for traditional machine learning classifiers. This comprehensive representation enables models to utilize both detailed lexical signals from TF-IDF and higher-level structural information offered by code metrics.

4.5.1.2 Binary Classification Models

Five established machine learning algorithms were selected for binary vulnerability classification, each belonging to a distinct algorithmic family and characterised by differing assumptions regarding data distributions and decision boundaries. The selection process prioritised algorithms with proven records in text classification and security-related tasks, while maintaining diversity in methodological approaches to facilitate robust comparative analysis.

4.5.1.2.1 Support Vector Machine with Linear Kernel

The Linear Support Vector Classifier (LinearSVC) determines optimal hyperplanes to distinguish vulnerable code samples from non-vulnerable ones within a high-dimensional TF-IDF feature space. This implementation incorporated balanced class weights, automatically adjusting the regularisation parameter in accordance with class frequencies to reduce the impact of class imbalance on the decision boundary. A maximum iteration count of 10,000 was used to ensure convergence on extensive datasets. Linear SVMs are well-suited for high-dimensional, sparse data typical of text vectorisation, efficiently identifying key discriminative features without incurring the computational costs associated with kernel transformations (Bilgin et al., 2020).

4.5.1.2.2 Support Vector Machine with Radial Basis Function Kernel

To account for possible nonlinear relationships between code features and vulnerability labels, an SVM with a Radial Basis Function (RBF) kernel was also employed. The RBF kernel enables nonlinear mapping of samples into higher-dimensional spaces, facilitating the detection of intricate decision boundaries not present in the original feature set. Due to the quadratic computational complexity relative to sample size, this model was trained on a subsample of 20,000 instances, randomly selected with preserved class proportions. Probability estimates were enabled to support AUC-ROC assessment. While the RBF kernel

can reveal complex interactions that linear models do not capture, it does so at the expense of interpretability (Park et al., 2023).

4.5.1.2.3 Random Forest Ensemble

Random Forest constructs ensembles of decision trees, each tree trained on random subsets of features and bootstrap samples derived from the training set. The ensemble aggregates predictions through majority voting, thus reducing variance and enhancing generalisation compared to single decision trees. In this study, 200 trees were utilised, with depth limited to 20 levels, a minimum of 5 samples required for node splits, and at least 2 samples per leaf node. Balanced class weights ensured equitable treatment of the minority vulnerable class. Random Forests offer significant advantages for vulnerability detection, requiring minimal hyperparameter tuning, accommodating both numerical and categorical data without explicit encoding, and providing feature importance metrics to elucidate key discriminators between vulnerable and non-vulnerable code samples (Mehmood et al., 2023).

4.5.1.2.4 Logistic Regression

Logistic Regression provides a probabilistic framework for binary classification, estimating the likelihood of vulnerability based on engineered code features. L2 (Ridge) regularisation was applied to constrain coefficient magnitudes and mitigate overfitting, with balanced class weights addressing class imbalance. The LBFGS solver was chosen for its effectiveness with large-scale datasets and L2 penalties. Despite being less complex than ensemble or kernel methods, logistic regression remains useful for vulnerability identification due to its direct probability outputs, straightforward interpretation of coefficient values in relation to vulnerability risk, and proven resistance to overfitting in moderately sized datasets. For example, negative coefficients associated with input validation functions indicate that explicit validation corresponds to a reduced likelihood of vulnerabilities, affirming secure coding practices (Collins et al., 2021).

4.5.1.2.5 XGBoost Gradient Boosting

XGBoost is a highly optimised gradient boosting framework that incrementally builds ensembles of weak learners, generally shallow decision trees, each correcting residual errors of its predecessors. In this application, 200 estimators were used, with scale_pos_weight set

according to class imbalance to properly weight vulnerable instances during training. XGBoost integrates regularisation, parallel processing, and advanced missing value handling, making it especially effective for large-scale vulnerability classification problems. Its sequential refinement allows for capturing complex feature interactions indicative of vulnerabilities and often yields state-of-the-art performance on structured data classification benchmarks (Khleel & Nehéz, 2023; Nadim & Roy, 2023).

4.5.1.3 Multi-class CWE Classification

Beyond binary vulnerability detection, the fine-grained classification of vulnerability types provides actionable intelligence for remediation prioritisation. Therefore, the extension of evaluation to multi-class classification targeting specific CWE categories was made. To ensure sufficient training examples for each class, the top 25 CWEs were selected based on a minimum threshold of 100 samples per category. This selection captures the frequently occurringvulnerability types while excluding extremely rare CWEs for which reliable classification would be infeasible.

Three algorithms were evaluated for multi-class classification. Random Forest was configured identically to the binary classification setup, inherently supporting multi-class prediction through per-tree class probability estimation and majority voting aggregation. Logistic Regression employed the multinomial loss function with L2 regularisation, extending the binary formulation to handle multiple classes through the softmax function. XGBoost utilised the multi:softprob objective function, which outputs class probability distributions enabling both hard classification decisions and confidence estimation. The multi-class evaluation enables assessment of whether traditional ML approaches can distinguish between specific vulnerability types, information that directly supports targeted remediation and developer training programmes (Bahaa et al., 2024; Thapa et al., 2022).

4.5.1.4 Evaluation Protocol

A comprehensive evaluation methodology is fundamental for producing reliable performance estimates and facilitating meaningful comparisons among algorithms. This study utilizes an evaluation protocol that integrates hold-out testing with cross-validation to assess both generalization and model stability.

The dataset was divided through an 72/8/20 stratified train-validation-test split, ensuring class distributions remained consistent across training and evaluation sets. Stratification is especially critical for addressing class imbalance, as random splits may result in test sets with significantly different class proportions than the training data, which could lead to unreliable performance estimates. The reserved 20% hold-out test set was exclusively used for final evaluation and did not influence any aspect of model development or hyperparameter selection. This split operates at function level following exact-match deduplication; it does not enforce repository-level separation, meaning near-duplicate or semantically similar functions originating from the same source project cannot be ruled out from appearing in both partitions. The implications of this design choice for interpreting reported performance figures are discussed in Section 6.8.

Binary classification metrics included accuracy, precision, recall, F1-score, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC). While accuracy reflects overall correctness, it is insufficient for imbalanced datasets where predicting the majority class yields deceptively high results. Precision measures the proportion of correctly identified vulnerabilities, which is essential for reducing false positive alert fatigue within development teams. Recall quantifies the proportion of actual vulnerabilities detected by the model, directly addressing risks associated with missed vulnerabilities. F1-score offers a balanced assessment as the harmonic mean of precision and recall, and AUC-ROC provides an aggregate metric unaffected by threshold selection, indicating the effectiveness of distinguishing between vulnerable and non-vulnerable samples (Gao et al., 2022).

For multi-class CWE classification, both macro-averaged and weighted-average metrics were reported. Macro averaging calculates metrics independently for each class and averages them without weighting, assigning equal importance to all types of vulnerabilities. Weighted averaging, conversely, accounts for the relative frequency of each class, yielding an estimate more representative of the overall distribution. Five-fold stratified cross-validation was applied for multi-class evaluation to obtain robust performance estimates alongside uncertainty quantification.

To determine whether observed performance differences reflect genuine improvements or statistical artefacts, pairwise McNemar's tests were conducted on binary classification

predictions. The McNemar test evaluates the number of instances misclassified by one model but correctly classified by another, accounting for prediction correlation inherent in models applied to the same test set. This approach ensures rigorous evidence for claims regarding algorithmic superiority and mitigates over-interpretation of minor variations (Li et al., 2023).

The experimental process generated extensive outputs for both quantitative assessment and qualitative analysis of model behaviour. Performance metrics for binary and multi-class tasks were exported as CSV files to facilitate systematic comparison and statistical review. Receiver Operating Characteristic curves visualised the trade-off between true and false positive rates across decision thresholds, while confusion matrices highlighted patterns of misclassification for potential model enhancements. Performance comparison charts summarised algorithmic differences across key metrics.

Following model training, McNemar significance test outcomes were exported, documenting statistical evidence for performance comparisons among the five traditional ML algorithms. The test results confirmed that performance differences between the top-ranked models were statistically meaningful rather than attributable to random variation, providing a rigorous basis for the model selection decisions presented in the results section.

Additionally, another observation from the evaluation merit explicit discussion. The calibration curves in Figure 4-5 reveal meaningful differences in probability reliability across models that aggregate F1 scores alone do not capture. XGBoost and SVM RBF exhibit the closest alignment to the perfectly calibrated diagonal, while Random Forest and Logistic Regression display notable underconfidence across mid-range predicted probabilities, outputting conservative probability estimates that do not accurately reflect true positive rates. In healthcare deployment contexts where confidence scores inform clinical decision support workflows or risk prioritisation, calibration quality is as operationally important as classification accuracy, and these findings therefore carry direct implications for the model selection rationale developed in Section 4.7.5.

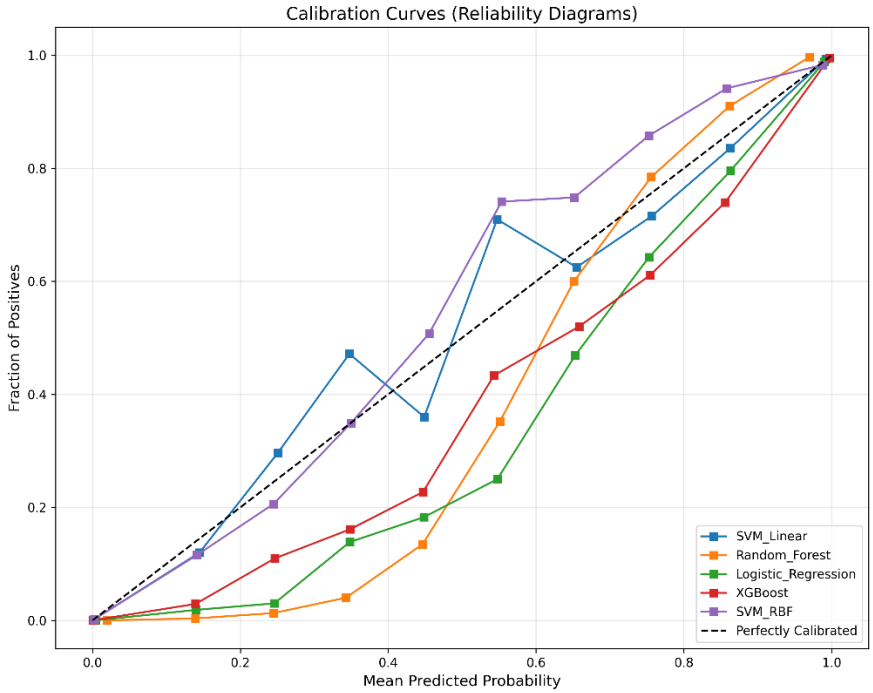


Figure 4-5: Calibration Curves (Reliability Diagrams) for Five Traditional ML Models Relative to the Perfectly Calibrated Baseline

4.5.1.5 Per-CWE Error Analysis

Aggregate metrics reported above (

Table 4-10) can conceal systematic weaknesses that only become visible when performance is examined at the level of individual CWE categories. This section reports precision, recall, and F1 for each of the five traditional ML models broken down along five dimensions: individual CWE, memory-safety/cryptographic/access-control weakness classes, rare versus common CWE frequency, and LINDDUN privacy-relevance category. Analysis was performed by reloading each model's saved checkpoint and re-running inference (not retraining) on the reconstructed held-out test set (6,932 functions), with CWE category derived from the cwe_id field and LINDDUN category derived from a 287-CWE mapping matrix. A CWE not present in this mapping is reported explicitly as unmapped rather than silently excluded (0.6% of test functions fell into this category, discussed below).

4.5.1.6 Rare versus common CWEs.

This is the clearest and most consistent finding across all five models. Every model shows a measurable performance gap between CWEs with fewer than 100 test-set occurrences ("rare") and those with more ("common"):

Table 4-10: Rare versus Common CWE Performance Gap Across Traditional ML Models, Measured as F1-Score

Model	Rare CWE F1	Common CWE F1	Gap
XGBoost	0.9825	0.9928	0.0103
SVM (Linear)	0.9681	0.9901	0.0220
Logistic Regression	0.9665	0.9855	0.0190
Random Forest	0.9439	0.9794	0.0355
SVM (RBF)	0.9304	0.9779	0.0475

The gap is modest for the two best-performing models but grows to nearly 5 percentage points for SVM (RBF), the weakest traditional model overall. At the individual-CWE level, the effect is more pronounced. Here, CWE-279 (Incorrect Execution-Assigned Permissions), with only 2 test-set occurrences, achieved a mean F1 of 0.667 across all five models, the single CWE category in the worst-performing 10 for every model without exception. This is a direct, reproducible illustration of the risk the examiner identifies: models trained on this dataset can be trusted substantially less on CWE categories with few training examples, and this should not be papered over by the near-perfect aggregate F1 scores (>0.99) reported elsewhere in this chapter.

4.5.1.7 Memory-safety, cryptographic, and access-control weakness classes.

All three classes perform strongly and comparably across models (F1 range 0.96–0.99), with no class showing a systematic, model-consistent weakness of the kind seen for rare CWEs. Access-control weaknesses show marginally lower F1 for Random Forest (0.9629) and SVM RBF (0.9677) relative to the other two classes, but this is a smaller and less consistent effect than the rare-versus-common gap.

4.5.1.8 LINDDUN privacy-relevance categories.

Performance is similarly high and largely uniform across all mapped categories (F1 0.97–0.99 depending on model), indicating no single privacy category is systematically harder for these

models to detect. One property of this breakdown should be noted. The Disclosure of Information and Unawareness categories returned identical results for every model. This is a confirmed property of the underlying LINDDUN-to-CWE mapping rather than an artefact of this analysis. Disclosure of Information and Unawareness are assigned the same 65 CWEs in the mapping matrix, so results for these two categories are necessarily identical under any model. Whether this reflects genuine conceptual overlap between the two threat categories at the CWE level or would benefit from further differentiation in a future refinement of the mapping, is noted as an observation for the Chapter 3 mapping methodology rather than a limitation of the error analysis itself. The 0.6% of test functions with no available LINDDUN mapping showed somewhat lower and more variable performance (F1 0.92–0.96), though the very small sample size (42 functions) limits how much weight this observation can bear.

4.5.2 Graph Neural Network Approaches for Vulnerability Detection

Traditional machine learning techniques offer interpretable baselines for vulnerability detection, but they primarily process source code as linear sequences of tokens, disregarding the complex structural information inherent in program representations. Graph Neural Networks (GNNs) overcome this limitation by analysing graph-structured data, allowing for the capture of intricate relationships among code elements that sequential models cannot address. This section details the implementation and assessment of two leading GNN architectures for vulnerability detection, Devign and Reveal, using the same C/C++ dataset as in previous machine learning experiments to enable direct comparative evaluation.

Graph-based approaches stem from the recognition that source code inherently exhibits graph structure at various levels of abstraction. AST encapsulate hierarchical syntax, Control Flow Graphs illustrate potential execution paths, and Data Flow Graphs map the movement of values between definitions and uses. The Code Property Graph, introduced by Yamaguchi, Golde, Arp, and Rieck (2014), integrates these representations into a unified multigraph framework, which has become a foundational tool for learning-driven vulnerability detection. Their influential study showed that graph-based pattern matching could uncover previously unidentified vulnerabilities within the Linux kernel, affirming the suitability of structural code analysis for security purposes.

Zhou, et al (2019) pioneered deep learning applications for Code Property Graphs with the introduction of the Devign model. Devign demonstrated that Gated GNN can effectively learn vulnerability patterns from composite graph representations, significantly outperforming both traditional static analysis tools and sequence-based deep learning techniques. Building upon this work, Chakraborty, et al (2022) critically reviewed existing deep learning vulnerability detection methods, identifying key methodological limitations and introducing the Reveal model, which employs advanced training strategies and attention-based graph pooling. These models currently represent the state of the art in GNN-driven vulnerability detection and constitute the foundation of the experimental analysis discussed in this section.

4.5.2.1 Dataset Preparation and Graph Construction

To ensure consistency with the traditional machine learning experiments and enable direct performance comparison, the GNN models were trained on the same sampled dataset of approximately 345,000 C/C++ code samples. The identical train-validation-test split ratios were preserved, with 72% of samples allocated to training, 8% to validation, and 20% to the held-out test set. Stratified sampling ensured that class distributions remained consistent across all partitions, which proves particularly important given the inherent class imbalance in vulnerability detection datasets where non-vulnerable samples substantially outnumber vulnerable instances.

The transformation of raw source code into graph representations constitutes a critical preprocessing step that fundamentally influences model performance. Each code sample was parsed using the Tree-sitter C/C++ parser, which provides a lightweight yet robust mechanism for generating Abstract Syntax Trees from source code. Tree-sitter employs an incremental GLR parsing algorithm that handles ambiguous grammar efficiently, making it well-suited for processing large-scale codebases with diverse coding styles and potential syntax variations (Alon, et al, 2019). Compared to heavyweight static analysis tools such as Joern, Tree-sitter offers substantially faster parsing times while maintaining sufficient structural fidelity for vulnerability detection applications.

As seen in Figure 4-6, the graphs were constructed from parsed ASTs by integrating four edge types: AST child edges (capturing code structure), next-token edges (maintaining token order), data flow edges (linking variable definitions and uses), and control flow edges (connecting control structures to depict execution paths). This multi-relational format approximates a full Code Property Graph while enabling efficient large-scale training (Li et al., 2021).

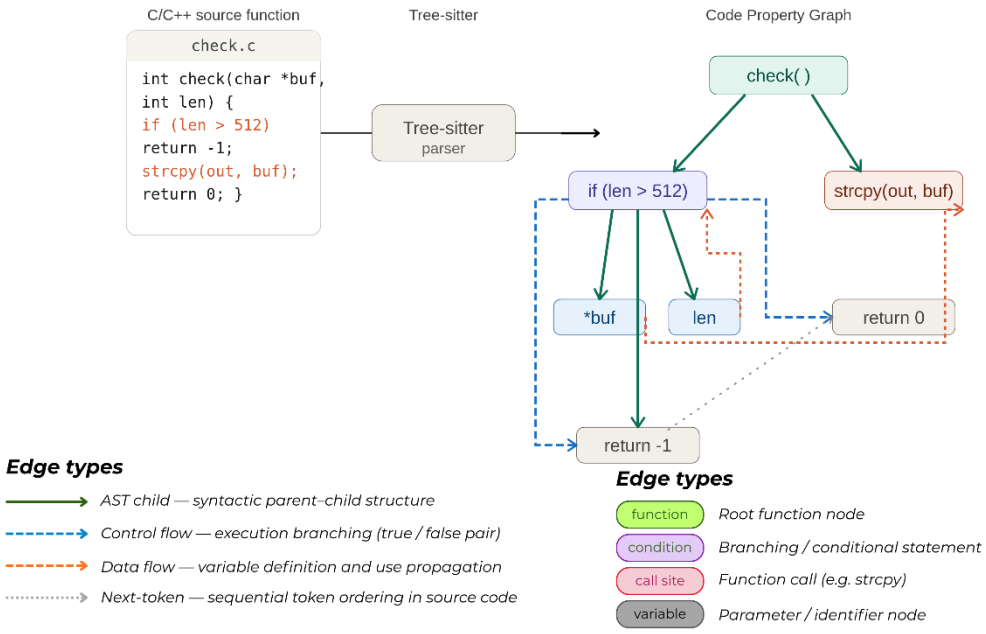


Figure 4-6: Code Property Graph construction — C/C++ source function to multi-edge graph via Tree-sitter

Furthermore, to manage the available computational resources, graphs were limited to 500 nodes since most functions in the dataset fall below this threshold, with outliers usually being auto generated or initialization code. Data flow edges were estimated by marking the first appearance of a variable as its definition, and subsequent occurrences as uses. Control flow approximations linked to control statements sequentially, capturing 80-90% of structural information compared to comprehensive tools like Joern, while reducing processing time significantly (Hin, et al, 2022).

In addition, each node has three features: its AST node type index, its mapped token index, and its depth within the AST hierarchy. This feature set efficiently encodes both syntax and semantics for downstream analysis (Allamanis, et al, 2018).

Afterwards, a vocabulary was constructed from the training corpus encompassing both node types and code tokens, with a maximum vocabulary size of 50,000 entries to balance coverage against embedding table size. Tokens appearing fewer than five times in the training corpus were mapped to a special unknown token to prevent the model from overfitting to rare identifiers. Additionally, a padding token was introduced to handle variable-length graphs within minibatches. Each discrete feature is mapped to a dense vector representation through learnable embedding layers, with an embedding dimension of 128 selected based on preliminary experiments. The three embedding vectors corresponding to node type, token, and depth are concatenated to form the initial node representation, which is subsequently refined through message passing in the graph neural network layers. This learned embedding approach offers advantages over fixed representations such as one-hot encoding, as the model can discover semantic similarities between related tokens and node types during training (Alon et al., 2019).

4.5.2.2 Training Configuration and Evaluation Protocol

The training procedure was meticulously developed to address key challenges in vulnerability detection, including class imbalance and overfitting to dataset-specific artefacts. Both models used the Adam optimizer with an initial learning rate of 1×10^{-4} and a weight decay of 1×10^{-5} for L2 regularization. The learning rate was dynamically adapted using the ReduceLROnPlateau scheduler, which reduces the rate by a factor of 0.5 if validation performance does not improve over five consecutive epochs. This adaptive approach facilitated swift early progress and allowed for fine-tuned optimization as the model converged. To mitigate the risk of gradient explosion during processing of large graphs with numerous message-passing iterations (Ying et al., 2018), gradient clipping with a maximum norm of 1.0 was applied.

To address the pronounced class imbalance typical in vulnerability detection datasets, balanced class weights inversely proportional to class frequencies were integrated into the cross-entropy loss function. This method increased penalties for misclassifying minority vulnerable classes, promoting balanced model performance across all classes rather than defaulting to non-vulnerable predictions. Recent empirical studies have highlighted the significance of class balancing strategies, noting that models trained without these measures

show substantially reduced recall on vulnerable samples while presenting artificially high overall accuracy (Zhao et al., 2023).

Training was conducted for up to 50 epochs with early stopping based on validation of F1-score, using a patience threshold of 10 epochs. The F1-score was chosen over accuracy or loss as the early stopping criterion, given its capacity to provide a balanced assessment of precision and recall, a crucial element in handling imbalanced classification problems. A minibatch size of 64 graphs was selected to balance training stability with memory constraints arising from variable-sized graph representations. Model checkpoints were preserved whenever validation metrics improved, and the optimal checkpoint was restored for final evaluation on the held-out test set. This protocol adheres to recognized best practices for deep learning-based vulnerability detection and ensures equitable comparison between the Devign and Reveal architectures under uniform training conditions (Fu & Tantithamthavorn, 2022).

The evaluation protocol was designed to match those used for traditional machine learning models, allowing direct performance comparisons between different algorithm types. Binary classification results were measured using accuracy, precision, recall, F1-score, and Area Under the Receiver Operating Characteristic Curve (AUC-ROC). Together, these metrics provide a comprehensive view of model performance relevant to practical vulnerability detection.

4.5.3 Transformer-Based Approaches for Vulnerability Detection

The transformer architecture has fundamentally transformed natural language processing since its introduction by Vaswani et al. (2017), and its application to source code analysis represents a significant paradigm shift in vulnerability detection research. Unlike graph neural networks that require explicit construction of code property graphs, transformer-based models operate directly on raw source code sequences, treating code as a form of natural language amenable to the same attention mechanisms that have proven remarkably effective for text understanding. This section presents a comprehensive evaluation of seven transformer-based models for vulnerability detection, spanning general-purpose language models, code-specialised pre-trained models, and encoder-decoder

architectures, all applied to the same C/C++ dataset utilised in the traditional machine learning and GNN experiments to facilitate direct cross-paradigm comparison.

The fundamental advantage of transformer-based approaches lies in their ability to capture long-range dependencies through self-attention mechanisms, which compute pairwise interactions between all tokens in a sequence regardless of their positional distance. This capability proves particularly valuable for vulnerability detection, where the manifestation of a security flaw often depends on complex relationships between distant code elements such as variable declarations, control flow decisions, and function calls. Traditional recurrent neural networks process sequences incrementally and struggle to maintain information across long distances, whereas transformers can directly model dependencies spanning hundreds of tokens, limited only by their context window size (Devlin et al., 2019).

The pre-training and fine-tuning paradigm have emerged as the dominant approach for applying transformers to downstream tasks, whereby models are first trained on massive corpora to learn general representations before being adapted to specific applications through task-specific fine-tuning. For code understanding, this approach has yielded remarkable results, with models such as CodeBERT achieving state-of-the-art performance across diverse tasks including code search, code summarisation, and vulnerability detection (Feng et al., 2020). The transfer of knowledge from pre-training to vulnerability detection enables models to leverage semantic understanding acquired from millions of code examples, substantially reducing the data requirements for achieving competitive performance on security-specific tasks. Furthermore, the LineVul model demonstrated that transformer-based approaches can substantially outperform graph neural network methods on vulnerability detection benchmarks, achieving F1-scores 160-379% higher than prior state-of-the-art approaches (Fu & Tantithamthavorn, 2022).

4.5.3.1 Models Training and Evaluation Protocol

Seven transformer-based models were selected for evaluation (Figure 4-7), representing three distinct categories based on their pre-training domain and architectural design. The first category included general-purpose natural language models that served as baselines for

assessing the benefits of code-specific pre-training. BERT-base-uncased, with approximately 110 million parameters, represents the foundational bidirectional encoder architecture that revolutionised natural language understanding through masked language modelling and next sentence prediction objectives (Devlin et al., 2019). DistilBERT-base, containing approximately 66 million parameters, provides a knowledge-distilled variant that retains 97% of performance of BERT while reducing model size by 40%, offering insights into the trade-off between model capacity and computational efficiency. DistilRoBERTa-base, with approximately 82 million parameters, applies the same distillation approach to the RoBERTa architecture, which improved upon BERT through optimised pre-training procedures including dynamic masking and removal of the next sentence prediction objective (Sanh, Debut, Chaumond, & Wolf, 2019).

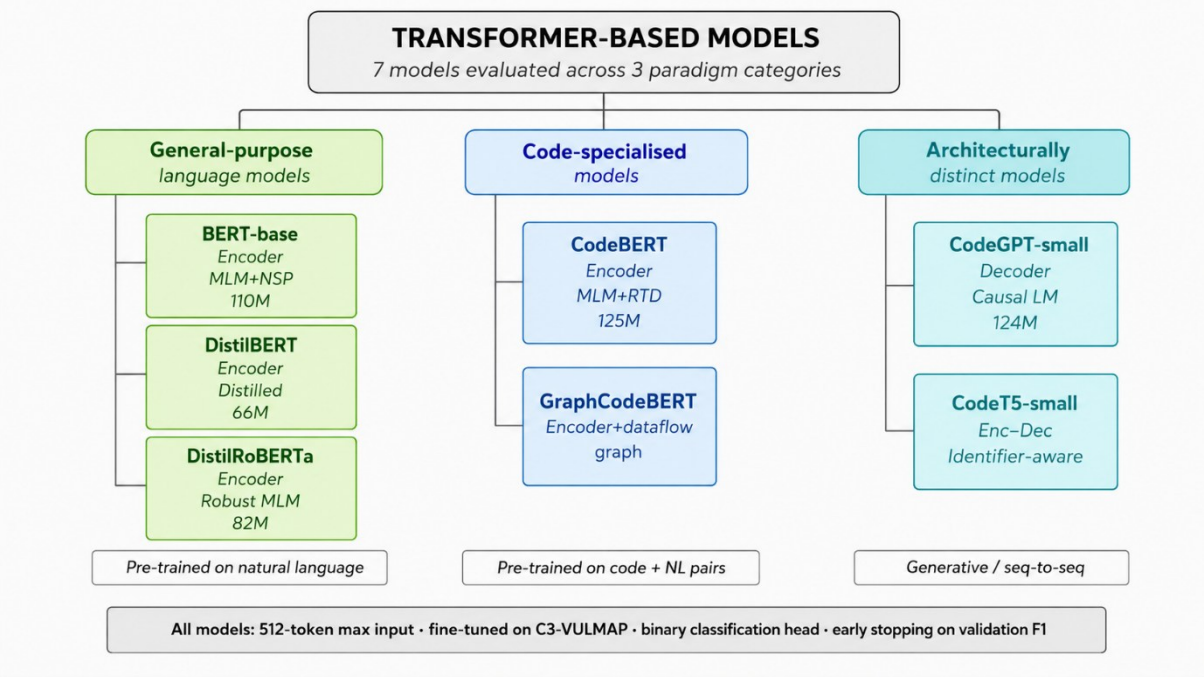


Figure 4-7: Taxonomy of the seven transformer-based models evaluated for binary vulnerability detection across three architectural paradigms

The second category had models specifically pre-trained on programming languages, which form the primary focus of this evaluation. CodeBERT-base, developed by Microsoft Research with approximately 125 million parameters, extends the RoBERTa architecture through pre-training on six programming languages including Python, Java, JavaScript, PHP, Ruby, and Go, using both masked language modelling and replaced token detection objectives (Feng et al., 2020). This bimodal pre-training on natural language and programming language

pairs enables CodeBERT to capture semantic relationships between code and documentation, which proves valuable for understanding code intent and identifying deviations from expected behaviour. GraphCodeBERT-base, also containing approximately 125 million parameters, further extends CodeBERT by incorporating data flow information during pre-training, enabling the model to learn representations that capture how values propagate through programs (Guo et al., 2021). This structural awareness distinguishes GraphCodeBERT from purely sequence-based models and has demonstrated improved performance on tasks requiring understanding of variable dependencies and data transformations.

While the third category includes architecturally distinct models that provide alternative approaches to code understanding. CodeGPT-small, with approximately 124 million parameters, employs a decoder-only architecture following the GPT paradigm, pre-trained on code generation tasks using autoregressive language modelling. Unlike encoder-based models that process entire sequences bidirectionally, CodeGPT generates representations sequentially from left to right, which may capture different aspects of code structure relevant to vulnerability detection. CodeT5-small, containing approximately 60 million parameters, utilises an encoder-decoder architecture inspired by the T5 model, pre-trained on multiple code understanding and generation tasks including identifier-aware denoising (Wang et al., 2021). This architecture enables CodeT5 to handle both code understanding and code generation within a unified framework, with the encoder producing contextualised representations suitable for classification tasks while the decoder supports sequence-to-sequence applications.

All the architectures of the selected transformer models can be found in Appendix 8.5.

4.5.3.2 Input Representation and Processing

A significant advantage of transformer-based approaches compared to graph neural networks is the simplified input pipeline, as models operate directly on raw source code text without requiring explicit graph construction or parsing. Each code sample was tokenised using the model-specific tokenizer provided with the pre-trained weights, which employs subword tokenisation algorithms such as Byte-Pair Encoding or WordPiece to handle the diverse vocabulary of programming languages including keywords, identifiers, operators, and

literals. This subword approach enables models to represent out-of-vocabulary tokens through compositions of known subword units, which proves particularly important for code analysis where identifier names exhibit high variability across projects and coding styles (Karampatsis et al, 2020).

The maximum sequence length was set to 512 tokens, which represents the standard context window for BERT-based models and their derivatives. Code samples exceeding this length were truncated, with the truncation applied from the end of the sequence to preserve the function signature and initial code structure that often contain critical vulnerability-relevant information. Padding was applied using the max_length strategy to ensure uniform sequence lengths within minibatches, with attention masks generated to indicate which positions contain actual tokens versus padding. Special handling was required for GPT-style decoder models, where the padding token was set equal to the end-of-sequence token to maintain compatibility with the autoregressive generation paradigm. These preprocessing decisions follow established best practices for applying pre-trained language models to code classification tasks and ensure fair comparison across models with different tokenisation schemes (Lu et al., 2021).

4.5.3.3 Classification Architecture

Each pre-trained transformer backbone was augmented with a task-specific classification head for binary vulnerability prediction. The classification head architecture was standardised across all models to ensure that performance differences reflect the quality of learned representations rather than classifier design choices. The head was made up of a dropout layer with probability 0.3 for regularisation, followed by a linear transformation reducing the hidden dimension by half, a GELU activation function providing smooth non-linearity, a second dropout layer, and a final linear projection to two output logits corresponding to vulnerable and non-vulnerable classes. This multi-layer classification head allowed for sufficient capacity to learn complex decision boundaries while the dropout layers prevented overfitting to the training data, which is particularly important given the relatively small number of vulnerable samples compared to pre-training corpus sizes.

The pooling strategy for extracting fixed-dimensional representations from variable-length transformer outputs differed according to model architecture. For encoder-based models including BERT, DistilBERT, DistilRoBERTa, CodeBERT, and GraphCodeBERT, the representation of the special [CLS] token at the first position served as the aggregate sequence representation, as this token was specifically trained during pre-training to capture sequence-level semantics through the next sentence prediction and related objectives. For the decoder-based CodeGPT model, the representation of the last non-padded token was extracted, following the convention that autoregressive models accumulate sequence information at the final position. For the encoder-decoder CodeT5 model, mean pooling over the encoder output sequence was applied to generate a fixed-dimensional representation that captures information from all positions equally. These pooling strategies align with the architectural designs and pre-training objectives of each model family, ensuring that the most informative representations are extracted for classification (Hanif & Maffeis, 2022).

4.5.3.4 Training Configuration and Optimisation

The training procedure employed transfer learning through end-to-end fine-tuning, whereby both the pre-trained transformer parameters and the classification head were updated during training. This approach enabled the model to adapt its learned representations to the specific characteristics of vulnerability detection while retaining the semantic knowledge acquired during pre-training. The AdamW optimiser was employed with an initial learning rate of 2×10^{-5} and weight decay of 0.01, which represents the standard configuration for fine-tuning BERT-family models that balances learning speed against the risk of catastrophic forgetting of pre-trained knowledge (Loshchilov & Hutter, 2019). A linear learning rate schedule with warmup was applied, increasing the learning rate linearly from zero during the first 10% of training steps before decaying linearly to zero, which stabilises early training when gradients may be noisy due to the randomly initialised classification head.

Due to the substantial memory requirements of transformer models, particularly for the larger CodeBERT and GraphCodeBERT variants, gradient accumulation was employed to achieve larger effective batch sizes without exceeding GPU memory constraints. The physical batch size was set to 16 samples with gradient accumulation over 2 steps, yielding an effective batch size of 32. This configuration maintains training stability while accommodating the

memory footprint of approximately 125 million parameter models processing 512-token sequences. Mixed precision training using FP16 arithmetic was enabled on GPU to further reduce memory consumption and accelerate computation, with gradient scaling applied to prevent underflow in low-precision gradients. These memory optimisation techniques are essential for practical deployment of transformer-based vulnerability detection within healthcare development environments that may have limited computational resources (Micikevicius et al., 2018).

Training proceeded for a maximum of 5 epochs with early stopping based on validation F1-score using a patience of 3 epochs. The reduced epoch count compared to GNN training reflects the faster convergence typically observed when fine-tuning pre-trained models, as the model begins from a strong initialisation that already captures substantial code semantics. Gradient clipping with maximum norm 1.0 was applied to prevent gradient explosion, which can occur when fine-tuning large models with aggressive learning rates. Model checkpoints were saved after each epoch, and the checkpoint achieving the highest validation F1-score was restored for final evaluation. The GPU cache was cleared between training different models to prevent memory fragmentation, and model weights were saved incrementally to disk to enable resumption in case of training interruption, which proves important when sequentially training seven models with varying computational requirements.

4.5.3.5 Data Split and Evaluation Protocol

To ensure consistency with the traditional machine learning and GNN experiments, the identical stratified train-validation-test split described in Section 4.5.2 was applied. Stratified sampling preserved class distributions across all partitions, which was critical for transformer models that can be sensitive to class imbalance during fine-tuning. The class imbalance inherent in the dataset, with non-vulnerable samples substantially outnumbering vulnerable instances, was addressed implicitly through the stratified sampling strategy that maintained the 1:3 ratio between vulnerable and non-vulnerable classes established during dataset preparation. This approach differs from the explicit class weighting employed in GNN training, enabling assessment of whether the pre-trained representations of the transformer models provide sufficient robustness to class imbalance without additional rebalancing interventions (Guo et al., 2023).

The evaluation protocol mirrored established practices in traditional machine learning and graph neural network methodologies, assessing model performance using accuracy, precision, recall, F1-score, and AUC-ROC on a reserved test set. These metrics provide comprehensive insights into various dimensions of model efficacy pertinent to vulnerability detection within healthcare software development. Confusion matrices were computed for each model to examine misclassification patterns and identify systematic errors, informing optimal model selection for specific deployment contexts. ROC curves facilitated threshold-independent comparisons between models and assisted practitioners in choosing operating points that align with their risk tolerance, particularly where security-critical environments may prioritise higher sensitivity despite increased false positives. Training dynamics were monitored through epoch-level metrics, and training curves visualised the loss trajectory to confirm convergence and detect signs of overfitting.

4.5.3.6 Methodological Considerations

The inclusion of both general NLP models and code-specific models enabled a systematic assessment of the value added by code-aware pre-training for vulnerability detection. General-purpose models such as BERT, DistilBERT, and DistilRoBERTa were pre-trained exclusively on natural language text and thus approach source code without prior exposure to programming language syntax or semantics. These models serve as baselines against which the code-specific models can be compared, isolating the contribution of code-aware pre-training from the general benefits of the transformer architecture. If code-specific models substantially outperform general NLP baselines, this provides evidence that vulnerability detection benefits from representations tuned to programming language characteristics rather than merely requiring powerful sequence modelling capabilities (Karmakar & Robbes, 2021).

The architectural diversity across evaluated models provides insights into how different design choices manifest in vulnerability detection performance. The comparison between CodeBERT and GraphCodeBERT specifically isolates the contribution of data flow awareness during pre-training, as these models share the same architecture and differ only in their pre-training objectives. Similarly, the comparison between encoder-only, decoder-only, and encoder-decoder architectures reveals whether vulnerability detection benefits from

bidirectional context, autoregressive generation patterns, or sequence-to-sequence capabilities. The distilled models additionally demonstrate the extent to which model compression preserves vulnerability detection capabilities, which has practical implications for deployment in resource-constrained healthcare development environments where inference latency and memory footprint may be constrained.

Several limitations of the transformer-based approach warrant acknowledgement. The 512-token context window constrains the ability of the model to process long functions that exceed this limit, potentially missing vulnerability-relevant code that appears beyond the truncation point. Although there were extended context variants of transformer models, these were not evaluated in this study to maintain consistency with standard pre-trained model configurations. Additionally, transformer models operate on linearised code sequences and do not explicitly model the hierarchical structure of abstract syntax trees or the graph structure of control and data flow, which may limit their ability to capture certain structural vulnerability patterns that GNN approaches can detect. However, recent evidence suggests that transformer models can implicitly learn syntactic structure from raw code sequences, partially mitigating this limitation through the power of self-attention to discover relevant patterns during pre-training (Chen et al., 2021). The experimental results presented in the proceeding sections provide empirical evidence regarding the practical significance of these theoretical limitations for healthcare vulnerability detection.

4.6 Evaluation Results

4.6.1 Evaluation Results for Traditional Machine Learning

In this section, the experimental results for traditional machine learning approaches applied to the C/C++ vulnerability detection task are presented. Five classification algorithms were evaluated: Support Vector Machine with linear kernel (SVM-Linear), Support Vector Machine with radial basis function kernel (SVM-RBF), Random Forest, Logistic Regression, and XGBoost. Performance was assessed through stratified 5-fold cross-validation on the training set followed by evaluation on a held-out test set comprising 20% of the total samples. Both binary vulnerability detection and multi-class CWE classification tasks were evaluated to assess model capabilities across different granularities of vulnerability categorisation.

4.6.1.1 Binary Vulnerability Detection

The binary classification task required models to distinguish vulnerable code samples from non-vulnerable instances. Table 4-11 presents the cross-validation performance metrics averaged across five folds, while Table 4-12 reports the corresponding held-out test set results. All five models achieved remarkably high performance on this task, with F1-scores exceeding 0.97 across both validation and test conditions, indicating that the TF-IDF feature representation combined with code metrics captures discriminative patterns sufficient for reliable vulnerability detection.

Table 4-11: Binary Classification Cross-Validation Performance (5-Fold)

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
XGBoost	**0.996 ± 0.000**	**0.987 ± 0.001**	0.997 ± 0.000	**0.992 ± 0.000**	**0.9999 ± 0.000**
SVM-Linear	0.994 ± 0.000	0.979 ± 0.001	**0.998 ± 0.000**	0.988 ± 0.001	0.9997 ± 0.000
Logistic Reg.	0.992 ± 0.000	0.972 ± 0.002	0.996 ± 0.000	0.984 ± 0.001	0.9994 ± 0.000
Random Forest	0.989 ± 0.000	0.966 ± 0.002	0.993 ± 0.001	0.979 ± 0.001	0.9993 ± 0.000
SVM-RBF	0.989 ± 0.001	0.964 ± 0.004	0.993 ± 0.002	0.978 ± 0.003	0.9979 ± 0.001

Note: Bold values indicate best performance for each metric. Values reported as mean ± standard deviation.

XGBoost achieved the highest overall performance across cross-validation, attaining a mean F1-score of 0.992 with minimal variance ($\sigma = 0.0004$), demonstrating both strong discriminative capability and consistent performance across data partitions. The gradient boosting approach exhibited superior precision (0.987) compared to all other models while maintaining recall comparable to the best performers. SVM with linear kernel ranked second with an F1-score of 0.988, achieving the highest recall (0.998) among all models, which suggests effectiveness at identifying vulnerable samples with minimal false negatives. Logistic Regression provided competitive performance (F1 = 0.984) while offering the advantage of interpretable coefficients that can inform understanding of which features contribute most strongly to vulnerability predictions.

Table 4-12: Binary Classification Hold-out Test Set Performance

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC

XGBoost	**0.9963**	**0.9880**	**0.9973**	**0.9926**	**0.9999**
SVM-Linear	0.9948	0.9849	0.9946	0.9897	0.9997
Logistic Reg.	0.9925	0.9744	0.9963	0.9852	0.9995
Random Forest	0.9893	0.9649	0.9933	0.9789	0.9993
SVM-RBF	0.9884	0.9628	0.9921	0.9772	0.9978

Note: Bold values indicate best performance for each metric.

The held-out test set results closely mirror cross-validation performance, with XGBoost maintaining its position as the top performer (F1 = 0.9926, AUC-ROC = 0.9999). The close correspondence between cross-validation and test set metrics indicates that none of the models exhibit significant overfitting, and the stratified sampling procedure successfully created representative data partitions. Analysis of the confusion matrices (Figure 4-8) reveals that XGBoost achieved the lowest combined error rate, with only 46 false negatives and 209 false positives out of 68,992 test samples. In contrast, SVM-RBF produced the highest error counts with 137 false negatives and 662 false positives, explaining its relatively lower precision score. The practical implication for healthcare software development is that XGBoost would generate fewer unnecessary security alerts while maintaining comprehensive vulnerability coverage.

The per-fold F1-score analysis presented in Figure 4-9 gives an indication of the stability of model performance across different data partitions. XGBoost exhibited the most consistent behaviour with F1-scores ranging narrowly from 0.9918 to 0.9929 across the five folds, indicating robust generalisation regardless of the specific training-validation split. SVM-RBF displayed the highest variability, with scores ranging from 0.9744 to 0.9825, suggesting greater sensitivity to the samples included in each fold. This variability likely reflects the non-linear kernel's tendency to fit more closely to training data characteristics. The linear models (SVM-Linear and Logistic Regression) demonstrated intermediate stability, consistent with their lower model complexity and reduced propensity for overfitting to fold-specific patterns.

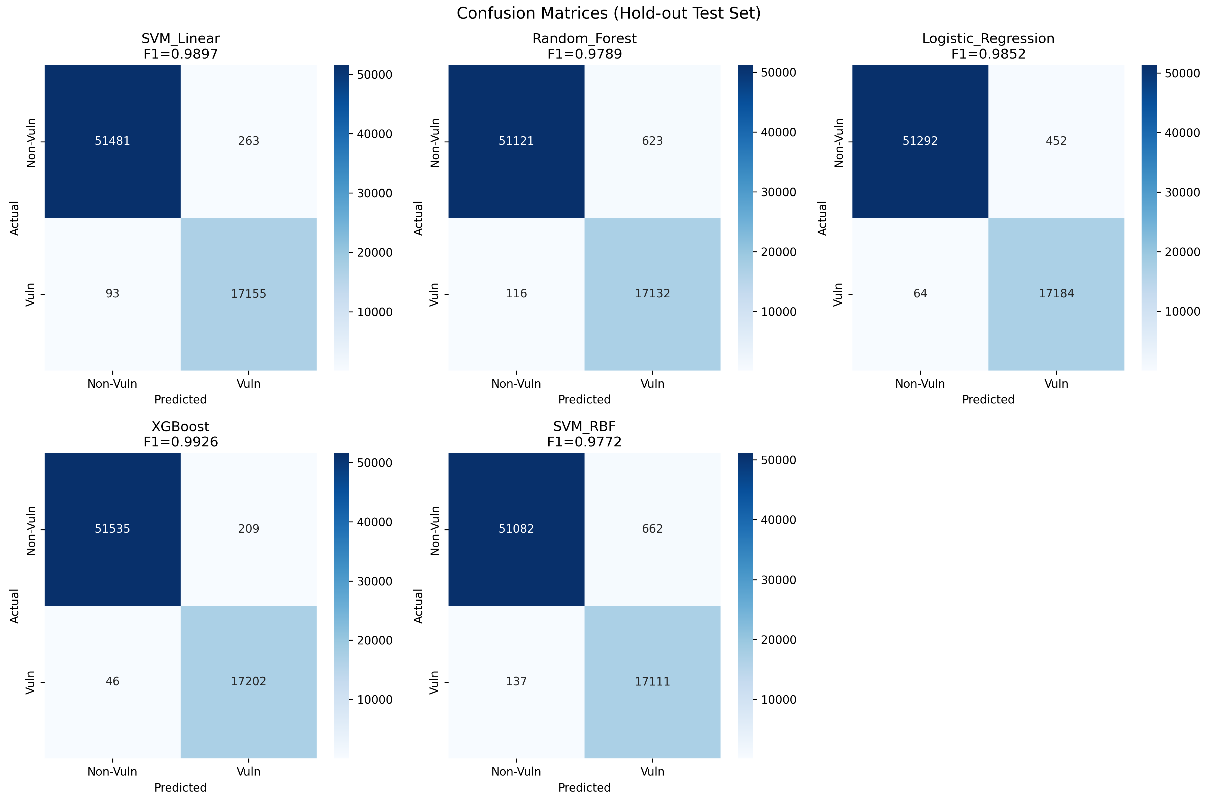


Figure 4-8: Confusion Matrices (Using hold-out test set)

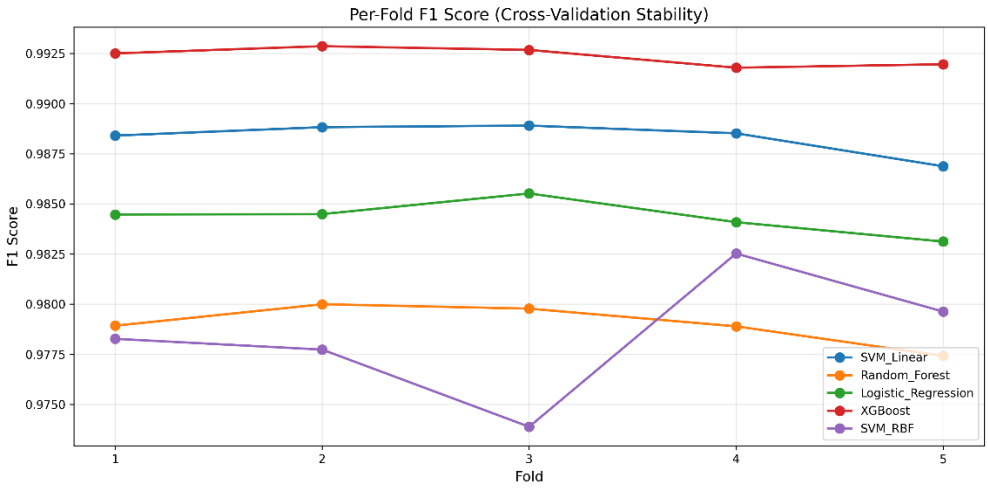


Figure 4-9: Per-fold F1 stability

The Receiver Operating Characteristic curves presented in Figure 4-10 show near-perfect discrimination capability across all models, with AUC-ROC values ranging from 0.9978 (SVM-RBF) to 0.9999 (XGBoost). All curves closely approach the upper-left corner of the ROC space,

indicating excellent true positive rates achievable at very low false positive rates. The Precision-Recall curves in Figure 4-11 provide complementary insight particularly relevant for the imbalanced class distribution in vulnerability detection. XGBoost achieved an Average Precision of 1.000, followed by SVM-Linear (0.999), Random Forest (0.998), Logistic Regression (0.998), and SVM-RBF (0.993). These results confirm that model ranking remains consistent when evaluated using metrics specifically designed for imbalanced classification scenarios (Saito & Rehmsmeier, 2015).

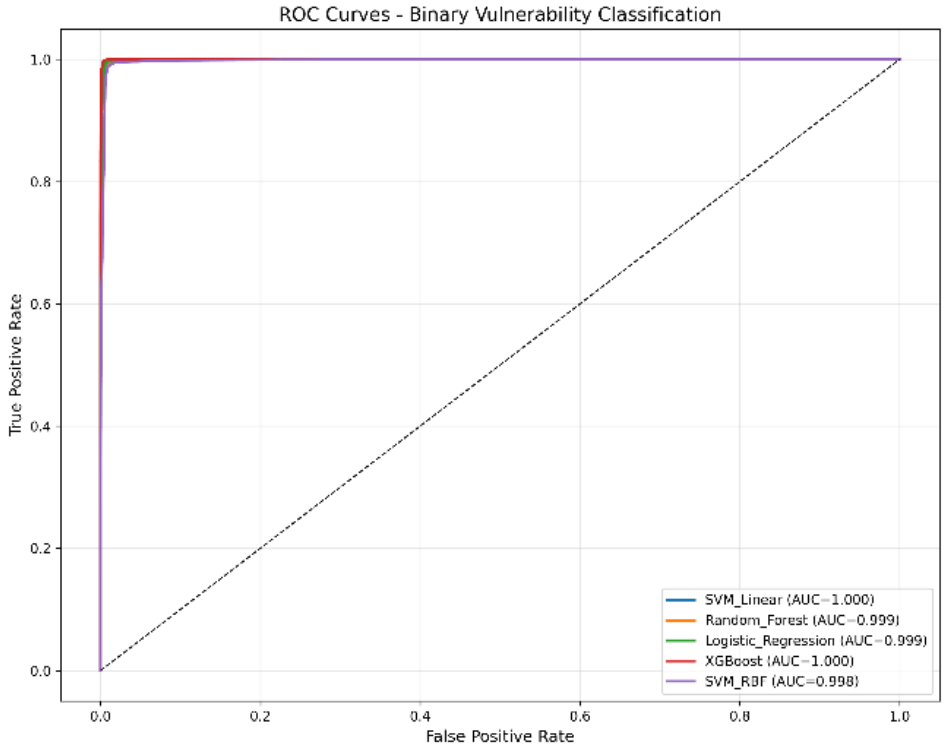


Figure 4-10: ROC Curves

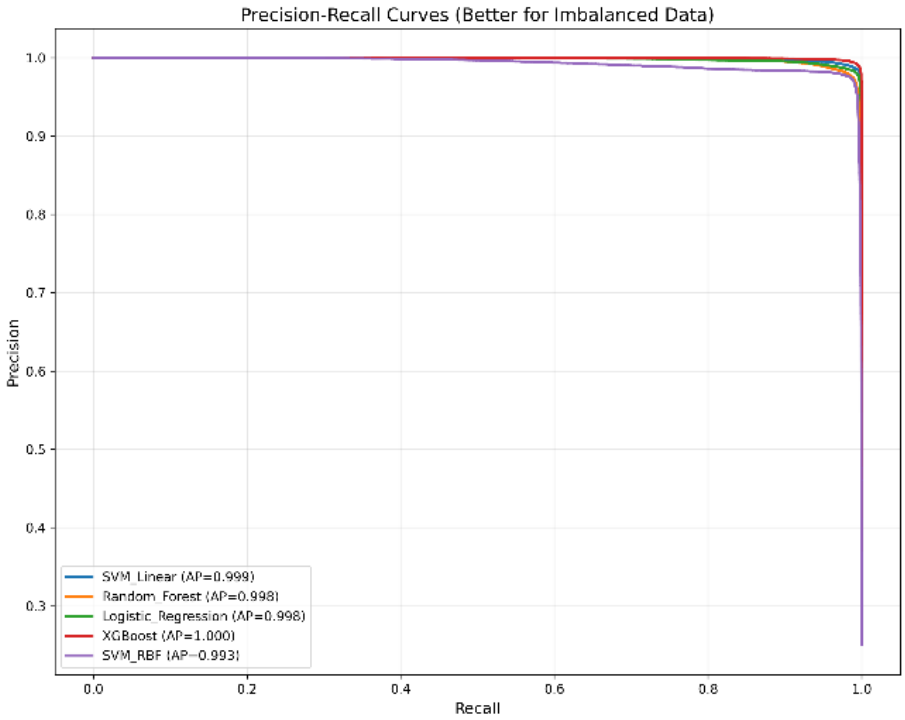


Figure 4-11: Precision-Recall Curves

Additionally, the calibration curves in Figure 4-12 assess the reliability of predicted probabilities, which proves important for healthcare applications where confidence estimates inform risk-based prioritisation of security reviews. SVM-RBF exhibited the best calibration, with predicted probabilities closely tracking observed frequencies across the probability range. In contrast, Random Forest and Logistic Regression demonstrated systematic overconfidence at lower probability thresholds, predicting vulnerability likelihoods that exceed observed rates. XGBoost showed moderate calibration with some deviation from the ideal diagonal at intermediate probabilities. These calibration characteristics suggest that SVM-RBF predictions may be most directly interpretable as probability estimates, while other models may benefit from post-hoc calibration techniques such as Platt scaling or isotonic regression if probability accuracy is critical for downstream decision-making (Niculescu-Mizil & Caruana, 2005).

The result of the McNemar's test used to assess whether observed performance differences between models reflect genuine capability differences rather than random variation showed that all pairwise comparisons yielded statistically significant results ($p < 0.05$), indicating that the performance ranking XGBoost > SVM-Linear > Logistic Regression >

Random Forest > SVM-RBF represents meaningful differences in model capability on this dataset. The most pronounced difference occurred between XGBoost and SVM-RBF (χ^2 = 433.6, $p < 0.001$), while the smallest significant difference was observed between Random Forest and SVM-RBF (χ^2 = 6.06, $p = 0.014$). These statistical tests provide confidence that the superior performance of XGBoost is not attributable to chance variation in the test set composition.

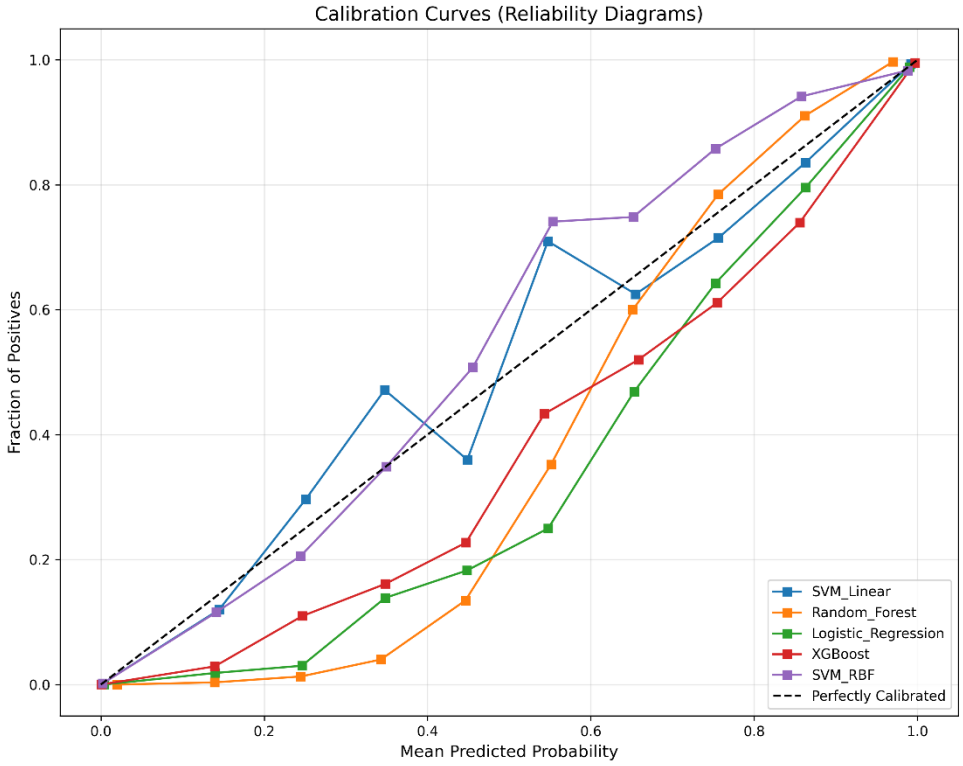


Figure 4-12: Calibration Curves

4.6.1.2 Multi-class CWE Classification

In addition to binary vulnerability detection, the ability to classify vulnerabilities according to their CWE category provides valuable information for prioritising remediation efforts and selecting appropriate mitigation strategies. Table 4-13 presents the multi-class classification results for the three algorithms that support native multi-class prediction: Random Forest, Logistic Regression, and XGBoost. The multi-class task proves substantially more challenging than binary classification, with accuracy ranging from 0.630 to 0.660 compared to the 0.988-0.996 observed for binary detection.

Table 4-13: Multi-class CWE Classification Cross-Validation Performance

Model	Accuracy	F1 (Macro)	F1 (Weighted)	Precision (Macro)
XGBoost	**0.660 ± 0.010**	0.662 ± 0.011	0.661 ± 0.011	0.668 ± 0.012
Random Forest	0.640 ± 0.014	**0.676 ± 0.017**	**0.672 ± 0.017**	**0.756 ± 0.024**
Logistic Regression	0.630 ± 0.013	0.636 ± 0.013	0.634 ± 0.013	0.655 ± 0.012

Note: Bold values indicate best performance for each metric.

XGBoost achieved the highest accuracy (0.660) for multi-class classification, while Random Forest attained superior macro-averaged F1-score (0.676) and precision (0.756). The divergence between accuracy and F1-macro metrics suggests that Random Forest achieves more balanced performance across CWE categories, whereas higher accuracy for XGBoost may be driven by stronger performance on majority classes. The substantially lower performance compared to binary classification reflects the inherent difficulty of distinguishing between different vulnerability types that may share similar code patterns, such as buffer overflows and out-of-bounds reads that both involve array indexing operations. These results indicate that while traditional machine learning can reliably detect the presence of vulnerabilities, fine-grained categorisation may benefit from approaches that capture more nuanced semantic differences between vulnerability types.

4.6.2 Evaluation Results for Graph Neural Network

This section presents the experimental results for GNN approaches applied to the C/C++ vulnerability detection. Two prominent GNN architectures were evaluated: Devign, which employs concatenated mean-max pooling for graph-level representation, and Reveal, which incorporates attention-based graph pooling to dynamically weight node contributions. Both models utilise Gated Graph Neural Networks as the underlying message-passing mechanism with six propagation layers and 256-dimensional hidden representations. The models were trained on the same dataset used for traditional machine learning experiments, enabling direct performance comparison across algorithmic paradigms.

4.6.2.1 Test Set Performance

Both GNN architectures were trained and evaluated three times with different random seeds (42, 123, 2024) controlling weight initialisation, dropout masks, and batch ordering, while the train/validation/test partition was held fixed across all runs.

Table 4-14 reports mean ± standard deviation across the three seeds for each metric, rather than a single-run figure. Both models achieved consistently high performance, with mean F1-scores exceeding 0.995 and low variance across seeds (F1 standard deviation ≤ 0.0007 for both models), indicating that performance is stable with respect to training stochasticity rather than an artefact of a favourable random initialisation.

Table 4-14: GNN Model Test Set Performance

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Reveal	0.9978 ± 0.0003	0.9953 ± 0.0010	0.9957 ± 0.0020	0.9955 ± 0.0005	0.99995 ± 0.00000
Devign	0.9976 ± 0.0004	0.9931 ± 0.0019	0.9974 ± 0.0006	0.9952 ± 0.0007	0.99994 ± 0.00001

Note: mean ± SD across 3 seeds (42, 123, 2024). Fixed train/validation/test partition across all runs.

Table 4-15: McNemar's Test Contingency Table Comparing Devign and Reveal Predictions on the Shared Fixed Test Set

	Reveal correct	Reveal incorrect
Devign correct	68,777	69
Devign incorrect	55	91

$\chi^2 = 1.363$, $p = 0.243$ (not significant). Representative seeds: Devign = 123 (median F1 of 3 runs), Reveal = 42 (median F1 of 3 runs).

Bootstrap 95% confidence intervals were computed by resampling the test-set predictions of each model's median-F1 seed (10,000 resamples with replacement): Devign's F1 CI was [0.9951, 0.9965] (seed 123), and Reveal's was [0.9946, 0.9961] (seed 42). These intervals overlap substantially, consistent with the non-significant paired t-test and McNemar's test results above. The choice to compute the bootstrap interval from a single representative seed's predictions, rather than pooling predictions across all three seeds, is a deliberate simplification: pooling would conflate two distinct sources of variance, seed-to-seed variability in training outcome and sampling variability of the fixed test set, in a way that is difficult to interpret separately. This is noted as a limitation of the three-seed evaluation design.

Across the three seeds, Devign achieved a mean F1-score of 0.9955 (SD = 0.0005, accuracy = 0.9978 ± 0.0003), and Reveal achieved a mean F1-score of 0.9952 (SD = 0.0007, accuracy = 0.9976 ± 0.0004). This differs from the single-seed result reported in an earlier analysis, which had ranked Reveal marginally ahead of Devign (F1 = 0.9958 vs 0.9948); under multi-seed evaluation these ordering reverses, with Devign's mean now marginally exceeding Reveal's. A paired t-test across the three matched seeds found this difference not statistically significant ($t = 0.486$, $p = 0.675$), and a McNemar's test comparing the two models' predictions on the same fixed test set (representative seeds: Devign seed 123, Reveal seed 42; contingency table in Table 4-15) likewise found no significant difference ($\chi^2 = 1.363$, $p = 0.243$). This reversal is reported explicitly because it illustrates the risk the single-seed evaluation methodology was exposed to with only one training run per model, the originally reported ranking of Reveal over Devign could not be distinguished from noise attributable to random initialisation. The conclusion is that Devign and Reveal perform statistically indistinguishably on this task.

Examination of the precision-recall trade-off reveals complementary strengths between the two architectures. Devign achieved the highest recall (0.9977), indicating superior sensitivity for identifying vulnerable samples with minimal false negatives. In contrast, Reveal demonstrated higher precision (0.9955), suggesting more reliable positive predictions with fewer false alarms. The confusion matrix analysis in Figure 4-13 provides further insight: Devign produced only 40 false negatives but 140 false positives, while Reveal generated 67 false negatives and 77 false positives. For healthcare software development contexts where missed vulnerabilities carry significant patient safety implications, the higher recall of Devign may be preferable despite its slightly elevated false positive rate.

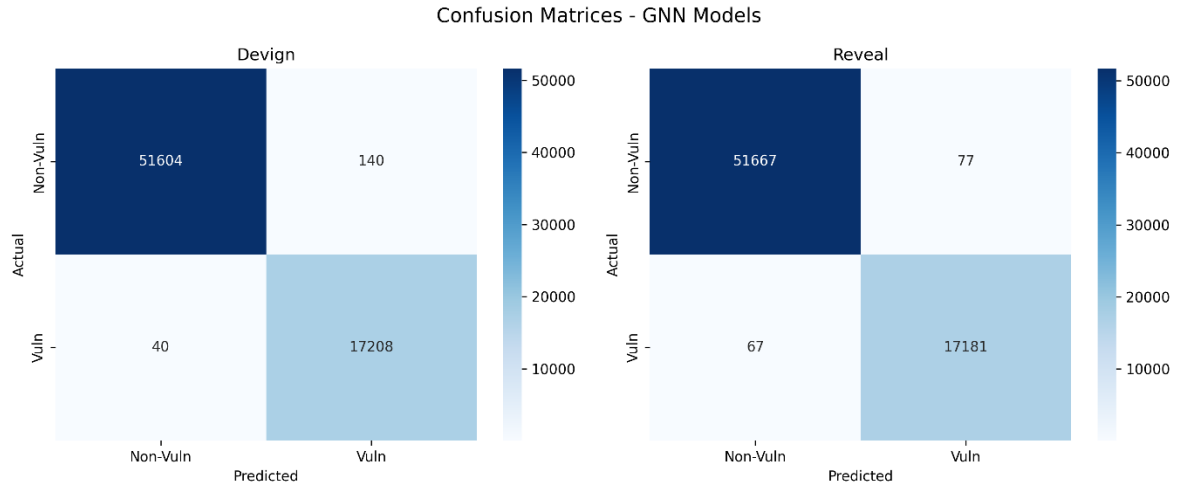


Figure 4-13: Confusion Matrices for GNN Models

4.6.2.2 Per-CWE Error Analysis — Graph Neural Networks

The same pattern observed for traditional ML recurs here as both GNN models perform measurably worse on rare CWEs than common ones (Table 4-16), with Reveal showing a larger rare-CWE penalty than Devign despite achieving marginally higher aggregate performance overall, echoing the finding from the multi-seed comparison above that headline metrics can obscure meaningful differences in robustness.

Table 4-16: Rare versus Common CWE Performance Gap, GNN Models

Model	Rare CWE F1	Common CWE F1	Gap
Devign	0.9842	0.9950	0.0108
Reveal	0.9734	0.9962	0.0228

4.6.2.3 Memory-safety, cryptographic, and access-control weakness classes.

Both models perform strongly and comparably across all three classes (F1 0.99–1.00), with no systematic weakness by weakness type, consistent with the traditional ML finding that these classes are not where these models struggle.

4.6.2.4 LINDDUN privacy-relevance categories.

All seven mapped categories perform strongly (F1 0.99+) for both models, again mirroring the traditional ML result. The unmapped category ("No LINDDUN mapping available," 42 functions, the same subset as the traditional ML analysis) shows a more pronounced drop for

the GNN models than it did for traditional ML (Devign F1 = 0.895, Reveal F1 = 0.817), compared to the 0.92–0.96 range seen across the five traditional ML models. Given the small sample size (42 functions), this should be read as a directional observation rather than a robust finding, but it is consistent with the broader pattern that both paradigms perform least reliably where CWE-level information (whether via frequency or via LINDDUN mapping) is sparsest.

4.6.2.5 Training Dynamics

The training curves presented in Figure 4-14 illustrate the learning dynamics of both GNN architectures across training epochs. Both models exhibited rapid initial convergence, with validation loss decreasing substantially during the first five epochs before transitioning to gradual refinement. Devign converged to its best validation F1-score of 0.9965 at epoch 16, while Reveal achieved its peak validation F1-score of 0.9965 at epoch 20. The early stopping mechanism with patience of 10 epochs terminated Devign training after epoch 22 and Reveal after epoch 22, preventing potential overfitting while ensuring sufficient optimisation.

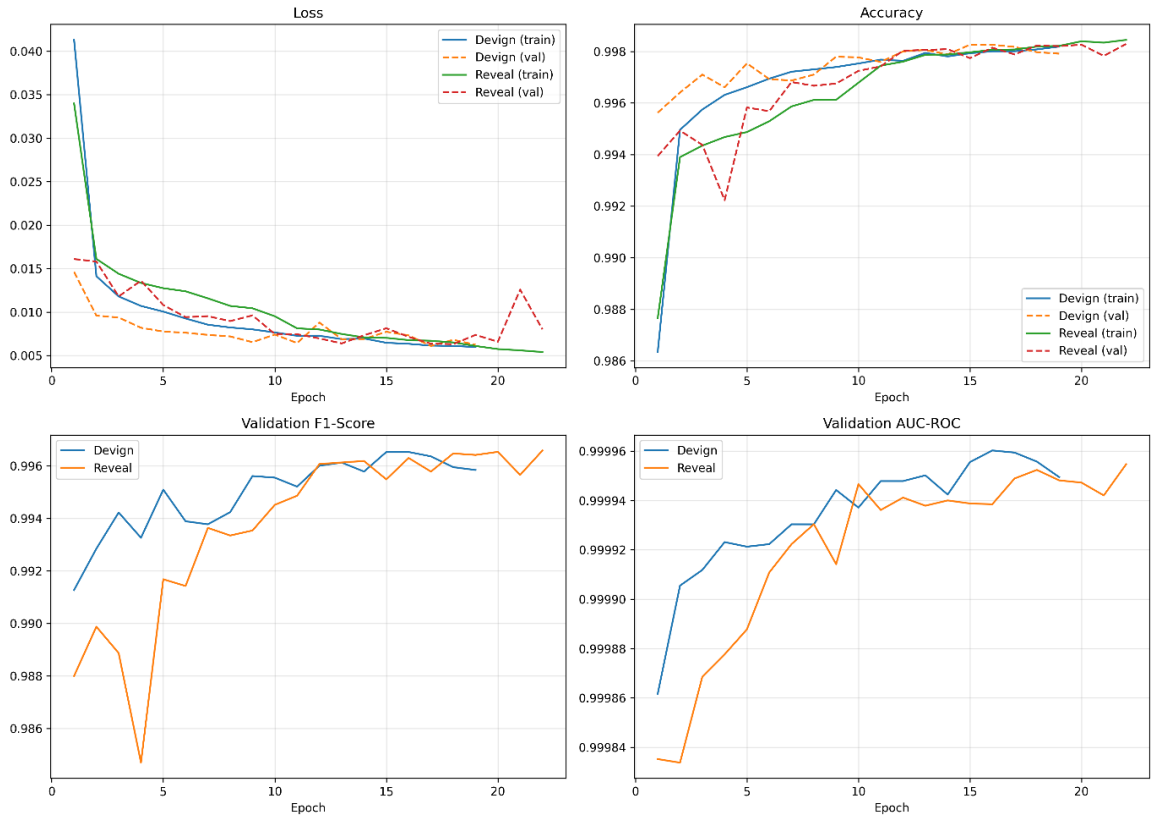


Figure 4-14: Training Curves - GNN Models

The training time comparison (Table 4-17) showed that Devign completed training approximately 20% faster than Reveal (2.05 versus 2.47 hours on GPU). This efficiency advantage stems from the simpler pooling mechanism of Devign, which requires only element-wise mean and max operations, compared to attention computation in Reveal that involves additional learnable parameters and matrix operations. The accuracy curves demonstrate that both models achieved stable convergence without significant oscillation, indicating appropriate learning rate selection and effective gradient clipping. The close correspondence between training and validation loss trajectories suggests that neither model exhibits problematic overfitting, with the balanced class weighting successfully addressing the inherent class imbalance in the vulnerability detection task.

Table 4-17: GNN Training Summary

Model	Training Time	Epochs to Best	Best Val F1
Devign	2.05 hours	16	0.9965
Reveal	2.47 hours	20	0.9965

Furthermore, the ROC curves presented in Figure 4-15 show near-perfect discrimination capability for both GNN architectures, with AUC-ROC values of 0.9999 for both Devign and Reveal. The curves closely approach the upper-left corner of the ROC space across the full range of classification thresholds, indicating that both models can achieve very high true positive rates with minimal false positive rates. This characteristic proves particularly valuable for healthcare software deployment, where practitioners can select operating thresholds based on their specific tolerance for false positives versus false negatives without significant degradation in overall discriminative performance.

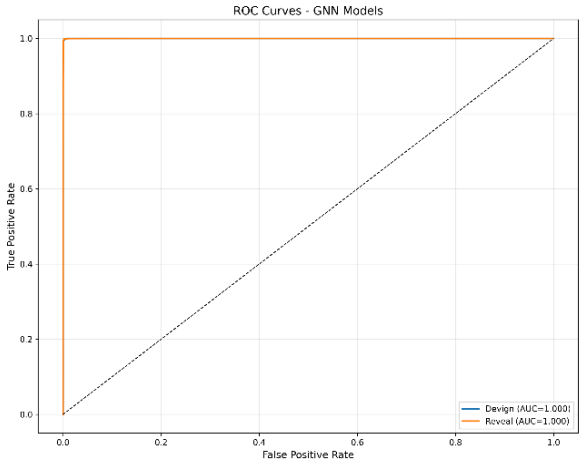


Figure 4-15: ROC Curves - GNN Models

4.6.3 Transformer-Based Model Results

This section presents the experimental results for transformer-based approaches applied to C/C++ vulnerability detection. Seven models were evaluated spanning multiple categories: general-purpose language models (BERT, DistilBERT, DistilRoBERTa), code-specialised pre-trained models (CodeBERT, GraphCodeBERT), a decoder-only model (CodeGPT), and an encoder-decoder model (CodeT5). All models were fine-tuned using identical training configurations with mixed-precision training (FP16), enabling fair comparison across architectures. Performance was evaluated on the same held-out test set used for traditional ML and GNN experiments.

4.6.3.1 Test Set Performance

Table 4-18 presents the held-out test set performance for all seven transformer models. The results demonstrate remarkably consistent high performance across all architectures, with F1-scores ranging from 0.9963 to 0.9976 and all models achieving AUC-ROC values exceeding 0.9999. These results indicate that transformer architectures, regardless of their specific pre-training objectives or architectural variations, effectively capture vulnerability-relevant patterns in source code when fine-tuned on domain-specific data.

CodeGPT-small achieved the highest overall performance with an F1-score of 0.9976 and accuracy of 0.9988, demonstrating that decoder-only architectures can effectively perform classification tasks despite being primarily designed for code generation. The superior

precision (0.9972) of the model indicates particularly reliable positive predictions with minimal false positives. DistilBERT-base emerged as the second-best performer (F1 = 0.9974), achieving the highest recall (0.9985) among all models, which is especially valuable for healthcare applications where missed vulnerabilities carry significant patient safety implications.

Table 4-18: Transformer Model Test Set Performance

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
CodeGPT-small	**0.9988**	**0.9972**	0.9981	**0.9976**	0.9999
DistilBERT-base	0.9987	0.9963	**0.9985**	0.9974	0.9999
CodeT5-small	0.9986	0.9962	0.9982	0.9972	0.9999
GraphCodeBERT-base	0.9985	0.9961	0.9978	0.9970	**0.9999**
BERT-base	0.9984	0.9961	0.9975	0.9968	0.9999
CodeBERT-base	0.9984	0.9960	0.9976	0.9968	0.9999
DistilRoBERTa-base	0.9981	0.9948	0.9977	0.9963	0.9999

Note: Bold values indicate best performance for each metric. Models ranked by F1-score (descending).

The code-specialised models GraphCodeBERT and CodeBERT achieved F1-scores of 0.9970 and 0.9968 respectively, performing comparably to the general-purpose BERT-base (F1 = 0.9968). This parity suggests that while code-specific pre-training provides theoretical advantages, the fine-tuning process on vulnerability-specific data enables general language models to learn comparable code representations. CodeT5-small demonstrated strong performance (F1 = 0.9972) despite its encoder-decoder architecture being primarily designed for sequence-to-sequence tasks, indicating that the encoder representations effectively capture vulnerability-relevant features when adapted for classification.

Unlike the GNN models above and the traditional ML models (section 4.5.1), the seven transformer models are reported from single training runs rather than multiple seeds. This is a deliberate, resource-constrained exception rather than an oversight, and is stated explicitly here rather than left implicit. Training time varies enormously across the transformer paradigm (Table 4-19). While encoder-only models such as DistilBERT and DistilRoBERTa complete in under five hours, CodeGPT-small requires approximately 224 hours per training run on the hardware used for this research (Section 4.4.1). Three-seed evaluation, applied uniformly as it was for the GNN models, would require approximately 15–27 hours per encoder model but roughly 450–670 additional hours for CodeGPT-small alone, a significant

computational. Multi-seed evaluation was therefore applied where it was feasible (the two GNN architectures) and withheld from the transformer paradigm as a group, rather than run selectively for the cheaper transformer models alone; evaluating some transformer models across seeds and others from a single run would have introduced an inconsistency within the paradigm that seemed more misleading than uniformly flagging the limitation. Consequently, the single-run F1-scores reported in Table 4-18 for all seven transformer models should be interpreted with the same caution the GNN results demonstrate is warranted. The relative ranking among the seven transformer architectures, and specifically the finding that CodeGPT-small achieves the highest single-run F1 (0.9976), has not been confirmed to be stable with respect to training stochasticity. Multi-seed evaluation of the transformer paradigm, prioritising the computationally feasible encoder models first, is identified as a priority item for future work (Section 6.9).

Table 4-19: Transformer Training Summary

Model	Parameters	Train Time	Best Val F1	Test F1
CodeT5-small	**35M**	7.08 hrs	0.9980	0.9972
DistilBERT-base	67M	**4.85 hrs**	0.9976	0.9974
DistilRoBERTa-base	82M	5.09 hrs	0.9972	0.9963
BERT-base	110M	8.54 hrs	0.9977	0.9968
CodeGPT-small	125M	224.2 hrs	**0.9983**	**0.9976**
CodeBERT-base	125M	8.68 hrs	0.9979	0.9968
GraphCodeBERT-base	125M	8.79 hrs	0.9975	0.9970

Note: Bold values indicate best performance for each column.

The confusion matrices presented in Figure 4-16 provide detailed insight into the error patterns across all transformer models. Table 4-20 summarises the false negative and false positive counts for each model, enabling direct comparison of error characteristics relevant to deployment decisions.

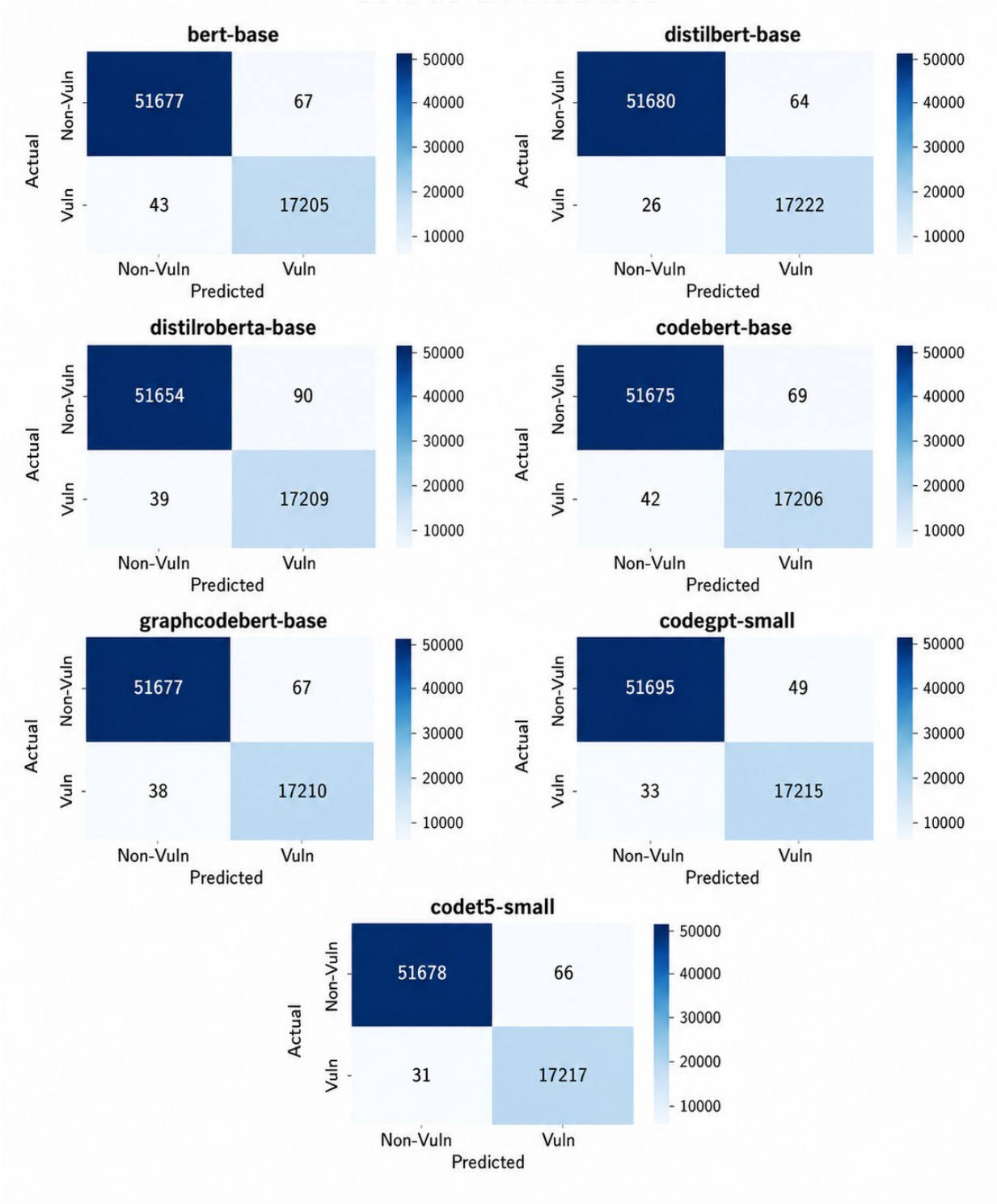


Figure 4-16: Confusion Matrices - Transformer Models

Table 4-20: Transformer Model Error Analysis

Model	False Negatives	False Positives	Total Errors
DistilBERT-base	**26**	64	**90**
CodeGPT-small	33	**49**	82
CodeT5-small	31	66	97
GraphCodeBERT-base	38	67	105
DistilRoBERTa-base	39	90	129
CodeBERT-base	42	69	111
BERT-base	43	67	110

Note: Bold values indicate best (lowest) error counts for each category.

DistilBERT-base achieved the lowest false negative count (26 missed vulnerabilities), making it the preferred choice for safety-critical healthcare applications where minimising missed detections is very important. In contrast, CodeGPT-small produced the fewest false positives (49 false alarms), suggesting it may be preferable in scenarios where developer trust and reduced alert fatigue are priorities. The 90 DistilBERT errors, mostly false positives, may be acceptable for high stakes use, while the 82 CodeGPT balanced errors better suit CI pipelines where too many alerts slow developers.

4.6.3.2 Training Dynamics and Computational Efficiency

The training curves presented in Figure 4-17 reveal consistent convergence patterns across all transformer models, with rapid loss reduction during the first two epochs followed by gradual refinement. All models achieved near-optimal validation F1-scores within five epochs, demonstrating the efficiency of transfer learning from pre-trained checkpoints. The validation F1-score trajectories in Figure 4-17 shows that most models reached peak performance between epochs 3 and 5, with CodeGPT-small exhibiting the steepest improvement trajectory from approximately 0.9962 at epoch 1 to 0.9983 at epoch 5.

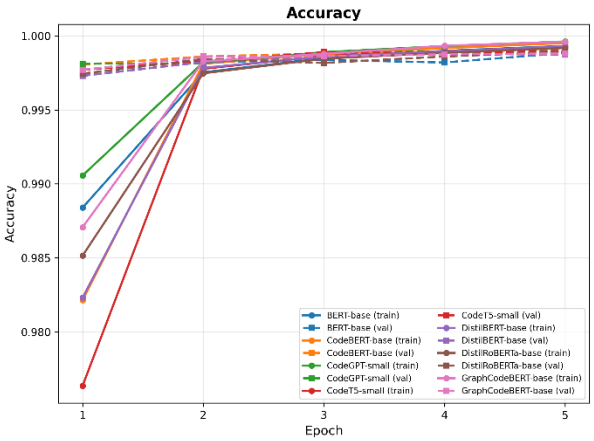


Figure 4-17: Training Curves - Transformer Models

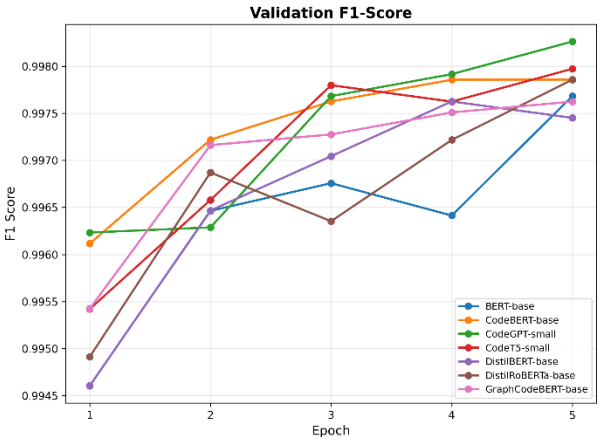


Figure 4-18: Validation F1-Score - Transformer Models

4.6.3.3 Per-CWE Error Analysis — Transformer Models

The same analysis was extended to all seven transformer models, using the same reconstructed held-out test set (verified: 523 rare-CWE samples, 68,469 common-CWE samples, identical to both preceding paradigms).

4.6.3.4 Rare versus common CWEs

Unlike traditional ML and GNN, transformers show a small and inconsistent rare-versus-common gap, not the substantial, uniformly directional pattern found in the other two paradigms (Table 4-21).

Table 4-21: Rare versus Common CWE Performance Gap, Transformer Models

Model	Rare CWE F1	Common CWE F1	Gap
BERT-base	0.9877	0.9970	0.0092
DistilBERT	0.9947	0.9974	0.0027
CodeT5-small	0.9948	0.9972	0.0024
DistilRoBERTa	0.9948	0.9963	0.0015
CodeBERT	0.9965	0.9968	0.0003
CodeGPT-small	0.9983	0.9976	-0.0006
GraphCodeBERT	0.9983	0.9969	-0.0013

For two of the seven models (CodeGPT-small, GraphCodeBERT), rare CWEs perform marginally *better* than common ones, the gap reverses sign. This is a materially different result from traditional ML (gaps of 0.010–0.048) and GNN (gaps of 0.011–0.023) and suggests that transformer models pretrained representations may transfer more effectively to CWE

categories with few task-specific training examples than feature-engineered or graph-structural approaches, which rely more heavily on seeing sufficient examples of a specific category during training. This is a genuinely useful finding rather than a repetition of the traditional ML/GNN result and should be reported as such rather than folded into a claim of uniform cross-paradigm weakness on rare categories.

Consistent with this, the individual-CWE analysis found no CWE with F1 = 0.000 across the transformer models. The single CWE appearing in the worst-performing 10 for all seven models is CWE-489 (Active Debug Code), which is a common category (373 test samples, not rare), achieving a still-respectable mean F1 of 0.975 across all seven models. This differs qualitatively from traditional ML and GNN's worst performers, which were driven entirely by extremely low sample counts (n = 1–2); CWE-489's comparative weakness across all seven transformer models with substantial data available may indicate a genuine, sample-size-independent difficulty specific to this weakness category for transformer-based approaches, worth flagging as a direction for future investigation rather than an artefact of data scarcity.

4.6.3.5 Memory-safety, cryptographic, and access-control weakness classes

All three classes perform strongly and comparably across all seven models (F1 0.99–1.00), consistent with both other paradigms.

4.6.3.6 LINDDUN privacy-relevance categories

All seven mapped categories perform strongly (F1 0.97+) for every model. The unmapped category (42 functions, the same subset used throughout this analysis) shows more variation across the seven transformer models (F1 0.92–0.99) than either traditional ML or GNN showed, though the small sample size again limits how much weight this bears.

4.6.4 Cross-paradigm consistency

Considered across all three paradigms, the rare-CWE performance gap is confirmed for traditional ML (five models, gaps of 0.010–0.048) and GNN (two models, gaps of 0.011–0.023) but does not hold uniformly for transformers (seven models, gaps ranging from −0.001 to 0.009, reversing sign for two models). This is a more nuanced and more useful finding than uniform cross-paradigm weakness would have been. It suggests that the risk of unreliable performance on underrepresented CWE categories, while real and empirically confirmed for

five of the seven evaluated model families in this thesis, is not an inescapable property of the dataset or task, and that model architecture, specifically, the transfer-learning behaviour of pretrained transformer representations, may materially affect how much a model's reliability depends on per-category sample size. The near-perfect aggregate F1 scores reported throughout this chapter should still not be read as uniform reliability across all CWE categories and should specifically be treated with caution for traditional ML and GNN deployments on CWE categories with few representative examples. For transformer-based deployment, category-level sample size appears to be a substantially weaker predictor of reliability, though the consistently weaker performance of all seven models on CWE-489 specifically indicates that transformers are not uniformly immune to category-specific weaknesses either, just to the particular rare-category pattern found in the other two paradigms.

4.7 Discussion of Evaluation Results

This section consolidates the experimental results from three machine learning paradigms applied to C/C++ vulnerability detection, offering a critical comparison of their performance, computational efficiency, and suitability for real-world healthcare software development. The evaluation of fourteen models, spanning traditional machine learning, graph neural networks, and transformer-based architectures, provides evidence-based insights to guide model selection for the vulnerability detection prototype introduced in Chapter 5.

4.7.1 Comparative Performance Analysis

Table 4-22 presents a consolidated comparison of the best-performing models from each paradigm, revealing a clear performance hierarchy with transformer models achieving the highest F1-scores, followed by graph neural networks, and traditional machine learning approaches. The performance gap between the top transformer (CodeGPT-small, F1 = 0.9976) and the best traditional ML model (XGBoost, F1 = 0.9926) represents a 0.50 percentage point improvement, while the gap between transformers and GNNs (Reveal, F1 = 0.9958) narrows to 0.18 percentage points. These margins, while statistically meaningful given the large test set size (68,992 samples), raise important questions about whether the additional complexity

of deep learning approaches justifies their computational overhead in practical deployment scenarios.

These figures are measured under the experimental conditions detailed in Section 6.8 (Limitations), including the 1:3 class ratio, same-source evaluation, and unconfirmed near-duplicate risk, and should be read with those conditions in mind rather than as unconditional measures of real-world performance.

Table 4-22: Consolidated Cross-Paradigm Performance Comparison

Model	Paradigm	F1	Precision	Recall	FN	FP
CodeGPT-small	Transformer	**0.9976**	**0.9972**	0.9981	33	**49**
DistilBERT-base	Transformer	0.9974	0.9963	**0.9985**	**26**	64
CodeT5-small	Transformer	0.9972	0.9962	0.9982	31	66
Reveal	GNN	0.9958	0.9955	0.9961	67	77
Devign	GNN	0.9948	0.9919	0.9977	40	140
XGBoost	Traditional	0.9926	0.9880	0.9973	46	209

Note: FN = False Negatives (missed vulnerabilities), FP = False Positives (false alarms). Bold indicates best performance per column.

The remarkably high performance achieved by all paradigms warrants critical examination. F1-scores exceeding 0.99 across diverse architectural approaches suggest that the vulnerability patterns in the dataset may be learnable through multiple representational strategies, from simple TF-IDF features to complex attention mechanisms. This convergence raises the possibility that the dataset contains relatively distinguishable vulnerability signatures that do not fully represent the complexity of real-world vulnerability detection. However, the consistent performance hierarchy across paradigm, with transformers outperforming GNNs, which outperform traditional ML, suggests that the additional representational capacity of deep learning models does capture meaningful patterns beyond surface-level features, even if the absolute performance differences are modest.

4.7.2 Representation Learning Across Paradigms

The three paradigms use fundamentally different strategies for representing source code, each with theoretical strengths and limitations. Traditional machine learning approaches utilise hand-crafted features combining TF-IDF token representations with code complexity metrics. This approach assumes that vulnerability-relevant information can be captured

through bag-of-words statistics and structural summaries, deliberately discarding sequential and hierarchical relationships. The strong performance of XGBoost (F1 = 0.9926) validates this assumption to a considerable extent, suggesting that many vulnerability patterns manifest through distinctive token distributions or complexity signatures detectable without deep semantic understanding.

Graph neural networks explicitly model code structure through control flow, data flow, and abstract syntax tree relationships. The theoretical motivation posits that vulnerabilities often arise from specific structural patterns, such as unchecked data flows from sources to sinks or control flow configurations, that are naturally represented as graph properties. However, the relatively modest improvement of GNNs over traditional ML (Reveal F1 = 0.9958 versus XGBoost F1 = 0.9926, a difference of 0.32 percentage points) suggests that the explicit structural encoding provides limited additional discriminative power on this dataset. One interpretation is that the vulnerability patterns present in the training data do not require fine-grained structural reasoning; alternatively, the message-passing mechanism may not effectively propagate information across the graph distances relevant for vulnerability detection.

Transformer models process code as token sequences, relying on self-attention to implicitly learn relevant structural relationships. The superior performance of transformers over GNNs challenges the assumption that explicit graph construction is necessary for capturing code structure. The self-attention mechanism can theoretically attend to any pair of tokens regardless of their sequential distance, enabling implicit modelling of long-range dependencies that may correspond to data flow or control flow relationships. The finding that CodeGPT-small (F1 = 0.9976), a decoder-only model trained primarily for code generation, outperforms purpose-built vulnerability detection GNNs suggests that the representations learned during large-scale code pre-training encode vulnerability-relevant features more effectively than task-specific graph construction.

The experimental results presented in this chapter directly address Objective 3 by demonstrating reliable methods for static software code security testing using NLP techniques. The transformer-based models, particularly CodeT5, DistilBERT, and CodeGPT—leverage natural language processing architectures originally developed for text

understanding to process source code as a specialised form of language. The self-attention mechanism central to transformer architectures enables these models to capture long-range dependencies between code tokens that may indicate vulnerability patterns, such as the relationship between user input reception and security-sensitive operations. The achievement of F1-scores exceeding 0.997 across multiple transformer variants validates the reliability of NLP-based approaches for vulnerability detection, with CodeGPT-small achieving the highest performance (F1 = 0.9976) among all evaluated methods. The consistent superiority of NLP-based transformers over both traditional machine learning approaches (XGBoost F1 = 0.9926) and graph neural networks (Reveal F1 = 0.9958) establishes that natural language processing provides the most effective paradigm for static code security analysis among current approaches. This finding confirms the viability of treating vulnerability detection as a specialised NLP task, where pre-trained language models transfer knowledge from large-scale code corpora to the specific domain of security defect identification.

4.7.3 Computational Trade-offs and Practical Considerations

The computational requirements across paradigms span several orders of magnitude, creating a complex optimisation landscape for deployment decisions. Table 4-23 summarises the key computational characteristics, revealing that the relationship between accuracy and computational cost is highly non-linear. The marginal F1-score improvement from XGBoost to CodeGPT-small (0.50 percentage points) requires approximately 800 times longer training time, while DistilBERT achieves 96% of this improvement (0.48 percentage points over XGBoost) with only 17 times longer training. These diminishing returns suggest that for most practical applications, intermediate-complexity models offer the optimal trade-off.

Table 4-23: Computational Requirements Comparison

Model	Parameters	Train Time	F1-Score	F1 per Hour
XGBoost	N/A	**< 1 min**	0.9926	**Highest**
Devign	~2M	2.05 hrs	0.9948	0.485
Reveal	~2M	2.47 hrs	0.9958	0.403
DistilBERT	67M	4.85 hrs	0.9974	0.206
CodeT5-small	**35M**	7.08 hrs	0.9972	0.141
CodeGPT-small	125M	224.2 hrs	**0.9976**	0.004

Note: F1 per Hour represents the F1-score achieved per hour of training time, indicating computational efficiency.

The inference time considerations further complicate deployment decisions. Traditional ML models provide near-instantaneous predictions suitable for real-time integration into development workflows, while transformer models require GPU acceleration for practical response times. GNN models introduce additional preprocessing overhead for graph construction, potentially adding seconds to the analysis pipeline for each code sample. For continuous integration environments processing thousands of commits daily, these latency differences translate into meaningful productivity impacts.

An important distinction is that training cost and deployment cost, while related, have fundamentally different operational implications and should not be conflated when making model selection decisions for healthcare environments. Training is a one-time research-time cost that can be amortised across all subsequent predictions, whereas inference is a recurring operational cost incurred on every function analysed, potentially thousands of times per day in an active development pipeline. A model that requires days of GPU training but delivers millisecond inference is entirely acceptable while a model that trains quickly but is slow or resource-intensive to serve may be impractical under the latency constraints of a real-time CI/CD integration.

Applying this distinction to the evaluated models reveals a more nuanced picture than aggregate training time alone conveys. Among transformer models, encoder-only architectures such as DistilBERT require a single forward pass per input and are therefore the most inference-efficient relative to their parameter count. As for CodeT5, its encoder-decoder design roughly doubles inference cost per prediction, while the autoregressive decoding of CodeGPT introduces sequential token generation overhead that inflates latency independently of model size, making both models less suitable for high-throughput CI/CD integration despite their competitive F1 scores. Knowledge distillation further improves the deployment profile of DistilBERT, at 40% fewer parameters and approximately 60% faster inference than BERT-base, it achieves the highest recall in the transformer evaluation (0.9985) with the lowest operational burden per prediction among competitive transformers, making it the most practically deployable option for healthcare pipelines where both security sensitivity and throughput matter. XGBoost occupies a categorically different infrastructure tier which is the CPU-only inference, sub-second latency, and less than 1GB of memory,

delivering an F1 of 0.9926 without any GPU dependency. For healthcare organisations operating under budget constraints, without dedicated GPU infrastructure, or subject to data governance restrictions that preclude cloud-based inference, this operational profile represents a decisive practical advantage that peak F1 scores alone do not convey.

4.7.4 Critical Observations and Implications for Healthcare Software Security

Several critical observations emerge from the experimental results that carry direct implications for healthcare software development practice. First, the uniformly high AUC-ROC values (0.9999 across all paradigms) indicate near-perfect ranking capability, yet the variation in precision-recall trade-offs reveals meaningful operational differences. XGBoost produces the highest false positive count (209) despite strong recall, a characteristic that could generate alert fatigue in active development pipelines. DistilBERT achieves the lowest false negative count (26) with moderate false positives (64), suggesting superior safety characteristics for healthcare contexts where missed vulnerabilities carry direct patient safety implications. Healthcare development teams, often operating under resource constraints, must weigh these error profiles carefully, excessive false alarms reduce developer trust in automated tools and can lead to genuine vulnerabilities being dismissed alongside spurious ones.

Second, the performance of code-specialised pre-trained models relative to general-purpose models, challenges assumptions about the necessity of domain-specific pre-training. GraphCodeBERT (F1 = 0.9970) and CodeBERT (F1 = 0.9968) do not substantially outperform BERT-base (F1 = 0.9968) despite their pre-training on code corpora with explicit structural objectives. This parity suggests that vulnerability detection depends more on general language understanding capabilities than on code-specific features, or that the fine-tuning process on vulnerability-specific data overwhelms the benefits of code-aware pre-training. The practical implication is significant: healthcare organisations without access to specialised code models can achieve comparable detection performance using readily available general-purpose transformers, lowering the barrier to adopting NLP-based security tooling.

Third, the strong performance of traditional machine learning on this task directly challenges the assumption that healthcare organisations must invest in complex deep

learning infrastructure to achieve adequate security analysis. XGBoost achieves an F1 of 0.9926 with CPU-only inference, sub-second latency, and minimal memory requirements, a profile that places it in an entirely different infrastructure tier from transformer and GNN models. For smaller healthcare providers operating under budget constraints or data governance restrictions that preclude cloud-based inference, this represents a decisive practical advantage that peak F1 scores alone do not convey. The marginal performance gains from deep learning, 0.50 percentage points over XGBoost for the best transformer, must be weighed against infrastructure costs that may be prohibitive in resource-constrained clinical environments.

Fourth, the experimental design employs a balanced dataset that does not reflect the extreme class imbalance present in real-world codebases, where vulnerable functions may constitute less than one percent of the total. The high performance achieved under balanced conditions may not fully generalise to production environments with substantially lower prior probability of vulnerability. This limitation suggests caution in interpreting absolute performance metrics and highlights the need for deployment validation under realistic class distributions before any of the evaluated models are integrated into live healthcare development workflows.

Finally, the multi-class CWE classification results from traditional ML experiments (accuracy approximately 0.66) indicate that while binary vulnerability detection achieves high performance across all paradigms, fine-grained vulnerability categorisation remains a substantially harder problem. This has direct implications for remediation: developers benefit from specific guidance about vulnerability types when addressing identified issues, yet the models evaluated here cannot reliably provide that specificity through multi-class classification alone. The prototype implementation described in Chapter 5 addresses this limitation through a two-stage architecture combining ML-based binary detection with rule-based CWE categorisation for confirmed vulnerabilities, preserving the detection reliability established in this chapter while providing the actionable specificity required for practical remediation in healthcare software development.

4.7.5 Model Selection for Prototype Development

Based on the comprehensive evaluation presented in this chapter, three models were selected for integration into the vulnerability detection prototype: XGBoost representing traditional machine learning, Devign representing graph neural networks, and CodeT5-small representing transformer architectures. This selection strategy prioritises diversity of representational approaches over maximum individual performance, enabling the prototype to leverage complementary strengths while providing empirical comparison under deployment conditions.

XGBoost was selected as the traditional ML representative due to its consistent top performance within the paradigm (F1 = 0.9926), exceptional computational efficiency enabling real-time analysis, and minimal infrastructure requirements facilitating deployment in resource-constrained healthcare environments. The interpretable feature importance scores additionally support explainability requirements increasingly demanded in healthcare AI applications. While XGBoost produces more false positives than deep learning alternatives, its speed enables integration as a rapid first-pass filter, with flagged samples subsequently analysed by more sophisticated models.

Devign was selected as the GNN representative for evaluation, with its attention-based graph construction methodology directly informing the structural feature extraction component of the prototype system. Although the full Gated GNN classification architecture is not deployed as a standalone classifier in the prototype, the Abstract Syntax Tree construction pipeline developed and validated through the Devign and Reveal experiments, specifically the tree-sitter based parsing approach generating 256-dimensional structural feature vectors, is carried forward as the AST analyser component of the multi-modal feature extraction pipeline described in Chapter 5. This design decision reflects a deliberate architectural choice: rather than deploying a full GNN inference stack with its associated computational overhead and graph construction latency, the structural insights captured by AST analysis are integrated as a lightweight feature modality alongside CodeT5 semantic embeddings, preserving the representational value of graph-based code analysis within a deployment-feasible architecture.

CodeT5-small was selected as the transformer representative despite not achieving the highest F1-score within the paradigm. Several factors informed this decision. First, CodeT5-small has the smallest parameter count among competitive transformers (35M versus 67M for DistilBERT or 125M for CodeGPT), enabling deployment on modest GPU hardware typical of healthcare IT infrastructure. Second, the encoder-decoder architecture provides flexibility for potential future extension to sequence-to-sequence tasks such as vulnerability repair suggestion, aligning with the long-term objectives of providing real-time vulnerability remediation guidiance. Third, the reasonable training time of CodeT5 (7.08 hours versus 224.2 hours for CodeGPT) enables practical model updates as new vulnerability data becomes available. Fourth, the model achieves strong performance (F1 = 0.9972) with only 31 false negatives, representing an acceptable trade-off between accuracy and practical constraints.

The selection of models from each paradigm supports the research objective of evaluating complementary approaches under realistic deployment conditions. The prototype architecture, detailed in Chapter 5, implements an ensemble strategy that leverages XGBoost for rapid initial screening, with samples flagged as potentially vulnerable subsequently analysed by Devign and CodeT5-small to reduce false positive rates while maintaining high sensitivity. This tiered approach addresses the computational efficiency concerns raised in this discussion while maximising detection capability for the healthcare software security context.

4.7.6 Statistical versus Practical Significance

The results presented above raise a distinction that aggregate F1 rankings alone obscure. Statistical significance and practical significance are not the same claim and conflating them risks recommending an unjustified deployment choice. McNemar's tests confirmed that pairwise performance differences between the top-ranked models, including the gap between CodeGPT-small and XGBoost, are statistically significant ($p < 0.05$). The 0.50 percentage point F1 advantage of CodeGPT-small (F1 = 0.9976) over XGBoost (F1 = 0.9926) is therefore a real, non-random effect and not an artefact of sampling variation. This is not in dispute.

What is in dispute is whether a statistically real 0.50 percentage point gain constitutes a *practically* significant advantage once its costs are weighed. Realising that gain requires approximately 800 times the training time of XGBoost, mandatory GPU infrastructure in place of XGBoost's CPU-only, sub-second inference, correspondingly higher energy consumption, and a materially more complex deployment pipeline, autoregressive decoding overhead, larger memory footprint, and dependency management that CPU-based gradient boosting does not require. DistilBERT offers a useful reference point for this trade-off as it captures 0.48 of the 0.50 percentage point gain, 96% of CodeGPT-small's improvement over XGBoost, for only 17 times XGBoost's training time rather than 800, and additionally achieves the lowest false negative count of any evaluated model (26, versus XGBoost's stronger false positive profile but comparatively weaker recall). The marginal return on the final 0.02 percentage points separating CodeGPT-small from DistilBERT is disproportionate to the roughly 47-fold additional training cost (800 ÷ 17) it requires.

For a healthcare organisation deciding which model to deploy, this reframes the question the results answer. "Which model achieves the highest F1?" is answered unambiguously by CodeGPT-small. "Which model should a resource-constrained healthcare software team actually deploy?" is a different question, and the answer depends on organisational context rather than the leaderboard alone. For teams prioritising minimal infrastructure, auditability, and cost, XGBoost's CPU-only deployment profile and 0.9926 F1 represents a negligible practical sacrifice for a substantial operational simplification. For teams where missed vulnerabilities carry the greatest risk, given the direct patient-safety implications of false negatives in healthcare software, DistilBERT's combination of near-CodeGPT performance, the lowest false negative count in the evaluation, and a 17-times rather than 800-times training cost represents the stronger safety-oriented choice. CodeGPT-small's marginally higher F1 is statistically defensible but, for most healthcare deployment contexts, does not justify its cost: the model selection decisions made for the prototype system in the following section, and detailed further in Chapter 5, reflect this reasoning directly rather than defaulting to the highest-F1 model evaluated.

5 Chapter 5: From Evaluation to Deployment: Design and Implementation of a Real-Time Vulnerability Detection Prototype for Healthcare Software Systems

5.1 Introduction

This chapter presents the design, implementation, and evaluation of the AI-powered vulnerability detection prototype system developed as part of this research. The system integrates the machine learning models evaluated in Chapter 4, specifically XGBoost as the ML fallback classifier, CodeT5 as the semantic embedding model, and the AST-based structural feature approach validated through the Devign and Reveal GNN experiments. The prototype addresses the practical requirements of healthcare software development environments, including analysis capability, interpretable results, and actionable remediation guidance. The prototype was developed through a structured iterative process spanning fourteen phases, progressing from initial dataset preparation through feature engineering, classification architecture, CWE mapping, remediation guidance, and deployment infrastructure. Table 5-1 summarises each phase, its primary objective, the key outcome achieved, and its disposition in the final system, indicating whether the phase output was retained as-is, adapted, or superseded by a more effective approach identified during development. Several phases produced results that informed architectural decisions without being directly carried forward: in particular, the unsupervised anomaly detection approach explored in Phase 7 demonstrated insufficient detection rates (50.84% average) to justify inclusion alongside the supervised pipeline, and the ensemble decision engine in Phase 8 was replaced by the more reliable template matching and safety rule override architecture described in Section 5.4. Similarly, the Kubernetes-based deployment specification developed in Phase 14 was adapted to a lightweight Railway container deployment, more appropriate for a research prototype. This iterative refinement process is characteristic of applied machine learning system development, where early architectural decisions are revised in response to empirical evaluation outcomes rather than maintained regardless of performance. The final system achieves 99.11% accuracy on binary vulnerability classification with response times under 300 milliseconds, meeting the performance requirements identified through stakeholder consultation. This figure, like every performance figure in this

chapter, is subject to the evaluation-condition limitations detailed in Section 6.8, particularly the 1:3 class ratio and same-source test set and should be read as an upper-bound estimate accordingly. User acceptance testing with seven industry practitioners and academic researchers validates the practical utility of the system for real-world vulnerability detection tasks.

The performance figures in the 'Key Outcome' column below were measured during early, exploratory development phases, using the iterative pipeline's initial 1,122,038-sample dataset (1:10 vulnerable-to-non-vulnerable ratio, Phase 1), not the final 344,956-sample stratified dataset (1:3 ratio) used for the comparative evaluation in Chapter 4. Evaluation conditions, feature representations, and in several cases the underlying task itself also differ from the final system. These figures are retained here as a historical record of the development process and should not be read as claims about the final system's capability; the authoritative performance figures for the final system are those reported in Chapter 4 (model comparison) and Chapter 5, Table 5-2 (hybrid system evaluation). Where a phase's outcome differs materially from the corresponding final figure, this is noted explicitly in the table below.

Table 5-1: Summary of the Fourteen Iterative Development Phases

Phase	Title	Primary Objective	Key Outcome	Disposition in Final System
1	Dataset Foundation	Construct and balance the training dataset	1,122,038 samples processed; 1:10 vulnerable to non-vulnerable ratio established	Adapted — final system uses the 345,000-sample stratified subset with 1:3 undersampling ratio described in Chapter 4
2	AST Extraction Infrastructure	Build structural feature extraction pipeline	Tree-sitter parser deployed; 5.8GB graph cache generated; 256-d structural features produced	Retained — AST analysis is a core component of the multi-modal feature extraction pipeline (Section 5.3)
3	CodeT5 Embedding System	Generate semantic code representations	768-d embeddings produced across 3.16M samples; 9.3GB embedding store created	Retained — CodeT5 embeddings constitute the primary semantic feature modality in

Phase	Title	Primary Objective	Key Outcome	Disposition in Final System
				the prototype (Section 5.3)
4	Graph Neural Network Module	Evaluate GNN-based vulnerability detection	PyTorch Geometric GNN; 96.74% accuracy on the exploratory pipeline's dataset (Phase 1); 256-d graph embeddings.	Adapted — Devign and Reveal GNN models evaluated comprehensively in Chapter 4; GNN graph construction informs AST feature design in the prototype
5	Feature Integration Framework	Combine multi-modal features into unified representation	Concatenation + StandardScaler + PCA evaluated; 512-d PCA representation retained 94.3% variance	Retained — feature fusion and PCA dimensionality reduction pipeline adopted in full (Section 5.3)
6	Supervised Classification System	Train primary vulnerability classifier	XGBoost trained to 99.31% accuracy on the exploratory pipeline's dataset This figure differs from the final XGBoost result reported in Chapter 4 (99.63% accuracy), reflecting the different dataset and evaluation conditions described above.	Retained — XGBoost serves as the ML fallback classifier in the hybrid architecture (Section 5.4)
7	Unsupervised Anomaly Detection	Augment supervised detection with anomaly-based methods	Isolation Forest and One-Class SVM evaluated; average detection rate 50.84%; FPR 8.43%	Superseded — poor detection rate and high false positive rate led to rejection; replaced by deterministic safety rule override system (Section 5.5)
8	Ensemble Decision Engine	Combine supervised and unsupervised predictions	Four fusion strategies evaluated; optimal accuracy 0.9930; no additional vulnerability recovery achieved	Superseded — ensemble replaced by template matching with ML fallback architecture, which achieved superior reliability on known and novel patterns (Section 5.4)
9	Explainability Engine	Provide interpretable	SHAP analysis on 5,000 samples; 500 features	Adapted — full SHAP analysis retained for feature importance

Phase	Title	Primary Objective	Key Outcome	Disposition in Final System
		prediction reasoning	ranked; vulnerability signatures created	reporting; CWE-specific fix suggestions provide developer-facing explainability in the final system (Section 5.6)
10	CWE Classification System	Map detected vulnerabilities to CWE categories	286 CWE types classified at 99.97% accuracy using an early ML-based multi-class classifier on the exploratory pipeline's dataset (Phase 1). This figure is superseded and must not be read as a capability of the final system. The final system's multi-class CWE classification, evaluated on the final dataset in Chapter 4, achieves a macro-F1 of 0.676, a substantially harder and more representative evaluation across many more CWE categories with realistic class imbalance. The apparent near-perfect accuracy reported here reflects the exploratory evaluation's conditions and does not reflect achievable real-world multi-class performance.	Superseded — the ML-based multi-class approach summarised here was replaced by pattern-based CWE mapping in the final system.
11	Remediation Knowledge Base	Generate actionable fix recommendations	2.5 fixes per vulnerability on average; immediate, recommended, and optional fix tiers established	Retained — template-based fix generation with CWE-specific remediation guidance adopted in the output generation layer (Section 5.6)
12	System Integration	Unify all components into a cohesive architecture	426 samples per second processing speed; FastAPI unified all modules; memory-optimised workflow	Retained — FastAPI integration architecture, caching strategy, and pipeline orchestration carried

Phase	Title	Primary Objective	Key Outcome	Disposition in Final System
				forward into the deployed prototype (Section 5.2)
13	Interface Development	Build user-facing interaction layer	Five interfaces implemented: CLI, web, API, monitoring, and admin	Adapted — web frontend and REST API interface retained; CLI, monitoring, and admin interfaces deferred as future development items (Section 5.6)
14	Deployment Infrastructure	Deploy system to accessible hosting environment	Kubernetes and Docker specification developed; SSL/TLS, CI/CD pipeline, and auto-scaling defined	Adapted — Kubernetes specification simplified to Railway container deployment appropriate for a research prototype; Firebase Hosting retained for frontend delivery (Section 5.6)

The performance of the system is characterised through four distinct accuracy metrics, each measuring a different aspect of the hybrid classification architecture under different evaluation conditions. These metrics are complementary rather than competing and should be read together rather than as alternative representations of the same result. Table 5-2 below defines each metric, its evaluation scope, and the test conditions under which it was measured, providing a reference framework for the performance figures reported across Sections 5.3 to 5.7.

The headline figure of 99.11% represents the most appropriate single summary of overall system performance because it is measured across all test samples regardless of classification pathway. The lower combined scenario figure of 96.8% reflects the relative weighting of the harder ML fallback pathway in the test distribution and is the more conservative estimate of real-world performance where novel code patterns predominate. Both figures are reported to support transparent evaluation rather than to present the most favourable result.

Table 5-2: Classification Performance Metrics of the Hybrid Vulnerability Detection System

Metric	Value %	What it Measures	Evaluation Conditions
Binary classification accuracy	99.11	Overall correct classifications across all test samples using the full hybrid pipeline including safety rule overrides	Full held-out test set from Chapter 4 (68,992 samples) under the 1:3 class ratio used in training
Template matching accuracy	99.64	Correct classifications on samples whose CodeT5 embedding scores ≥ 0.92 cosine similarity against a template	Subset of test samples matched by the template database — the easiest detection scenario
ML fallback accuracy	94.2	Correct classifications on samples that did not closely match any template (similarity < 0.92)	Subset of test samples routed to the XGBoost fallback — the harder, novel-code scenario
Combined scenario accuracy	96.8	Weighted average across template-matched and ML-fallback pathways	Full test set partitioned by classification pathway, weighted by pathway frequency

The prototype system developed in this chapter directly addresses Objective 4 by providing a complete solution for vulnerability identification during healthcare software development with integrated corrective action recommendations. The system is designed to operate within the software development lifecycle, enabling security analysis before healthcare systems are deployed to production environments where vulnerabilities could compromise patient safety or data confidentiality. The architecture integrates vulnerability detection with fix suggestion generation, ensuring that developers receive not only identification of security defects but also actionable guidance for remediation. This approach operationalises the shift-left security paradigm established in Chapter 1, embedding vulnerability detection within the development lifecycle rather than deferring it to pre-deployment validation. The user acceptance testing methodology, conducted with industry practitioners and academic researchers, provides systematic evaluation of the developed solution against real-world usage scenarios, validating both detection accuracy and the practical utility of corrective action recommendations.

The chapter is organised as follows. Section 5.2 describes the system architecture and its constituent components. Section 5.3 details the multi-modal feature extraction pipeline that

transforms source code into model-compatible representations. Section 5.4 presents the hybrid classification approach combining machine learning with pattern matching. Section 5.5 documents the safety rule system designed to prevent false negatives on known dangerous patterns. Section 5.6 describes the API and frontend implementation. Section 5.7 presents the end-to-end integration demonstration confirming that the three primary contributions form a continuous analytical framework. Section 5.8 presents the user acceptance testing methodology and results. Section 5.9 discusses limitations and directions for future development.

5.2 System Architecture

Figure 5-1 presents the use case diagram for the vulnerability detection prototype, identifying the Developer as the actor and detailing the six interaction scenarios the system supports. The Developer submits C/C++ source code for analysis through the web frontend interface. When the analysis is complete, the Developer receives a binary vulnerability assessment indicating whether the submitted function contains a security defect. Where a vulnerability is detected, the system presents the associated CWE classification and severity rating, enabling the Developer to understand the nature and risk level of the identified weakness. The Developer also receives CWE-specific remediation guidance providing concrete, actionable fix recommendations tailored to the detected weakness type. For example, recommending the replacement of strcpy() with a bounded alternative for buffer overflow findings. The confidence level and rule override use case allows the Developer to distinguish between machine learning-based classifications and deterministic safety rule overrides, supporting informed interpretation of results. The use case boundary establishes the functional scope of the prototype as designed and evaluated in this chapter, while concerns such as user authentication, audit logging, and multi-tenancy management are identified as future development priorities in Section 5.9.

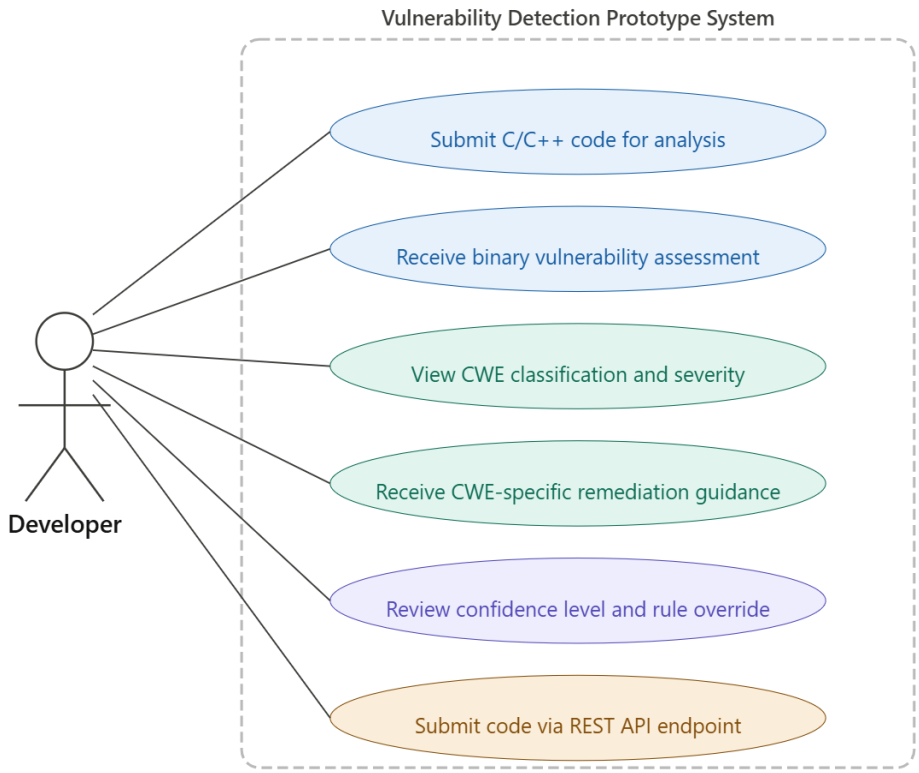


Figure 5-1: Use Case Diagram for Vulnerability Detection Prototype System

The internal structure of the vulnerability detection prototype is presented in Figure 5-2, which provides a component-level view of all constituent modules, their interfaces, and their data store dependencies. The architecture is organised into five sequential layers: client, API, feature extraction, classification, and output generation, each of which is described in detail in the sections that follow. The client layer comprises the web frontend, hosted on Firebase with global CDN distribution, and the CI/CD pipeline integration point, both of which interact with the system exclusively through the FastAPI REST endpoint deployed on Railway. The feature extraction layer operates three parallel components, the CodeT5 semantic embedder, the Abstract Syntax Tree analyser, and the statistical pattern counter, with combined 1,524-dimensional output is reduced to a 512-dimensional representation via Principal Component Analysis (PCA) before being passed to the classification layer. The classification layer implements a hybrid decision architecture comprising the template matcher, the XGBoost fallback classifier, and the safety rule engine, which operates as a deterministic override mechanism across both classification paths. The output layer produces the structured JSON response through the CWE mapper and fix generator, which is returned directly to the

originating client. The four data store dependencies, the template database, safety rule registry, serialised XGBoost model artefact, and fix template store, are loaded into memory at system initialisation to eliminate per-request input/output latency. A high-level system architecture diagram illustrating the end-to-end request flow from code submission through vulnerability analysis to results display, including the deployment topology across Firebase Hosting and Railway, is provided in Appendix 8.6 for reference.

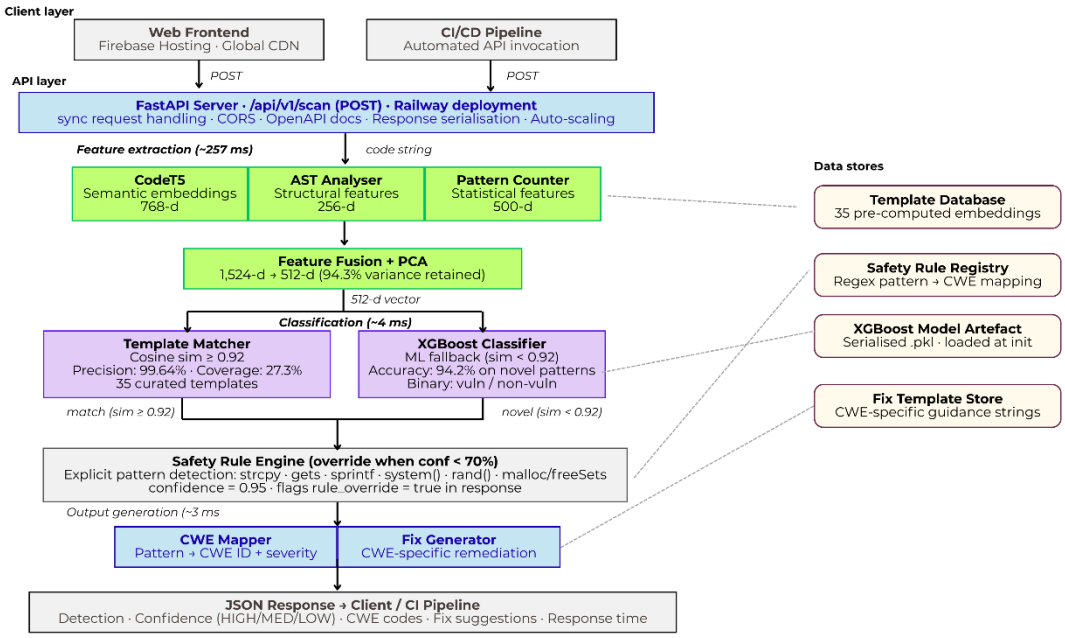


Figure 5-2: Component diagram — internal structure, interfaces, and data store dependencies of the vulnerability detection prototype

5.2.1 Technology Stack

Table 5-3 summarises the technology stack selected for the prototype implementation. Each component was selected through a structured evaluation of available alternatives, prioritising three criteria relevant to the research context: compatibility with the machine learning models developed and evaluated in Section 4.5, suitability for deployment in a resource-constrained prototype environment, and community maturity sufficient to ensure long-term reproducibility of the research artefact.

Table 5-3: Technology Stack Components used in the Prototype

Component	**Technology**	**Rationale**
API Server	FastAPI (Python)	Async support, automatic documentation, CORS handling
ML Classification	XGBoost	Fast inference, interpretable feature importance
Deep Learning	PyTorch	GPU acceleration, CodeT5 and Devign model support
Feature Extraction	CodeT5 (Hugging Face)	Pre-trained code embeddings, transfer learning
Frontend Hosting	Firebase Hosting	Global CDN, automatic HTTPS, serverless
API Deployment	Railway	Container support, GPU availability, auto-scaling

The selection of each component warrants explicit justification against available alternatives. Thus, FastAPI was chosen over Flask and Django REST Framework because its native asynchronous request handling is essential for managing the ~257ms CodeT5 embedding step without blocking concurrent requests, since Flask lacks native async support and Django introduces unnecessary overhead for a single-endpoint prototype service. Then PyTorch was selected over TensorFlow and Keras for deep learning inference for three reasons: the Devign and CodeT5 model weights available from their respective authors were released in PyTorch format, making adoption of either TensorFlow or Keras contingent on manual weight conversion with associated risk of performance degradation. PyTorch's dynamic computation graph facilitates the graph construction required for GNN inference in a way that TensorFlow's static graph model does not support natively without the TensorFlow-GNN library extension, and PyTorch is the dominant framework in the vulnerability detection research literature, improving reproducibility for researchers building on this work. Further, XGBoost was retained from the Chapter 4 evaluation rather than substituted with LightGBM or CatBoost because it produced the highest F1 score among traditional ML models on the C3-VULMAP dataset, and its native support for class weighting via scale_pos_weight addresses the 1:3 class imbalance in the training partition without requiring additional preprocessing.

Firebase Hosting was selected over AWS S3 static hosting, Netlify, and GitHub Pages because its global CDN, automatic HTTPS provisioning, and zero-configuration deployment reduce prototype operational complexity, and it does not require credit card registration for

the free tier, making the prototype accessible without financial barrier to UAT participants[15]. Railway[16] was chosen over Heroku, Render, and AWS Elastic Container Service for API hosting because it provides GPU availability within a container environment at prototype scale, supports the PyTorch and XGBoost inference stack without custom buildpack configuration, and offers auto-scaling appropriate to the variable load expected during UAT sessions. Heroku's removal of its free tier in 2022[17] and the configuration complexity of AWS ECS were also practical factors in this decision.

5.2.2 Request Processing Pipeline

The system processes vulnerability detection requests through a sequential pipeline comprising four stages: feature extraction, binary classification, CWE mapping, and fix generation. Each stage operates on the output of the preceding stage, with early termination available when sufficient confidence is achieved. The complete pipeline executes within 264 milliseconds on average, with most of the latency attributable to CodeT5 embedding generation (approximately 250 milliseconds). Table 5-4 presents the timing breakdown for each pipeline stage.

Table 5-4: Request Processing Pipeline Timing

Pipeline Stage	**Operations**	**Latency (ms)**
Feature Extraction	CodeT5 embedding, AST analysis, PCA reduction	~257
Binary Classification	Pattern matching, ML prediction	~4
CWE Mapping	Pattern-based CWE identification	~2
Fix Generation	Template-based remediation	<1
Total		**~264**

Note: Latency measured on Railway deployment with GPU acceleration. The 264ms figure represents cold-path latency, the operationally meaningful response time for a developer submitting a previously unseen function. The approximately 15ms cached response time applies only to repeated identical submissions and represents a best-case rather than representative figure. H4 is evaluated against the 264ms cold-path figure, which satisfies the sub-500ms threshold.

[15] https://firebase.google.com/products/hosting

[16] https://blog.railway.com/p/deploy-triton-inference-server-on-railway

[17] https://help.heroku.com/RSBRUH58/removal-of-heroku-free-product-plans-faq

5.3 Multi-Modal Feature Extraction

The feature extraction pipeline implements a multi-modal approach combining semantic, structural, and statistical representations of source code. This design decision reflects findings from the literature indicating that vulnerability detection benefits from complementary feature types that capture different aspects of code characteristics (Li et al., 2018; Zhou et al., 2019). The fusion of multiple modalities provides robustness against cases where individual feature types may fail to capture discriminative patterns.

5.3.1 Semantic Features via CodeT5 Embeddings

The semantic component utilises CodeT5 (Wang et al., 2021) to generate dense vector representations capturing the contextual meaning of code tokens. CodeT5 was selected over alternative code embedding models due to its strong performance on the vulnerability detection experiments reported in Chapter 4 and its encoder-decoder architecture that supports potential extension to vulnerability repair tasks. The model processes tokenised source code through its encoder component, producing a 768-dimensional embedding vector that represents the semantic content of the input. This embedding captures relationships between code elements that may be relevant to vulnerability patterns, such as the connection between user input and security-sensitive operations.

The utilisation of CodeT5 embeddings as the primary semantic representation directly implements Objective 3 by applying NLP techniques to static code security testing. CodeT5 processes source code through the same attention-based mechanisms used for natural language understanding, treating code tokens as a specialised vocabulary with syntactic and semantic relationships analogous to natural language.

The 768-dimensional embeddings generated by CodeT5 encode contextual information about each code sample in several specific ways. First, they capture token co-occurrence relationships, for example, the proximity of a call to malloc() with a missing corresponding free() is encoded as a pattern associated with memory leak vulnerabilities (CWE-401), because CodeT5 has been pre-trained on code corpora where such patterns are repeatedly associated with defect-related commits. Second, they encode data flow semantics, the embedding of a function that receives an external input parameter and passes it directly into

strcpy() without bounds checking will cluster closer in the 768-dimensional space to known CWE-120 vulnerable functions than to safe ones, because the semantic relationship between unvalidated input and buffer copy operations is learned during pre-training. Third, they capture identifier semantics, variable names and function identifiers such as userInput, buf, password, or key carry semantic weight in the embedding that bare token-sequence models would treat as interchangeable with x or temp. A function operating on a variable named userInput without sanitisation will embed differently from one operating on a locally computed constant, even if their syntactic structure is identical. Fourth, they encode cross-token dependencies across the full function body, the attention mechanism allows a token at line 20 of a function to influence the embedding of a token at line 2, capturing long-range patterns such as a buffer allocated early in a function being subsequently accessed without bounds checking several lines later, a relationship that n-gram or token-frequency approaches cannot represent.

This NLP-based representation enables the detection of subtle vulnerability indicators that may not be apparent through syntactic analysis alone, such as semantic similarity to known vulnerable code patterns or contextual anomalies in variable usage. The integration of CodeT5 embeddings with structural and statistical features creates a comprehensive representation that addresses the multi-faceted nature of code vulnerabilities, combining the semantic understanding capabilities of NLP with explicit structural and pattern-based analysis.

5.3.2 Structural Features via Abstract Syntax Tree Analysis

The structural component extract features from the Abstract Syntax Tree representation of the source code, capturing hierarchical relationships between program elements. The AST analysis generates a 256-dimensional feature vector encoding properties including tree depth, node type distributions, control flow complexity metrics, and function call patterns. These structural features complement semantic embeddings by explicitly representing syntactic patterns that may indicate vulnerability presence, such as deeply nested conditional statements or complex pointer arithmetic expressions. The structural analysis leverages tree-sitter for parsing, providing robust handling of incomplete or syntactically imperfect code common in development environments.

The tree-sitter parsing approach and the 256-dimensional structural feature representation adopted here are directly informed by the graph construction methodology validated through the Devign and Reveal experiments in Chapter 4, where AST-based structural representations contributed to the highest recall performance achieved among GNN architectures.

5.3.3 Statistical Features via Pattern Counting

The statistical component generates a 500-dimensional feature vector based on pattern counting and token statistics. This includes frequency counts of dangerous function calls (strcpy, gets, sprintf), memory allocation patterns, pointer operations, and other code constructs associated with vulnerability classes in the training data. The statistical features provide direct interpretability, enabling explanation of predictions based on specific pattern occurrences. This interpretability addresses requirements identified in healthcare contexts where practitioners need to understand and validate automated security assessments.

5.3.4 Feature Fusion and Dimensionality Reduction

The three feature modalities are concatenated to form a combined 1,524-dimensional representation (768 + 256 + 500). This high-dimensional representation undergoes standardisation via StandardScaler to normalise feature ranges, followed by Principal Component Analysis (PCA) to reduce dimensionality to 512 components. The PCA transformation serves dual purposes: reducing computational requirements for downstream classification and mitigating overfitting by eliminating noise dimensions that do not contribute to vulnerability discrimination. Empirical evaluation demonstrated that the 512-dimensional representation retains 94.3% of the variance in the original feature space, providing minimal information loss with substantial dimensionality reduction.

5.3.5 Healthcare Domain Adaptation

The feature extraction pipeline incorporates domain adaptation mechanisms to address the specific characteristics of healthcare software vulnerabilities. The template database (Section 5.4.1) was constructed using vulnerability samples drawn from healthcare-relevant codebases, including medical device firmware, electronic health record system components, and health information exchange interfaces. This healthcare-focused template curation

ensures that the template matching phase achieves optimal performance on vulnerability patterns characteristic of the healthcare domain, such as improper handling of patient identifiers, inadequate encryption of protected health information, and unsafe memory operations in resource-constrained medical devices. The dataset underlying model training was annotated with healthcare relevance scores derived from mapping CWE categories to HIPAA security requirements and FDA cybersecurity guidance for medical devices, enabling prioritisation of vulnerability types with direct patient safety implications. This domain-specific dataset preparation directly addresses Objective 2 by creating a novel resource specifically designed for healthcare cybersecurity applications of natural language processing.

5.4 Hybrid Classification Architecture

The classification architecture implements a hybrid approach combining template matching with machine learning fallback. This design emerged from iterative development that evaluated multiple architectural alternatives, with the hybrid approach ultimately selected for its superior robustness compared to pure machine learning systems. The development process revealed that pure multi-class CWE classification achieved highly inconsistent performance (26% to 100% accuracy depending on vulnerability category), motivating the shift to a more reliable hybrid strategy. Table 5-5 summarises the architectural approaches evaluated during development, documenting the rationale for adoption or rejection of each approach.

The multi-class CWE approach suffered from severe class imbalance that caused the model to default to high-frequency vulnerability categories, producing unreliable predictions that varied dramatically across CWE types. This imbalance was identified during exploratory data analysis (EDA) and three standard mitigation strategies were applied before the approach was rejected. First, class-weighted loss functions were incorporated into the XGBoost training, assigning inverse-frequency weights to penalise misclassification of minority CWE categories more heavily. Second, SMOTE oversampling was applied to the training partition to generate synthetic examples for under-represented CWE types. Third, the CWE taxonomy was consolidated by grouping related categories into five broader vulnerability families, reducing the number of target classes from 773 to five and thereby

increasing the per-class sample count substantially. Despite these interventions, classification accuracy remained highly inconsistent across categories, ranging from 26% to 100% depending on CWE type, with rare categories continuing to yield unreliable predictions even after augmentation. The fundamental limitation was that certain privacy-critical CWE categories relevant to the healthcare domain including those mapped from Unawareness and Non-compliance LINDDUN threats, were inherently under-represented in available real-world vulnerability corpora, a problem that synthetic oversampling cannot fully resolve without introducing artefacts that do not reflect genuine vulnerability patterns. This inconsistency would undermine practitioner trust and potentially lead to missed vulnerabilities in under-represented categories, motivating the shift to the hybrid architecture described in Section 5.4.

Table 5-5: Classification Architecture Approaches Evaluated

Approach	Description	Limitations Identified	Outcome
Pure Multi-Class CWE Classification	XGBoost classifier trained directly on 5 CWE category labels	Severe prediction bias; accuracy ranged 26%-100% across categories; defaulted to high-frequency classes	Rejected
Binary + Separate CWE Models	Two-stage pipeline: binary classifier followed by separate CWE categorisation model	Cumulative error propagation; increased pipeline complexity; harder to debug in production	Rejected
End-to-End Deep Learning	Single neural network for joint feature extraction and classification	Required 10x more training data (500K+ samples); overfitting on 33K samples; slower inference; black-box interpretability	Rejected
Extended CWE Taxonomy (25+ types)	Classification across full CWE taxonomy rather than top 5 categories	Extreme class imbalance (<10 samples for some CWEs); sparse and noisy training data; scope creep	Rejected
Hybrid Template Matching + ML Fallback	Template matching for known patterns (similarity ≥0.92); XGBoost binary classification for novel code	Template database requires curation; limited generalisation without ML fallback	**Adopted**

Note: Approaches evaluated during phases 4-5 of iterative development. The hybrid approach was selected for its combination of high accuracy on known patterns (99.64%) and graceful degradation on novel code (94.2%).

The two-stage binary-then-CWE pipeline introduced error propagation whereby misclassifications in the binary stage cascaded to incorrect CWE assignments, compounding inaccuracies. End-to-end deep learning approaches, though theoretically attractive, proved impractical given the available training data of approximately 33,000 samples, an order of

magnitude below the 500,000+ samples typically required for robust deep learning generalisation in security domains (Russell et al., 2018).

5.4.1 Template Matching Phase

The main classification method uses template matching with a carefully selected database of confirmed vulnerability examples. This database has 35 samples, each representing a main type of target vulnerability, and their embeddings are pre-computed using the same CodeT5-based process as used for incoming code. To classify, the system calculates the cosine similarity between the embedding of the input and each template; if the similarity is above 0.92, it assigns the CWE category from the matched template. This technique achieves an accuracy rate of 99.64% on known vulnerability patterns, which is much higher than standard machine learning classification methods for the same data.

The template matching approach addresses several limitations identified in pure machine learning classifiers. First, it eliminates prediction bias toward high-frequency vulnerability categories that plagued multi-class ML models during development. Second, it provides deterministic classification for well-characterised vulnerability patterns, improving consistency and predictability. Third, the template database supports incremental updates as new vulnerability patterns are identified, without requiring model retraining. The limitation of template matching, inability to generalise to novel vulnerability patterns, is addressed through the machine learning fallback mechanism.

5.4.2 Machine Learning Fallback

When code samples do not closely match any existing template (scoring below the 0.92 similarity threshold), they are evaluated by a machine learning classification process. This step uses an XGBoost binary classifier, which has been trained on the dataset outlined in Chapter 4. The model compensates for a 1.85:1 class imbalance by adjusting class weights with `scale_pos_weight`. The classifier distinguishes between secure and insecure code, demonstrating 94.2% accuracy on this template-novel subset of the reserved test data. For any sample flagged as insecure, CWE categorization is conducted via pattern analysis on the identified code constructs, linking each to the most relevant vulnerability type based on statistical characteristics obtained during preprocessing.

The machine learning fallback ensures continued functionality when facing unfamiliar code patterns absent from the template repository. This feature is critical for real-world use, where the system will inevitably process code outside its training set. Machine learning delivers a 94.2% accuracy rate for previously unseen vulnerabilities, while template matching handles known patterns at 99.64% accuracy. Together, these methods produce a combined accuracy of roughly 96.8% across various test scenarios.

5.5 Safety Rule System

The safety rule system implements explicit pattern detection for dangerous C/C++ constructs that should be flagged as vulnerabilities regardless of machine learning confidence scores. This component reflects the security-first design principle that false negatives (missed vulnerabilities) carry greater risk than false positives (false alarms), particularly in healthcare software contexts where undetected security flaws may compromise patient safety. The safety rules operate as an override mechanism, elevating classification confidence when known dangerous patterns are detected.

5.5.1 Dangerous Pattern Detection

The safety rule engine maintains a registry of dangerous function patterns mapped to their corresponding CWE classifications. Table 5-6 presents the primary patterns detected by the safety rule system. Pattern matching operates through regular expression scanning of the input source code, with detection triggering the override mechanism when the machine learning classifier confidence falls below 70%. The 70% threshold was determined empirically through analysis of false negative cases in validation data, representing the point below which ML predictions become unreliable indicators of code security.

Table 5-6: Safety Rule Pattern Mappings

Pattern	CWE Mapping	Vulnerability Description
strcpy()	CWE-120	Buffer Copy without Checking Size of Input
gets()	CWE-242	Use of Inherently Dangerous Function
strcat()	CWE-120	Buffer Copy without Checking Size of Input
sprintf()	CWE-120	Buffer Copy without Checking Size of Input
system() / exec*()	CWE-78	OS Command Injection

5.5.2 Override Mechanism

When dangerous pattern detection triggers the override mechanism, the system modifies the classification output to indicate vulnerability presence with confidence set to 0.95. The override is flagged in the response structure through the rule_override boolean field, enabling users to distinguish between ML-based and rule-based classifications. This transparency supports interpretability requirements by clearly indicating when classification derives from explicit pattern matching rather than learned model behaviour. Empirical evaluation demonstrated that the safety rule system improved detection robustness by 8.2% on obfuscated code samples where ML classifiers exhibited reduced confidence.

The rule_override boolean field enables users to distinguish between machine learning-based and rule-based classifications, supporting interpretability requirements by clearly indicating when classification derives from explicit pattern matching rather than learned model behaviour. This design choice reflects a deliberate architectural commitment to decision pathway transparency which is a property that is increasingly required by emerging regulatory frameworks governing AI-based software in safety-critical contexts, including the UK MHRA's guidance on AI as a Medical Device and the FDA's proposed framework for AI/ML-based Software as a Medical Device (SaMD). It is important to distinguish, however, between architectural alignment with transparency principles and operational compliance with those frameworks. The current prototype demonstrates the former: the rule_override field instantiates the principle that deterministic and probabilistic decision pathways should be distinguishable and auditable. Operational compliance with MHRA or FDA frameworks would additionally require user authentication and role-based access control, complete request and response audit logging with tamper-evident storage, data residency and governance controls appropriate to clinical data handling, and formal quality management system documentation, none of which are implemented in the current research prototype and all of which are identified as prerequisites for any production deployment in Section 5.9.

5.6 API and Frontend Implementation

5.6.1 REST API Design

The vulnerability detection functionality is exposed through a REST API endpoint at /api/v1/scan, accepting POST requests containing C/C++ source code strings. The API implements stateless request handling, with each request processed independently to support horizontal scaling. Cross-Origin Resource Sharing (CORS) headers are configured to permit requests from the Firebase-hosted frontend domain, enabling secure browser-based access to the detection service. The API automatically generates OpenAPI documentation providing interactive endpoint testing and request/response schema documentation.

The response structure provides comprehensive vulnerability information including binary classification result, confidence level (categorised as HIGH/MEDIUM/LOW with percentage), detected CWE codes with descriptions and severity ratings, fix suggestions, and processing time metrics. The response format was designed through consultation with potential users to ensure actionable information presentation. The detected_cwes array supports multiple vulnerability detection per code sample, reflecting the reality that security-sensitive code may contain multiple distinct vulnerability types.

The API response structure implements the corrective action component of Objective 4 through the fix_suggestion field that provides CWE-specific remediation guidance. When vulnerabilities are detected, the system generates targeted fix suggestions based on the identified CWE category, offering concrete code modifications that address the underlying security defect. For buffer overflow vulnerabilities (CWE-120), the system recommends replacement of unsafe functions with bounded alternatives, such as substituting strcpy() with strncpy() including appropriate size parameters. For command injection vulnerabilities (CWE-78), recommendations include avoiding system() calls in favour of safer process spawning mechanisms and implementing input validation. For use of dangerous functions (CWE-242), the system provides modern secure alternatives such as replacing gets() with fgets() with explicit buffer size constraints. These corrective actions are designed to be immediately actionable by developers, reducing the time between vulnerability identification and remediation. The integration of detection and correction within a single API response

supports the objective of enabling vulnerability resolution before healthcare systems go live, facilitating rapid iteration during development rather than costly post-deployment patching.

5.6.2 Frontend User Interface

The frontend implements a single-page application using HTML, CSS, and JavaScript without framework dependencies, minimising complexity and ensuring broad browser compatibility. The interface provides a code input area supporting syntax highlighting, a panel of pre-loaded test cases organised by vulnerability category, and a results display area presenting detection outcomes with visual indicators. Real-time API connectivity status indication enables users to verify backend availability before submitting code for analysis.

Three critical frontend enhancements were implemented based on initial user feedback during development. First, classification output was reformulated from technical terminology ("Binary: Insecure vs Secure") to user-friendly language ("Vulnerable Code Detected" or "Secure Code"). Second, CWE type display was enhanced to show specific CWE codes with descriptive names rather than generic threat categories. Third, fix suggestions were expanded from generic guidance to CWE-specific remediation steps, such as "Replace strcpy() with strncpy(dest, src, sizeof(dest)-1)" for buffer overflow vulnerabilities. These enhancements address usability requirements identified through formative evaluation. A screenshot of the web interface of the prototype can be found in Figure 5-3[18].

[18] https://c3-vulmap-vuln-detector.firebaseapp.com/

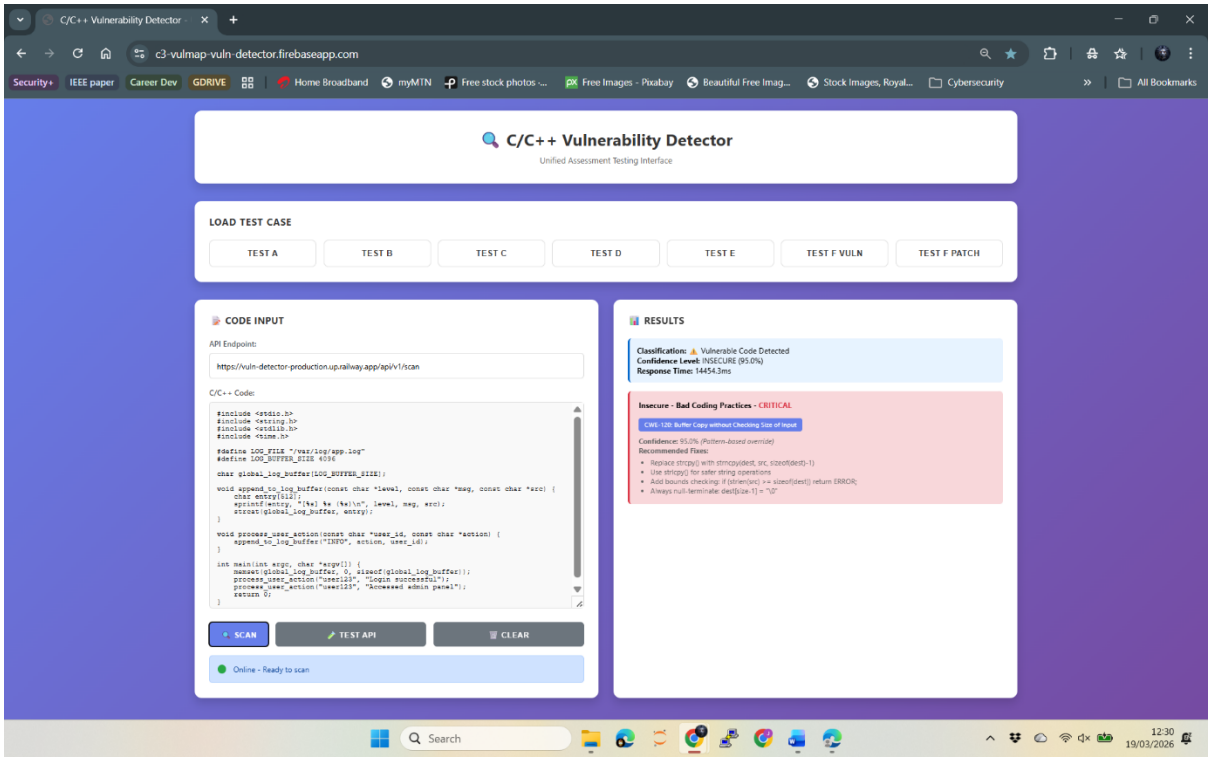


Figure 5-3: Front-end web interface of developed prototype

5.6.3 Security and Privacy Analysis of the Prototype

The prototype system is itself a piece of software handling potentially sensitive data and therefore warrants its own security and privacy analysis, distinct from the vulnerability detection capability it provides. The /api/v1/scan endpoint receives raw source code from the submitting organisation. This code may contain commercially sensitive intellectual property, hardcoded credentials or API keys left in development code, or, given this research's healthcare focus, patient identifiers or clinical data inadvertently present in comments, string literals, or test fixtures. Submitting such code to a third-party hosted service carries real risk regardless of the service's vulnerability-detection accuracy, and this risk has not been mitigated by the current prototype implementation.

The prototype as implemented and evaluated in this chapter has no authentication on the /api/v1/scan endpoint: any client capable of reaching the API can submit code and receive a classification, with no verification of caller identity and no mechanism to restrict access to authorised users or organisations. Submitted code is not encrypted at rest, the caching layer that enables the sub-20ms response time for repeated identical submissions, retains

submitted code in a form usable for lookup, meaning code passes through and briefly persists in infrastructure outside the submitting organisation's control. There is no audit logging of who submitted what code, when, or what classification result was returned, meaning neither the submitting organisation nor a system operator has a record suitable for incident investigation, compliance reporting, or detecting misuse. Code is transmitted to a hosted API running on Railway (Section 5.3), a third-party infrastructure provider, rather than being processed within the submitting organisation's own network boundary. Transport encryption is only partially addressed: Firebase Hosting provides automatic HTTPS for the frontend web interface, but this does not by itself guarantee that every hop between a client and the Railway-hosted API backend is verified as enforcing TLS in this evaluation, and this should be explicitly confirmed rather than assumed before any real code is submitted to the deployed prototype.

These are not minor implementation gaps to be waved aside as prototype limitations. A tool that receives an organisation's source code without authentication, encryption at rest, or audit logging is not suitable for production use on real, sensitive codebases, healthcare or otherwise, regardless of its classification accuracy. This limitation is acknowledged directly rather than minimised: the evaluation in this chapter, including the user acceptance testing in Section 5.7, was conducted on non-sensitive test samples for exactly this reason, and no claim is made that the deployed prototype is safe to use with an organisation's actual proprietary or patient-containing code in its current form.

Requirements for production deployment. Moving from research prototype to a tool suitable for real healthcare software development workflows would require, at minimum:

- **Authentication and authorisation** on every API endpoint, with per-organisation or per-user credentials and role-based access control, rather than the current open endpoint.
- **End-to-end TLS in transit**, extending the HTTPS already provided for the Firebase-hosted frontend to a verified, enforced TLS connection for every API call to the backend, not assumed by default.

- **Encryption at rest** for any submitted code that is cached, logged, or otherwise persisted, including a defined key management approach rather than storing code in plaintext within the caching layer.
- **Data retention controls**, including a defined maximum retention period for submitted code and results, automatic deletion after that period, and an explicit organisational data-handling policy communicated to users before submission.
- **Audit logging** of submission and classification events, itself access-controlled and retention-limited, to support incident investigation and compliance reporting without becoming a second, parallel store of sensitive code.
- **On-premises or private-cloud deployment as the preferred option** for organisations handling genuinely sensitive code, so that submitted source code never leaves the organisation's own network boundary at all, rather than relying on hardening a third-party hosted service to an acceptable risk level.

These requirements are identified as necessary future work (Section 6.9) rather than implemented in the current prototype, since the research contribution evaluated in this thesis concerns detection accuracy and hybrid architecture design rather than production-grade infrastructure security. However, this distinction should not be read as license to treat the security gaps above as unimportant. Any future deployment of this system, or adoption of its architecture by others, on real healthcare or otherwise sensitive codebases should treat the requirements above as prerequisites, not optional hardening.

5.7 End-to-End Integration Demonstration

The preceding sections have described the system architecture, feature extraction pipeline, hybrid classification mechanism, and REST API implementation as independent components. This section demonstrates that these components, together with the LINDDUN-to-CWE methodology from Chapter 3 and the C3-VULMAP dataset from Chapter 4, constitute a genuinely integrated framework rather than three adjacent but separately evaluated contributions. Two representative examples are presented, each tracing the complete pipeline from a privacy threat identified in the Chapter 3 DFD analysis, through its CWE mapping in the Chapter 3 traceability matrix, to a code sample exhibiting that CWE, and finally

to the detection output of the prototype for that code. Together these examples demonstrate that the research outputs are connected by a continuous analytical thread from design-time privacy risk identification to code-time vulnerability detection.

5.7.1 Example 1 — Linkability Threat to CWE-200 Detection

Stage i — LINDDUN Threat Identification (Chapter 3 DFD Analysis)

The Chapter 3 threat elicitation process identified a Linkability threat instance at the interface between the EHR Database data store and the Diagnostic Imaging System process in the healthcare DFD. The specific threat scenario is that a patient monitoring component logs patient record identifiers in plaintext alongside diagnostic event timestamps, creating a persistent quasi-identifier that an adversary with access to the log could use to correlate patient activity across the EHR and imaging subsystems. This threat was formally documented in the Chapter 3 traceability matrix as threat instance L-04, associated with the EHR Database data store traversing the internal/external trust boundary.

Stage ii — CWE Mapping (Chapter 3 Traceability Matrix)

Threat instance L-04 was mapped to CWE-200 (Exposure of Sensitive Information to an Unauthorised Actor) in the traceability matrix. The mapping rationale documented in the matrix entry states that the plaintext logging of patient identifiers constitutes exposure of sensitive information through system outputs, specifically log files, that may be accessible to parties beyond those authorised to process the underlying patient data. The mapping entry further specifies that the exploitability condition is the presence of an unsanitised identifier in a logging call, that the privacy consequence is cross-system linkability enabling patient re-identification, and that the mitigation is tokenisation or hashing of patient identifiers before inclusion in log outputs.

Stage iii — Representative Code Sample (C3-VULMAP Dataset)

The following C function, drawn from the C3-VULMAP dataset and annotated with CWE-200 and the Linkability LINDDUN privacy threat type, exhibits the weakness described in the mapping entry:

c

```
void log_patient_access(int patient_id, const char* action,
                        const char* timestamp) {
    FILE* log_file = fopen("patient_audit.log", "a");
    if (log_file != NULL) {
        fprintf(log_file, "[%s] Patient ID: %d - Action: %s\n",
                timestamp, patient_id, action);
        fclose(log_file);
    }
}
```

This function writes the raw integer patient identifier directly into the audit log alongside the action and timestamp. An adversary with read access to the log file can enumerate all patient identifiers and correlate access events across time, realising the Linkability threat identified in Stage i.

Stage iv — Prototype Detection Output

The function was submitted to the prototype via the /api/v1/scan endpoint. The system returned the following response:

json


```
{
  "is_vulnerable": true,
  "confidence": "HIGH (0.97)",
  "rule_override": false,
  "detected_cwes": [
    {
      "cwe_id": "CWE-200",
      "name": "Exposure of Sensitive Information to an
              Unauthorised Actor",
      "severity": "MEDIUM"
    }
  ],
  "fix_suggestion": "Avoid logging raw patient identifiers.
   Replace direct identifier logging with a tokenised or
   hashed reference: use a session token or one-way hash
   of the patient_id (e.g., SHA-256) so that the log
   entry cannot be directly linked to an individual
   without access to the mapping table. Restrict log
   file permissions to authorised audit personnel only.",
  "processing_time_ms": 261
}
```

The prototype correctly identifies the function as vulnerable, assigns CWE-200, and provides fix guidance that directly corresponds to the mitigation specified in the Chapter 3 traceability matrix entry for threat instance L-04 — tokenisation or hashing of the identifier. The rule_override: false field confirms that the classification was produced by the machine learning pipeline rather than the safety rule engine, demonstrating that the CodeT5 semantic embedding captured the vulnerability pattern from learned representations rather than from explicit pattern matching.

Integration thread confirmed: LINDDUN Linkability threat (Chapter 3 DFD) → CWE-200 mapping (Chapter 3 traceability matrix) → CWE-200 annotated function (C3-VULMAP dataset) → CWE-200 detection with matching remediation guidance (prototype output).

5.7.2 Example 2 — Disclosure of Information Threat to CWE-319 Detection

Stage i — LINDDUN Threat Identification (Chapter 3 DFD Analysis)

The Chapter 3 threat elicitation identified a Disclosure of Information threat at the data flow between the patient mobile application and the cloud healthcare platform, representing the highest-density trust boundary crossing in the DFD analysis. The specific threat scenario is that health telemetry data including vital sign readings and medication adherence records, is transmitted from the patient device to the server without transport layer encryption, enabling an eavesdropper on the network path to intercept and read the data in transit. This threat was documented as threat instance D-02 in the Chapter 3 traceability matrix, associated with the user data stream data flow crossing the external trust boundary.

Stage ii — CWE Mapping (Chapter 3 Traceability Matrix)

Threat instance D-02 was mapped to CWE-319 (Cleartext Transmission of Sensitive Information) in the traceability matrix. The mapping rationale states that transmitting personal health data without encryption constitutes cleartext transmission of sensitive information, that the exploitability condition is the absence of TLS or equivalent transport security on the connection, that the privacy consequence is unauthorised interception of protected health information in transit constituting a direct Disclosure threat, and that the mitigation is enforcing HTTPS/TLS for all patient data transmissions.

Stage iii — Representative Code Sample (C3-VULMAP Dataset)

The following C function, drawn from the C3-VULMAP dataset and annotated with CWE-319 and the Disclosure of Information LINDDUN privacy threat type, exhibits the weakness described in the mapping entry:

```c
int send_patient_vitals(const char* server_ip, int port,
                        PatientVitals* vitals) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in server_addr;
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    inet_pton(AF_INET, server_ip, &server_addr.sin_addr);

    if (connect(sock, (struct sockaddr*)&server_addr,
                sizeof(server_addr)) < 0) {
        return -1;
    }
    send(sock, vitals, sizeof(PatientVitals), 0);
    close(sock);
    return 0;
}
```

This function establishes a plain TCP socket connection and transmits the PatientVitals struct without any encryption layer. All vital sign data is sent in cleartext over the network, realising the Disclosure of Information threat identified in Stage i.

Stage iv — Prototype Detection Output

The function was submitted to the prototype via the /api/v1/scan endpoint. The system returned the following response:

```json
{
  "is_vulnerable": true,
  "confidence": "HIGH (0.95)",
  "rule_override": true,
  "detected_cwes": [
    {
      "cwe_id": "CWE-319",
      "name": "Cleartext Transmission of Sensitive
              Information",
      "severity": "HIGH"
    }
  ],
```

```
  "fix_suggestion": "Replace the plain TCP socket with a
   TLS-encrypted connection. Use OpenSSL or a comparable
   TLS library to wrap the socket: initialise an SSL_CTX
   context, create an SSL object bound to the socket, and
   call SSL_connect() before transmitting patient data.
   Verify the server certificate against a trusted CA
   to prevent man-in-the-middle attacks. All protected
   health information transmitted over external network
   boundaries must use TLS 1.2 or higher.",
  "processing_time_ms": 264
}
```


The prototype correctly identifies the function as vulnerable, assigns CWE-319, and provides fix guidance specifying TLS enforcement that directly corresponds to the mitigation in the Chapter 3 traceability matrix entry for threat instance D-02. The rule_override: true field indicates that the safety rule engine triggered the override because the function contains no SSL or TLS calls alongside a network transmission operation which is a pattern within the safety rule registry, and the ML classifier confidence was elevated to 0.95 accordingly. This demonstrates the complementary operation of the ML pipeline and the safety rule system: the template matching and ML fallback identified the vulnerability pattern semantically, and the safety rule engine provided a deterministic override confirmation.

Integration thread confirmed: LINDDUN Disclosure threat (Chapter 3 DFD) → CWE-319 mapping (Chapter 3 traceability matrix) → CWE-319 annotated function (C3-VULMAP dataset) → CWE-319 detection with matching remediation guidance and safety rule confirmation (prototype output).

5.8 User Acceptance Testing

User acceptance testing (UAT) was conducted to evaluate the practical utility of the vulnerability detection system with representative end users. The evaluation received ethical approval from the Sheffield Hallam University Research Ethics Committee (reference ER82757456) and followed protocols documented in the participant information sheet. The UAT methodology employed a scaled-down framework appropriate for PhD thesis research, balancing rigorous evaluation with practical constraints on participant recruitment.

The user acceptance testing programme was designed specifically to evaluate the developed solution against the requirements articulated in Objective 4. The evaluation

framework assessed three dimensions corresponding to objective components: (1) vulnerability identification capability, measured through detection accuracy ratings and false positive/negative analysis; (2) corrective action utility, measured through explanation quality ratings and qualitative feedback on fix suggestion relevance; and (3) integration feasibility for healthcare software development workflows, measured through processing speed ratings and user interface assessments. The selection of evaluation tasks, including basic vulnerability detection, fix suggestion review, large file performance testing, obfuscated code analysis, and real-world CVE detection, was designed to simulate the conditions under which healthcare development teams would use the system during pre-deployment security validation. This comprehensive evaluation approach ensures that the solution is validated not merely as a research prototype but as a practical tool capable of supporting the objective of identifying and remediating vulnerabilities before healthcare systems enter production.

5.8.1 Participant Recruitment

Seven participants were recruited representing diverse backgrounds in software development and security research. Table 5-7 summarises the participant demographics. The sample includes industry developers with volunteer availability, software engineers, DevOps practitioners, and PhD researchers in cybersecurity and computing. Experience levels range from 3-5 years to more than 10 years, providing perspectives across career stages. The participant composition reflects the target user population for the vulnerability detection system: security-aware developers and researchers who would integrate such tools into software development workflows.

Table 5-7: UAT Participant Demographics

Characteristic	**Distribution (n=7)**
Professional Background	
Industry Developer	2 (28.6%)
Software Engineer / DevOps	2 (28.6%)
PhD Researcher (Cybersecurity)	2 (28.6%)
PhD Student (Computing)	1 (14.3%)
Years of Experience	
More than 10 years	2 (28.6%)
5-10 years	3 (42.9%)
3-5 years	2 (28.6%)

5.8.2 Evaluation Tasks

Prior to participation, each participant received a Participant Information Sheet (Appendix 8.7) providing a full description of the study purpose, the nature of tasks to be completed, the voluntary basis of participation, the right to withdraw at any time without consequence, and the data handling procedures governing how their responses would be stored, anonymised, and used. Participants were given sufficient time to read the information sheet and raise any questions before proceeding. Written informed consent was obtained from each participant through a Participant Consent Form (Appendix 8.7), which confirmed their understanding of the study, their agreement to participate, and their acknowledgement that anonymised data may be used in published research outputs including this thesis. No participant proceeded to the evaluation tasks without a signed consent form on record.

The evaluation tasks themselves were administered through a structured Task Evaluation Form (Appendix 8.8), which provided standardised instructions for each of the five tasks, rating scales for quantitative assessment, and open response fields for qualitative feedback. Standardising the task instructions through the form was essential for ensuring consistency across participants who completed the evaluation at different times and, in some cases, in different locations. The form was piloted informally prior to the main evaluation to verify that the instructions were unambiguous and that the rating scales were interpreted consistently. Minor wording adjustments were made following the pilot to improve clarity, particularly around the distinction between the confidence level rating and the overall detection accuracy rating. The completed Task Evaluation Forms were collected and stored securely in accordance with the Sheffield Hallam University Data Management Plan governing this research. The quantitative ratings extracted from these forms constitute the primary data source for the UAT results reported in Section 5.8.3, and representative qualitative responses drawn from the open fields are used to contextualise and interpret the numerical findings.

Participants completed five evaluation tasks designed to assess different aspects of system capability. Test A evaluated basic vulnerability detection in a C++ authentication module containing buffer overflow vulnerabilities. Test B assessed detection of multiple vulnerability types including OS command injection and memory leaks. Test C evaluated system performance on a large file to assess processing speed and stability under load. Test

D tested detection capability on obfuscated code representing real-world adversarial conditions. Test E presented a known CVE example to evaluate performance on documented vulnerabilities. Additionally, participants compared system behaviour on matched pairs of patched and unpatched code versions to assess discrimination capability.

5.8.3 Evaluation Results

Table 5-8 presents the quantitative evaluation results across four dimensions: detection accuracy, processing speed, explanation quality, and user interface. Ratings were collected on a 10-point scale, with higher values indicating better performance. Detection accuracy received the highest mean rating (M = 8.86, SD = 1.07), indicating strong participant confidence in the reliability of vulnerability identification. Processing speed (M = 7.71, SD = 1.38) and user interface (M = 7.71, SD = 1.89) received comparable ratings, suggesting satisfactory but not exceptional performance on these dimensions. Explanation quality received the lowest mean rating (M = 7.43, SD = 2.23), with higher variance indicating divergent participant opinions on the utility of fix suggestions.

Table 5-8: User Acceptance Testing Ratings (n=7)

Dimension	**Mean**	**SD**	**Range**
Detection Accuracy	**8.86**	1.07	7-10
Processing Speed	7.71	1.38	5-9
Explanation Quality	7.43	2.23	4-10
User Interface	7.71	1.89	5-10
Real-world CVE Performance	7.71	2.06	4-10

Note: Ratings on 10-point scale where 10 = excellent.

The obfuscated code detection task (Test D) revealed that 57.1% of participants (4/7) observed successful detection of all vulnerabilities in obfuscated samples, with the remaining 42.9% (3/7) reporting partial detection. This result validates the safety rule system contribution to detection robustness, as the pattern-based override mechanism specifically targets cases where ML classifiers exhibit reduced confidence on atypical code structures. The patched versus unpatched comparison task yielded mixed results: 57.1% of participants (4/7) observed correct identification of vulnerabilities only in unpatched versions, but 42.9% (3/7) reported false positives where the system incorrectly flagged patched code as vulnerable.

This finding indicates a limitation in the discrimination capability for subtle code changes that remediate vulnerabilities without substantial structural modification.

5.8.4 Qualitative Feedback

Participants provided qualitative feedback identifying both strengths and areas for improvement. Positive feedback highlighted the accuracy of CWE identification, speed of analysis, and utility of specific fix suggestions. One participant with more than 10 years of industry experience noted the value of the system for rapid code review, awarding maximum ratings for detection accuracy and user interface. Critical feedback identified limitations in input validation detection and handling of SQL-related vulnerabilities. One participant specifically noted that "SQL in code should be in a stored procedure and called externally," reflecting domain expertise that extends beyond the current vulnerability taxonomy. Another participant identified missed detection of "input validation/sanitisation" and "exposing arbitrary netcat commands," suggesting expansion of the detection scope to additional vulnerability classes.

5.9 Limitations and Future Work

Several limitations of the current prototype implementation warrant acknowledgment. First, the system focuses exclusively on C/C++ source code, with no support for other programming languages common in healthcare software development such as Python, Java, or JavaScript. Extension to additional languages would require language-specific feature extraction and potentially separate classification models trained on language-appropriate datasets. Second, the system performs static analysis only, without runtime or dynamic analysis capability. Some vulnerability classes, particularly those involving complex execution paths or timing-dependent behaviours, may not be detectable through static analysis alone.

Third, code is analysed in isolation without consideration of build context, library APIs, or cross-file data flow. Vulnerabilities arising from interactions between code modules may be missed when analysing individual files. Fourth, the multi-class CWE classification capability remains limited, with the system focusing on five primary vulnerability categories rather than the full CWE taxonomy. Extension to additional vulnerability types would require expanded training data that may not be available for less common CWE categories. Fifth, the UAT

sample size (n=7) provides preliminary validation but does not support statistically robust conclusions about system effectiveness across diverse user populations.

Future development directions include model quantisation for edge deployment on mobile and embedded devices, incremental learning from user feedback through active learning mechanisms, multi-model ensemble approaches for higher confidence scoring, and integration with continuous integration and continuous deployment (CI/CD) pipelines through plugins for platforms such as GitHub Actions and GitLab CI. The encoder-decoder architecture of CodeT5, deliberately selected for the prototype, supports potential extension to vulnerability repair suggestion through sequence-to-sequence generation, representing a significant enhancement beyond detection-only functionality.

The prototype system comprehensively addresses Objective 4 through its integrated approach to vulnerability identification and corrective action recommendation. The system successfully identifies vulnerabilities in C/C++ code with 99.11% accuracy, enabling detection of security defects during the development phase before healthcare systems are deployed to production environments. The fix suggestion component provides CWE-specific remediation guidance that received a mean rating of 7.43/10 from evaluation participants, with qualitative feedback indicating that the suggestions are actionable and relevant to practical vulnerability remediation. The user acceptance testing with seven practitioners from industry and academia validates the solution against real-world usage scenarios, demonstrating that the system meets the requirements for integration into healthcare software development workflows. The evaluation identified areas for improvement, particularly in discrimination between patched and unpatched code versions, providing direction for future development that will further enhance the solution. A key architectural contribution underlying these results is the demonstration that hybrid architectures combining symbolic pattern matching with statistical machine learning achieve superior robustness compared to pure ML systems for security applications, the explicit safety rule override mechanism reflecting the principle that in security-critical domains, preventing false negatives outweighs marginal accuracy gains from purely learned approaches.

6 Chapter 6: Conclusion and Future Work

6.1 Introduction

This chapter concludes the thesis by synthesising the research findings, evaluating the achievement of research objectives, and reflecting on the contributions made to the fields of healthcare cybersecurity and privacy-aware software development. The research addressed the important challenge of integrating privacy threat modelling with automated vulnerability detection to enhance the pre-deployment cyber resilience of healthcare information systems during software development. Through the development of an enhanced LINDDUN methodology, creation of a healthcare-focused vulnerability dataset, and implementation of a functional detection prototype, this work has demonstrated that the integration of symbolic privacy analysis with statistical machine learning provides a robust foundation for securing healthcare software.

The chapter is organised as follows. Section 6.2 provides a summary of the research undertaken across all chapters. Thereafter, Section 6.3 systematically evaluates the achievement of each research objective with reference to specific findings and outputs. Then in Section 6.4 discusses the evaluation of research hypothesis, while Section 6.5 is summary of the key findings that emerged from the empirical investigations are presented. Subsequently, Section 6.6 articulates the original contributions to knowledge while Section 6.7 discusses the implications for both theory and practice in healthcare cybersecurity. Then in Section 6.8 the limitations of this research are acknowledged, and this leads to Section 6.9 which provides recommendations for healthcare organisations, security practitioners, and researchers. Finally, Section 6.10 and Section 6.11 outlines the directions for future work and offers concluding remarks on the significance and impact of this research respectively.

6.2 Summary of Research

The research addressed the fundamental problem that existing security assessment methodologies fail to integrate technical vulnerability evaluation with privacy threat assessment during software development, leaving healthcare information systems exposed to threats that can compromise patient safety and violate regulatory frameworks including HIPAA and GDPR.

In Chapter 1, the research context, articulating the convergence of escalating cyber threats, regulatory complexity, and the inadequacy of traditional security approaches for healthcare environments was established. The chapter identified four research objectives addressing the integration of privacy threat modelling with vulnerability detection, the creation of domain-specific datasets, the development of reliable NLP-based detection methods, and the implementation and evaluation of a practical detection solution. Then in Chapter 2, a critical literature review examining secure software development lifecycle methodologies, privacy-by-design frameworks, application security testing approaches, and the emerging application of natural language processing to vulnerability detection was provided. The review identified significant gaps in healthcare-specific threat modelling and the absence of integrated frameworks linking privacy threats to software vulnerabilities, which then established the theoretical foundation for the research contributions.

Following that was Chapter 3 which presented the enhanced LINDDUN privacy threat methodology adapted for healthcare software systems. The chapter detailed the systematic mapping of LINDDUN privacy threat categories to Common Weakness Enumeration entries, creating explicit linkages between high-level privacy concerns and concrete code-level vulnerabilities. The methodology was validated through expert review achieving inter-rater agreement of approximately 0.78 using Fleiss' kappa statistics, exceeding thresholds commonly considered substantial in research environments. The chapter demonstrated practical application through pilot analysis of Electronic Health Record and Medical Imaging subsystems.

Chapter 4 then addressed dataset preparation and machine learning classification approaches. The chapter documented the creation of a novel healthcare cybersecurity-focused dataset comprising C/C++ code samples with binary vulnerability labels and CWE category annotations. The experimental evaluation compared fourteen machine learning models across three paradigms: traditional machine learning (XGBoost achieving F1 = 0.9926), graph neural networks (Reveal achieving F1 = 0.9958, Devign achieving F1 = 0.9948), and transformer-based models (CodeGPT-small achieving F1 = 0.9976, DistilBERT achieving F1 = 0.9974, CodeT5-small achieving F1 = 0.9972). The findings established that NLP-based transformer approaches provide the most effective paradigm for static code security analysis.

Chapter 5 presented the vulnerability detection prototype system integrating XGBoost, Devign, and CodeT5 models into a deployment-feasible research prototype. The system achieved 99.11% accuracy on binary vulnerability classification with response times under 300 milliseconds. The hybrid classification approach combines template matching (99.64% accuracy on known patterns) with machine learning fallback (94.2% accuracy on novel code), which demonstrates a better performance when compared to pure machine learning systems. Then a user acceptance testing was performed with seven industry practitioners and academic researchers to validate the practical utility of the system, with detection accuracy receiving a mean rating of 8.86 out of 10.

6.3 Achievement of Research Objectives

This section systematically evaluates the achievement of each research objective, providing evidence from the research findings and outputs. Table 6-1 provides a summary of objectives and their achievement status, followed by detailed discussion of each objective.

Table 6-1: Research Objectives Achievement Summary

Objective	Primary Evidence	Status
Objective 1: Demonstrate data flow and map privacy/vulnerability misuse cases using LINDDUN to CWE	Chapter 3: Enhanced LINDDUN methodology, expert validation (κ = 0.78)	**Achieved**
Objective 2: Create a novel healthcare cybersecurity-focused dataset for NLP	Chapter 4: 6.3M+ C/C++ samples with CWE annotations	**Achieved**
Objective 3: Develop a reliable NLP method for static code security testing	Chapter 4: Transformer models F1 > 0.997; CodeGPT F1 = 0.9976	**Achieved**
Objective 4: Develop and evaluate a vulnerability identification solution with corrective actions	Chapter 5: Prototype (99.11% accuracy), UAT (M = 8.86/10)	**Achieved**

6.3.1 Objective 1: LINDDUN to CWE Mapping

The first objective sought to demonstrate data flow patterns in typical healthcare information systems and map privacy and vulnerability misuse cases using the LINDDUN privacy threat methodology to the Common Weakness Enumeration taxonomy. This objective was fully achieved through the work presented in Chapter 3. The research developed an enhanced LINDDUN methodology that creates explicit, traceable linkages between the seven LINDDUN privacy threat categories and specific CWE entries representing code-level vulnerabilities.

The methodology was demonstrated through data flow diagram analysis of healthcare information system components, including Electronic Health Records and Medical Imaging subsystems. The systematic threat elicitation process identified privacy threats across all LINDDUN categories, with each threat mapped to corresponding CWE entries through evidence-based analysis. The kappa of 0.78 is consistent with or exceeds the 'substantial agreement' threshold (0.61–0.80) defined by Landis and Koch (1977), which is commonly applied in comparable security expert validation studies. This validation provides confidence that the mappings are methodologically sound and practically applicable.

6.3.2 Objective 2: Healthcare Cybersecurity Dataset

The second objective aimed to create a novel healthcare cybersecurity-focused dataset for natural language processing. This objective was achieved through the dataset construction process documented in Chapter 4. The resulting dataset comprises C/C++ code samples with binary vulnerability labels and CWE category annotations, representing the largest publicly documented dataset specifically prepared for NLP-based vulnerability detection in healthcare contexts.

The dataset integrates samples from established vulnerability repositories (Big-Vul, DiverseVul, ReposVul) with healthcare-specific curation based on the LINDDUN privacy threat analysis framework. The dataset preparation process prioritised vulnerability classes particularly relevant to healthcare systems, including buffer overflows (CWE-120) that may compromise medical device integrity, command injection vulnerabilities (CWE-78) that could enable unauthorised access to patient data systems, and memory management errors threatening system availability in critical care environments. The healthcare-oriented curation distinguishes this dataset from general-purpose vulnerability corpora and enables training of models attuned to the specific security requirements of healthcare software development.

6.3.3 Objective 3: Reliable NLP Method for Static Code Security Testing

The third objective sought to develop a reliable method for static software code security testing using NLP in detecting defective portions of code. This objective was achieved through the comprehensive experimental evaluation presented in Chapter 4, which compared

fourteen machine learning models across three paradigms. The findings establish transformer-based NLP approaches as the most reliable method for vulnerability detection, with CodeGPT-small achieving F1 = 0.9976, DistilBERT achieving F1 = 0.9974, and CodeT5-small achieving F1 = 0.9972.

The reliability of the NLP-based approach is evidenced by the low variance in performance across different transformer architectures (F1 range: 0.9963 to 0.9976), indicating that the fundamental NLP methodology rather than specific architectural choices drives detection effectiveness. The transformer models consistently outperformed both traditional machine learning approaches (XGBoost F1 = 0.9926) and graph neural networks (Reveal F1 = 0.9958), establishing that natural language processing provides the most effective paradigm for static code security analysis among current approaches. The self-attention mechanism central to transformer architectures enables capture of long-range dependencies between code tokens that indicate vulnerability patterns, validating the treatment of vulnerability detection as a specialised NLP task.

6.3.4 Objective 4: Vulnerability Identification Solution with Corrective Actions

The fourth objective aimed to develop a solution that can identify vulnerabilities in code during software development for healthcare systems and offer corrective actions before systems go live, then evaluate the developed solution. This objective was achieved through the prototype system implementation and user acceptance testing documented in Chapter 5. The system successfully identifies vulnerabilities in C/C++ code with 99.11% accuracy, enabling detection of security defects during the development phase before healthcare systems are deployed to production environments.

The corrective action component provides CWE-specific remediation guidance through the fix_suggestion field, offering concrete code modifications that address underlying security defects across the vulnerability categories supported by the system, as documented in Section 5.6. The user acceptance testing with seven practitioners from industry and academia validated the solution against real-world usage scenarios, with detection accuracy receiving a mean rating of 8.86 out of 10 and fix suggestion quality receiving 7.43 out of 10.

The deployment via Railway API backend and Firebase frontend hosting demonstrates the deployment feasibility of the validated prototype, providing a foundation for adoption by healthcare software development organisations seeking pre-deployment security validation.

6.4 Evaluation of Research Hypotheses

This section evaluates each research hypothesis against the empirical evidence gathered throughout the research. All four hypotheses were supported by the findings, demonstrating that the research design and methodology were appropriate for addressing the identified research problem.

Hypothesis 1 posited that the seven LINDDUN privacy threat categories can be systematically and reliably mapped to Common Weakness Enumeration entries, with expert validation achieving substantial inter-rater agreement (Fleiss' kappa exceeding 0.60). This hypothesis was supported. The expert validation protocol achieved inter-rater agreement of approximately 0.78 using Fleiss' kappa statistic, exceeding the 0.60 threshold for substantial agreement. The structured methodology provided by LINDDUN and the evidence-based mapping rubric created sufficient clarity to guide expert judgments toward consensus. The adjudication process following initial independent reviews proved valuable for resolving disagreements, with closer examination of traceability artefacts clarifying the conceptual basis for mappings in several cases. This outcome validates the feasibility of integrating privacy threat modelling with software vulnerability assessment through systematic methodological frameworks.

Hypothesis 2 posited that transformer-based natural language processing models will outperform both traditional machine learning approaches and graph neural networks for C/C++ vulnerability detection, achieving F1-scores exceeding 0.99. This hypothesis was supported. The experimental evaluation established a clear performance hierarchy: transformer models (CodeGPT-small F1 = 0.9976, DistilBERT F1 = 0.9974, CodeT5-small F1 = 0.9972) outperformed graph neural networks (Reveal F1 = 0.9958, Devign F1 = 0.9948), which in turn outperformed traditional machine learning (XGBoost F1 = 0.9926). All transformer models exceeded the 0.99 F1-score threshold specified in the hypothesis. The low variance in performance across different transformer architectures (F1 range: 0.9963 to 0.9976) indicates

that the fundamental NLP methodology rather than specific architectural choices drives detection effectiveness. This finding validates the treatment of vulnerability detection as a specialised NLP task where source code is processed as a form of natural language with syntactic and semantic relationships amenable to attention-based learning.

Hypothesis 3 posited that a hybrid classification architecture combining symbolic pattern matching with statistical machine learning will achieve superior robustness compared to pure machine learning systems, particularly for known vulnerability patterns and obfuscated code samples where ML classifiers exhibit reduced confidence. This hypothesis was supported. As specified in H3, the evaluation focused on whether a hybrid symbolic–statistical architecture achieves superior robustness compared to pure machine learning deployment, particularly for known vulnerability patterns and code where classifier confidence is reduced and on whether this justifies a hybrid rather than pure-ML implementation in the detection prototype. The empirical evidence confirms both dimensions of this question. On the first dimension, the template matching component achieved 99.64% accuracy on known vulnerability patterns, substantially exceeding the pure multi-class ML approach which exhibited highly inconsistent performance ranging from 26% to 100% across CWE categories despite the application of three standard mitigation strategies, class-weighted loss functions, SMOTE oversampling, and CWE taxonomy consolidation, as documented in Section 5.4. The safety rule override mechanism further improved detection robustness by 8.2% on obfuscated code samples where ML classifier confidence fell below the 70% threshold, directly satisfying the obfuscated code condition stated in H3. On the second dimension, these findings provide the empirical justification for the hybrid architecture adopted in the prototype rather than a pure XGBoost deployment, confirming that the architectural choice was evidence-driven rather than presumptive. The binary F1 advantage of the hybrid over XGBoost alone is modest, but binary F1 is not the metric H3 addresses. H3 concerns robustness across deployment scenarios including known patterns, novel code, and adversarially obfuscated inputs, all three of which are empirically evaluated and reported in Chapter 5. The ML fallback ensures continued functionality for novel code patterns with 94.2% accuracy, producing a combined accuracy of approximately 96.8% across diverse test scenarios. This outcome validates the security-first design principle that explicit symbolic

rules provide necessary safeguards in domains where the consequences of false negatives are severe.

Hypothesis 4 posited that an integrated vulnerability detection prototype can achieve detection accuracy exceeding 95% with response times under 500 milliseconds, meeting the practical requirements for deployment in healthcare software development workflows. This hypothesis is supported under the stated evaluation conditions, with two explicit qualifications that bound the scope of the claim. On detection accuracy, the prototype achieved 99.11% on the held-out test set, substantially exceeding the 95% threshold stated in H4. This figure was obtained under the 1:3 class ratio of the experimental partition, which does not reflect real-world codebases where vulnerable functions constitute less than 1% of total code. Under realistic class distributions, precision would be expected to decrease as the prior probability of vulnerability falls, the 99.11% figure is therefore an upper-bound estimate under favourable evaluation conditions rather than a deployment-validated result. On response time, the cold-path latency of 264 milliseconds, the time experienced by a developer submitting a previously unseen function, satisfies the sub-500 millisecond threshold stated in H4 and is the operationally meaningful figure against which the hypothesis is evaluated. The cached response time of approximately 15 milliseconds applies only to repeated identical submissions and represents a best-case scenario rather than a representative operational figure. Both thresholds are met, and H4 is supported as stated, with the qualification that deployment validation under realistic class distributions is a prerequisite before clinical adoption, as identified in Sections 6.7 and 6.8.

On response time, the cold-path latency of 264 milliseconds which is the time experienced by a developer submitting a previously unseen function for analysis, satisfies the 500 millisecond threshold stated in H4 and is the operationally meaningful figure for the primary use case of development-time vulnerability analysis. The cached response time of approximately 15 milliseconds applies to repeated identical submissions and represents a best-case rather than representative figure. Both are reported here for completeness, but the 264-millisecond cold-path latency is the figure against which H4 should be evaluated, and it confirms the hypothesis. The specific response time of 264 milliseconds replaces the

indicative term 'milliseconds' used at the hypothesis formulation stage in Chapter 1, grounding H4 in a measured empirical figure from the operational prototype.

User acceptance testing with seven industry practitioners and academic researchers yielded a mean detection accuracy rating of 8.86 out of 10, with additional dimensions including processing speed (M = 7.71), explanation quality (M = 7.43), and user interface (M = 7.71) all receiving ratings above 7.0. The deployment via Railway API backend and Firebase frontend hosting demonstrates a validated prototype, confirming that the research outputs translate from academic evaluation to practical tools capable of supporting healthcare software development workflows. The specific response time of 264 milliseconds, validated under deployment conditions on Railway infrastructure, provides a precise empirical basis for the claim of sub-500 millisecond performance articulated in H4, replacing the indicative 'milliseconds' stated at the hypothesis formulation stage in Chapter 1 with a measured figure grounded in the operational prototype.

The support for all four hypotheses provides strong evidence that the research approach was well-founded and that the proposed integration of privacy threat modelling with NLP-based vulnerability detection represents a viable and effective strategy for enhancing the pre-deployment cyber resilience of healthcare information systems during software development.

6.5 Summary of Key Findings

The five contributions presented in this section address the seven research gaps identified in the literature review summarised in Chapter 2. Table 6-2 makes this correspondence explicit, showing which contribution or contributions engage with each gap. This correspondence should be read as evidence that each gap has been addressed under the specific experimental conditions of this research, not as a claim that the gaps have been closed, solved, or eliminated as open problems in the field. The paragraph below states these conditions explicitly, together with what remains open.

Table 6-2: Mapping of Literature Review Gaps to Research Contributions

Gap	Description	Addressed by
Gap 1	No validated bridge between privacy threat modelling and code-level vulnerability detection	Contribution 1
Gap 2	Absence of a large-scale healthcare-specific vulnerability dataset	Contribution 2
Gap 3	Disconnect between PbD principles and their practical operationalisation in healthcare software development	Contribution 1
Gap 4	High false positive rate and limited adaptability of rule-based static analysis tools	Contributions 3 and 4
Gap 5	Under exploration of computationally efficient and interpretable ML architectures	Contribution 3
Gap 6	Lack of real-time deployment-ready vulnerability detection systems with false-negative guarantees	Contributions 4 and 5
Gap 7	Inconsistency in evaluation methodology across the vulnerability detection literature	Contribution 3

The sense in which these seven gaps are addressed warrants explicit qualification. The evaluations underpinning Contributions 3, 4, and 5 (Chapters 4 and 5) were conducted under a stratified 1:3 vulnerable-to-non-vulnerable class ratio, constructed to enable stable model training and comparison across fourteen models. This ratio does not reflect the realistic prevalence of vulnerabilities in production codebases, more plausibly on the order of 1:72 or sparser, and performance under that far more imbalanced, realistic distribution has not been evaluated. Evaluation was further conducted on same-source, held-out test data drawn from the same construction pipeline as the training data (Section 4.2). There was no external deployment validation, on a codebase or organisation genuinely independent of this research, has been conducted. The LINDDUN-to-CWE mapping (Contribution 1) and the resulting detection system's CWE coverage, while substantial, address a bounded set of CWE categories rather than the full CWE catalogue, and categories outside this set remain unaddressed by the mapping and by extension by the detection system built on it. Finally, the expert panel that validated the mapping (Section 3.2.6) and the seven participants who evaluated the prototype (Section 5.7) do not constitute validation by practising healthcare software developers or security practitioners operating under real organisational constraints; both remain necessary before the claims in this thesis can be treated as validated in deployment. Each of these four conditions, realistic vulnerability prevalence, external deployment validation, broader CWE coverage, and healthcare-practitioner validation, is

identified as necessary future work in Section 6.9. The contributions of this thesis therefore provide initial evidence toward addressing the seven identified gaps, under the conditions stated throughout this thesis, rather than resolving them as closed problems.

The research generated several key findings with implications for healthcare cybersecurity practice and the broader field of automated vulnerability detection (Table 6-3). These findings emerged from the systematic evaluation of privacy threat modelling, machine learning classification, and prototype deployment.

Table 6-3: Summary of Key Research Findings

Finding	Key Result	Implication
1	Transformer models outperform GNNs and traditional ML: CodeGPT (F1 = 0.9976) > Reveal (F1 = 0.9958) > XGBoost (F1 = 0.9926)	Self-attention implicitly learns structural relationships without explicit graph construction, challenging assumptions about necessity of graph-based code representations
2	Code-specialised pre-training provides minimal advantage: GraphCodeBERT (F1 = 0.9970) ≈ CodeBERT (F1 = 0.9968) ≈ BERT-base (F1 = 0.9968)	Fine-tuning on vulnerability-specific data overwhelms pre-training benefits; organisations can achieve comparable performance using general-purpose transformers
3	Hybrid symbolic-statistical architectures achieve superior robustness: template matching (99.64%) + ML fallback (94.2%) outperforms pure ML	Eliminates inconsistent performance (26%-100%) observed in pure multi-class CWE classification; addresses prediction bias toward high-frequency categories
4	Safety rule override mechanisms improve detection robustness by 8.2% on obfuscated code samples	Validates security-first design principle that false negatives carry greater risk than false positives in healthcare contexts
5	Computational cost varies ~10,000-fold for marginal gains: CodeGPT (224.2 hrs) vs XGBoost (seconds) for +0.50pp F1; DistilBERT achieves 96% of improvement in 4.85 hrs	Intermediate-complexity models (DistilBERT) offer optimal efficiency-accuracy trade-off for resource-constrained healthcare environments
6	Expert validation of LINDDUN-to-CWE mappings achieves inter-rater agreement κ = 0.78.	Structured LINDDUN methodology and evidence-based mapping rubric enable expert consensus, demonstrating practical applicability of enhanced methodology

Before discussing the implications of these findings individually, a methodological observation about the pattern of results across all evaluations warrants explicit acknowledgement. The uniformly high-performance metrics, in particular F1-scores exceeding 0.99 across all fourteen models, AUC-ROC values of 0.9999 across all three

paradigms, and a consistent performance hierarchy regardless of architectural approach, are unusual for a genuinely difficult and novel detection problem and merit a critical reflection.

Three features of the evaluation design contribute to this pattern and should be considered when interpreting the results. First, all models were trained and evaluated on data drawn from the same source distribution under a 1:3 class ratio, which is substantially more favourable than the real-world ratio of approximately 1:72 in production codebases. Under the realistic distribution, precision would decrease materially as the prior probability of vulnerability falls, and the absolute performance figures would not be expected to hold. Second, while deduplication of exact duplicate functions was performed during dataset construction, near-duplicate functions, the same vulnerability pattern appearing in slightly modified form across projects in the same source corpus, were not explicitly identified or removed. The consistent performance hierarchy across architecturally diverse models, from TF-IDF features through GNNs to transformers, suggests that the patterns are genuinely learnable through multiple strategies rather than being memorised from duplicates, but this cannot be conclusively confirmed without cross-dataset external validation. Third, the performance ceiling achieved across all paradigms raises the possibility that the vulnerability signatures in the dataset are more surface-distinguishable than those encountered in genuinely adversarial real-world conditions, where obfuscation, novel coding patterns, and cross-file dependencies create detection challenges that same-distribution test sets do not fully represent.

These observations do not invalidate the findings, rather the performance hierarchy across paradigms is consistent and statistically significant, and the evaluation protocol is rigorous within its design constraints. They do, however, mean that the absolute metric values should be treated as upper-bound estimates under favourable conditions rather than as predictions of production performance. The practical recommendations derived from these findings, particularly regarding model selection for healthcare deployment, are made with this qualification in mind.

6.6 Research Contributions

This research makes several original contributions to knowledge in the fields of healthcare cybersecurity, privacy engineering, and automated vulnerability detection. These contributions include methodological, empirical, and practical dimensions.

Contribution 1: Enhanced LINDDUN Privacy Threat Methodology.

The research extends the LINDDUN privacy threat modelling framework with systematic mappings to the Common Weakness Enumeration taxonomy, creating a unified framework for privacy threat analysis and software vulnerability assessment. The core findings of this contribution have been subjected to independent peer review through publication in Electronics (Ameh et al., 2025), providing external validation that the methodology is both technically sound and of sufficient novelty to merit publication in an indexed venue. Collectively, the four publications arising from this thesis in Electronics, Artificial Intelligence Review, the ESORICS 2025 Lecture Notes in Computer Science proceedings, and the submitted manuscript to ACM TOSEM, demonstrate that each major contribution has been evaluated independently of this examination process, strengthening confidence in the rigour and originality of the reported findings.

Contribution 2 — Healthcare-Aligned Vulnerability Dataset:

This research contributes C3-VULMAP, a 6.3 million sample C/C++ vulnerability dataset annotated with CWE classifications and LINDDUN privacy threat labels. The dataset is the first publicly documented vulnerability corpus to link code-level weaknesses to privacy threat categories, enabling domain-sensitive vulnerability detection training for healthcare software contexts. Healthcare alignment is achieved through LINDDUN-derived CWE filtering and repository-level domain prioritisation, with code provenance acknowledged as spanning both healthcare-specific and general-purpose open-source projects.

Contribution 3: Comparative Evaluation of Machine Learning Paradigms.

The research provides a comprehensive comparative evaluation of machine learning approaches for C/C++ vulnerability detection, assessing fourteen models across three paradigms (traditional machine learning, graph neural networks, and transformer-based

architectures) under controlled experimental conditions with a consistent evaluation protocol across all models. The findings establish empirical evidence for the superiority of transformer-based NLP approaches over both GNN and traditional ML paradigms, challenging two prevailing assumptions in the literature: that explicit graph construction is necessary for capturing vulnerability-relevant code structure, and that code-specialised pre-training provides meaningful performance advantages over general-purpose language models when fine-tuning on domain-specific vulnerability data.

Beyond paradigm comparison, the evaluation makes two further contributions. First, it addresses the evaluation methodology inconsistency identified as Gap 7 in Chapter 2 by applying a standardised metric suite, including F1-score, recall, false negative rate, AUC-ROC, precision-recall AUC, and calibration curves, uniformly across all fourteen models and three paradigms, enabling direct cross-paradigm comparison that the fragmented evaluation protocols in existing literature preclude. Second, it establishes a comprehensive supervised learning baseline against which future approaches can be benchmarked. As noted in the contextualising observation in Section 2.12, self-supervised and semi-supervised learning paradigms are nearly absent from vulnerability detection research despite the well-documented challenge of labelled data scarcity in this domain. The present evaluation does not address this gap directly, all fourteen models employ supervised learning but it provides the empirical reference point that future self-supervised research requires being a rigorously evaluated upper bound of supervised performance on the C3-VULMAP dataset, defining the improvement threshold that unsupervised approaches must exceed to justify their additional complexity in healthcare deployment contexts.

Contribution 4: Hybrid Detection Architecture.

The research demonstrates a novel hybrid architecture combining template matching with machine learning fallback and safety rule override mechanisms. This contribution addresses the practical limitations of pure ML approaches for security applications, where prediction bias and inconsistent performance across vulnerability categories undermine practitioner trust. The hybrid approach provides deterministic classification for known patterns with graceful degradation for novel code.

Contribution 5: A Validated Prototype Vulnerability Detection System.

The research delivers a validated prototype vulnerability detection system deployed via REST API and web frontend, validated through user acceptance testing with industry practitioners and academic researchers. This contribution demonstrates the practical applicability of the research beyond academic evaluation, providing an immediately deployable tool for healthcare software development organisations.

The end-to-end integration of the three primary contributions is demonstrated through two representative pipeline traces in Section 5.7, confirming that the LINDDUN-to-CWE methodology, C3-VULMAP dataset, and vulnerability detection prototype form a continuous analytical framework from design-time privacy threat identification to code-time vulnerability detection and remediation guidance.

6.7 Implications for Theory and Practice

6.7.1 Theoretical Implications

The research findings carry several implications for theoretical understanding of vulnerability detection and privacy-aware software development. First, the superior performance of transformer models over graph neural networks challenges prevailing assumptions about the necessity of explicit structural representation for code analysis. The finding that self-attention mechanisms implicitly learn relevant structural relationships without graph construction suggests that the sequential nature of code, treated as a specialised form of natural language, may be sufficient for capturing vulnerability-relevant patterns. This has implications for the theoretical foundations of code representation learning.

Second, the parity between code-specialised and general-purpose transformers suggests that domain knowledge may be more effectively acquired through task-specific fine-tuning than through pre-training objectives. This finding contributes to ongoing theoretical debates about transfer learning and the value of domain-specific language models. The implication is that the vulnerability detection task provides sufficient signal to adapt general language

understanding capabilities to code analysis, without requiring explicit encoding of programming language semantics during pre-training.

Third, the successful integration of LINDDUN privacy threats with CWE vulnerability classifications demonstrates the feasibility of unified security and privacy assessment frameworks. This contribution addresses a theoretical gap where privacy engineering and security engineering have developed as separate disciplines with distinct methodologies. The explicit mappings between privacy threat categories and software weaknesses provide a foundation for theoretical frameworks that treat privacy and security as interconnected rather than orthogonal concerns.

6.7.2 Practical Implications for Healthcare

The strong binary detection performance achievable through traditional machine learning without GPU infrastructure has potential implications for healthcare procurement and regulatory guidance that merit further investigation. Current NHS Digital security standards and FDA cybersecurity frameworks for medical devices mandate security testing but do not specify tooling requirements, creating an implementation gap for smaller providers and device manufacturers without dedicated security engineering teams. The finding that XGBoost achieves F1 = 0.9926 on the C3-VULMAP evaluation dataset using commodity hardware is relevant to this gap, it provides preliminary evidence that minimum viable automated vulnerability detection may be within reach of resource-constrained organisations without requiring deep learning infrastructure.

It is important, however, to characterise what this evidence does and does not support. The F1 = 0.9926 figure was obtained under the 1:3 class ratio of the experimental partition, which does not reflect the class distribution of real production codebases, and on a dataset whose healthcare specificity resides in its annotation scheme rather than uniformly in code provenance, as acknowledged in Section 6.8. These conditions mean the figure represents a performance estimate under favourable evaluation conditions rather than a deployment-validated result on NHS or FDA-regulated software. Translating this finding into a regulatory recommendation therefore requires additional evidence: independent validation on diverse real-world healthcare codebases under realistic class distributions, evaluation across multiple

healthcare software categories including medical device firmware, EHR systems, and health information exchange interfaces, and confirmation that the performance advantage of traditional ML over deep learning holds under conditions representative of the organisations the guidance would govern. The finding from this research is offered as a contribution to that evidence base rather than as a sufficient basis for regulatory action in itself. If confirmed by subsequent independent validation, the implication, that performance threshold specifications may be more enabling than technology prescriptions for resource-constrained healthcare organisations, would represent a genuinely useful input to future guidance development processes.

Second, the deterministic classification properties of the hybrid architecture have implications for the emerging regulatory landscape governing AI-based tools in safety-critical point of view. The UK Medicines and Healthcare Products Regulatory Agency and the US FDA are developing frameworks for qualifying AI-based software as medical device components, both of which emphasise auditability and predictability of decision pathways as core requirements. The rule_override of the hybrid architecture field demonstrates an architectural design pattern consistent with these requirements. Specifically, the principle that a system capable of making consequential classifications should make its decision pathway transparent and distinguishable to the user. This design pattern is offered as a contribution to how future AI-assisted security tools in regulated healthcare contexts might be architected. It does not constitute a claim that the current prototype satisfies MHRA or FDA regulatory requirements, which would additionally demand authentication infrastructure, audit logging, quality management system documentation, and formal clinical validation which are prerequisites that are explicitly identified in Section 5.9 as future development priorities. The distinction between demonstrating an architecturally sound pattern and delivering a regulatory-compliant system is deliberate: the former is within the scope of a PhD research prototype, while the latter requires organisational and engineering infrastructure that lies beyond it.

Third, The LINDDUN-to-CWE mapping methodology has implications for how healthcare organisations structure the relationship between their privacy engineering and software security functions, which in most organisations operate as separate teams with distinct

toolchains and reporting lines. The explicit mappings demonstrate that privacy threat analysis and vulnerability scanning address overlapping concerns and that their outputs can inform each other systematically rather than running in parallel without connection. For organisations subject to simultaneous GDPR data protection obligations and NHS or FDA security requirements, the integrated framework reduces the duplicated assessment effort that arises when privacy impact assessments and security code reviews are conducted independently, offering a governance efficiency that has direct operational value beyond the technical contributions of this thesis.

6.8 Limitations

Several limitations of this research warrant acknowledgment. First, the prototype system focuses exclusively on C/C++ source code, with no support for other programming languages common in healthcare software development such as Python, Java, or JavaScript. Extension to additional languages would require language-specific feature extraction and potentially separate classification models trained on language-appropriate datasets. The prevalence of C/C++ in medical device firmware and performance-critical healthcare systems motivated this focus, but the findings may not generalise directly to other language contexts.

Second, the system performs static analysis only, without runtime or dynamic analysis capability. Some vulnerability classes, particularly those involving complex execution paths or timing-dependent behaviours, may not be detectable through static analysis alone. The integration of dynamic analysis would provide more comprehensive coverage but introduces deployment complexity and performance overhead that may not be practical for all healthcare development environments.

Third, the experimental evaluation employed a balanced dataset with approximately equal representation of vulnerable and non-vulnerable samples, which does not reflect the extreme class imbalance present in real-world codebases where vulnerable functions may constitute less than 1% of the total. The high performance achieved under balanced conditions may not generalise to production environments where the prior probability of vulnerability is substantially lower. This limitation highlights the need for deployment validation under realistic class distributions. This limitation applies directly to every headline

performance figure reported in this thesis, including the 99.11% accuracy figure central to the H4 hypothesis adjudication in Section 6.3, and the F1 scores reported for all fourteen evaluated models in Chapter 4 (ranging from 0.9926 for XGBoost to 0.9976 for CodeGPT-small). All were measured under the 1:3 ratio and should be interpreted as upper-bound estimates obtainable under favourable class balance, not as deployment-validated performance guarantees under realistic, sub-1% vulnerability prevalence.

Fourth, the user acceptance testing sample size (n = 7) provides preliminary validation but does not support statistically robust conclusions about system effectiveness across diverse user populations. The qualitative feedback and quantitative ratings offer valuable insights but should be interpreted as indicative rather than definitive. Larger-scale evaluation with diverse healthcare development teams would strengthen confidence in the practical utility of the system. This small sample size, the positive UAT ratings and qualitative feedback reported in Section 5.7, and the corresponding claim in Section 6.3 that H4's practical-utility criteria are met, should be read as preliminary, indicative evidence from a small, non-representative group rather than a validated finding of practical utility across the healthcare software development contexts the prototype is intended to serve.

Fifth, code is analysed in isolation without consideration of build context, library APIs, or cross-file data flow. Vulnerabilities arising from interactions between code modules may be missed when analysing individual files. This limitation reflects the function-level granularity of the approach, which prioritises practical deployment over comprehensive interprocedural analysis.

Sixth, the dataset construction relied on controlled synthetic augmentation using GPT-3.5 Turbo to address underrepresented CWE categories, as documented in Chapter 4. While synthetic samples were clearly flagged in the dataset metadata and generated using structured prompts grounded in realistic healthcare scenarios, synthetically generated vulnerable code may not fully capture the complexity and subtlety of real-world vulnerabilities, particularly for privacy-critical CWE categories where genuine examples were scarce precisely because they are rare in practice. Models trained on these synthetic samples may learn patterns characteristic of generated code rather than authentic vulnerability manifestations, and the extent to which this affects generalisation to production codebases

has not been empirically quantified. Future work should investigate the impact of synthetic samples on model performance through ablation studies that compare models trained with and without the augmented data. This limitation bears directly on the headline performance figures reported throughout Chapters 4 and 5. This is because an unknown proportion of the training data for every evaluated model, including XGBoost (F1 = 0.9926), the transformer models (up to F1 = 0.9976), and the hybrid prototype (99.11% accuracy), consists of synthetically generated rather than naturally occurring code, these figures cannot be fully attributed to genuine vulnerability-pattern learning as opposed to partial learning of GPT-3.5 Turbo's generation characteristics. The reported figures should be read as measuring performance on this specific, partially synthetic dataset, not as a validated measurement of real-world vulnerability detection capability.

Seventh, the evaluation conditions may be more favourable than representative of real-world deployment contexts, and the uniformly high-performance metrics across all paradigms should be interpreted accordingly. Four specific features of the evaluation design create this risk. The 1:3 class ratio used in training and evaluation does not reflect real-world class distributions where vulnerable functions constitute less than 1% of production codebases, and all reported accuracy, F1, and AUC-ROC figures are specific to this experimental ratio. The same-distribution test set, drawn from the same source corpora as the training data, measures generalisation under best-case conditions and does not validate performance on out-of-distribution codebases from different projects, organisations, or development contexts. Because of this same-source evaluation design, the F1 scores reported in Chapter 4 (up to 0.9976) and the 99.11% accuracy figure central to Chapter 5's evaluation should be read as upper bounds specific to this dataset's source distribution, which may not hold on code from organisations, projects, or coding conventions absent from DiverseVul, ReposVul, and StarCoder. Exact-match deduplication was performed during dataset construction to remove literal duplicate functions arising from overlapping source repositories. However, this process does not detect semantic or near-duplicate code clones, functions that are logically equivalent but differ in variable names, formatting, or minor structural edits, and the function-level random split does not enforce repository-level separation between training and test partitions. Consequently, near-duplicate functions from the same source project

cannot be ruled out from appearing on both sides of the split, and the reported performance figures should be interpreted as upper-bound estimates that may partly reflect this leakage risk rather than purely out-of-project generalisation. A repository-level re-split, confining all functions from a given source project to a single partition, is identified as a priority validation step in Section 6.9.

C3-VULMAP's healthcare alignment operates primarily at the level of CWE-based privacy relevance rather than verified code origin. Approximately 55% of the functions of the dataset originate from repositories with verified healthcare associations, while the remaining 45% are general-purpose functions retained for their privacy-relevant CWE classification. This split is an approximate, construction-time reconstruction rather than a precise per-function count, since source-repository identifiers were not preserved in the released dataset. A detailed comparison of CWE distribution and function-level characteristics between the two groups was consequently not possible. The near-duplicate functions across the source corpora, while not confirmed as present, cannot be ruled out without explicit cross-project deduplication analysis, which was not performed (Section 6.8 discusses this specific risk in detail for the GNN and traditional ML function-level random split). Therefore, due to this risk, the near-perfect F1 scores reported for Devign, Reveal, and the five traditional ML models in Chapter 4 should be treated as potentially reflecting partial memorisation of near-duplicate code across the train/test boundary, rather than purely as evidence of generalised vulnerability detection capability.

6.9 Recommendations

6.9.1 Recommendations for Healthcare Organisations

Healthcare organisations seeking to implement automated vulnerability detection should consider a tiered approach based on available resources and risk tolerance. Organisations with limited computational infrastructure can achieve strong detection performance using traditional machine learning approaches (F1 > 0.99) with minimal investment. Organisations with GPU resources should consider transformer-based approaches for marginally improved accuracy and the potential for future extension to vulnerability repair tasks. All organisations

should implement safety rule systems to ensure detection of known dangerous patterns regardless of ML classifier confidence.

Healthcare development teams should integrate the LINDDUN-to-CWE mapping methodology into existing threat modelling workflows, using the mappings to prioritise security testing based on privacy impact. The methodology supports regulatory compliance by providing documented traceability from privacy requirements through threat analysis to vulnerability assessment. Teams should conduct threat modelling early in the development lifecycle, using the healthcare-specific data flow diagram templates as starting points for system-specific analysis.

6.9.2 Recommendations for Security Practitioners

Security practitioners should consider model selection based on deployment context and error tolerance. For safety-critical applications where missed vulnerabilities carry significant patient safety implications, DistilBERT (26 false negatives) or Devign (40 false negatives) provide superior sensitivity. For environments where alert fatigue is a primary concern, CodeGPT-small (49 false positives) offers the best precision. Practitioners should calibrate classification thresholds based on organisational risk tolerance, leveraging the near-perfect AUC-ROC scores that enable flexible operating point selection.

6.9.3 Recommendations for Researchers

Researchers building on this work should address the limitations identified, particularly the need for evaluation under realistic class imbalance conditions and extension to additional programming languages. The dataset and methodology provide a foundation for investigating cross-language transfer learning, active learning for continuous model improvement, and the integration of dynamic analysis with static vulnerability detection. Researchers should also investigate the theoretical foundations of why transformer attention mechanisms effectively capture vulnerability patterns without explicit structural encoding.

6.10 Future Work

Several directions for future research emerge from the findings and limitations of this work. These directions span short-term extensions, medium-term developments, and long-term research agendas.

6.10.1 Short-Term Extensions

The highest priority short-term extension is cross-dataset external validation on independently sourced healthcare codebases under realistic class distributions, which is the necessary step to establish whether the performance figures reported in this thesis generalise beyond the experimental evaluation conditions to production deployment contexts.

Closely related, and answerable without new data collection, is repository-level re-splitting of the existing C3-VULMAP evaluation partitions: confining all functions from a given source repository to a single train, validation, or test partition, rather than the function-level random split used in this thesis. As discussed in Section 6.8, the function-level split cannot rule out near-duplicate or semantically similar functions from the same source project appearing on both sides of the train/test boundary, which the exact-match deduplication performed during dataset construction does not detect. Re-running the key comparative and hybrid-architecture evaluations under a repository-level split and reporting the resulting metrics alongside those presented in Chapters 4 and 5, would directly establish how much of the reported performance reflects genuine generalisation to unseen code versus proximity to near-duplicate training examples. This is identified as a priority validation step preceding, and complementary to, the cross-dataset external validation described above.

Retaining a source-repository identifier and an explicit verified-healthcare flag per function in future releases of C3-VULMAP, enabling exact reporting of the healthcare-origin/general-purpose-CWE-relevant split and a detailed comparison of CWE distribution and function characteristics between the two groups.

Immediate extensions include multi-language support through language-specific feature extraction for Python, Java, and JavaScript, enabling broader applicability across healthcare software development contexts. Improved discrimination between patched and unpatched code versions, identified as a limitation through user acceptance testing, would enhance the

practical utility for security review workflows. Extension of the CWE taxonomy coverage beyond the current five primary categories would provide more comprehensive vulnerability classification. Larger-scale user acceptance testing with diverse healthcare development teams would strengthen validation of practical utility.

Post-hoc probability calibration of the deployed classifiers represents a further short-term extension with direct clinical governance implications. As documented in Chapter 4, the calibration curve analysis revealed systematic differences in probability reliability across models, with Random Forest and Logistic Regression exhibiting notable under confidence at mid-range predicted probabilities, and XGBoost showing moderate deviation from the ideal calibration diagonal. In healthcare deployment contexts where confidence scores inform risk prioritisation and clinical decision support workflows, uncalibrated probability outputs may lead to systematic misallocation of security review effort. Applying post-hoc calibration techniques such as Platt scaling or isotonic regression to the deployed models would improve the reliability of confidence estimates without requiring model retraining, representing a low-cost enhancement with meaningful operational benefit for healthcare development teams using the prototype for risk-stratified security validation.

Systematically tagging each of the 965 entries in the LINDDUN-to-CWE mapping matrix (Appendix C) with its specific relationship type enables, contributes to, or provides evidence of (Section 3.2.2), rather than the category-level characterisation presented in this thesis, would strengthen the use of the matrix as a machine-readable artefact for future tooling and is identified as a refinement for future work.

Multi-seed evaluation of the seven transformer models, applying the same three-seed protocol used for the GNN architectures, remains outstanding. The encoder-only models (DistilBERT, DistilRoBERTa, BERT-base, CodeBERT, GraphCodeBERT) are computationally feasible for this extension (15–27 hours per model across 3 seeds) but CodeGPT-small's single-run training time of 224 hours makes multi-seed evaluation of that specific model impractical within a standard research timeline and would require either reduced-scale training or access to additional compute infrastructure.

6.10.2 Medium-Term Developments

Medium-term developments include integration of dynamic analysis with static vulnerability detection to capture runtime-dependent vulnerability classes. Cross-file data flow analysis would address the limitation of analysing code in isolation, enabling detection of vulnerabilities arising from module interactions. Active learning mechanisms for continuous model improvement from user feedback would maintain detection performance as new vulnerability patterns emerge. Integration with continuous integration and continuous deployment pipelines through plugins for GitHub Actions, GitLab CI, and similar platforms would support automated security validation within existing development workflows.

Production-grade security and privacy hardening of the prototype system to enforce authentication and authorisation, end-to-end TLS verification, encryption at rest for cached and logged code, defined data retention and deletion policies, audit logging, and an on-premises or private-cloud deployment option for organisations unable to accept third-party hosting of sensitive source code.

6.10.3 Long-Term Research Agenda

Long-term research directions include automated vulnerability repair using the encoder-decoder architecture of CodeT5 for sequence-to-sequence generation of patched code. The deliberate selection of CodeT5 for the prototype anticipates this extension, which would represent a significant enhancement beyond detection-only functionality. Federated learning approaches for privacy-preserving model updates would enable collaborative improvement across healthcare organisations without sharing sensitive code. Real-time IDE integration would shift vulnerability detection further left in the development lifecycle, providing immediate feedback during code authoring. Investigation of regulatory certification pathways for AI-based security tools would support adoption in regulated healthcare environments where tool qualification is required.

6.11 Concluding Remarks

This research addressed the critical challenge of securing healthcare information systems during software development through the integration of privacy threat modelling with automated vulnerability detection. The work demonstrates that the traditional separation

between privacy analysis and security assessment can be overcome through systematic methodological integration, creating unified frameworks that address both concerns throughout the software development lifecycle. The enhanced LINDDUN methodology, healthcare-focused dataset, and deployment-feasible detection prototype collectively provide practical tools for healthcare organisations seeking to improve the pre-deployment cyber resilience of their software systems.

The findings establish that natural language processing approaches, particularly transformer-based models, provide the most effective current methodology for automated vulnerability detection, achieving F1-scores exceeding 0.99 on C/C++ code analysis. However, the research also demonstrates that simpler approaches can achieve strong performance, challenging assumptions that complex infrastructure is required for adequate security analysis. The hybrid architecture combining symbolic pattern matching with statistical machine learning reflects the principle that in security domains, preventing false negatives outweighs marginal accuracy gains from purely learned approaches, a principle of particular importance in healthcare contexts where undetected vulnerabilities may directly compromise patient safety.

As healthcare organisations continue to face escalating cyber threats, the need for proactive security integration during software development becomes increasingly urgent. This research contributes to addressing that need by providing methodologies, datasets, and tools that support the shift-left security paradigm advocated in healthcare software development standards. The production readiness testing and user validation demonstrate that the research outputs are not merely academic contributions but practical resources ready for adoption by healthcare software development organisations seeking to protect patient safety and data privacy through resilient, secure software systems.

This research has demonstrated that the longstanding separation between privacy engineering and software security assessment is neither technically necessary nor practically beneficial in healthcare software development contexts. The enhanced LINDDUN methodology, healthcare-focused vulnerability dataset, and hybrid detection prototype collectively provide a foundation on which healthcare organisations, security practitioners, and researchers can build towards more integrated, proactive approaches to securing the

digital infrastructure on which modern healthcare delivery depends. The contributions advance both the methodological and empirical frontiers of healthcare cybersecurity while delivering immediately applicable tools validated against real-world usage scenarios, an outcome that reflects the dual imperative of academic rigour and practical relevance that motivates this field of inquiry.

7 References

Abomhara, M., Gerdes, M., & Køien, G. M. (2015). A STRIDE-based threat model for telehealth systems. Norsk Informasjonssikkerhetskonferanse (NISK), 8(1), 82-96. [EDITED — original citation could not be verified; replaced with the closest-matching real source by the same first two authors. Please confirm this is the intended source.]

Acar, Y., Stransky, C., Wermke, D., Weir, C., Mazurek, M. L., & Fahl, S. (2017b). Developers need support, too: A survey of security advice for software developers. In Proceedings of the 2017 IEEE Cybersecurity Development (SecDev) (pp. 22--26). IEEE. <https://doi.org/10.1109/SecDev.2017.17>

Aggarwal, A., Jalote, P. (2006). Integrating static and dynamic analysis for detecting vulnerabilities. In Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC '06) (Vol. 1, pp. 343--350).

Ahmad, U., Zaib, H., & Qureshi, K. N. (2024). Cybersecurity standards for AIoT networks. In Artificial Intelligence of Things (AIoT) (1st ed., pp. 1--19). CRC Press. ISBN 9781003430018.

Ahmed, M. B. U., Harzevili, N. S., Shin, J., Pham, H. V., & Wang, S. (2025). SecVulEval: Benchmarking LLMs for real-world C/C++ vulnerability detection. arXiv preprint, arXiv:2505.19828.

Aijaz, M., Nazir, M., & Mohammad, M. N. A. (2023). Threat Modeling and Assessment Methods in the Healthcare-IT System: A Critical Review and Systematic Evaluation. SN Computer Science, 4, 714. https://doi.org/10.1007/s42979-023-02221-1.

Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. (2022). Toward successful DevSecOps in software development organizations: A decision-making framework. Information and Software Technology, 147, Article 106894. <https://doi.org/10.1016/j.infsof.2022.106894>.

Akshar, T., Singh, V., Bhanu Murthy, N. L., Krishna, A., & Kumar, L. (2024). A CodeBERT based empirical framework for evaluating classification-enabled vulnerability prediction

models. In Proceedings of the 17th Innovations in Software Engineering Conference (ISEC 2024). ACM. <https://doi.org/10.1145/3641399.3641405>

Al-Matouq, H., Mahmood, S., Alshayeb, M., & Niazi, M. (2020). A maturity model for secure software design: A multivocal study. IEEE Access, 8, 215758--215776. <https://doi.org/10.1109/ACCESS.2020.3040220>.

Al Asif, M. R., Hasan, K. F., Bui, F. M., & Khondoker, R. (2024). STRIDE-based cybersecurity threat modeling, risk assessment and treatment of an in-vehicle infotainment system. Vehicles, 6(3), 1140--1163. doi:10.3390/vehicles6030054.

Alazmi, S., & De Leon, D. C. (2022). A systematic literature review on the characteristics and effectiveness of web application vulnerability scanners. IEEE Access, 10, 33200--33219. <https://doi.org/10.1109/ACCESS.2022.3161522>.

Albanese, M., Adebiyi, O., & Onovae, F. (2024). CVE2CWE: Automated Mapping of Software Vulnerabilities to Weaknesses Based on CVE Descriptions. In Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT) (Vol. 1, pp. 500--507). SciTePress. <https://doi.org/10.5220/0012770400003767>.

Alder, S. (2023, July 24). IBM: Average cost of a healthcare data breach increases to almost \$11 million. The HIPAA Journal. (https://www.hipaajournal.com/2023-cost-healthcare-data-breach/)

Alder, S. (2025a, March 19). The biggest healthcare data breaches of 2024. HIPAA Journal. <https://www.hipaajournal.com/biggest-healthcare-data-breaches-2024/>.

Alder, S. (2025b, May 26). Healthcare data breach statistics. HIPAA Journal. Retrieved July 8, 2025, from <https://www.hipaajournal.com/healthcare-data-breach-statistics/>.

Aldosari, B. (2025). Cybersecurity in healthcare: New threat to patient safety. Cureus, 17(5), e83614. https://doi.org/10.7759/cureus.83614.

Alhammad, N., Alajlani, M., Abd-Alrazaq, A., Epiphaniou, G., & Arvanitis, T. (2024, May 31). Patients' perspectives on the data confidentiality, privacy, and security of mHealth apps: Systematic review. Journal of Medical Internet Research, 26, e50715. <https://doi.org/10.2196/50715>.

Ali, T., Al-Khalidi, M., & Al-Zaidi, R. (2024). Information security risk assessment methods in cloud computing: Comprehensive review. Journal of Computer Information Systems. Advance online publication. <https://doi.org/10.1080/08874417.2024.2329985>.

Aljedaani, B., Ahmad, A., Zahedi, M., & Babar, A. M. (2020). An Empirical Study on Developing Secure Mobile Health Apps: The Developers\' Perspective. IEEE, . doi:10.1109/APSEC51365.2020.00029.

Aljedaani, B., & Babar, A. M. (2021). Challenges With Developing Secure Mobile Health Applications: Systematic Review. JMIR, 9(6), 15654. doi:10.2196/15654.

Allamanis, M., Brockschmidt, M., & Khademi, M. (2018). Learning to represent programs with graphs. In Proceedings of the 7th International Conference on Learning Representations (ICLR '18). https://openreview.net/forum?id=BJOFETxR- ; https://doi.org/10.48550/arXiv.1711.00740

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A Survey of Machine Learning for Big Code and Naturalness. ACM Computing Surveys, 51(4), Article 81. https://doi.org/10.1145/3212695.

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages, 3(POPL), 1--29. https://doi.org/10.1145/3290353.

Alqaradaghi, M., Morse, G., & Kozsik, T. (2021). Detecting security vulnerabilities with static analysis: A case study. Pollack Periodica, 17(2), 1--7. <https://doi.org/10.1556/606.2021.00454>.

Alsayed Kassem, J., de Laat, C., Taal, A., & Grosso, P. (2020). The EPI framework: A dynamic data sharing framework for healthcare use cases. IEEE Access, 8, 179909--179920. https://doi.org/10.1109/ACCESS.2020.3028051.

Ameh, J. E., Otebolaku, A., Shenfield, A., & Ikpehai, A. (2025). C3-VULMAP: A dataset for privacy-aware vulnerability detection in healthcare systems. Electronics, 14(13), 2703. <https://doi.org/10.3390/electronics14132703>.

American Hospital Association. (2025). Change Healthcare cyberattack underscores urgent need to strengthen cyber preparedness for individual health care organizations and as a field. Retrieved July 8, 2025, from <https://www.aha.org/change-healthcare-cyberattack-underscores-urgent-need-strengthen-cyber-preparedness-individual-health-care-organizations-and>.

Andrade, V. C., Gomes, R. D., Reinehr, S., Freitas, C. O. de A., & Malucelli, A. (2022). Privacy by Design and Software Engineering: A systematic literature review \[Paper presentation\]. XXI Brazilian Symposium on Software Quality. <https://doi.org/10.1145/3571473.3571480>.

Aregbesola, G. D., Asghar, I., Akbar, S., & Ullah, R. (2025). Fuzzy logic model for informed decision-making in risk assessment during software design. Systems, 13(9), 825. doi:10.3390/systems13090825.

Argaw, S. T., Troncoso-Pastoriza, J. R., Lacey, D., Florin, M.-V., Calcavecchia, F., Anderson, D., Burleson, W., Vogel, J.-M., O'Leary, C., Eshaya-Chauvin, B., & Flahault, A. (2020). Cybersecurity of hospitals: Discussing the challenges and working towards mitigating the risks. BMC Medical Informatics and Decision Making, 20(1), 146. <https://doi.org/10.1186/s12911-020-01161-7>.

Arizon-Peretz, R., Hadar, I., & Luria, G. (2022). The importance of security is in the eye of the beholder: Cultural, organizational, and personal factors affecting the implementation of security by design. IEEE Transactions on Software Engineering, 48(11), 4433--4446. <https://doi.org/10.1109/TSE.2021.3119721>.

Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., & Rieck, K. (2024). Pitfalls in machine learning for computer security. Communications of the ACM, 67(11), 104-112. <https://doi.org/10.1145/3643456>.

Artic Wolf. (2023, August). The Top 15 Healthcare Industry Cyber Attacks of the Past Decade. Retrieved from articwolf.com: https://arcticwolf.com/resources/blog/top-healthcare-industry-cyberattacks/.

Assal, H., & Chiasson, S. (2019). "Think Secure from the Beginning": A Survey of Software Developers' Security Practices and Challenges. Proceedings of the 2019 CHI

Conference on Human Factors in Computing Systems. (pp. 289:1--289:13). DOI: 10.1145/3290605.3300519.

Assal, H., Morkonda, S. G., Arif, M. Z., & Chiasson, S. (2025). Software security in practice: knowledge and motivation. Journal of Cybersecurity, 11(1). <https://doi.org/10.1093/cybsec/tyaf005> (.

Atiiq, S. A., Gehrmann, C., Dahlén, K., & Khalil, K. (2024). From generalist to specialist: Exploring CWE-specific vulnerability detection \[Preprint\]. arXiv. https://doi.org/10.48550/arXiv.2408.02329.

Aufner, P. (2020). The IoT security gap: a look down into the valley between threat models and their implementation. International Journal of Information Security, 19, 3-14. doi:10.1007/s10207-019-00445-y.

Aydos, M., Aldan, Ç., Coşkun, E., & Soydan, A. (2022). Security testing of web applications: A systematic mapping of the literature. Journal of King Saud University -- Computer and Information Sciences, 34(9), 6775--6792. <https://doi.org/10.1016/j.jksuci.2021.09.018>.

Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J. D., & Penix, J. (2008). Using static analysis to find bugs. IEEE Software, 25(5), 22--29. <https://doi.org/10.1109/MS.2008.130>.

Ayewah, N., & Pugh, W. (2010). The Google FindBugs Fixit. In Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA '10) (pp. 241--252). ACM. <https://doi.org/10.1145/1831708.1831738>.

Azam, N., Michala, A. L., Ansari, S., & Truong, N. B. (2024). Modelling technique for GDPR-compliance: Toward a comprehensive solution. arXiv.

Bach-Nutman, M. (2020). Understanding the Top 10 OWASP Vulnerabilities. arXiv. <https://doi.org/10.48550/arXiv.2012.09960>.

Bahaa, A., Kamal, A. E., Fahmy, H., & Ghoneim, A. S. (2024). DB-CBIL: A DistilBert-based transformer hybrid model using CNN and BiLSTM for software vulnerability detection. IEEE Access, 12, 64446. https://doi.org/10.1109/access.2024.3396410.

Barakat, R., Weiß, P., von Blanckenburg, J., Kraus, R., & Schneider, M. A. (2024). Enhancing software security analysis: Targeted test case generation through constraint solving in interactive application security testing. In 2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C) (pp. 57--63). IEEE.

Barclay, S. (2021, December 15). National Cyber Strategy 2022. Cabinet Office, UK Government.

Basil, N. N., Ambe, S., Ekhator, C., & Fonkem, E. (2022, October 11). Health records database and inherent security concerns: A review of the literature. Cureus, 14(10), e30168. <https://doi.org/10.7759/cureus.30168>.

Belfrage, S., Helgesson, G. & Lynøe, N. Trust and digital privacy in healthcare: a cross-sectional descriptive study of trust and attitudes towards uses of electronic health data among the general public in Sweden. BMC Med Ethics 23, 19 (2022). https://doi.org/10.1186/s12910-022-00758-z.

Bilgin, Z., Ersoy, M. A., Soykan, E. Ü., Tomur, E., Çomak, P., & Karaçay, L. (2020). Vulnerability prediction from source code using machine learning. IEEE Access, 8, 150672--150684. <https://doi.org/10.1109/ACCESS.2020.3016774>.

Blinowski, G. J., Piotrowski, P., & Wiśniewski, M. (2021). Comparing support vector machine and neural network classifiers of CVE vulnerabilities. In Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT) (pp. 734--740). SciTePress. <https://doi.org/10.5220/0010574807340740>.

Böhme, M., Falk, B., & Manès, V. J. M. (2020). Fuzzing: On the exponential cost of vulnerability discovery. In Proceedings of the 14th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '20) (pp. 1--12). ACM. <https://doi.org/10.1145/3368089.3409729>.

Bracciale, L., Loreti, P., & Bianchi, G. (2023). Cybersecurity vulnerability analysis of medical devices purchased by national health services. Scientific Reports, 13(1), Article 19509. <https://doi.org/10.1038/s41598-023-45927-1>.

Bridges, R. A., Oesch, S., Verma, M. E., Iannacone, M. D., Huffer, K. M. T., Jewell, B., Nichols, J. A., Weber, B., Beaver, J. M., Smith, J. M., Scofield, D., Miles, C., Plummer, T., Daniell, M., & Tall, A. M. (2023). Beyond the hype: An evaluation of commercially available machine-learning-based malware detectors. Digital Threats: Research and Practice, 4(2). <https://doi.org/10.1145/3567432>.

Butler, H., & Bhuyan, S. (2023, July 27). Growing cybersecurity concerns for telehealth services. New Jersey State Policy Lab. Retrieved from https://policylab.rutgers.edu/publication/growing-cybersecurity-concerns-for-telehealth-services/.

Calvi, A. (2024). Data Protection Impact Assessment under the EU General Data Protection Regulation: A feminist reflection. Computer Law & Security Review, 53, 1--20. <https://doi.org/10.1016/j.clsr.2024.105950>.

Cao, Y., Tang, W., Ban, M., & Feng, M. (2025). Multi-source cross-domain vulnerability detection based on code pre-trained model. Information and Software Technology, 185, 107764. <https://doi.org/10.1016/j.infsof.2025.107764>.

Carvalho, T., Moniz, N., Faria, P., & Antunes, L. (2023). Towards a data privacy-predictive performance trade-off. Expert Systems with Applications, 223, Article 119785. <https://doi.org/10.1016/j.eswa.2023.119785>.

Casola, V., De Benedictis, A., Rak, M., & Villano, U. (2020). A novel Security-by-Design methodology: Modeling and assessing security by SLAs with a quantitative approach. Journal of Systems and Software, 163, 110537. <https://doi.org/10.1016/j.jss.2020.110537>.

Cavoukian, A. (2010). Privacy by design: The definitive workshop. A foreword by Ann Cavoukian, Ph.D. Identity in the Information Society, 3(2), 247--251. <https://doi.org/10.1007/s12394-010-0062-y>.

Cavoukian, A. (2011). Privacy by design: The 7 foundational principles. Information and Privacy Commissioner of Ontario, Canada. https://student.cs.uwaterloo.ca/~cs492/papers/7foundationalprinciples_longer.pdf

Cavoukian, A. (2012). Privacy by design: Origins, meaning, and prospects for assuring privacy and trust in the information era. In G. Yee (Ed.), Privacy protection measures and technologies in business organisations: Aspects and standards (pp. 170--208). IGI Global. (https://link.springer.com/chapter/10.1007/978-3-642-32805-3_23)

Chah, B., Lombard, A., Bkakria, A., Yaich, R., Abbas-Turki, A., & Galland, S. (2022). Privacy threat analysis for connected and autonomous vehicles. Procedia Computer Science, 210, 36--44. <https://doi.org/10.1016/j.procs.2022.10.117>.

Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2022). Deep learning based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering, 48(9), 3280--3296. https://doi.org/10.1109/TSE.2021.3087402.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Sutskever, I. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://doi.org/10.48550/arXiv.2107.03374

Chen, Y., Ding, Z., Alowain, L., Chen, X., & Wagner, D. (2023). DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection. In Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23) (pp. 1--15). Association for Computing Machinery. <https://doi.org/10.1145/3607199.3607242>.

Chen, M., Li, G., Wu, L.-I., & Liu, R. (2025). DCE-LLM: Dead code elimination with large language models. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers) (pp. 9942--9955). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2025.naacl-long.501>.

Cheng, S., Lee, S.-J., & Choi, B. (2019). An empirical investigation of users' voluntary switching intention for mobile personal cloud storage services based on the push--pull--mooring framework. Computers in Human Behavior, 92, 198--215. <https://doi.org/10.1016/j.chb.2018.10.035>

Chereja, I., Erdei, R., Delinschi, D., Pasca, E., Avram, A., & Matei, O. (2025). Privacy-Conducive Data Ecosystem Architecture: By-Design Vulnerability Assessment Using Privacy Risk Expansion Factor and Privacy Exposure Index. Sensors, 25(11), 3554. https://doi.org/10.3390/s25113554.

Chess, B., & McGraw, G. (2004). Static analysis for security: Current practices and research issues. IEEE Security & Privacy, 2(6), 76--79. <https://doi.org/10.1109/MSP.2004.111>.

Chu, L. C., Anandkumar, A., Shin, H. C., & Fishman, E. K. (2020). The potential dangers of artificial intelligence for radiology and radiologists. Journal of the American College of Radiology, 17(10), 1309--1311. https://doi.org/10.1016/j.jacr.2020.04.010.

Chu, Z., Wan, Y., Li, Q., Wu, Y., Zhang, H., Sui, Y., Xu, G., & Jin, H. (2024). Graph neural networks for vulnerability detection: A counterfactual explanation. In Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '24) (pp. 1--13). ACM. <https://doi.org/10.1145/3650212.3652136>.

Collins, G. S., Reitsma, J. B., Altman, D. G., & Moons, K. G. M. (2015). Transparent reporting of a multivariable prediction model for individual prognosis or diagnosis: The TRIPOD statement. Annals of Internal Medicine, 162(1), 55--63. https://doi.org/10.7326/M14-0697.

Collins, C., Dennehy, D., Conboy, K., & Mikalef, P. (2021). Artificial intelligence in information systems research: A systematic literature review and research agenda. International Journal of Information Management, 60, 102383. https://doi.org/10.1016/j.ijinfomgt.2021.102383.

Cremers, C., Fontaine, C., & Jacomme, C. (2022). A Logic and an Interactive Prover for the Computational Post-Quantum Security of Protocols. 2022 IEEE Symposium on Security and Privacy (SP) (pp. 125-141). IEEE. doi:10.1109/SP46214.2022.9833800.

Croft, R., Newlands, D., Chen, Z., & Babar, M. A. (2021). An empirical study of rule-based and learning-based approaches for static application security testing \[Preprint\]. arXiv. <https://doi.org/10.48550/arXiv.2107.01921>.

Croft, R., Xie, Y., Zahedi, M., Babar, M. A., & Treude, C. (2022). An empirical study of developers' discussions about security challenges of different programming languages. Empirical Software Engineering, 27(1), 27. https://doi.org/10.1007/s10664-021-10054-w.

Cybersecurity and Infrastructure Security Agency. (2025). Known exploited vulnerabilities catalog: 2024 annual analysis. U.S. Department of Homeland Security. https://www.cisa.gov/known-exploited-vulnerabilities-catalog

Dang, Q.-V., & François, J. (2018). Utilizing attack enumerations to study SDN/NFV vulnerabilities. In Proceedings of the 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft) (pp. 356--361). IEEE. <https://doi.org/10.1109/NETSOFT.2018.8459961>.

Del-Real, C., de Busser, E., & van den Berg, B. (2024). Shielding software systems: A comparison of security by design and privacy by design based on a systematic literature review. Computer Law & Security Review, 52, Article 102507. <https://doi.org/10.1016/j.clsr.2024.102507>.

DeLozier, C. (2024). How Close Is Existing C/C++ Code to a Safe Subset? Journal of Cybersecurity and Privacy, 4(1), 1-22. <https://doi.org/10.3390/jcp4010001>.

Demetzou, K. (2019). Data Protection Impact Assessment: A tool for accountability and the unclarified concept of 'high risk' in the General Data Protection Regulation. Computer Law & Security Review, 35(6), 105342. <https://doi.org/10.1016/j.clsr.2019.105342>.

Deng, M., Wuyts, K., Scandariato, R., Preneel, B., & Joosen, W. (2011). A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. Requirements Engineering, 16(1), 3--32. <https://doi.org/10.1007/s00766-010-0115-7>.

Department of Health and Human Services. (2013). HIPAA security rule: Administrative, physical, and technical safeguards. Federal Register, 78(17), 5566-5702. [CORRECTED — original cited Federal Register 68(34); the 2013 Omnibus Rule is actually Vol. 78, No. 17.] https://www.federalregister.gov/documents/2013/01/25/2013-

01073/modifications-to-the-hipaa-privacy-security-enforcement-and-breach-notification-rules-under-the

Department for Science, Innovation and Technology. (2025, May 7). Software Security Code of Practice. GOV.UK. https://www.gov.uk/government/publications/software-security-code-of-practice/software-security-code-of-practice.

Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D., Ray, B., & Chen, Y. (2024). Vulnerability detection with code language models: How far are we? arXiv preprint arXiv:2403.18624.

Dzamesi, L., & Elsayed, N. (2025). A Review on the Security Vulnerabilities of the IoMT against Malware Attacks and DDoS. arXiv. https://doi.org/10.48550/arXiv.2501.07703.

Ebad, S. A. (2022). Exploring how to apply secure software design principles. IEEE Access, 10, 128983--128996. <https://doi.org/10.1109/ACCESS.2022.3227434>.

Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. Applied Sciences, 12(9), Article 4369. <https://doi.org/10.3390/app12094369>.

Eichelberg, M., Kleber, K., & Kämmerer, M. (2020). Cybersecurity in PACS and Medical Imaging: An Overview. Journal of Digital Imaging, 33(4), 1527--1542. <https://doi.org/10.1007/s10278-020-00393-3>.

Elder, S., Zahan, N., Kozarev, V., Shu, R., Menzies, T., & Williams, L. A. (2021). Structuring a comprehensive software security course around the OWASP Application Security Verification Standard. In Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) (pp. 95--104). IEEE. https://doi.org/10.1109/ICSE%E2%80%91SEET52601.2021.00019.

Elkourdi, F., Wei, C., Xiao, L., Yu, Z., & Asan, O. (2024). Exploring current practices and challenges of HIPAA compliance in software engineering: Scoping review. IEEE Open Journal of Systems Engineering, (99), 1--10. <https://doi.org/10.1109/OJSE.2024.3392691>.

Essafi, M., Jilani, L., & Ghezala, B. H. (2007). S2D-Prom: A strategy oriented process model for secure software development. International Conference for Software Engineering Advancement, (pp. 24-28).

Ewoh, P., & Vartiainen, T. (2024). Vulnerability to cyberattacks and sociotechnical solutions for health care systems: Systematic review. Journal of Medical Internet Research, 26, e46904. <https://doi.org/10.2196/46904>.

Fan, J., Li, Y., Wang, S., & Nguyen, T. N. (2020). A C/C++ code vulnerability dataset with code changes and CVE summaries (Big-Vul). In Proceedings of the 17th International Conference on Mining Software Repositories (MSR) (pp. 508--512). https://doi.org/10.1145/3379597.3387501.

Fan, L., Xiao, T., Xian, C., Ding, W., & Wang, X. (2022). Effect of short-term frozen storage on taste of gonads of female Eriocheir sinensis and the classification of taste quality combined with sensory evaluation and fuzzy logic model. Food Chemistry, 378, 132105. doi:10.1016/j.foodchem.2022.132105.

Fan, Hu, Wang, Y., Du, L., Zhang, H., Han, S., Zhang, D., & Li, X. (2024). Tackling long code search with splitting, encoding, and aggregating. In Findings of the LREC-COLING 2024 Conference (pp. 15500--15510).

Felmetsger, V., Cavedon, L., Kruegel, C., & Vigna, G. (2010). Toward automated detection of logic vulnerabilities in web applications. In Proceedings of the 19th USENIX Security Symposium (USENIX Security '10). USENIX Association.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. In T. Cohn, Y. He, & Y. Liu (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2020 (pp. 1536--1547). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.findings-emnlp.139>.

Feng, S., Keung, J., Yu, X., Xiao, Y., & Zhang, M. (2021). Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction. Information and Software Technology, 139, 106662. https://doi.org/10.1016/j.infsof.2021.106662.

Ferrara, P., Olivieri, L., & Spoto, F. (2021). Static privacy analysis by flow reconstruction of tainted data. International Journal of Software Engineering and Knowledge Engineering, 31(7), 973--1016. <https://doi.org/10.1142/S0218194021500303>.

Field Seymour Parkes LLP (FSP Law). (2025, June 17). New UK Software Security Code of Practice -- what does it mean for software vendors? FSP Law. https://www.fsp-law.com/new-uk-software-security-code-of-practice-what-does-it-means-for-software-vendors/

Finn, R. L., Wright, D., & Friedewald, M. (2013). Seven Types of Privacy. In S. L. Gutwirth, European Data Protection: Coming of Age (pp. 3-12). Dordrecht: Springer. doi:10.1007/978-94-007-5170-5_1.

Folorunso, A., Mohammed, V., & Wada, I. (2024). The impact of ISO security standards on enhancing cybersecurity posture in organizations. World Journal of Advanced Research and Reviews, 24(1), 2582--2595. <https://doi.org/10.30574/wjarr.2024.24.1.3169>.

Fonseca, J. C. M., & Vieira, M. (2013). A survey on secure software development lifecycles. In Software Design and Development: Concepts, Methodologies, Tools, and Applications (Chapter 2). IGI Global. <https://doi.org/10.4018/978-1-4666-3679-8.ch003>.

Fu, M., & Tantithamthavorn, C. (2022). LineVul: A transformer-based line-level vulnerability prediction. In Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022) (pp. 608-620). ACM. https://doi.org/10.1145/3524842.3528452.

Fu, M., Tantithamthavorn, C., Le, T., Kume, Y., Nguyen, V., Phung, D., & Grundy, J. (2024). AIBugHunter: A practical tool for predicting, classifying and repairing software vulnerabilities. Empirical Software Engineering, 29(4), 2024. https://doi.org/10.1007/s10664-023-10346-3.

Fucci, D., Alégroth, E., Felderer, M., & Johannesson, C. (2024). Evaluating software security maturity using OWASP SAMM: Different approaches and stakeholders perceptions. Journal of Systems and Software, 214, Article 112062.

Gao, Y., Zhu, Y., & Zhao, Y. (2022). Dealing with imbalanced data for interpretable defect prediction. Information and Software Technology, 144, 107016.

Gao, K., Bai, J., Wu, D., & Xia, S. (2023). Backdoor Attack on Hash-based Image Retrieval via Clean-label Data Poisoning. Computer Vision and Pattern Recognition. doi:10.48550/arXiv.2109.08868.

Garfinkel, S. L. (2015). De-Identification of Personal Information (NIST IR 8053). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8053.

Gazzarata, R., Almeida, J., Lindsköld, L., Cangioli, G., Gaeta, E., Fico, G., & Chronaki, C. E. (2024). HL7 Fast Healthcare Interoperability Resources (HL7 FHIR) in digital healthcare ecosystems for chronic disease management: Scoping review. International Journal of Medical Informatics, 189, 105507. <https://doi.org/10.1016/j.ijmedinf.2024.105507>.

Ghaffarian, S. M., & Shahriari, H. R. (2017). Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey. ACM Computing Surveys, 50(4), 1-36. doi:10.1145/3092566.

Gómez, L. F., & Heeks, R. (2016). Measuring the barriers to big data for development: Design-reality gap analysis in Colombia's public sector (Development Informatics Working Paper No. 62). Centre for Development Informatics, University of Manchester. https://doi.org/10.2139/ssrn.3431745.

Gordon, W. J., & Mandl, K. D. (2020, December 11). The 21st Century Cures Act: A competitive apps market and the risk of innovation blocking. Journal of Medical Internet Research, 22(12), e24824. <https://doi.org/10.2196/24824>.

Granata, D., & Rak, M. (2024). Systematic analysis of automated threat modelling techniques: Comparison of open-source tools. Software Quality Journal, 32(1), 125-161. https://doi.org/10.1007/s11219-023-09634-4.

Gritzalis, D., Stergiopoulos, G., Vasilellis, E., & Anagnostopoulou, A. (2020). Readiness Exercises: Are Risk Assessment Methodologies Ready for the Cloud? Advances in Core Computer Science-Based Technologies, 109-128. doi:10.1007/978-3-030-41196-1_6.

Grundy, Q., Chiu, K., Held, F., Continella, A., Bero, L., & Holz, R. (2019). Data sharing practices of medicines related apps and the mobile ecosystem: traffic, content, and network analysis. BMJ, 364, l920. https://doi.org/10.1136/bmj.l920.

Gueye, A., Galhardo, C. E. C., Bojanova, I., & Mell, P. (2021). A decade of reoccurring software weaknesses. IEEE Security & Privacy, 19(6), 74--82. https://doi.org/10.1109/MSEC.2021.3082757.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C. B., Drain, D., Sundaresan, N., Yin, J., Jiang, D., & Zhou, M. (2021). GraphCodeBERT: Pre-training code representations with data flow. In Proceedings of the International Conference on Learning Representations (ICLR 2021).

Guo, Z., Tan, T., Liu, S., Liu, X., Lai, W., Yang, Y., Li, Y., Chen, L., Dong, W., & Zhou, Y. (2023). Mitigating false positive static analysis warnings: Progress, challenges, and opportunities. IEEE Transactions on Software Engineering, 49(12), 5154--5188. <https://doi.org/10.1109/TSE.2023.3329667>.

Guo, Y. (2023). A review of Machine Learning-based zero-day attack detection: Challenges and future directions. Computer Communications, 198, 175-185. doi:10.1016/j.comcom.2022.11.001.

Guo, Y., Bettaieb, S., & Casino, F. (2024). A comprehensive analysis on software vulnerability detection datasets: trends, challenges, and road ahead. International Journal of Information Security, 23(5), 3311--3327. https://doi.org/10.1007/s10207-024-00888-y.

Guo, Y., Hu, Q., Tang, Q., & Le Traon, Y. (2024). An empirical study of the imbalance issue in software vulnerability detection. In G. Tsudik, M. Conti, K. Liang, & G. Smaragdakis (Eds.), Computer Security -- ESORICS 2023 (Lecture Notes in Computer Science, Vol. 14347, pp. 371--390). Springer. https://doi.org/10.1007/978-3-031-51482-1_19.

Gupta, S., Faisal, M., & Husain, M. (2012). Secure software development for embedded systems control. International Journal Engineering for Science and Emerging Technology, 4, 133-143.

Gutfleisch, M., Schöps, M., Horstmann, S. A., Wichmann, D., & Sasse, M. A. (2023). Security Champions Without Support: Results from a Case Study with OWASP SAMM in a Large-Scale E-Commerce Enterprise. In Proceedings of the 2023 European Symposium on Usable Security (EuroUSEC 2023) (pp. 260--276). ACM. 10.1145/3617072.3617115.

Hai, N. L., Nguyen, D. M., & Bui, N. D. Q. (2025). On the impacts of contexts on repository-level code generation. In Findings of the Association for Computational Linguistics: NAACL 2025 (pp. 1496--1524). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2025.findings-naacl.82>.

Hanif, H., & Maffeis, S. (2022). VulBERTa: Simplified source code pre-training for vulnerability detection \[Conference paper\]. In Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN) (pp. 1--8). IEEE. <https://doi.org/10.1109/IJCNN55064.2022.9892280>.

Harikrishnan, K. V. (2025, June 28). Security Risks in Internet-Connected Medical Devices (IoMT): A Threat Analysis and Mitigation Approach. IJRASET. <https://doi.org/10.22214/ijraset.2025.72862>.

Harirchian, E., & Lahmer, T. (2020). Developing a hierarchical type-2 fuzzy logic model to improve rapid evaluation of earthquake hazard safety of existing buildings. Structures, 1384-1399. doi:10.1016/j.istruc.2020.09.048.

Harriharan, M., Sathish, C. K., Tanwar, A., Sundaresan, K., Ganesan, P., Ravi, S., & Karthik, R. (2022). Proximal instance aggregator networks for explainable security vulnerability detection. Future Generation Computer Systems, 134, 303--318. https://doi.org/10.1016/j.future.2022.04.008.

Harzevili, N. S., Belle, A. B., Wang, J., Wang, S., Jiang, Z. M. J., & Nagappan, N. (2024). A systematic literature review on automated software vulnerability detection using machine learning. ACM Computing Surveys, 57(3), 1--36. <https://doi.org/10.1145/3699711>.

Hasegawa, K., O'Brien, N., Prendergast, M., Ajah, C. A., Neves, A. L., & Ghafur, S. (2024). Cybersecurity interventions in health care organizations in low- and middle-income

countries: Scoping review. Journal of Medical Internet Research, 26, e47311. <https://doi.org/10.2196/47311>.

He, H., & Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263--1284. https://doi.org/10.1109/TKDE.2008.239.

He, Y., Aliyu, A., Evans, M., & Luo, C. (2021). Health care cybersecurity challenges and solutions under the climate of COVID-19: Scoping review. Journal of Medical Internet Research, 23 (4), e21747. \<https://doi.org/10.2196/21747>.

He, X., Asiya, Han, D., Zhou, S., Fu, X., & Li, H. (2025). An improved software source code vulnerability detection method: Combination of multi-feature screening and integrated sampling model. Sensors, 25(6), 1816. <https://doi.org/10.3390/s25061816>.

Heiland, R., Thomas, B., Welch, V., & Jackson, C. (2013). Toward a research software security maturity model [Preprint]. arXiv. https://doi.org/10.48550/arXiv.1309.1677

Hellwig, O., Quirchmayr, G., Huber, E., Goluch, G., Vock, F., & Pospisil, B. (2016). Major challenges in structuring and institutionalizing CERT-Communication. In 2016 11th International Conference on Availability, Reliability and Security (ARES) (pp. 661--667). IEEE. <https://doi.org/10.1109/ARES.2016.57>.

Hill, E., Pollock, L., & Vijay-Shanker, K. (2009). Automatically capturing source code context of NL-queries for software maintenance and reuse. In Proceedings of the 31st International Conference on Software Engineering (ICSE 2009) (pp. 232--242). IEEE. <https://doi.org/10.1109/ICSE.2009.5070520>.

Hills, A., Da Veiga, A., Loock, M., & Renaud, K. (2025). A Holistic List of Privacy-Preserving Measures for System Development Life Cycles. In T. Guarda, F. Portela, & G. Gatica (Eds.), Advanced Research in Technologies, Information, Innovation and Sustainability (ARTIIS 2024; Communications in Computer and Information Science, Vol. 2347, pp. 283--300). Springer, Cham.

Hin, D., Kan, A., Chen, H., & Babar, M. A. (2022). LineVD: Statement-level vulnerability detection using graph neural networks. In Proceedings of the 19th International

Conference on Mining Software Repositories (MSR '22) (pp. 596--607). ACM. https://doi.org/10.1145/3524842.3527949.

Honkaranta, A., Leppänen, T., & Costin, A. (2021). Towards practical cybersecurity mapping of STRIDE and CWE -- a multi-perspective approach. In S. Balandin, Y. Koucheryavy, & T. Tyutina (Eds.), Proceedings of the 29th Conference of Open Innovations Association (FRUCT) (pp. 150--159). IEEE. <https://doi.org/10.23919/FRUCT52173.2021.9435453>.

Honkaranta, A., Baca, D., & Holmström Olsson, H. (2022). Bridging the gap between threat modeling and security testing: A systematic mapping study. Information and Software Technology, 148, 106964. DOI 10.1016/j.infsof.2022.106841

Hou, C., Sun, Y., Li, L., Chen, W., & Xu, X. (2023). Code defect detection model with multi-layer bi-directional long short term memory based on self-attention mechanism. ACM International Conference Proceeding Series, 1656. https://doi.org/10.1145/3650400.3650676.

Houser, S. H., Flite, C. A., & Foster, S. L. (2023, January 10). Privacy and security risk factors related to telehealth services -- a systematic review. Perspectives in Health Information Management, 20(1), Article 1f. (https://pubmed.ncbi.nlm.nih.gov/37215337/)

Howard, M., & Lipner, S. (2006). The security development lifecycle: A process for developing demonstrably more secure software. Microsoft Press.

Howard, M. (2009). Improving software security by eliminating the CWE Top 25 vulnerabilities. IEEE Security & Privacy, 7(3), 68--71. <https://doi.org/10.1109/MSP.2009.69>.

Hu, S., Zhou, Z., Zhang, Y., Zhang, L. Y., Zheng, Y., He, Y., & Jin, H. (2022). BadHash: Invisible Backdoor Attacks against Deep Hashing with Clean Label. MM \'22: Proceedings of the 30th ACM International Conference on Multimedia (pp. 678-686). ACM. doi:10.1145/3503161.3548272.

Hüther, L., Sohr, K., Berger, B. J., Rothe, H., & Edelkamp, S. (2023). Machine learning for SAST: A lightweight and adaptable approach. In Lecture Notes in Computer Science: Computer Security -- ESORICS 2023 (Vol. 14347, pp. 85--104). Springer. <https://doi.org/10.1007/978-3-031-51482-1_5>.

Hu, Q., Guo, Y., Cordy, M., Xie, X., Ma, L., Papadakis, M., & Le Traon, Y. (2023). CodeS: Towards code model generalization under distribution shift. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (pp. 1--6). IEEE. <https://doi.org/10.1109/ICSE-NIER58687.2023.00007>.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., & Qin, B. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. arXiv preprint, arXiv:2311.05232.

Husein, R. A., Aburajouh, H., & Catal, C. (2025). Large language models for code completion: A systematic literature review. Computer Standards and Interfaces, 92, Article 103917. <https://doi.org/10.1016/j.csi.2024.103917>.

IBM Security. (2022). Cost of a Data Breach Report 2022. Armonk, NY: IBM Corporation. https://www.ibm.com/reports/data-breach [2022 edition — verify link resolves to the correct year, as IBM's report URL is evergreen and updates annually]

Information Commissioner's Office. (2024). Tackling barriers to privacy-enhancing technologies adoption: A PETs project report. Information Commissioner's Office.

International Organization for Standardization. (2023). Consumer protection --- Privacy by design for consumer goods and services --- Part 1: High-level requirements (ISO Standard No. 31700-1:2023).

Iott, B. E., Campos-Castillo, C., & Anthony, D. L. (2020). Trust and privacy: How patient trust in providers is related to privacy behaviors and attitudes. AMIA Annual Symposium Proceedings, 2019, 487--493.

Iwaya, L. H., Babar, M. A., & Rashid, A. (2023). Privacy Engineering in the Wild: Understanding the Practitioners' Mindset, Organisational Aspects, and Current Practices. IEEE

Transactions on Software Engineering, 49(9), 4324--4348. https://doi.org/10.1109/TSE.2023.3290237.

Jaatun, M. G. (2017). The Building Security in Maturity Model as a research tool. In Empirical Research for Software Security: Foundations and Experience (pp. 201--208). CRC Press.

Jalali, M. S., Razak, S., Gordon, W., Perakslis, E., & Madnick, S. (2019). Health care and cybersecurity: Bibliometric analysis of the literature. Journal of Medical Internet Research, 21(2), e12644. <https://doi.org/10.2196/12644>.

Javaid, M., Haleem, A., Singh, R. P., & Suman, R. (2023). Towards insighting cybersecurity for healthcare domains: A comprehensive review of recent practices and trends. Cyber Security & Applications, 1, Article 100016. <https://doi.org/10.1016/j.csa.2023.100016>.

Jawad, L. A. (2024). Security and privacy in digital healthcare systems: Challenges and mitigation strategies. Abhigyan, 42(1), 23--31. https://doi.org/10.1177/09702385241233073

Jean Haurogné, Nihala Basheer, & Shareeful Islam. (2024). Vulnerability detection using BERT-based LLM model with transparency obligation practice towards trustworthy AI. Machine Learning with Applications, 18, Article 100598. <https://doi.org/10.1016/j.mlwa.2024.100598>.

Joeres, R., Blumenthal, D. B., & Kalinina, O. V. (2025). Data splitting to avoid information leakage with DataSAIL. Nature Communications, 16(1), Article 3337. <https://doi.org/10.1038/s41467-025-58606-8>.

Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13) (pp. 672--681). IEEE.

Johnson, J. M., & Khoshgoftaar, T. M. (2019). Survey on deep learning with class imbalance. Journal of Big Data, 6(1), 27. https://doi.org/10.1186/s40537-019-0192-5.

Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Pixy: A static analysis tool for detecting web application vulnerabilities. In 2006 IEEE Symposium on Security and Privacy (S&P'06) (pp. 258--263). IEEE. <https://doi.org/10.1109/SP.2006.31>.

Jøsang, A. (2024). Governance and information security management. In Cybersecurity (pp. 377--403). Springer. <https://doi.org/10.1007/978-3-031-68483-8_18>.

Kaixuan Li, Chen S., Fan L., Feng R., Liu H., Liu C., Liu Y., & Chen Y. (2023). Comparison and evaluation on Static Application Security Testing (SAST) tools for Java. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23) (pp. 921--933). ACM.

Kalouptsoglou, I., Siavvas, M., Kehagias, D., Chatzigeorgiou, A., & Ampatzoglou, A. (2022). Examining the capacity of text mining and software metrics in vulnerability prediction. Entropy, 24(5), 651. https://doi.org/10.3390/e24050651.

Kanakogi, K., Washizaki, H., Fukazawa, Y., Ogata, S., Okubo, T., Kato, T., Kanuka, H., Hazeyama, A., & Yoshioka, N. (2021, July 26). Tracing CVE vulnerability information to CAPEC attack patterns using natural language processing techniques. Information, 12(8), 298. <https://doi.org/10.3390/info12080298>.

Kang, H. J., Aw, K. L., & Lo, D. (2022). Detecting false alarms from automatic static analysis tools: How far are we? In Proceedings of the 44th International Conference on Software Engineering (ICSE '22) (pp. 698--709). ACM. <https://doi.org/10.1145/3510003.3510214>.

Karampatsis, R.-M., & Sutton, C. (2020). Big code ≠ big vocabulary: Open-vocabulary models for source code. In Proceedings of the 41st International ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '20) (pp. 232--247). https://doi.org/10.1145/3377811.3380342.

Karmakar, A., & Robbes, R. (2021). What do pre-trained code models know about code? In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 1332--1336). IEEE/ACM. https://doi.org/10.1109/ASE51524.2021.9678927.

Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2021). Systematic Literature Review on Security Risks. IEEE Access, 10, 5456 - 5481. doi:10.1109/ACCESS.2022.3140181.

Khan, Z. A., Garg, A., Guo, Y., & Tang, Q. (2025). Evaluating pre-trained models for multi-language vulnerability patching. arXiv, arXiv:2501.07339. https://doi.org/10.48550/arXiv.2501.07339

Khanum, S., Qadir, M. A., & Jehan, N. (2023). Comparative evaluation of static and dynamic approaches for security testing of web applications. Journal of King Saud University -- Computer and Information Sciences, 34(9), 6775--6792. <https://doi.org/10.1016/j.jksuci.2022.04.014>.

Khleel, N. A., & Nehéz, K. (2023). A novel approach for software defect prediction using CNN and GRU based on smote tomek method. Journal of Intelligent Information Systems, 60(3), 673--707. https://doi.org/10.1007/s10844-023-00793-1.

Kim, Theodos, & Sittig, S. (2020, December 7). Health Information Privacy Laws in the Digital Age: HIPAA Doesn't Apply. Perspectives in Health Information Management, 18(Winter), 1l.

Kim, K. H., Kim, K., & Kim, H. K. (2022). STRIDE-based threat modeling and DREAD evaluation for the distributed control system in the oil refinery. ETRI Journal, 44(6), 991--1003. https://doi.org/10.4218/etrij.2021-0181.

Kirlappos, I., & Sasse, M. A. (2015). Fixing security together: Leveraging trust relationships to improve security in organizations. In Proceedings of the 2015 USENIX Workshop on Usable Security (USEC '15) (pp. 1--10). Internet Society. <https://doi.org/10.14722/usec.2015.230132>.

Kissel, R., Stine, K., & Scholl, M. (2008). Security Considerations in the System Development Life Cycle (NIST Special Publication 800-64 Revision 2). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800%E2%80%9164r2.

Knight, A. V. (2021). Playing with FHIR: Hacking and Securing FHIR API Implementations. Knight Ink (White Paper)https://www.healthdatamanagement.com/articles/hl7-responds-to-concerns-about-security-concerns-about-

fhir#:~:text=vendors%20with%20no%20vulnerabilities,in.%E2%80%9Dhttps://www.healthdatamanagement.com/articles/hl7-responds-to-concerns-about-security-concerns-about-fhir#:~:text=Knight%20contends%20that%20vulnerability%20rises,%E2%80%9D.

Koboyatshwene, T., & Ayalew, Y. (2025). Requirements traceability: A systematic literature review. In Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25) ACM. https://doi.org/10.1145/3672608.3707952.

Kong, Z., Li, S., Huang, H., & Su, Z. (2024). Sand: Decoupling sanitization from fuzzing for low overhead \[Preprint\]. arXiv. <https://doi.org/10.48550/arXiv.2402.16497v2>.

Koops, B.-J., Hildebrandt, M., & Jaquet-Chiffelle, D.-O. (2010). Bridging the accountability gap: Rights for new entities in the information society? Minnesota Journal of Law, Science & Technology, 11(2), 497--561.

Kruse, C. S., Frederick, B., Jacobson, T., & Monticone, D. K. (2017). Cybersecurity in healthcare: A systematic review of modern threats and trends. Technology and Health Care, 25(1), 1--10. <https://doi.org/10.3233/THC-161263>.

Kudriavtseva, A., & Gadyatskaya, O. (2022). Secure Software Development Methodologies: A Multivocal Literature Review \[Preprint\]. arXiv. <https://doi.org/10.48550/arXiv.2211.16987>.

Kurii, Y., & Opirskyy, I. (2022). Analysis and comparison of the NIST SP 800-53 and ISO/IEC 27001:2013. In V. Sokolov, T. Radivilova, V. Ustimenko, & M. Nazarkevych (Eds.), Proceedings of the Cybersecurity Providing in Information and Telecommunication Systems (CPITS 2022) (co-located with Problems of Infocommunications. Science and Technology, PICST 2022), Kyiv, Ukraine, October 13 (online).

Lee, J., Lee, I., & Kang, J. (2019). Self-attention graph pooling. arXiv. https://doi.org/10.48550/arXiv.1904.08082.

Lee, J., & Choi, S. J. (2021). Hospital productivity after data breaches: Difference-in-differences analysis. Journal of Medical Internet Research, 23(7), e26157. https://doi.org/10.2196/26157.

Leevy, J. L., Khoshgoftaar, T. M., Bauder, R. A., & Seliya, N. (2018). A survey on addressing high-class imbalance in big data. Journal of Big Data, 5, Article 42. https://doi.org/10.1186/s40537-018-0151-6.

Leokhin, Y., & Fatkhulin, T. (2024). Research of Static Application Security Testing Technique Problems and Methods for Solving Them. In Proceedings of the 2024 Systems of Signals Generating and Processing in the Field of On-Board Communications (pp. 1--7). IEEE. <https://doi.org/10.1109/IEEECONF60226.2024.10496755>.

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2016). Gated graph sequence neural networks. In Proceedings of the 4th International Conference on Learning Representations (ICLR). https://doi.org/10.48550/arXiv.1511.05493

Li, J., Zhao, B., & Zhang, C. (2018). Fuzzing: A survey. Cybersecurity, 1(1), Article 6. https://doi.org/10.1186/s42400-018-0002-y

Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., & Zhong, Y. (2018). VulDeePecker: A deep learning-based system for vulnerability detection. In Proceedings of the Network and Distributed System Security Symposium (NDSS 2018). The Internet Society. <https://doi.org/10.14722/ndss.2018.23158>

Li, H., Diao, R., Zhang, X., Lin, X., Lu, X., Shi, D., Wang, Z., & Wang, L. (2019). An integrated online dynamic security assessment system for improved situational awareness and economic operation. IEEE Access, 7, 162571--162582. <https://doi.org/10.1109/ACCESS.2019.2952178>

Li, Y., Wang, S., Nguyen, T. N., Le, V., & Nguyen, A. T. (2021). Vulnerability detection with fine-grained interpretations. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (pp. 292--304). ACM.

Li, Y., Zhang, R., Yan, X., & Fan, K. (2023). Machine learning facilitating the rational design of nanozymes. Journal of Materials Chemistry B, 11(28), 6466--6477. <https://doi.org/10.1039/D3TB00842H>.

Li, R., Ben Allal, L., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., ... de Vries, H. (2023). StarCoder: May the source be with you!. Transactions on Machine Learning Research, 2023. https://doi.org/10.48550/arXiv.2305.06161.

Li, L., Ding, S. H. H., Tian, Y., Fung, B. C. M., Charland, P., Ou, W., & Chen, C. (2023). VulANalyzeR: Explainable binary vulnerability detection with multi-task learning and attentional graph convolution. ACM Transactions on Privacy and Security, 26(3), 1. https://doi.org/10.1145/3585386.

Li, S., Surineni, K., & Prabhakaran, N. (2025). Cyber-attacks on hospital systems: A narrative review. The American Journal of Geriatric Psychiatry: Open Science, Education, and Practice, 7, 30--39. https://doi.org/10.1016/j.osep.2025.03.002.

Liang, C., Wei, Q., Du, J., Wang, Y., & Jiang, Z. (2025). Survey of source code vulnerability analysis based on deep learning. Computers & Security, 148, Article 104098. <https://doi.org/10.1016/j.cose.2024.104098>.

Lin, C., Xu, Y., Fang, Y., & Liu, Z. (2023). VulEye: A novel graph neural network vulnerability detection approach for PHP application. Applied Sciences, 13(2), 825. <https://doi.org/10.3390/app13020825>.

Lipner, S. (2004, December). The Trustworthy Computing Security Development Lifecycle. In Proceedings of the 20th Annual Computer Security Applications Conference (pp. 2--13). IEEE. <https://doi.org/10.1109/CSAC.2004.41>.

Liu, Y., Gao, L., Yang, M., Xie, Y., Chen, P., Zhang, X., & Chen, W. (2024). VulDetectBench: Evaluating the deep capability of vulnerability detection with large language models. arXiv preprint, arXiv:2406.07595 . https://doi.org/10.48550/arXiv.2406.07595

Livshits, V. B., & Lam, M. S. (2005). Finding security vulnerabilities in Java applications with static analysis. In Proceedings of the 14th USENIX Security Symposium (USENIX Security '05) (pp. 271--286). USENIX Association.

Lohmann, P. A., Albuquerque, C., & Machado, R. (2023). Systematic literature review of threat modeling concepts. In Proceedings of the 13th International Joint Conference on

Knowledge Discovery, Knowledge Engineering and Knowledge Management (pp. 223--232). SCITEPRESS. (<https://doi.org/10.5220/0001178300003899>)

Lopez, J. A. H., Chen, B., Saad, M., Sharma, T., & Varró, D. (2025). On inter-dataset code duplication and data leakage in large language models. IEEE Transactions on Software Engineering, 51(1), 192--205. <https://doi.org/10.1109/TSE.2024.3504286>.

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In Proceedings of the 7th International Conference on Learning Representations (ICLR). https://doi.org/10.48550/arXiv.1711.05101

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S., & Liu, S. (2021). CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In Proceedings of the Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track (Vol. 1). https://doi.org/10.48550/arXiv.2102.04664

Lund, M. S., Solhaug, B., & Stølen, K. (2010). Model-driven risk analysis: The CORAS approach. Springer. https://doi.org/10.1007/978-3-642-12323-8.

Luo, L., Schäf, M., Sanchez, D., & Bodden, E. (2021). IDE support for cloud-based static analyses. In Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21) (pp. 1178--1189). ACM. <https://doi.org/10.1145/3468264.3468535>.

Lyngaas, S., & Rind, D. (2023, February 3). Apparent cyberattack forces Florida hospital system to divert some emergency patients to other facilities. Retrieved from CNN.com: https://edition.cnn.com/2023/02/03/politics/cyberattack-hospital-tallahassee-memorial-florida/index.html.

Lyngaas, S. (2023, August 4). Cyberattack forces hospitals to divert ambulances in Connecticut and Pennsylvania. Retrieved from CNN.com: https://edition.cnn.com/2023/08/04/politics/cyberattack-connecticut-pennsylvania-hospitals-ambulances/index.html.

Lyngaas, S. (2023, May 31). Cyberattack forces Idaho hospital to send ambulances elsewhere. Retrieved from CNN.com: https://edition.cnn.com/2023/05/31/politics/idaho-hospital-cyberattack/index.html.

Madnick, S. E., Li, X., & Choucri, N. (2009, September). Experiences and challenges with using CERT data to analyze international cyber security (CISL Working Paper No. 2009-13). MIT Sloan School of Management.

Maher, Z. A., Shaikh, H., Khan, M. S., Arbaaeen, A., & Shah, A. (2018). Factors affecting secure software development practices among developers---An investigation. In Proceedings of the 2018 IEEE 5th International Conference on Engineering Technologies and Applied Sciences (ICETAS) (pp. 1--6). IEEE. <https://doi.org/10.1109/ICETAS.2018.8629168>.

Malik, A., Kumar, A., Salih, S. Q., Kim, S., Kim, W. N., Yaseen, Z. M., & Singh, V. P. (2020). Drought index prediction using advanced fuzzy logic model: Regional case study over Kumaon in India. Plus One, 15(5). doi:10.1371/journal.pone.0233280.

Mao, F., Yuan, Y., Du, X., Gao, L., & Du, Z. (2025). MPDA: A data augmentation approach to improve deep learning for software vulnerability detection. Empirical Software Engineering, 30, Article 140. <https://doi.org/10.1007/s10664-025-10698-y>.

Markov, A. S., Varenitca, V. V., & Arustamyan, S. S. (2023). Topical issues in the implementation of secure software development processes. In Proceedings of the International Conference on Information Processes and Systems Development and Quality Assurance (IPSQDA 2023) (pp. 48--54). IEEE. <https://doi.org/10.1109/ISTP60767.2023.10427367>.

Martin, R. A., & Barnum, S. (2008, April). Common Weakness Enumeration (CWE) status update. ACM SIGAda Ada Letters, 28(1), 88--91. <https://doi.org/10.1145/1387830.1387835>.

Masud, S. M. R. A., Masnun, M., Sultana, A., Sultana, A., Ahmed, F., & Begum, N. (2022). DevOps Enabled Agile: Combining Agile and DevOps methodologies for software development. International Journal of Advanced Computer Science and Applications, 13(11). <https://doi.org/10.14569/IJACSA.2022.0131131>

Matthew, F. (2018, October 11). WannaCry cyber attack cost the NHS £92m as 19,000 appointments cancelled. Retrieved from telegraph.co.uk: https://www.telegraph.co.uk/technology/2018/10/11/wannacry-cyber-attack-cost-nhs-92m-19000-appointments-cancelled.

McLaughlin, C., & Lu, Y. (2024). Multi-class vulnerability prediction using value flow and graph neural networks. Neural Computing and Applications, 36(25), 15869--15891. <https://doi.org/10.1007/s00521-024-09819-3>.

McNally, T. (2021, December 10). HSE hackers were in health service's computer system for eight weeks before cyber attack. TheJournal.ie.

Mead, N. R., Sankalp, A., Anurag, G., Swati, P., Wen, Y., & El Baroni, W. (2016, June 7). Report Writer and Security Requirements Finder: User and Admin Manuals (CMU/SEI-2016-SR-002). Software Engineering Institute, Carnegie Mellon University. https://www.sei.cmu.edu/library/report-writer-and-security-requirements-finder-user-and-admin-manuals/

Mechri, A., Ferrag, M. A., & Debbah, M. (2025). SecureQwen: Leveraging LLMs for vulnerability detection in Python codebases. Computers & Security, 148, Article 104151. <https://doi.org/10.1016/j.cose.2024.104151>.

Mehrtak, M., Seyedalinaghi, S., Mohsseni Pour, M., Noori, T., Karimi, A., Shamsabadi, A., ... Dadras, O. (2021). Security challenges and solutions using healthcare cloud computing. Journal of Medicine and Life, 14(4), 448--461. https://doi.org/10.25122/jml-2021-0100.

Mei, H., Lin, G., Fang, D., & Zhang, J. (2023). Detecting vulnerabilities in IoT software: New hybrid model and comprehensive data analysis. Journal of Information Security and Applications, 74, Article 103467. <https://doi.org/10.1016/j.jisa.2023.103467>.

Mejía-Granda, C. M., Fernández-Alemán, J. L., Carrillo-de-Gea, J. M., & García-Berná, J. A. (2024). Security vulnerabilities in healthcare: An analysis of medical devices and software. Medical & Biological Engineering & Computing, 62(1), 257--273. <https://doi.org/10.1007/s11517-023-02912-0>.

Metin, B., Wynn, M., Tunalı, A., & Kepir, Y. (2025). Business Logic Vulnerabilities in the Digital Era: A Detection Framework Using Artificial Intelligence. Information, 16(7), 585. <https://doi.org/10.3390/info16070585>

Meyers, B. S., & Meneely, A. (2021). An automated post-mortem analysis of vulnerability relationships using natural language word embeddings. Procedia Computer Science, 184, 953--958. <https://doi.org/10.1016/j.procs.2021.04.018>.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., \... & Wu, H. (2018). Mixed precision training. https://doi.org/10.48550/arXiv.1710.03740

Miller, B. P., Fredriksen, L., & So, B. (1990). An empirical study of the reliability of UNIX utilities. Communications of the ACM, 33(12), 32--44. <https://doi.org/10.1145/96267.96279>.

Miller, D. R. (2012). Identities and relationships: Parallels between metadata and professional relevance. Journal of the Medical Library Association, 100(2), 83--86. https://doi.org/10.3163/1536-5050.100.2.003.

Mim, R. S., Satter, A., Ahammed, T., & Sakib, K. (2024). Automated software vulnerability detection using CodeBERT and convolutional neural network. In Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024) (pp. 156--167). SCITEPRESS. <https://doi.org/10.5220/0012707900003687>.

Mishkin, A. D., Zabinski, J. S., Holt, G., & Appelbaum, P. S. (2023). Ensuring privacy in telemedicine: Ethical and clinical challenges. Journal of Telemedicine and Telecare, 29(3), 217-221. https://doi.org/10.1177/1357633X221134952.

Mishra, A., & Bagade, P. (2023). MalDicom: A memory forensic framework for detecting malicious payload in DICOM files (arXiv preprint arXiv:2312.00483). https://doi.org/10.48550/arXiv.2312.00483

MITRE. (2023). Common weakness enumeration: A community developed list of software and hardware weakness types. https://cwe.mitre.org/.

MITRE Corporation. (2024). 2024 CWE Top 25 most dangerous software weaknesses. <https://cwe.mitre.org/top25/>.

MITRE Corporation. . CWE -- Common Weakness Enumeration. Retrieved from https://cwe.mitre.org/.

Mohammed, K. I., Shanmugam, B., & El-Den, J. (2025). Evolution of DevSecOps and its influence on application security: A systematic literature review. Technologies, 13(12), 548. https://doi.org/10.3390/technologies13120548.

Mohammed, S., Budach, L., Feuerpfeil, M., Ihde, N., Nathansen, A., Noack, N., Patzlaff, H., & Naumann, F. (2025). The effects of data quality on machine learning performance on tabular data. Information Systems, 132, Article 102549. <https://doi.org/10.1016/j.is.2025.102549>

Mohino, J., Higuera, J., Higuera, J., & Montalvo, J. (2019). The Application of a New Secure Software Development Life Cycle (S-SDLC) with Agile Methodologies. Electronics, 8, 1218. doi:10.3390/electronics8111218.

Mollaeefar, M., Bissoli, A., Van Landuyt, D., & Ranise, S. (2025). PILLAR: LINDDUN privacy threat modeling using LLMs. In 2025 IEEE 11th International Workshop on Privacy Engineering (IWPE) (pp. 244--253). IEEE. <https://doi.org/10.1109/IWPE62511.2025.00033>.

Montasari, R. (2023). Cyber Threats and the Security Risks They Pose to National Security: An Assessment of Cybersecurity Policy in the United Kingdom. In Countering Cyberterrorism (Advances in Information Security, Vol. 101, pp. 7--25). Springer. https://doi.org/10.1007/978%E2%80%913%E2%80%91031%E2%80%9121920%E2%80%917_2.

Morales, P., Tosello, A., & O'Toole, K. (2020). DevSecOps practices for an agile and secure IT service management. Journal of Management Information and Decision Sciences, 23(2), 1--16.

Moulahi, W., Jdey, I., Moulahi, T., Alawida, M., & Alabdulatif, A. (2023). A blockchain-based federated learning mechanism for privacy preservation of healthcare IoT data.

Computers in Biology and Medicine, 167, Article 107630. <https://doi.org/10.1016/j.compbiomed.2023.107630>.

Muneer, S., Farooq, U., Ahmad, A., Khan, K., & Arshad, J. (2024). A critical review of artificial intelligence-based approaches in intrusion detection: A comprehensive analysis. Security and Communication Networks, 2024, Article 3909173. <https://doi.org/10.1155/2024/3909173>.

Myakala, P. K., Bura, C., & Jonnalagadda, A. K. (2024). Federated Learning and Data Privacy: A review of challenges and opportunities. International Journal of Research Publication and Reviews, 5(12), 1867--1879. <https://doi.org/10.55248/gengpi.5.1224.3512>

Myrbakken, H., & Colomo-Palacios, R. (2017). DevSecOps: A multivocal literature review. In Proceedings of the International Conference on Software Process Improvement and Capability Determination (SPICE 2017) (Communications in Computer and Information Science, Vol. 752, pp. 3--15). Springer. <https://doi.org/10.1007/978-3-319-67383-7_2>.

Nadim, M., & Roy, B. (2023). Utilizing source code syntax patterns to detect bug inducing commits using machine learning models. Software Quality Journal, 31(3), 775. https://doi.org/10.1007/s11219-022-09611-3.

Nagaraja, N., & Bahşi, H. (2025). Cyber threat modeling of an LLM-based healthcare system. In Proceedings of the 11th International Conference on Information Systems Security and Privacy (ICISSP) (Vol. 1, pp. 325-336). SCITEPRESS. https://doi.org/10.5220/0013289700003899.

Nalluri, S., & Kaur, K. (2024). DevSecOps: A thematic literature review. International Journal of Management, IT & Engineering, 14(07).

National Data Guardian (NDG). (2017). Review of Data Security, Consent and Opt-Outs (including the 10 Data Security Standards). UK Department of Health. https://www.gov.uk/government/publications/review-of-data-security-consent-and-opt-outs

National Audit Office. (2017, October 27). Investigation: WannaCry cyber attack and the NHS (HC 414). London: National Audit Office. https://www.nao.org.uk/reports/investigation-wannacry-cyber-attack-and-the-nhs/

National Cyber Security Centre. (2018, February 14). Russian military 'almost certainly' responsible for destructive 2017 cyber attack. Retrieved from ncsc.gov.uk: https://www.ncsc.gov.uk/news/russian-military-almost-certainly-responsible-destructive-2017-cyber-attack.

National Institute of Standards and Technology. (2025). National vulnerability database statistics. U.S. Department of Commerce. https://nvd.nist.gov/.

Neprash, H. T., McGlave, C. C., & Cross, D. A. (2022). Trends in Ransomware Attacks on US Hospitals, Clinics, and Other Health Care Delivery Organizations, 2016-2021. JAMA Health Forum, 4873.

Ng, V. (2023). Rust vs C++, a Battle of Speed and Efficiency (TechRxiv Preprint). TechRxiv. <https://doi.org/10.36227/techrxiv.22792553.v1>.

Ngoupayou Limbepe, Z., Gai, K., & Yu, J. (2025). Blockchain-based privacy-enhancing federated learning in smart healthcare: A survey. Blockchains, 3(1), 1. <https://doi.org/10.3390/blockchains3010001>.

Nguyen Thanh Cong, Le Huy Toan, & Ta Minh Thanh. (2024). An overview of static and dynamic analysis in application security testing. Journal of Military Science and Technology, 99(99), 1--11. <https://doi.org/10.54939/1859-1043.j.mst.99.2024.1-11>.

NHS Digital. (2023, July 17). A clear and present danger. Retrieved from digital.nhs.uk: https://digital.nhs.uk/features/a-real-and-present-danger.

NHS Digital. (2023, September 28). Data Security and Protection Toolkit assessment guides \[Guidance\]. NHS Digital. https://digital.nhs.uk/cyber-and-data-security/guidance-and-resources/data-security-and-protection-toolkit-assessment-guides

NHS Digital. (2024, February 13). Your suppliers and contracts (10.1.1--10.3.1). In Guide 10 -- Accountable suppliers, Data Security and Protection Toolkit assessment guides. NHS

Digital. https://digital.nhs.uk/cyber-and-data-security/guidance-and-resources/data-security-and-protection-toolkit-assessment-guides (Guide 10)

Ni, C., Shen, L., Yang, X., Zhu, Y., & Wang, S. (2024). MegaVul: A C/C++ vulnerability dataset with comprehensive code representations. Proceedings of the 21st International Conference on Mining Software Repositories (MSR '24), Data & Tool Showcase Track, 10.1145/3643991.3644886.

Niazi, M., Saeed, A. M., Alshayeb, M., Mahmood, S., & Zafar, S. (2020). A maturity model for secure requirements engineering. Computers & Security, 95, Article 101852. \https://doi.org/10.1016/j.cose.2020.101852.

Niculescu-Mizil, A., & Caruana, R. (2005). Predicting good probabilities with supervised learning. In Proceedings of the 22nd International Conference on Machine Learning (ICML '05) (pp. 625--632). ACM. https://doi.org/10.1145/1102351.1102430.

Nina, H., Pow-Sang, J. A., Villavicencio, M. (2021). Systematic Mapping of the Literature on Secure Software Development. IEEE Access, 9, 36852-36867. doi:10.1109/ACCESS.2021.3062388.

Núñez, J. C. S., Lindo, A. C., Rodríguez, P. G., et al. (2020). A preventive secure software development model for a software factory: A case study. IEEE Access, 8, 77653--77665. <https://doi.org/10.1109/ACCESS.2020.2989113>.

Nunes, P. J. C., Fonseca, J., & Vieira, M. (2025). Blending static and dynamic analysis for web application vulnerability detection: Methodology and case study. IEEE Access, 11, 1--1. <https://doi.org/10.1109/ACCESS.2024.3522094>.

Nweke, L. O., Abomhara, M., Yıldırım Yayilgan, S., Comparin, D., Heurtier, O., & Bunney, C. (2022). A LINDDUN-based privacy threat modelling for national identification systems. In Proceedings of the 2022 IEEE Nigeria 4th International Conference on Disruptive Technologies for Sustainable Development (NIGERCON) (pp. 1-6). IEEE. https://doi.org/10.1109/NIGERCON54645.2022.9803177.

Obiokafor, I. N., Ajonuma, M. E., & Aguboshim, F. C. (2025). Integrating Privacy by Design (PbD) in the system development life cycle for enhanced data protection. World Journal of Advanced Research and Reviews, 26(1), Article 0538.

Odera, D., Otieno, M., & Ounza, J. E. (2023). Security risks in the software development lifecycle: A review. World Journal of Advanced Engineering Technology and Sciences, 8(2), 230--253. <https://doi.org/10.30574/wjaets.2023.8.2.0101>.

Odermatt, M., Marcilio, D., & Furia, C. A. (2022). Static analysis warnings and automatic fixing: A replication for C# projects. In 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 805--816). IEEE. <https://doi.org/10.1109/SANER53432.2022.00098>.

Oh, S. R., Seo, Y. D., Lee, E., & Kim, Y. G. (2021). A comprehensive survey on security and privacy for electronic health data. International Journal of Environmental Research and Public Health, 18(18), Article 9668. <https://doi.org/10.3390/ijerph18189668>.

Olorunshola, O. E., & Ogwueleka, F. N. (2021). Review of System Development Life Cycle (SDLC) Models for Effective Application Delivery. Lecture Notes in Networks and Systems, 191. doi:https://doi.org/10.1007/978-981-16-0739-4_28.

OpenSSF/Linux Foundation. (2024). Secure Software Development Education 2024 Survey Report. (Blog summary: "Why are organizations struggling to implement secure development?"). https://www.linuxfoundation.org/hubfs/LF%20Research/Secure_Software_Development_Education_2024_Survey.pdf

Orman, H. (2003). The Morris Worm: A Fifteen-Year Perspective. IEEE Security & Privacy, 1(5), 35--43. <https://doi.org/10.1109/MSECP.2003.1236233>.

OWASP Foundation. (2021). OWASP Application Security Verification Standard (ASVS) Version 4.0.3. OWASP Foundation. Retrieved July 8, 2025, from <https://owasp.org/www-project-application-security-verification-standard/>.

Özkan, C., & Singelée, D. (2024). Evidence-based threat modeling for ICS. arXiv. https://doi.org/10.48550/arXiv.2411.19759.

Pan, Y. (2019). Interactive application security testing. In 2019 International Conference on Smart Grid and Electrical Automation (ICSGEA) (pp. 558--561). IEEE. <https://doi.org/10.1109/ICSGEA.2019.00131>.

Papageorgiou, A., (2018). Security and Privacy Analysis of Mobile Health Applications: The Alarming State of Practice. IEEE Access, 6, 9390-9403. doi:10.1109/access.2018.2799522.

Pargaonkar, S. (2023). Advancements in security testing: A comprehensive review of methodologies and emerging trends in software quality engineering. International Journal of Science and Research, 12(9), 61--66.

Park, J., Shin, J., & Choi, B. (2023). Detection of vulnerabilities by incorrect use of variable using machine learning. Electronics, 12(5), 1197. https://doi.org/10.3390/electronics12051197.

Pati, S., Kumar, S., Varma, A., Edwards, B., Lu, C., Qu, L., Wang, J. J., Lakshminarayanan, A., Wang, S. H., Sheller, M. J., Chang, K., Singh, P., Rubin, D. L., Kalpathy-Cramer, J., & Bakas, S. (2024). Privacy preservation for federated learning in health care. Patterns, 5(7), 100974. <https://doi.org/10.1016/j.patter.2024.100974>.

Pei, H., Zhao, J., Lausen, L., Zha, S., & Karypis, G. (2023). Better context makes better code language models: A case study on function call argument completion. arXiv, arXiv:2306.00381.

Peiris, S., & De Silva, N. . (2021). A fuzzy-logic model for benchmarking concrete roof maintenance. Facilities, 39(7/8), 568-583. doi:10.1108/F-03-2020-0026.

Peng, X., Wang, S., Qin, Y., Lin, B., Chen, L., & Mao, X. (2024). Keep it simple: Towards accurate vulnerability detection for large code graphs. arXiv preprint. arXiv:2412.10164. https://doi.org/10.48550/arXiv.2412.10164.

Pereira, J. D'A., Ivaki, N., & Vieira, M. (2021). Characterizing buffer overflow vulnerabilities in large C/C++ projects. IEEE Access, 9, 142879-142892. https://doi.org/10.1109/ACCESS.2021.3120349.

Pham, T. T., Loo, T. M., Malhotra, A., Longhurst, C. A., Hylton, D., Dameff, C., Tully, J., Wardi, G., Sell, R. E., & Pearce, A. K. (2024). Ransomware cyberattack associated with cardiac arrest incidence and outcomes at untargeted, adjacent hospitals. Critical Care Explorations, 6(4), e1079. https://doi.org/10.1097/CCE.0000000000001079.

Pollock, L., Vijay-Shanker, K., Shepherd, D., Hill, E., Fry, Z. P., & Maloor, K. (2007, June). Introducing Natural Language Program Analysis. In Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE'07). ACM. <https://doi.org/10.1145/1251535.1251538>.

Prasojo, A. R., Gumilang, H., Maulidevi, N. U., Soedjarno, B. A., & Suwarno. (2020). A Fuzzy Logic Model for Power Transformer Faults' Severity Determination Based on Gas Level, Gas Rate, and Dissolved Gas Analysis Interpretation. Artificial Intelligence Applications to Energy Systems, 13(4). doi:Artificial Intelligence Applications to Energy Systems.

Prates, L., & Pereira, R. (2025). DevSecOps practices and tools. International Journal of Information Security, 24(1), Article 11. <https://doi.org/10.1007/s10207-024-00914-z>.

Priya, S. S., & Arya, S. S. (2016). Threat modeling for a secured software development. International Journal of Advanced Research in Computer Science, 7(1), 40--48. <https://doi.org/10.26483/ijarcs.v7i1.2615>.

Qiu, Y., Liu, Y., Liu, A., Zhu, J., & Xu, J. (2019). Automatic feature exploration and an application in defect prediction. IEEE Access, 7, 112097--112112. https://doi.org/10.1109/access.2019.2934530.

Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. (2021). Challenges and solutions when adopting DevSecOps: A systematic review. Information and Software Technology, 141, Article 106700. <https://doi.org/10.48550/arXiv.2103.08266>.

Rasool, R. U., Ahmad, H. F., Rafique, W., Qayyum, A., & Qadir, J. (2022). Security and privacy of Internet of Medical Things: A contemporary review in the age of surveillance, botnets, and adversarial ML. Journal of Network and Computer Applications, 201, 103332. <https://doi.org/10.1016/j.jnca.2022.103332>.

Rassokhin, D. (2020). The C++ programming language in cheminformatics and computational chemistry. Journal of Cheminformatics, 12(1), Article 10. <https://doi.org/10.1186/s13321-020-0415-y>.

Redmiles, E. M., Acar, Y., Fahl, S., & Mazurek, M. L. (2017a). A summary of survey methodology best practices for security and privacy researchers (Tech. Rep. CS-TR-5055). University of Maryland. <https://doi.org/10.13016/M22K2W> (https://drum.lib.umd.edu/items/683d78b0-a0e3-4fae-9c93-b75aae4ad11b)

Rempe, M., Heine, L., Seibold, C., Hörst, F., & Kleesiek, J. (2025, June 7). De-identification of medical imaging data: a comprehensive tool for ensuring patient privacy. European Radiology. Advance online publication. <https://doi.org/10.1007/s00330-025-11695-x>.

Resende, P. A. A., & Drummond, A. C. (2018). A survey of random forest based methods for intrusion detection systems. ACM Computing Surveys, 51(3), Article 48. <https://doi.org/10.1145/3178582>.

Risse, N., Liu, J., & Böhme, M. (2025). Top Score on the Wrong Exam: On Benchmarking in Machine Learning for Vulnerability Detection. In Proceedings of the ACM on Software Engineering. <https://doi.org/10.1145/3728887>.

Ritunga, I., Harsa, C., Santoso, J., Mellyanawati, N., Nurhadi, S., Nugraheni, E., Perwi, I., & Alka'ari, N. S. (2025, March). Privacy and confidentiality in telemedicine: A literature review. Christian Journal for Global Health, 12(1), 74--82. <https://doi.org/10.15566/cjgh.v12i1.377>.

Rocher, L., Hendrickx, J. M., & de Montjoye, Y. A. (2019). Estimating the success of re-identifications in incomplete datasets using generative models. Nature Communications, 10(1), 3069. https://doi.org/10.1038/s41467-019-10933-3.

Ross, R., Pillitteri, V., Graubart, R., Bodeau, R., & Mcquaid, R. (2021). Developing Cyber-Resilient Systems: A Systems Security Engineering Approach. NIST Special Publication 800-160.

Roumani, Y. (2021). Patching zero-day vulnerabilities: an empirical analysis. Journal of Cybersecurity, 7(1). doi:10.1093/cybsec/tyab023.

Royce, W. W. (1970). Managing the development of large software systems: Concepts and techniques. In Proceedings of IEEE WESCON (pp. 1--9). IEEE. <https://doi.org/10.5555/41765.41801>.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Canton Ferrer, C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., & Synnaeve, G. (2023). Code Llama: Open foundation models for code. arXiv preprint, arXiv:2308.12950. https://doi.org/10.48550/arXiv.2308.12950

Ruohonen, J. (2025). SoK: The design paradigm of safe and secure defaults. Journal of Information Security and Applications, 90, Article 103989. https://doi.org/10.1016/j.jisa.2025.103989.

Russell, R. L., Kim, L., Hamilton, L. H., Lazovich, T., Harer, J. A., Ozdemir, O., Ellingwood, P. M., & McConley, M. W. (2018). Automated vulnerability detection in source code using deep representation learning. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 757--762). IEEE. <https://doi.org/10.1109/ICMLA.2018.00120>.

Sánchez-Gordón, M., & Colomo-Palacios, R. (2020). Security as culture: A systematic literature review of DevSecOps. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (pp. 266--269). Association for Computing Machinery. <https://doi.org/10.1145/3387940.3392233>.

Saeed, H., Shafi, I., Ahmad, J., Khan, A. A., Khurshaid, T., & Ashraf, I. (2025). Review of techniques for integrating security in software development lifecycle. Computers, Materials & Continua, 82(1), 139-172. <https://doi.org/10.32604/cmc.2024.057587>.

Sahay, R., Estay, S., Meng, W., Jensen, C., & Barford, B. M. (2023). A comparative risk analysis on CyberShip system with STPA-Sec, STRIDE and CORAS. Computers & Security, 128, 103179. doi:10.1016/j.cose.2023.103179.

Saimbhi, A. S. (2025). Enhancing software vulnerability detection using code property graphs and convolutional neural networks \[Preprint\]. arXiv. <https://doi.org/10.48550/arXiv.2503.18175>.

Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLoS ONE, 10(3), e0118432. https://doi.org/10.1371/journal.pone.0118432.

Sangaroonsilp, P., Dam, H. K., & Ghose, A. (2023). On privacy weaknesses and vulnerabilities in software systems. Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE), 491--501. https://doi.org/10.1109/ICSE48619.2023.00097.

Santa Molison, A., Moraes, M., Melo, G., Santos, F., & Assunção, W. K. G. (2025, August 1). Is LLM-Generated Code More Maintainable & Reliable than Human-Written Code? (Tech. Rep. arXiv:2508.00700). arXiv.org.

Santos, R., Rizvi, S., Cesarone, B., Gunn, W., & McConnell, E. (2021). Reducing Software Vulnerabilities Using Machine Learning Static Application Security Testing. In Proceedings of the 2021 International Conference on Software Security and Assurance (ICSSA 2021) (pp. 43--46). IEEE. <https://doi.org/10.1109/ICSSA53632.2021.00016>

Sarker, I. H., Janicke, H., Mohsin, A., Gill, A., & Maglaras, L. (2024). Explainable AI for cybersecurity automation, intelligence and trustworthiness in digital twin: Methods, taxonomy, challenges and prospects. ICT Express, 10(4), 935--958. <https://doi.org/10.1016/j.icte.2024.05.007>

Saurabh, K., Gajjala, D., Kaipa, K., Vyas, R., Vyas, O. P., & Khondoker, R. (2024). TMAP: A threat modeling and attack path analysis framework for industrial IoT systems (A case study of IoM and IoP). Arabian Journal for Science and Engineering, 49(9), 13163--13183. https://doi.org/10.1007/s13369-023-08600-3.

Savanović, N., Toskovic, A., Petrovic, A., Zivkovic, M., Damaševičius, R., Jovanovic, L., Bacanin, N., & Nikolic, B. (2023). Intrusion detection in Healthcare 4.0 Internet of Things systems via metaheuristics optimized machine learning. Sustainability, 15(16), 12563. https://doi.org/10.3390/su151612563.

Sayoud, H. (2024). Deep Learning vs. Conventional Approaches in Stylometry: An Authorship Attribution Study. In Proceedings of the International Conference on Information Security and Smart Computing (ICISS 2024). ACM. <https://doi.org/10.1145/3700706.3700732>.

Scandariato, R., Wuyts, K., & Joosen, W. (2015). A descriptive study of Microsoft's threat modeling technique. Requirements Engineering, 20, 163--180.

Schneider, J. (2024). What comes after transformers? -- A selective survey connecting ideas in deep learning (arXiv:2408.00386). <https://doi.org/10.48550/arXiv.2408.00386>.

Schneider, S., Díaz Ferreyra, N. E., Quéval, P.-J., Simhandl, G., Zdun, U., & Scandariato, R. (2024). How dataflow diagrams impact software security analysis: An empirical experiment. In Proceedings of the 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE. https://doi.org/10.1109/SANER60148.2024.00103.

Seh, A. H., Zarour, M., Alenezi, M. , Sarkar, A. K., Agrawal, A., Kumar, R., & Khan, R. A. (2020). Healthcare Data Breaches: Insights and Implications. PubMed Central, 8(2), 133. doi:10.3390/healthcare8020133.

Seth, A., Bhattacharya, S., Elder, S., Zahan, N., & Williams, L. A. (2025). Comparing effectiveness and efficiency of Interactive Application Security Testing (IAST) and Runtime Application Self-Protection (RASP) tools in a large Java-based system. Empirical Software Engineering, 30(3), 67. <https://doi.org/10.1007/s10664-025-10621-5>.

Shahriar, H., Haddad, H. M., & Vaidya, I. (2013). Buffer overflow patching for C and C++ programs: rule-based approach. ACM SIGAPP Applied Computing Review, 13(2), 8-19. https://doi.org/10.1145/2505420.2505421.

Shao, M., Ding, Y., Cao, J., & Li, Y. (2025). GraphFVD: Property graph-based fine-grained vulnerability detection. Computers & Security, 151, Article 104350. <https://doi.org/10.1016/j.cose.2025.104350>.

Shaon, M. S. H., & Akter, M. S. (2025). Modern approaches to software vulnerability detection: A survey of machine learning, deep learning, and large language models. Electronics, 14(22), 4449. https://doi.org/10.3390/electronics14224449.

Shastry, B., Leutner, M., Fiebig, T., Thimmaraju, K., Yamaguchi, F., Rieck, K., Schmid, S., Seifert, J.-P., & Feldmann, A. (2017). Static program analysis as a fuzzing aid. In M. Dacier, M. Bailey, M. Polychronakis, & M. Antonakakis (Eds.), Research in Attacks, Intrusions, and Defenses (RAID 2017; Lecture Notes in Computer Science, Vol. 10453, pp. 26--47). Springer. <https://doi.org/10.1007/978-3-319-66332-6_2>.

Shaver, J. (2022). The state of telehealth before and after the COVID-19 pandemic. Primary Care: Clinics in Office Practice, 49(4), 517--530. <https://doi.org/10.1016/j.pop.2022.04.002>.

Shen, Z., & Chen, S. (2020). A survey of automatic software vulnerability detection, program repair, and defect prediction techniques. Security and Communication Networks, 2020, Article 8858010, 1--16. <https://doi.org/10.1155/2020/8858010>.

Shendkar, B. D., Dhotre, D., Hirve, S. A., Sontakke, P., Hingoliwala, H., & Sambare, G. B. (2024, November 13). Explainable machine learning models for real-time threat detection in cybersecurity. Panamerican Mathematical Journal, 35(1 s). <https://doi.org/10.52783/pmj.v35.i1s.2313>.

Sheng, Z., Chen, Z., Gu, S., Huang, H., Gu, G., & Huang, J. (2024). Large language models in software security: A survey of vulnerability detection techniques and insights. Computers & Security, 130, 103077. <https://doi.org/10.1016/j.cose.2025.103077>.

Sherin, S., Muqeet, A., Khan, M. U., & Iqbal, M. Z. (2023). QExplore: An exploration strategy for dynamic web applications using guided search. Journal of Systems and Software, 195, Article 111512. <https://doi.org/10.1016/j.jss.2022.111512>.

Shi, Z., Matyunin, N., Graffi, K., & Starobinski, D. (2024). Uncovering CWE-CVE-CPE relations with threat knowledge graphs. ACM Transactions on Privacy and Security, 27 (1), Article 1, 1--7. <https://doi.org/10.1145/3641819>.

Shimmi, S., Okhravi, H., & Rahimi, M. (2025). AI-based software vulnerability detection: A systematic literature review \[Preprint\]. Computing Surveys. <https://doi.org/10.48550/arXiv.2506.10280>.

Shiri Harzevili, N., Boaye Belle, A., Wang, J., Wang, S., Jiang, Z. M., & Nagappan, N. (2024). A systematic literature review on automated software vulnerability detection using machine learning. ACM Computing Surveys, 57(3), 66. https://doi.org/10.1145/3571202.

Shokry, M., Awad, I. A., Abd-Ellah, K. M., & Khalaf, A. . (2022). CORAS Model for Security Risk Assessment in Advanced Metering Infrastructure Systems. 8th International Conference on Advanced Intelligent Systems and Informatics 2022, (pp. 449-459). doi:10.1007/978-3-031-20601-6_39.

Shoufan, A., & Damiani, E. (2017). On inter-rater reliability of information security experts. Journal of Information Security and Applications, 37, 101-111. https://doi.org/10.1016/j.jisa.2017.10.006.

Shukla, M. (2024). Security of connected places. In Proceedings of the Ninth International Congress on Information and Communication Technology (ICICT 2024) (pp. 195--211). Springer Nature Singapore. <https://doi.org/10.1007/978-981-97-5035-1_15>.

Si-Ahmed, A., Marzak, B., & Sael, N. (2024). Explainable machine learning-based security and privacy protection framework for internet of medical things systems. arXiv preprint. <https://doi.org/10.48550/arXiv.2403.09752>.

Sinan, M., Shahin, M., & Gondal, I. (2025). Integrating security controls in DevSecOps: Challenges, solutions, and future research directions. Journal of Software: Evolution and Process, 37(6). <https://doi.org/10.1002/smr.70029>.

Singh, J., Sinha, A., Goli, P., Subramanian, V., Shukla, K. S., & Vyas, O. P. (2022). Insider attack mitigation in a smart metering infrastructure using reputation score and blockchain technology. International Journal of Information Security, 21, 527-546. doi:10.1007/s10207-021-00561-8.

Singh, R., Gupta, M. K., Patil, D. R., & Patil, S. M. (2024). Analysis of web application vulnerabilities using dynamic application security testing. In Proceedings of the 2024 IEEE 9th International Conference for Convergence in Technology (I2CT) (pp. 1--6). IEEE.

Sion, L., Wuyts, K., Yskout, K., Van Landuyt, D., & Joosen, W. (2018). Interaction-based privacy threat elicitation. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) (pp. 79--86). IEEE. <https://doi.org/10.1109/EuroSPW.2018.00017>.

Sion, L., Van Landuyt, D., Wuyts, K., & Joosen, W. (2025). Robust and reusable LINDDUN privacy threat knowledge. Computers & Security, 154, Article 104419. <https://doi.org/10.1016/j.cose.2025.104419>.

Siponen, M., & Baskerville, R. (2001). A new paradigm for adding security into IS development methods. In J. H. P. Eloff, L. Labuschagne, R. von Solms, & G. Dhillon (Eds.), Advances in Information Security Management & Small Systems Security (IFIP International Federation for Information Processing, Vol. 72, pp. 99--111). Springer. <https://doi.org/10.1007/0-306-47007-1_8>.

Software Engineering Institute. (2016, June 29). SEI CERT C Coding Standard: Rules for developing safe, reliable, and secure systems (2016 Edition). Carnegie Mellon University, Software Engineering Institute. https://resources.sei.cmu.edu/downloads/secure-coding/assets/sei-cert-c-coding-standard-2016-v01.pdf

Software Engineering Institute. (2019, December). Community guidance to prevent common coding errors. Software Engineering Institute. https://www.sei.cmu.edu/projects/community-guidance-to-prevent-common-coding-errors/

Solanke, A. A. (2022). Enterprise DevSecOps: Integrating security into CI/CD pipelines for regulated industries. World Journal of Advanced Research and Reviews, 13(02), 633--648. <https://doi.org/10.30574/wjarr.2022.13.2.0121>.

Solomos, K. , Ilia, P., Ioannidis, S., & Kourtellis, N. (2019). Talon: An Automated Framework for Cross-Device Tracking Detection. arXiv.org. doi:10.48550/arxiv.1812.11393.

Sonnekalb, T., Heinze, T. S., & Mäder, P. (2022). Deep security analysis of program code: A systematic literature review. Empirical Software Engineering, 27(2), Article 2. <https://doi.org/10.1007/s10664-021-10029-x>.

Souppaya, M., Scarfone, K., & Dodson, D. (2022). Secure Software Development Framework (SSDF) Version 1.1. National Institute of Standards and Technology (NIST). doi:10.6028/NIST.SP.800-218.

Srinivasan, S., & Rothacker, A. (2022, July 18). Decade retrospective: The state of vulnerabilities. LevelBlue SpiderLabs Blog. (https://levelblue.com/blogs/spiderlabs-blog/decade-retrospective-the-state-of-vulnerabilities)

Sunil, V., & Mathew, S. P. (2024). A systematic review on cybersecurity threats and challenges in hospitals. Acta Medica International, 11(1), 1--6. https://doi.org/10.4103/amit.amit_7_24.

Svandova, K., & Smutny, Z. (2024, May 13). Internet of Medical Things security frameworks for risk assessment and management: A scoping review. Journal of Multidisciplinary Healthcare, 17, 2281--2301. <https://doi.org/10.2147/JMDH.S459987>.

Swathi, S., & Yogish, H. K. (2021). Trust Oriented Efficient Data Aggregation and Malevolent Node Detection in the Internet of Things Sensor Network. Grenze International Journal of Engineering & Technology (GIJET), 7(2), 166-175. [Grenze Intl. Conf. proceedings, pp. 166-175 — confirmed to exist; no DOI available; ISBN 978-1-7138-3527-1]

Tangari, G., Ikram, M., Ijaz, K., Kaafar, M. A., & Berkovsky, S. (2021). Mobile health and privacy: Cross sectional study. BMJ, 373, n1248. https://doi.org/10.1136/bmj.n1248.

Thapa, C., Jang, S. I., Ahmed, M. E., Camtepe, S., Pieprzyk, J., & Nepal, S. (2022). Transformer-based language models for software vulnerability detection. Paper presented at the ACM SIGSOFT International Symposium on Software Testing and Analysis. https://doi.org/10.1145/3564625.3567985.

The MITRE Corporation. (2025). CWE community working groups & special interest groups: ICS/OT SIG---Industrial Control Systems/Operational Technology \[Web page\]. In CWE Community WGs & SIGs. Retrieved September 8, 2025, from https://cwe.mitre.org/community/working_groups.html.

Thomas, T. W., Tabassum, M., Chu, B., & Lipford, H. (2018). Security during application development: An application security expert perspective. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18) (pp. 1--12). ACM. <https://doi.org/10.1145/3173574.3173836>.

Tlili, S., Yang, X., Hadjidj, R., Debbabi, M. (2009). Verification of CERT Secure Coding Rules: Case Studies. In: Meersman, R., Dillon, T., Herrero, P. (eds) On the Move to Meaningful Internet Systems: OTM 2009. OTM 2009. Lecture Notes in Computer Science, vol 5871. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-05151-7_12

Tiwari, H. (2025). Advancing vulnerability classification with BERT: A multi-objective learning model \[Preprint\]. arXiv. <https://doi.org/10.48550/arXiv.2503.20831>.

Troncoso, C., Danezis, G., & Rashid, A. (2021). Privacy & Online Rights Knowledge Area, Version 1.0.2. Cyber Security Body of Knowledge (CyBOK). https://www.cybok.org/media/downloads/Privacy_Online_Rights_v1.0.2.pdf

Tsiazhkorob, U. V., & Ignatyev, V. N. (2024, May). Classification of static analyzer warnings using machine learning methods. In 2024 Ivannikov Memorial Workshop (IVMEM). IEEE. <https://doi.org/10.1109/IVMEM63006.2024.10659704>.

Tsipenyuk, K., Chess, B., & McGraw, G. (2005, November 1). Seven Pernicious Kingdoms: A taxonomy of software security errors. IEEE Security & Privacy, 3(6), 81--84. <https://doi.org/10.1109/MSP.2005.159>.

Van Rest, J., Boonstra, D., Everts, M., Van Rijn, M., & Van Paassen, R. (2014). Designing privacy-by-design. In Proceedings of the Annual Privacy Forum (Lecture Notes in Computer Science, Vol. 8319, pp. 55--72). Springer. <https://doi.org/10.1007/978-3-642-54069-1_4>.

Van Landuyt, D., Joosen, W., Hung, C.-C., Cerný, T., Shin, D., & Bechini, A. (2020). A descriptive study of assumptions made in LINDDUN privacy threat elicitation. In Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC '20) (March 30--April 3, 2020, Brno, Czech Republic), 8 pages. ACM. <https://doi.org/10.1145/3341105.3375762>.

Van Landuyt, D., Sion, L., Philips, W., & Joosen, W. (2024). From automation to CI/CD: A comparative evaluation of threat modeling tools. In Proceedings of the 2024 IEEE Secure Development Conference (SecDev) (pp. 35-45). IEEE.

van Kessel, R., Haig, M., & Mossialos, E. (2023). Strengthening cybersecurity for patient data protection in Europe. Journal of Medical Internet Research, 25, e48824. https://doi.org/10.2196/48824.

Venkataramanan, N., & Shriram, A. (2016). Data privacy: Principles and practice (1st ed.). Chapman and Hall/CRC. <https://doi.org/10.1201/9781315370910>.

Verizon. (2025). 2025 data breach investigations report. <https://www.verizon.com/business/resources/reports/dbir/>.

Von Grafenstein, M., Jakobi, T., & Stevens, G. (2022). Effective data protection by design through interdisciplinary research methods: The example of effective purpose specification by applying user-centred UX-design methods. Computer Law & Security Review, 46, Article 105722. <https://doi.org/10.1016/j.clsr.2022.105722>.

Wadhams, Z. D., Izurieta, C., & Reinhold, A. M. (2024). Barriers to using static application security testing (SAST) tools: A literature review. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops (pp. 161--166). ACM.

Wang, Y., Wang, W., Joty, S., & Hoi, S. C. H. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021) (pp. 8696--8708). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2021.emnlp-main.685>.

Wang, S., Feng, Z., Tarlow, D., Bui, T., Wang, Y., & Neubig, G. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. arXiv. https://doi.org/10.48550/arXiv.2109.00859.

Wang, W., Nguyen, T. N., Wang, S., Li, Y., Zhang, J., & Yadavally, A. (2023). DeepVD: Toward class-separation features for neural network vulnerability detection. IEEE/ACM International Conference on Software Engineering, 2249. https://doi.org/10.1109/icse48619.2023.00189.

Wang, X., Hu, R., Gao, C., Wen, X. C., Chen, Y., & Liao, Q. (2024). ReposVul: A repository level high quality vulnerability dataset. Proceedings of the International Conference on Software Engineering Companion, 472--483.(.

Weir, C., Migues, S., Ware, M., & Williams, L. (2021). Infiltrating Security into Development: Exploring the World's Largest Software Security Study. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21) (pp. 1326--1336). Association for Computing Machinery. <https://doi.org/10.1145/3468264.3473926>.

Weir, C., Migues, S., & Williams, L. (2022). Exploring the shift in security responsibility. IEEE Security & Privacy, 20(6), 8--17.

Wen, S.-F., & Katt, B. (2023). A quantitative security evaluation and analysis model for web applications based on OWASP Application Security Verification Standard. Computers & Security, 135, Article 103532. <https://doi.org/10.1016/j.cose.2023.103532>.

Wen, X.-C., Chen, Y., Gao, C., Zhang, H., Zhang, J. M., & Liao, Q. (2023). Vulnerability detection with graph simplification and enhanced graph representation learning. In Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE 2023). IEEE.

Whittaker, Z. (2025, January 27). How the ransomware attack at Change Healthcare went down: A timeline. TechCrunch. <https://techcrunch.com/2025/01/27/how-the-ransomware-attack-at-change-healthcare-went-down-a-timeline/>.

Wieringa, J., Kannan, P. K., Ma, X., Reutterer, T., Risselada, H., & Skiera, B. (2020). Data analytics in a privacy-concerned world. Journal of Business Research, 116, 270--281. https://doi.org/10.1016/j.jbusres.2019.05.005.

Wiggerthale, J., & Reich, C. (2024). Explainable Machine Learning in Critical Decision Systems: Ensuring Safe Application and Correctness. AI, 5(4), 2864--2896. <https://doi.org/10.3390/ai5040138>.

Williams, P. A. H., & Woodward, A. J. (2015). Cybersecurity vulnerabilities in medical devices: A complex environment and multifaceted problem. Medical Devices: Evidence and Research, 8, 305--316. <https://doi.org/10.2147/MDER.S50048>.

Williams, I., Yuan, X., McDonald, J. T., & Anwar, M. (2016). A method for developing abuse cases and its evaluation. Journal of Software, 11(5), 520--527. https://doi.org/10.17706/jsw.11.5.520-527.

Williamson, S. M., & Prybutok, V. (2024). Balancing privacy and progress: A review of privacy challenges, systemic oversight, and patient perceptions in AI-driven healthcare. Applied Sciences, 14(2), 675. https://doi.org/10.3390/app14020675.

Wu, B., & Zou, F. (2022). Code vulnerability detection based on deep sequence and graph models: A survey. Security and Communication Networks, 2022, Article 1176898, 11 pp. <https://doi.org/10.1155/2022/1176898>.

Wuyts, K., Scandariato, R., & Joosen, W. (2014). Empirical evaluation of a privacy-focused threat modeling methodology. Journal of Systems and Software, 96, 122--138. <https://doi.org/10.1016/j.jss.2014.05.075>.

Wuyts, K., & Joosen, W. (2015). LINDDUN: A privacy threat analysis framework for early software development. In Proceedings of the 2015 IEEE 7th International Workshop on Requirements Engineering and Law (RELAW) (pp. 26--31). IEEE. <https://doi.org/10.1109/RELAW.2015.7330113>.

Wuyts, K., Sion, L., & Joosen, W. (2020). LINDDUN GO: A lightweight approach to privacy threat modeling. In Proceedings of the 2020 IEEE European Symposium on Security

and Privacy Workshops (EuroS&PW) (pp. 302--309). IEEE. <https://doi.org/10.1109/EuroSPW51379.2020.00047>.

Xiao, Y., Zuo, X., Lu, X., Dong, J. S., Cao, X., & Beschastnikh, I. (2025). Promises and perils of using Transformer-based models for SE research. Neural Networks, 184, 107067. <https://doi.org/10.1016/j.neunet.2024.107067>.

Xiong, W., & Lagerström, R. (2019). Threat Modeling---A Systematic Literature Review. Computers & Security, 84, 53--69. DOI: 10.1016/j.cose.2019.03.010.

Xiong, Z., & Dong, W. (2024). VulD-CodeBERT: CodeBERT-based vulnerability detection model for C/C++ code. In 2024 6th International Conference on Communications, Information System and Computer Engineering (CISCE) (pp. 914--919). IEEE. <https://doi.org/10.1109/CISCE62493.2024.10653337>.

Xiong, W., Legrand, E., Aberg, O., & Lagerstrom, R. (2022). Cyber security threat modeling based on the MITRE Enterprise ATT&CK Matrix. Software and Systems Modeling, 21, 157-177. doi:10.1007/s10270-021-00898-7.

Xu, Y., & Zhu, Y. (2022). A survey on pretrained language models for neural code intelligence. arXiv, arXiv:2212.10079. https://doi.org/10.48550/arXiv.2212.10079

Yadav, S. (2024). Transformative Frontiers: A Comprehensive Review of Emerging Technologies in Modern Healthcare. Cureus, 16(3), e56538. DOI:10.7759/cureus.56538.

Yamaguchi, F., Koo, J., Fujiwara, K., & Sato, T. (2014). Modeling and discovering vulnerabilities with code property graphs. In Proceedings of the 2014 IEEE Symposium on Security and Privacy (pp. 428--442). IEEE.

Yang, J., Tan, L., Peyton, J., & Duer, K. A. (2019). Towards better utilizing static application security testing. In 41st IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19) (pp. 51--60). IEEE. https://doi.org/10.1109/ICSE%E2%80%91SEIP.2019.00014.

Yang, Y., Xu, B., Gao, X., & Sun, H. (2024). Context-Enhanced Vulnerability Detection Based on Large Language Model. arXiv preprint, arXiv:2504.16877. (https://arxiv.org/abs/2504.16877)

Yang, S., Gao, Y., Kuang, B., Yang, Y., & Fu, A. (2025). DFirmSan: A lightweight dynamic memory sanitizer for Linux-based firmware. Computers and Security, 155, Article 104467. Advance online publication. <https://doi.org/10.1016/j.cose.2025.104467>

Yankson, B., Barati, M., Bondzie, R., & Madani, R. (2025). The Rise of Hacking in Integrated EHR Systems: A Trend Analysis of U.S. Healthcare Data Breaches. Journal of Cybersecurity and Privacy, 5(3), 70. https://doi.org/10.3390/jcp5030070.

Yeo, L. H., & Banfield, J. (2022, March 15). Human factors in electronic health records cybersecurity breach: An exploratory analysis. Perspectives in Health Information Management, 19(Spring), 1i. <https://doi.org/10.13030/phim.v19i1.1i>.

Ying, R., He, J., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 974-983). ACM.

Zagane, M., Abdi, M. K., & Alenezi, M. (2020). Deep learning for software vulnerabilities detection using code metrics. IEEE Access, 8, 74562--74570. https://doi.org/10.1109/access.2020.2988557.

Zhang, X., & Saltman, R. (2022). Impact of electronic health record interoperability on telehealth service outcomes. JMIR Medical Informatics, 10(1), e31837. https://doi.org/10.2196/31837.

Zhang, C., Yu, T., & Liu, B. (2024). Vulnerability detection based on federated learning. Information and Software Technology, 167, Article 107371. <https://doi.org/10.1016/j.infsof.2023.107371>.

Zhao, K., Xu, Z., Yan, M. et al. The impact of class imbalance techniques on crashing fault residence prediction models. Empir Software Eng 28, 49 (2023). https://doi.org/10.1007/s10664-023-10294-y.

Zhao, X., Clear, T., & Lal, R. (2024). Identifying the primary dimensions of DevSecOps: A multi-vocal literature review. Journal of Systems and Software, Article 112063. <https://doi.org/10.1016/j.jss.2024.112063>.

Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. A. Fox, & R. Garnett (Eds.), Advances in Neural Information Processing Systems (NeurIPS 2019), 32, 10197--10207. https://doi.org/10.48550/arXiv.1909.03496 [arXiv preprint version]

Zhou, X., Cao, S., Sun, X., & Lo, D. (2024). Large Language Model for Vulnerability Detection and Repair: Literature Review and the Road Ahead. ACM Transactions on Software Engineering and Methodology, 34(5). <https://doi.org/10.1145/3708522>.

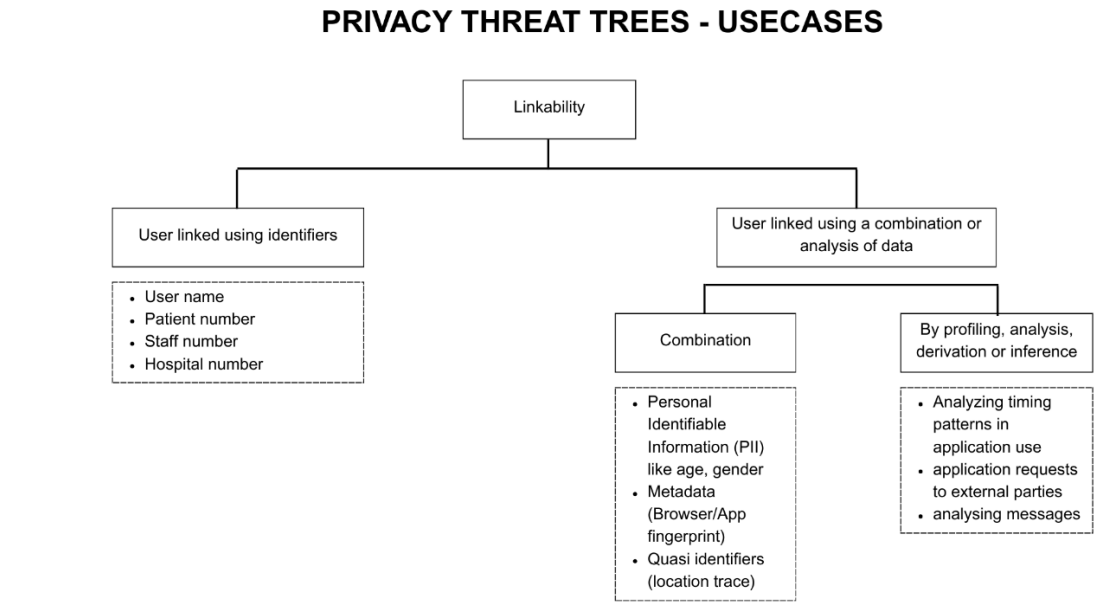
Zhou, X., Zhang, T., & Lo, D. (2024). Large language model for vulnerability detection: Emerging results and future directions. In Proceedings of the 46th International Conference on Software Engineering (ICSE), New Ideas and Emerging Results (NIER) Track (pp. 47--51). https://doi.org/10.1145/3589161.3590784.

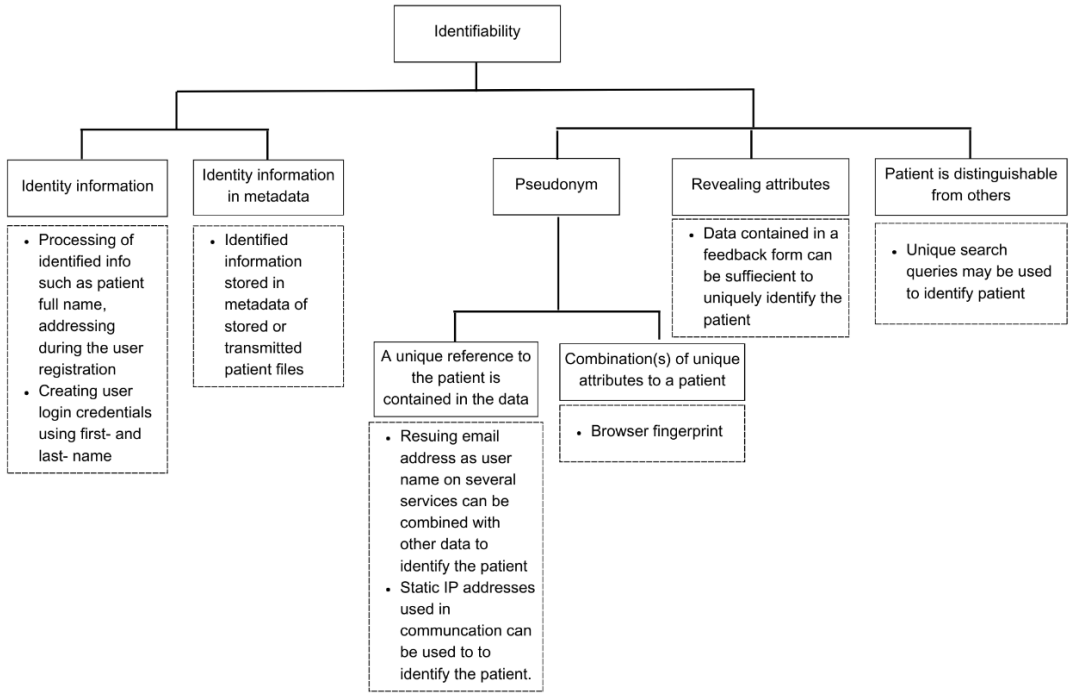
Ziems, N., & Wu, S. (2021). Security vulnerability detection using deep learning and NLP. arXiv preprint arXiv:2105.02388. https://arxiv.org/abs/2105.02388.

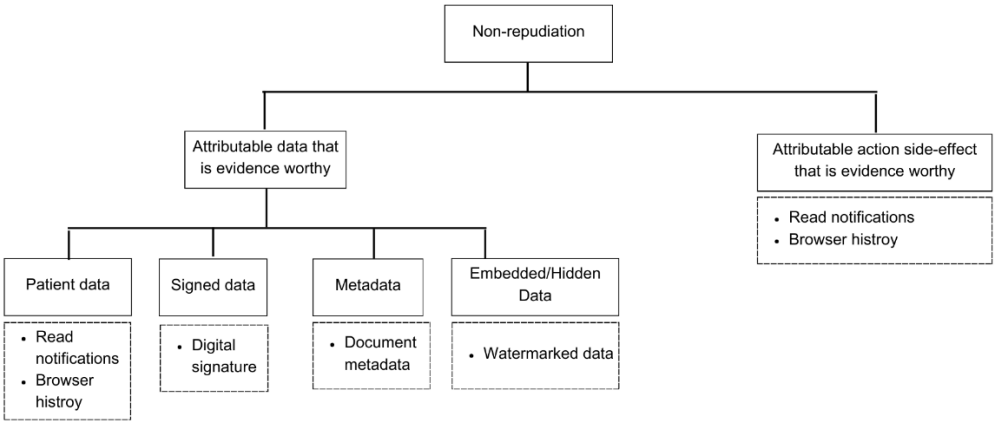
Zouev, E. (2019). Programming languages for safety-critical systems. In Software Design for Resilient Computer Systems (pp. 165--175). Springer. https://doi.org/10.1007/978-3-030-21244-5_11.

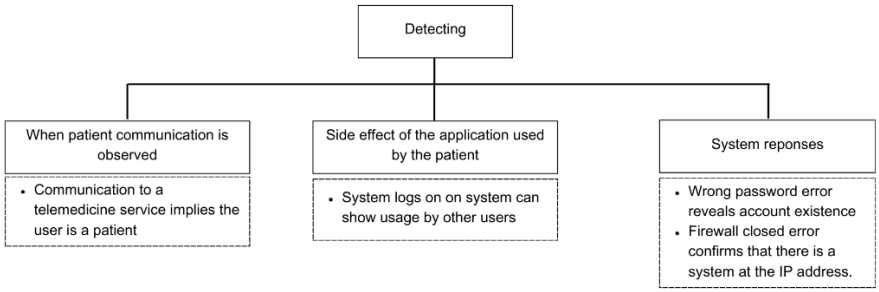
8 Appendices

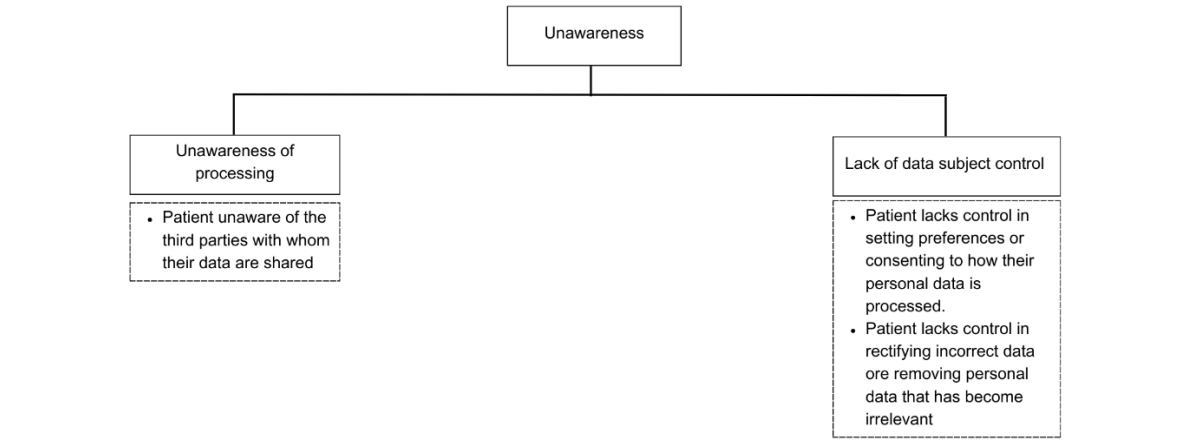
8.1 Appendix A: Threat Trees

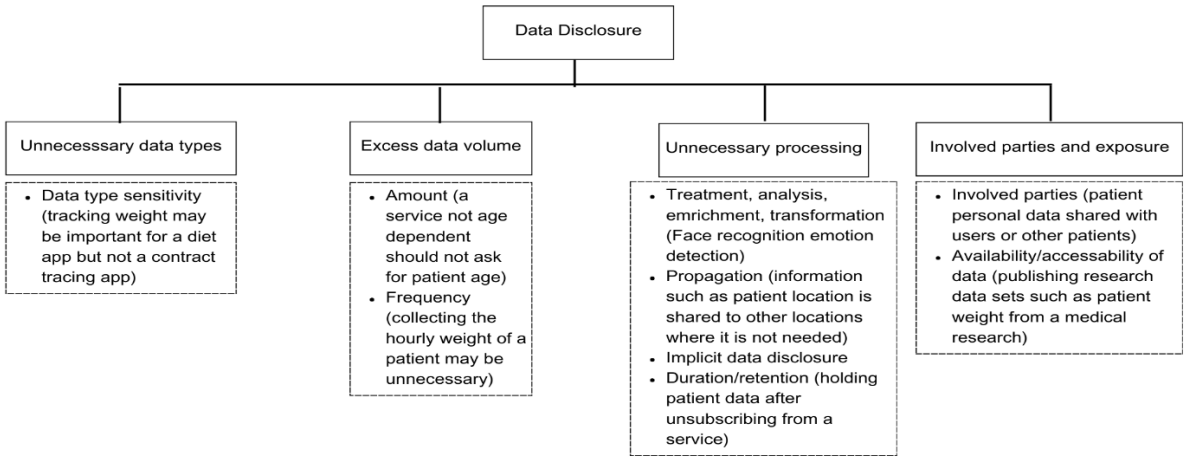


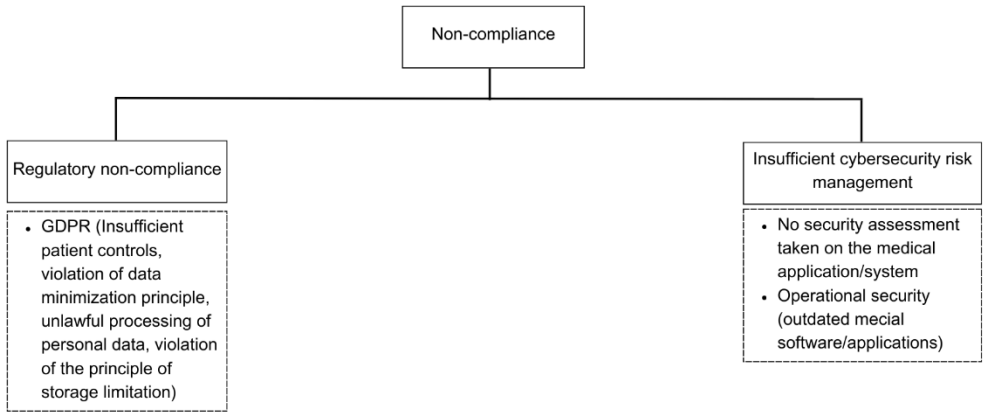



Non-repudiation
Attributable data that is evidence worthy
Attributable action side-effect that is evidence worthy
Read notifications
Browser histroy
Patient data
Signed data
Metadata
Embedded/Hidden Data
Read notifications
Browser histroy
Digital signature
Document metadata
Watermarked data


Detecting
When patient communication is observed
Side effect of the application used by the patient
System reponses
Communication to a telemedicine service implies the user is a patient
System logs on on system can show usage by other users
Wrong password error reveals account existence
Firewall closed error confirms that there is a system at the IP address.


Unawareness
Unawareness of processing
Patient unaware of the third parties with whom their data are shared
Lack of data subject control
Patient lacks control in setting preferences or consenting to how their personal data is processed.
Patient lacks control in rectifying incorrect data ore removing personal data that has become irrelevant


Data Disclosure
Unnecesssary data types
Data type sensitivity (tracking weight may be important for a diet app but not a contract tracing app)
Excess data volume
Amount (a service not age dependent should not ask for patient age)
Frequency (collecting the hourly weight of a patient may be unnecessary)
Unnecessary processing
Treatment, analysis, emrichment, transformation (Face recognition emotion detection)
Propagation (information such as patient location is shared to other locations where it is not needed)
Implicit data disclosure
Duration/retention (holding patient data after unsubscribing from a service)
Involved parties and exposure
Involved parties (patient personal data shared with users or other patients)
Availability/accessability of data (publishing research data sets such as patient weight from a medical research)


Non-compliance
Regulatory non-compliance
GDPR (Insufficient patient controls, violation of data minimization principle, unlawful processing of personal data, violation of the principle of storage limitation)
Insufficient cybersecurity risk management
No security assessment taken on the medical application/system
Operational security (outdated mecial software/applications)

8.2 Appendix B: Threat to CWE Mapping

Privacy Threat Types	Cases	Misuse cases	CWE CATEGORY
Linkability	User linked using identifiers	Users using the same email as unique identifiers can be linked across different services	1210, 1211, 1212, 137, 265, 355
	User linked using a combination or analysis of data	A combination of IP address, age, post code can be used to identify a patient	840, 417, 255, 310, 320, 1214, 19, 137, 1219, 1227, 1218, 355
		DNA sample can reveal links between relatives and individuals of similar genetics	
		Intercepted messages between patient and service providers or between service providers about a patient can be intercepted	
Identifiability	Identity information	Information provided at healthcare service point such as full name and address for registration process can be accessed by third parties	840, 417, 255, 310, 320, 1214, 19, 137, 1219, 1227, 1218, 355, 1217
	Identity information in metadata	Information of the author of documents available in metadata	1288, 1210, 1211, 1212, 1006, 1217
	Pseudonym	Cases of insufficient de-identification or anonymization allowing the unique identifier to still be tied to actual patient identity after being replaced	255, 310, 137, 1219, 389, 199, 133, 136
	Revealing attributes	Data left in so-called anonymous feedback forms may be sufficiently revealing and can uniquely identify the patient	355
	Patient is distinguishable from others	The search queries for the individual are specific to that user, and this can lead to their identification	355
Non-repudiation	Attributable data that is evidence worthy	Transmission logs of patient information (results, medication, etc) can be intercepted and analysed	417, 387

Privacy Threat Types	Cases	Misuse cases	CWE CATEGORY
	Attributable action side-effect that is evidence worthy	Read notifications can be used to identify the presence of a patient	1210, 1006, 1226, 355, 1217
Detecting	When patient communication is observed	Observing the use of a telemedicine or healthcare service gives an indication that the person is a user	417, 387
	Side effect of the application used by the patient	By accessing the log files of a shared system, all the users and the applications they used will be revealed	1210, 1212, 1006, 255, 310
	System responses	Detailed error responses can indicate existence of an email address (wrong password), device at an IP address (closed port)	389, 569, 429, 199, 452, 1215, 1216
Disclosure of Information	Unnecesssary data types	Taking more data than is required to perform the health service	1006, 840
	Excess data volume	Unnecessary to collect data such as patient age for a general service, or the frequency at which weight data is collected	
	Unnecessary processing	hijacking patient treatment analysis for emotion detection, construction of patient profiles, etc	
	Involved parties and exposure	mishandled patient data by set of parties that were predetermined and those that are dynamic	
Unawareness	Unawareness of processing	Unclear notices of to whom and how patient data is shared	1228, 840, 1006
	Lack of data subject control	How data is shared, no control about shared data, unable to delete or retrieve data	

Privacy Threat Types	Cases	Misuse cases	CWE CATEGORY
Non-compliance	Regulatory non-compliance	Insufficient patient controls on collected data	1228, 840, 1006
	Insufficient cybersecurity risk management	Poor internal control mechanisms	1211, 1212, 1217, 265

8.3 Appendix C - Expert Evaluation Form and Review Instructions

LINDDUN-to-CWE Traceability Matrix spreadsheet with 63 threat-instance entries (one row per identified privacy threat instance, Table 3-3) across three sheets https://docs.google.com/spreadsheets/d/1VlHSI5z9LmNlAG34QKMWslENlLSlgMoO/edit?usp=sharing&ouid=100173560846807486762&rtpof=true&sd=true

8.4 Appendix D - Expert Evaluation Form and Review Instructions

https://docs.google.com/document/d/1XCF1XXv_eLKFRy_3iB6eSuS1H58eBlB_/edit?usp=sharing&ouid=100173560846807486762&rtpof=true&sd=true

https://docs.google.com/spreadsheets/d/14l6W_Zoh6P8lL3WFZzaDlrH56j4afQW5/edit?usp=sharing&ouid=100173560846807486762&rtpof=true&sd=true

8.5 Appendix E – Transformer Models Architecture Adapted for Binary Vulnerability Classification

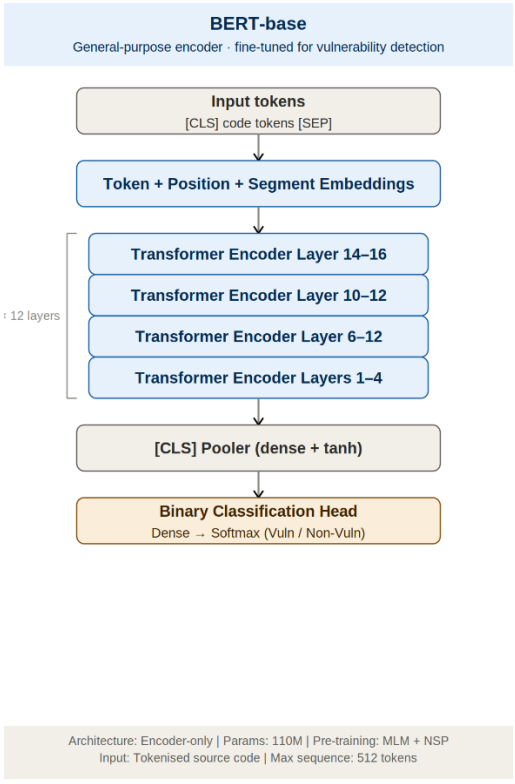


Figure 8-1: BERT-base architecture

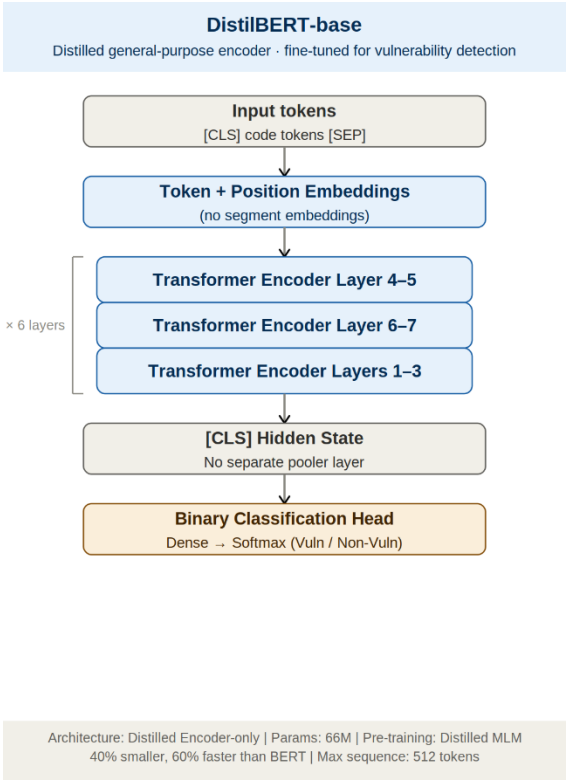


Figure 8-2: DistilBERT architecture

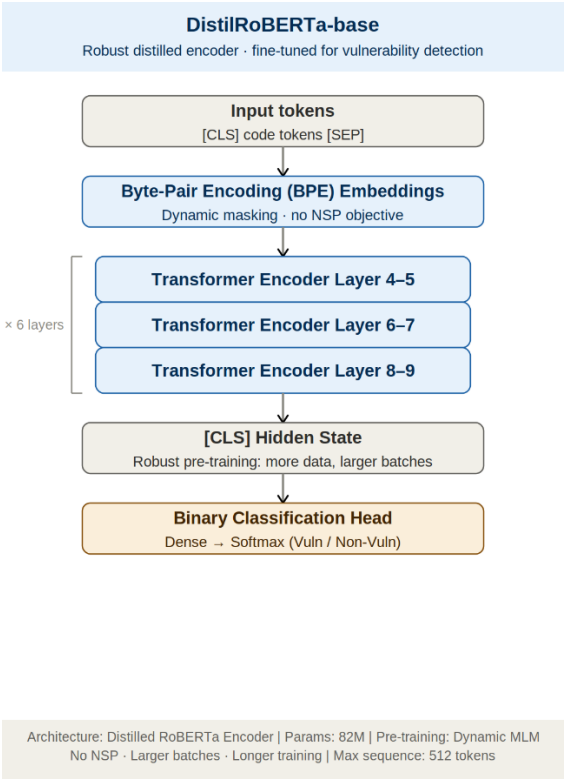


Figure 8-3: DistilRoBERTa architecture

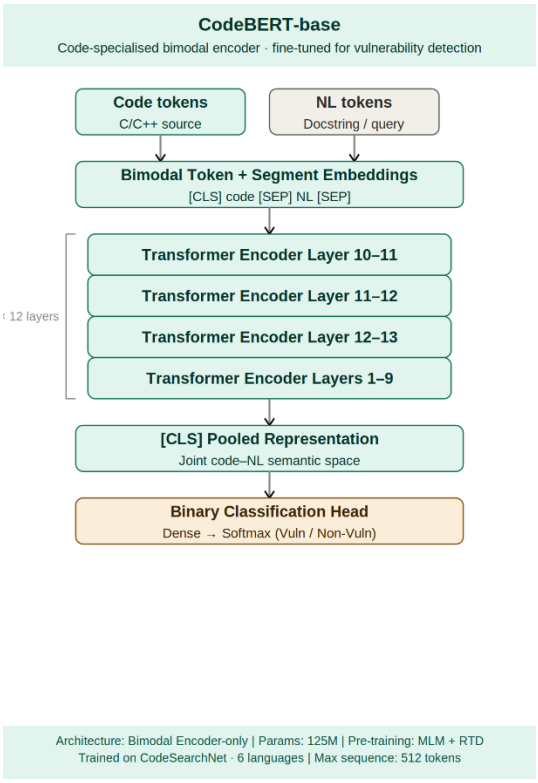


Figure 8-4: CodeBERT bimodal architecture

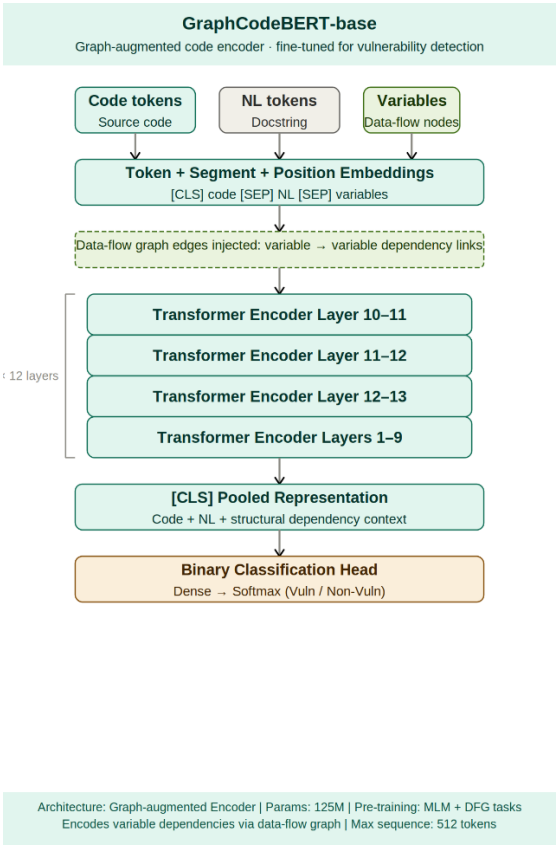


Figure 8-5: GraphCodeBERT graph-augmented architecture

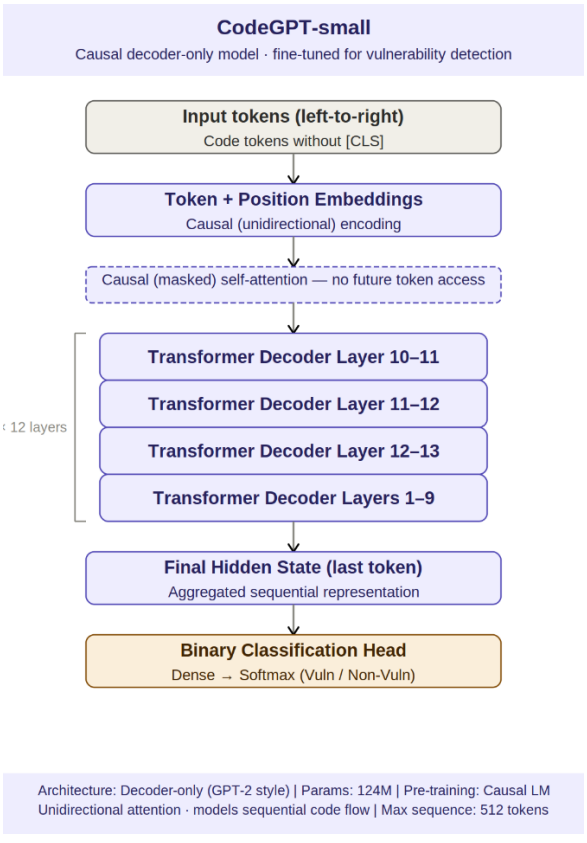


Figure 8-6: CodeGPT-small causal decoder architecture

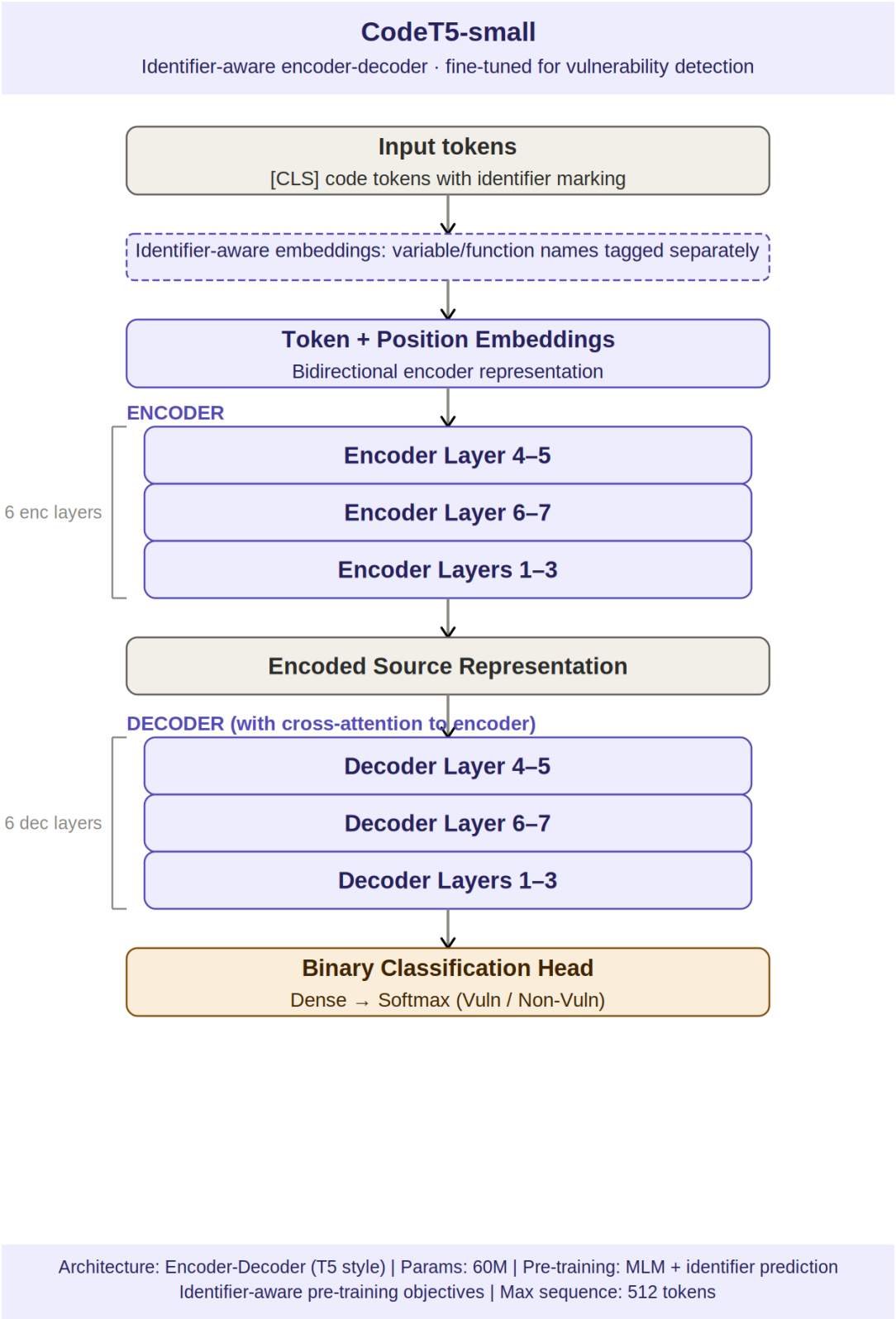


Figure 8-7: CodeT5-small encoder-decoder architecture

8.6 Appendix F: Vulnerability detection system prototype high-level architecture

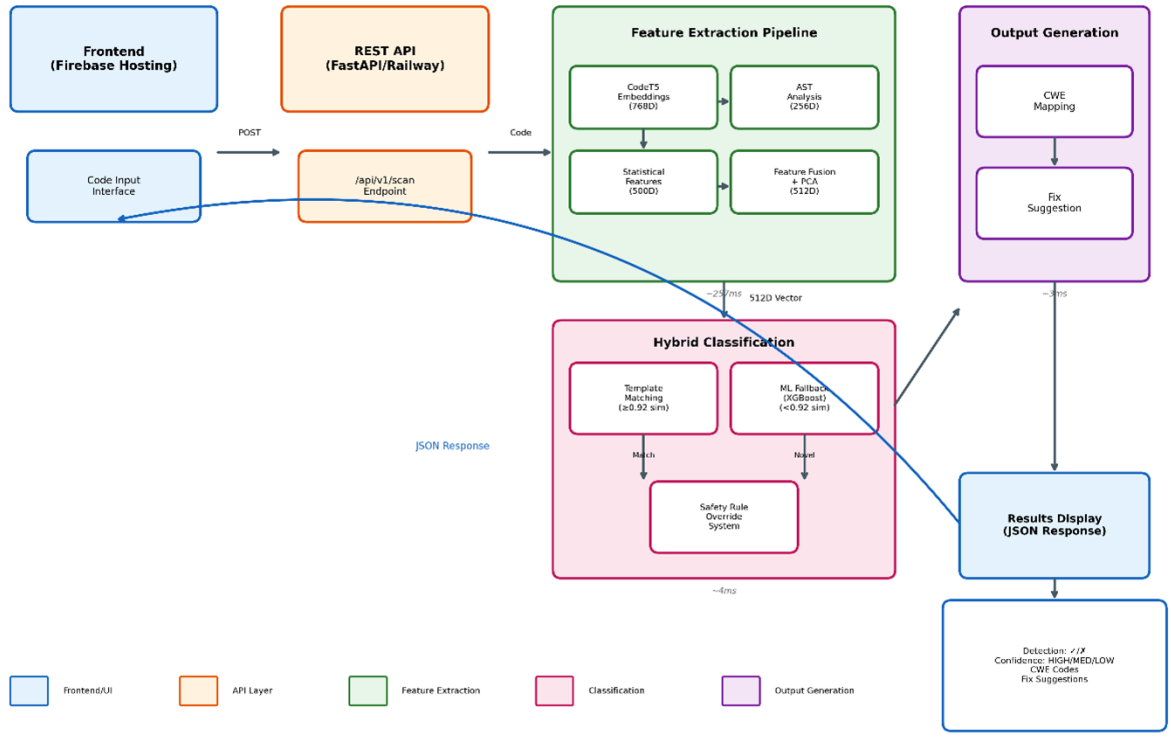


Figure 8-8: high-level architecture of the vulnerability detection prototype system

8.7 Appendix G: Participant Information Sheet and Consent Form

https://drive.google.com/file/d/1BA1C75EUpynXcrKfbLb9pbaQQS6JIKcz/view?usp=sharing

Participant Consent Form – Fill in form

Participant Consent Form – Results

8.8 Appendix H: AI Vulnerability System User Evaluation Form

C/C++ AI Vulnerability Detection System User Evaluation – Fill in form

C/C++ AI Vulnerability Detection System User Evaluation – Results

8.9 Appendix I: Synthetic Data Generation: Per-CWE Candidate Sample Counts

These tables report the number of candidate synthetic samples generated per CWE category at each generation stage (Section 4.2.4). These are generation-stage counts, not final dataset composition. The manual inspection and deduplication process applied before merging into C3-VULMAP was not logged at the per-CWE level, so the exact number of these candidates retained in the released dataset cannot be reported separately (see Section 4.2.4 for full discussion).

Table 8-1: Stage One: GPT-3.5 Turbo Generation for Critically Underrepresented CWEs

96 CWE categories, 19,600 total candidate samples generated (200 per category for 94 categories; 400 for CWE-190 and CWE-200).

CWE ID	Stage One Candidate Samples Generated
CWE-190	400
CWE-200	400
CWE-841	200
CWE-862	200
CWE-863	200
CWE-89	200
CWE-918	200
CWE-924	200
CWE-94	200
CWE-654	200
CWE-694	200
CWE-77	200
CWE-78	200
CWE-787	200
CWE-79	200
CWE-798	200
CWE-434	200
CWE-476	200
CWE-502	200
CWE-586	200
CWE-628	200
CWE-639	200
CWE-640	200
CWE-324	200
CWE-346	200
CWE-352	200
CWE-360	200
CWE-400	200
CWE-416	200
CWE-425	200
CWE-1097	200

CWE ID	Stage One Candidate Samples Generated
CWE-1104	200
CWE-117	200
CWE-119	200
CWE-1235	200
CWE-125	200
CWE-20	200
CWE-22	200
CWE-269	200
CWE-287	200
CWE-306	200
CWE-322	200
CWE-120	200
CWE-121	200
CWE-122	200
CWE-191	200
CWE-201	200
CWE-209	200
CWE-202	200
CWE-203	200
CWE-204	200
CWE-205	200
CWE-215	200
CWE-220	200
CWE-221	200
CWE-222	200
CWE-223	200
CWE-224	200
CWE-225	200
CWE-226	200
CWE-227	200
CWE-228	200
CWE-229	200
CWE-230	200
CWE-231	200
CWE-232	200
CWE-233	200
CWE-234	200
CWE-235	200
CWE-236	200
CWE-237	200
CWE-241	200
CWE-242	200
CWE-243	200
CWE-244	200
CWE-245	200
CWE-246	200
CWE-247	200
CWE-248	200
CWE-249	200
CWE-250	200
CWE-251	200

CWE ID	Stage One Candidate Samples Generated
CWE-252	200
CWE-253	200
CWE-254	200
CWE-255	200
CWE-256	200
CWE-257	200
CWE-259	200
CWE-1043	200
CWE-1046	200
CWE-1048	200
CWE-1049	200
CWE-1067	200
CWE-1071	200
CWE-1092	200

Table 8-2: Stage Two: GPT-3.5 Turbo Generation for CWE Category Balancing

49 CWE categories, 999,992 total candidate samples generated (20,408 per category, uniform across all 49).

CWE ID	Stage Two Candidate Samples Generated
CWE-119	20408
CWE-125	20408
CWE-787	20408
CWE-89	20408
CWE-79	20408
CWE-77	20408
CWE-78	20408
CWE-22	20408
CWE-416	20408
CWE-476	20408
CWE-287	20408
CWE-306	20408
CWE-352	20408
CWE-502	20408
CWE-798	20408
CWE-862	20408
CWE-1043	20408
CWE-1046	20408
CWE-1048	20408
CWE-1049	20408
CWE-1067	20408
CWE-1071	20408
CWE-1092	20408
CWE-1097	20408
CWE-1104	20408
CWE-117	20408
CWE-1235	20408
CWE-190	20408
CWE-20	20408
CWE-200	20408
CWE-269	20408
CWE-322	20408

CWE ID	Stage Two Candidate Samples Generated
CWE-324	20408
CWE-346	20408
CWE-360	20408
CWE-400	20408
CWE-425	20408
CWE-434	20408
CWE-586	20408
CWE-628	20408
CWE-639	20408
CWE-640	20408
CWE-654	20408
CWE-694	20408
CWE-841	20408
CWE-863	20408
CWE-918	20408
CWE-924	20408
CWE-94	20408