AN AI-INTEGRATED METHODOLOGY FOR SECURE SOFTWARE AND SYSTEM DEVELOPMENT

by

Ian Matthew Campbell Coston

A Dissertation Submitted to the Faculty of

The College of Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Florida Atlantic University

Boca Raton, FL

August 2026

Copyright 2026 by Ian Matthew Campbell Coston

AN AI-INTEGRATED METHODOLOGY FOR SECURE SOFTWARE AND SYSTEM DEVELOPMENT

by

Ian Matthew Campbell Costan

This dissertation was prepared under the direction of the candidate's dissertation advisor, Dr. Mehrdad Nojoumian, Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

SUPERVISORY COMMITTEE:

Mehrdad Nojoumian, Ph.D.
Dissertation Advisor

Taghi Khoshgoftaar, Ph.D.

Borko Furht, Ph.D.

Mohammad Ilyas, Ph.D.

Hari Kalva, Ph.D.
Chair, Electrical Engineering and Computer Science

Stella Batalama, Ph.D.
Dean, The College of Engineering and Computer Science

Robert W. Stackman Jr, Ph.D.
Dean, Graduate College

May 27, 2026
Date

ACKNOWLEDGEMENTS

This dissertation came together because of the support and guidance of many people, and I want to take a moment to recognize them here.

First, my deepest thanks go to my advisor and committee chair, Dr. Mehrdad Nojoumian. Your guidance, patience, and steady encouragement carried me through this process from start to finish. You shaped the way I think about this field, and your mentorship was essential to completing this work.

I am also grateful to my committee members, Dr. Taghi Khoshgoftaar, Dr. Borko Furht, and Dr. Mohammad Ilyas. Your feedback and constructive critiques made this dissertation stronger. Thank you for your time, your expertise, and your support.

To Eadan Plotnizky and Karl David Hezel: thank you. You both stepped up when the work got heavy. As Cybectr LLC interns working under my direction, you each brought real skill to this project. Eadan, your background in penetration testing saved me from blind spots I didn't even know existed. You pushed the hardware and network layers harder than I would have on my own. Karl, your coding work kept the pipeline running while I was stuck on theory. You caught bugs in the Sentinel implementation that would have been difficult to find later. I appreciated your dedication throughout, but more than that, I appreciated your willingness to challenge me when the work needed it. Both Eadan and Karl have provided written permission for the use of their testing data, penetration test findings, and development contributions in this dissertation.

To my wife and family: thank you. Your patience, understanding, and belief in me kept me going through the long nights of research and writing. You even read

this dissertation, which says a lot. I love you, and your support means the world to me.

To our son Ezra Maurice Coston, who we lost last year: you were with me through so much of this journey, even if you never got to see it finished. I carry you with me always, and part of this work belongs to you.

And to Reese Martin Campbell Coston, who we are waiting to meet: you are already a source of hope and joy. I cannot wait to share this accomplishment with you someday and tell you about everyone who helped make it possible.

Finally, a personal note for my grandfather, Jose Miguel Martin, and my grandmother, Paula Martin, who we also lost last year. Papa and Paula, I finished. You can call me Dr. Coston now. I know this was something you were looking forward to, and I hope I made you proud. I like to think you are looking down, happy with what I accomplished.

This journey was challenging, but it was also rewarding. I am grateful to everyone who played a part in it.

ABSTRACT

Author:	Ian Matthew Campbell Coston
Title:	An AI-Integrated Methodology for Secure Software and System Development
Institution:	Florida Atlantic University
Dissertation Advisor:	Dr. Mehrdad Nojoumian
Degree:	Doctor of Philosophy
Year:	2026

Securing interconnected software systems requires more than layering existing frameworks on top of each other. Most current Secure Software and System Development Lifecycle (S-SDLC) models treat security as a phase rather than a design condition, leaving real gaps in governance, access control, and automated enforcement that become critical failure points in Internet of Things (IoT) environments where devices are resource-constrained, long-lived, and frequently insecure by default.

This dissertation introduces the Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D) methodology, a novel approach that unifies DevSecOps automation, the National Institute of Standards and Technology (NIST) Risk Management Framework (RMF), and Zero Trust (ZT) architecture into a single cohesive lifecycle with Artificial Intelligence (AI) as the orchestrating engine. AZTRM-D applies Zero Trust principles not only to users and devices but to its own AI components, addressing a blind spot present in every comparable framework evaluated. The methodology is operationalized through Cybectr Sentinel, an AI enforcement platform developed by Cybectr LLC, a United States Department of Defense contracting company, featuring six AI subsystems with formal mathemati-

cal specifications: Isolation Forest for behavioral anomaly detection, XGBoost with SHapley Additive exPlanations (SHAP) for vulnerability triage, Sentence Transformers for unknown asset inference, Proximal Policy Optimization (PPO) for AI-guided penetration testing, Retrieval-Augmented Generation (RAG) with a Large Language Model (LLM) for mitigation generation, and Diverse Counterfactual Explanations (DiCE) for adversarial robustness testing.

Empirical validation was conducted on physical NVIDIA Jetson Orin Nano hardware across three progressive hardening stages, with adversarial testing spanning hardware, radio frequency (RF), network, software, insider, privileged insider, and AI-assisted attack vectors. Testing was performed by three testers under a blind protocol with inter-rater agreement analysis (Gwet Agreement Coefficient 1 (AC1) = 0.888). All three testers are affiliated with Cybectr LLC; independent third-party laboratory validation is the highest-priority future-work deliverable. Factory-default devices were fully compromised to root level in under five minutes. Following full AZTRM-D hardening, zero successful breaches were recorded across any tested vector. The five-modality Continuous Integration/Continuous Delivery (CI/CD) scanning pipeline achieved a measured Vulnerability Detection Rate (VDR) of 96.8% (Wilson 95% Confidence Interval: [0.891, 0.991]). Cybectr Sentinel delivered 94.1% precision, 91.8% recall, a 3.1% aggregate False Positive Rate (FPR), and a 4.2-minute average Time to Initial Detection (TTID) within 12–18% Central Processing Unit (CPU) overhead on constrained edge hardware. The 93.7% adversarial detection rate is evaluated under a black-box threat model using DiCE-generated counterfactual inputs; gradient-based white-box evaluation is scoped to future work. A 14-capability comparative analysis against five established secure development frameworks found seven capabilities present in AZTRM-D that are absent from every evaluated alternative: AI-Assisted Code Analysis, Zero Trust Network Architecture, AI-Guided Penetration Testing, Unknown Asset Similarity Analysis, MITRE Detection, Denial, and

Disruption Framework Empowering Network Defense (D3FEND) Mapping, MITRE Adversary Engagement, Deception, and Denial Framework (ENGAGE) Active Defense, and Post-Quantum Readiness Planning [69].

This dissertation also presents a post-quantum integration framework addressing the Harvest Now, Decrypt Later (HNDL) threat model, with concrete guidance for transitioning AZTRM-D's cryptographic controls to NIST post-quantum standards including Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) under Federal Information Processing Standards (FIPS) 203, Module-Lattice-Based Digital Signature Algorithm (ML-DSA) under FIPS 204, and Stateless Hash-Based Digital Signature Algorithm (SLH-DSA) under FIPS 205.

AN AI-INTEGRATED METHODOLOGY FOR SECURE SOFTWARE AND SYSTEM DEVELOPMENT

List of Figures xvii

1 Introduction 1

1.1 Overview 1

1.2 Motivation and Contribution 3

1.3 Dissertation Outline 5

2 The Constrained-Edge Threat Environment as a Test Bed for AZTRM-D 7

2.1 Introduction to IoT 7

2.1.1 Overview of IoT 7

2.1.2 IoT Architectures 10

2.2 IoT Common Communication Protocols 14

3 Attack Surface Taxonomy and Countermeasures in the Validation Environment 23

3.1 IoT Vulnerability Layers 23

3.1.1 Hardware Vulnerabilities 26

3.1.2 Software Vulnerabilities 35

3.1.3 Network Vulnerabilities 38

4 Related Work and Foundational Concepts 51

4.1 Software and System Development Lifecycles 51

4.1.1 Waterfall Method 51

4.1.2 Agile Method 52
4.1.3 V-Model 52
4.1.4 Spiral Method 52
4.1.5 DevSecOps Method 53
4.1.6 Secure Development Methodologies and Frameworks: Where Things Stand 54
4.2 The Rationale for AZTRM-D's Integrated Approach 58

5 Understanding the Risk Management Framework 60
5.1 NIST Risk Management Framework 60
5.1.1 Preparation Phase 62
5.1.2 Categorization Phase 62
5.1.3 Selection Phase 64
5.1.4 Implementation Phase 65
5.1.5 Assessment Phase 65
5.1.6 Authorization Phase 65
5.1.7 Monitoring Phase 66

6 The Guiding Philosophy: Zero Trust Architecture in AZTRM-D 67
6.1 Zero Trust 67
6.1.1 Identity 71
6.1.2 Devices 72
6.1.3 Networks 72
6.1.4 Applications and Workloads 73
6.1.5 Data 73
6.1.6 Cross-Cutting Capabilities: Governance, Automation, and Visibility 73
6.1.7 Integrating RMF, ZT, and DevSecOps in AZTRM-D 74

7 Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D) . . . 75

7.1 Phases of AZTRM-D . . . 76

7.1.1 Planning Phase . . . 76

7.1.2 Development Phase . . . 78

7.1.3 Build Phase . . . 79

7.1.4 Test Phase . . . 80

7.1.5 Release and Deliver Phase . . . 80

7.1.6 Deployment Phase . . . 81

7.1.7 Operate, Monitor and Feedback Phase . . . 81

7.2 NIST Artificial Intelligence Risk Management Framework . . . 82

7.3 Implementation of Zero Trust Methodologies on Artificial Intelligence Models/Features . . . 84

8 AZTRM-D Simulated Theoretical Scenario . . . 86

8.1 Initial Scenario . . . 86

8.1.1 Scenario Description . . . 86

8.1.2 AZTRM-D Walkthrough . . . 87

8.1.3 AZTRM-D Enabled System Versus the Hacker . . . 88

9 Lab-Built Scenario and Benchmarking . . . 91

9.1 Implemented Security Posture . . . 91

9.2 Security Testing Campaign . . . 92

9.2.1 Security Testing Scenario: Factory Default Configuration . . . 92

9.2.2 Security Testing Scenario: After Initial Hardening . . . 94

9.2.3 Security Testing Scenario: After Final Hardening . . . 95

9.2.4 AI-Assisted Insider Attack Validation . . . 96

9.3 Quantitative Benchmarking Results . . . 97

9.3.1 Vulnerability Detection Rate Measurement Methodology . . . 97

9.3.2 Supply Chain Vulnerability Quantification . . . 102

9.3.3 Scalability Projection Methodology 104
9.3.4 System Technical Overview 106
9.3.5 User Interaction with AZTRM-D Implemented System 107
9.3.6 Future Setup Additions 111

10 Cybectr Sentinel: Architecture, AI Stack, and Operational Mechanics 114
10.1 The Four Capability Gaps Sentinel Fills 115
10.2 Sentinel Workflow End-to-End 116
10.3 AI Subsystem Specifications 118
10.4 Explainable AI Implementation and Claude Integration 121
10.5 How Sentinel Uses AI: A Single Finding Through the Full Stack . . . 125
10.6 Sentinel Performance Metrics 126
10.7 Sentinel versus Manual and Commercial Alternatives 130
10.8 What Sentinel Does and Does Not Replace 131

11 Comparative Analysis, Validation, and Quantitative Benchmarking of the AZTRM-D Framework 136
11.1 Introduction 136
11.2 Conflict of Interest and Independent Validation Status 137
11.3 AZTRM-D versus Conventional SDLC Methodologies and Secure Frameworks 139
11.3.1 Methodology Comparison 140
11.3.2 Secure Framework Capability Comparison 141
11.3.3 Fourteen-Capability Matrix 142
11.3.4 Consolidated SDLC and Security Framework Comparison . . . 145
11.3.5 Implementation Cost Comparison 147
11.3.6 Performance Overhead: Putting the Numbers in Context . . . 155
11.4 AI Stack Selection: Why These Algorithms 158
11.4.1 Behavioral Anomaly Detection: Isolation Forest over Alternatives 158

11.4.2 Vulnerability Triage: XGBoost over Random Forest and Neural Classifiers . . . 162
11.4.3 AI-Guided Pen Testing: PPO over Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG) . . . 166
11.4.4 GNN-Augmented SAST: Why Graph Neural Networks over Pattern Matching . . . 169
11.4.5 Unknown Asset Inference: Sentence Transformers and Cosine Similarity . . . 171
11.4.6 Adversarial Robustness: DiCE Counterfactuals . . . 173
11.4.7 Adversarial Threat Model and Scope of the 93.7% Detection Rate 174
11.4.8 Comparative Performance Against Published External Baselines 176
11.5 Tool Stack Selection: Why These Specific Tools . . . 179
11.6 Testing Methodology and Team Structure . . . 185
11.6.1 Inter-Rater Reliability and Procedural Bias Mitigation . . . 185
11.6.2 Attack Vector Methodology by Layer . . . 190
11.6.3 Tester Roles and Rotation . . . 194
11.6.4 Attack Surface Coverage . . . 195
11.7 Multi-Vector Penetration Testing Results . . . 197
11.7.1 Stage 1: Factory Default Configuration . . . 198
11.7.2 Stage 2: After Initial Network Hardening . . . 201
11.7.3 Stage 3: Full AZTRM-D Hardening . . . 203
11.8 Quantitative Benchmarking . . . 206
11.8.1 Vulnerability Detection Rate: Reference and Ablation Setup . 206
11.8.2 Per-Modality Ablation: Each Scanner's Unique Contribution . 206
11.8.3 Security Effectiveness Metrics . . . 209
11.8.4 Resource and Scalability Metrics . . . 211
11.9 Generalizability Beyond the NVIDIA Jetson Orin Nano Test Platform 213
11.9.1 Three Categories of Transfer Claims . . . 213
11.9.2 How Different Hardware Architectures Would Shift the CPU Overhead and PEP Latency Metrics . . . 215

11.9.3 Why the Pipeline Controls Transfer Directly 216
11.9.4 Zero Trust Enforcement in Enterprise Contexts 217
11.9.5 Alignment with Federal Zero Trust Policy and the Civilian-Agency Mandate 217
11.9.6 Applicability to National Security Systems, DoD, and Intelligence Community Environments 218
11.9.7 What AZTRM-D Inherits from Enterprise and Government Practice, Not the Reverse 220
11.9.8 Insider Threat Coverage in Enterprise Environments 221
11.9.9 Bounded Scope of the Generalizability Claim 221
11.10 Datasets, Preprocessing, and Training Protocols 221
11.11 Cybectr Sentinel: Validation Summary 225
11.12 Zero Trust Architecture Enforcement 226
11.13 Cryptographic Enforcement Across Deployment Models 228
11.13.1 Full-Disk Encryption: AES-256-XTS Implementation 228
11.13.2 Transport Security: TLS 1.3 and WPA3-SAE 229
11.13.3 Digital Signatures: ECDSA with RFC 6979 Deterministic Nonces 230
11.13.4 Service Identity: SPIFFE/SPIRE SVIDs 230
11.13.5 Cryptographic Control Validation in the Pipeline 231
11.14 NIST RMF Phase Mapping 233
11.15 Consolidated Results 235
11.16 Discussion and Limitations 238
11.16.1 Affiliation of All Three Testers with Cybectr LLC 239
11.16.2 Single Device Class 239
11.16.3 Adversarial Evaluation Breadth 240
11.16.4 Vulnerability Test Corpus Size 240
11.16.5 False Positive Rate Per-Class Decomposition 241
11.16.6 Comparative Framework Evaluation Scope 241
11.16.7 Proprietary Platform and Reproducibility 242

11.16.8 Self-Measured Performance Metrics 242

12 The Future of AZTRM-D: Preparing for the Post-Quantum Era 244

12.1 The Quantum Threat to Modern Cryptography 245

12.1.1 Shor's Algorithm and the Threat to Asymmetric Cryptography 245

12.1.2 Grover's Algorithm and the Impact on Symmetric Cryptography 246

12.1.3 The Harvest Now, Decrypt Later Threat Model 247

12.1.4 Categorizing Data Sensitivity for HNDL Risk Assessment . . . 248

12.2 Post-Quantum Cryptographic Standards 249

12.2.1 NIST Post-Quantum Cryptography Standardization Process . 249

12.2.2 Understanding Lattice-Based Cryptography 251

12.2.3 Alternative Post-Quantum Cryptographic Approaches 252

12.3 Integrating Post-Quantum Cryptography into AZTRM-D 253

12.3.1 Modifications to the Planning Phase 254

12.3.2 Modifications to the Development Phase 256

12.3.3 Modifications to the Build and Test Phases 257

12.3.4 Modifications to the Deploy and Operate Phases 258

12.3.5 Considerations for IoT and Constrained Devices 258

12.3.6 ML-KEM Performance Projection for NVIDIA Jetson Orin . . 260

12.4 Zero Trust and Cryptographic Agility 262

12.4.1 Cryptographic Agility as a Zero Trust Capability 262

12.4.2 Zero Trust Principles Applied to Post-Quantum AI Components 263

12.5 Migration Challenges and Strategic Recommendations 264

12.5.1 Technical Migration Challenges 264

12.5.2 Organizational Migration Challenges 265

12.5.3 Recommended Migration Strategy for AZTRM-D Implementations 265

12.6 Regulatory and Compliance Considerations 267

13 Significance and Conclusion 270
13.1 Summary of Contributions 270
13.2 Limitations and Challenges 272
13.2.1 Organizational and Resource Challenges 272
13.2.2 AI-Specific Limitations 273
13.2.3 Integration and Interoperability Challenges 274
13.2.4 Scope and Generalizability Considerations 275
13.2.5 Threats Don't Stand Still 275
13.3 Significance and Broader Impact 276
13.4 Future Work 278
13.4.1 Advanced AI Techniques 278
13.4.2 Blockchain Integration 278
13.4.3 Adversarial AI Defenses 279
13.4.4 Post-Quantum Cryptography Evolution 279
13.4.5 Domain-Specific Adaptations 279
13.4.6 Empirical Validation at Scale 280
13.5 Final Remarks 280

Abbreviations 282

Bibliography 291

LIST OF FIGURES

2.1 Five-layer architecture. This image was inspired by [344]. 9

2.2 IoT applications within smart cities. This image was inspired by [338]. 10

3.1 IoT vulnerability flow. This image was inspired by [222]. 23

3.2 Bluetooth man-in-the-middle attack. This image was inspired by [124]. 29

3.3 Reverse engineering and inspection attacks. This image was inspired by [252]. 30

3.4 Trojan activation modes. This image was inspired by [161]. 33

3.5 Working process of a simple hardware Trojan. This image was inspired by [161]. 34

3.6 Distributed Denial of Service (DDoS) attack. This image was inspired by [26]. 39

3.7 Types of attacks. This image was inspired by [315]. 40

3.8 WiFi over-the-air attack. This image was inspired by [18]. 41

3.9 Wireless sensor networks DoS attacks. This image was inspired by [290]. 43

3.10 Blackhole attack. This image was inspired by [13]. 44

3.11 Wormhole attack tunnel. This image was inspired by [321]. 45

3.12 System model of a typical smart cloud-based home deployment. This image was inspired by [101]. 46

3.13 Phantom-delay attack. This image was inspired by [101]. 47

4.1 The DevSecOps Lifecycle, integrating security into every phase from planning to feedback. This image was inspired by [77]. 53

5.1 The cyclical nature of the RMF, where monitoring provides continuous feedback into the preparation for the next cycle. This image was inspired by [297, 188]. 63

5.2 A conceptual diagram illustrating a system's Authorization Boundary, which contains critical assets, and its connections to the wider Environment of Operation. This mapping is a key activity in the RMF Categorization phase. This image was inspired by [297]. 64

6.1 A general overview of the Zero Trust Architecture concept, where all access requests from users, devices, or applications are untrusted by default and must pass through a policy enforcement gateway for verification. This image was inspired by [266, 128]. 68

6.2 The five foundational pillars of Zero Trust, supported by three cross-cutting capabilities. The goal is to advance each pillar from a traditional, reactive security posture to an optimal, proactive state. This image was inspired by [72, 107]. 69

7.1 A high-level mapping of how Zero Trust and Risk Management Framework processes align with the phases of the DevSecOps pipeline within the AZTRM-D framework. 76

7.2 AZTRM-D AI Architecture and Orchestration Engine 77

8.1 AZTRM-D attack-chain analysis for the Company A simulated scenario. 88

9.1 VDR breakdown by scanning modality: seeded versus organically discovered vulnerabilities (left) and unique versus cross-modality shared detections (right) from the 63-vulnerability test corpus. 100

9.2 The internal security process flow for a device operating under the AZTRM-D model, from secure boot to continuous monitoring and policy enforcement. 108

9.3 The secure Git server workflow within AZTRM-D, illustrating the mandatory security checks and administrative approvals required for code progression. 110

9.4 The multi-layered user authentication and session management flow for accessing a device-hosted server under AZTRM-D's strict ZT policies. 112

10.1 Cybectr Sentinel versus manual analyst workflows and commercial tooling across five normalized capability dimensions. 131

11.1 Fourteen-capability support matrix across six secure development frameworks. 145

11.2 MTTR comparison for four vulnerability categories under conventional DevSecOps versus AZTRM-D Stage 3 (left), and implementation cost by framework showing initial setup versus ongoing sprint hours (right). The asterisk on the right panel indicates that the AZTRM-D setup cost is front-loaded and non-recurring. 155

11.3 AZTRM-D runtime performance on the NVIDIA Jetson Orin Nano (Stage 3): device CPU overhead versus comparable approaches (left), ZT policy enforcement latency across endpoint counts (center), and key measured metrics (right). Green metric values in the right panel indicate measurements within their target operating envelope; the red value flags the false positive rate, which is the only metric whose triage cost an operator must monitor over time. 157

11.4 Penetration-test findings across three hardening stages on the NVIDIA Jetson Orin Nano: total findings and tester agreement (left), and attack vector success rate by stage (right). 189

11.5 Security posture progression across the three AZTRM-D hardening stages for four key risk metrics (0 = fully exposed, 100 = fully mitigated). 198

11.6 Vulnerability detection rate breakdown across the five CI/CD scanning modalities from Stage 3 pipeline results. 209

11.7 Security posture across five key metrics: Stage 1 factory-default baseline versus Stage 3 full AZTRM-D hardening on the NVIDIA Jetson Orin Nano. 211

12.1 AZTRM-D cryptographic component migration map. 254

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Somewhere between the convenience of a connected world and its consequences, security got left behind. Software ships fast. Systems scale fast. Attack surfaces grow faster than most organizations can track them. And the security frameworks built to manage that problem are often fighting the last war, relying on manual checkpoints at the end of development cycles, perimeter defenses that assume trust once someone is inside the network, and risk management processes that happen on paper rather than in the pipeline.

The Internet of Things (IoT) makes this concrete in a way that's hard to ignore. Billions of sensors, actuators, embedded controllers, and smart devices now operate across smart cities, industrial control systems, healthcare infrastructure, and military environments [68]. Many were designed for function, not security. They ship with weak authentication, unpatched firmware, no secure update mechanisms, and communication protocols that were never built to resist a motivated adversary. An IoT device is often a gateway into critical infrastructure, and the attack surface it represents is enormous.

S-SDLC models acknowledge the problem. Few actually solve it. Waterfall-era security reviews arrive after the architecture is already locked in [50]. Agile sprints move too fast for meaningful manual security integration [235]. DevSecOps gets the automation philosophy right but doesn't prescribe how to govern risk formally or enforce access control continuously [347]. ZT addresses the trust model but

leaves implementation entirely to the engineer [266]. NIST RMF provides rigorous governance but needs an agile engine to run it at modern development pace [297]. Each framework solves part of the problem.

AZTRM-D is a synthesis built on the premise that DevSecOps automation, NIST RMF governance, and Zero Trust access control work better as a single integrated system than as separate practices applied in sequence. AI runs throughout the framework as the orchestrating engine, automating threat modeling, driving static and dynamic analysis, enforcing continuous access policy, and maintaining security posture through the operational life of deployed systems. The goal is a methodology that makes security not a phase of development but the fundamental condition under which development happens.

A note on scope before the chapters that follow. AZTRM-D is a domain-general methodology, not an IoT-specific one. Its constituent practices, DevSecOps automation, the NIST RMF, Zero Trust architecture, and AI-driven orchestration, originated in and currently govern federal enterprise and National Security Systems (NSS) environments. NIST SP 800-207 was written for enterprise security architects [266]; the NIST RMF (SP 800-37 Rev. 2) governs federal information systems [297]; Executive Order 14028 and OMB Memorandum M-22-09 set Zero Trust as a federal requirement across civilian agencies [310, 234]; NSM-8 extends those requirements to DoD and the Intelligence Community [312]; and the DoD Zero Trust Reference Architecture and DoD Zero Trust Strategy operationalize the same control set inside cleared environments [75, 78]. AZTRM-D applies that established federal practice under tighter resource constraints than any of those environments impose, not the other way around. The empirical validation reported in this dissertation uses constrained edge hardware as the test bed because it represents the most demanding deployment context: limited compute headroom, an exposed physical attack surface, and an RF layer that does not exist in cloud or enterprise data center environments. Fed-

eral IoT cybersecurity guidance (NIST SP 800-213, SP 800-213A, and NISTIR 8228) establishes that the same controls AZTRM-D enforces are also expected of federal IoT systems, which means the validation environment is itself a federally recognized cybersecurity context rather than a peripheral one [86, 87, 41]. Chapters 2 and 3 characterize that test environment in detail, covering the architectural layers, communication protocols, and attack surface that the validation campaign in Chapter 9 actively exercises. Demonstrating that the methodology's controls hold under those conditions establishes a floor on what AZTRM-D can support. Enterprise software, healthcare, defense, financial services, and critical infrastructure deployments inherit the same architectural controls with more compute headroom, more mature identity infrastructure, and far smaller physical attack surfaces (typically zero exposed Universal Asynchronous Receiver-Transmitter (UART), General Purpose Input/Output (GPIO), or RF interfaces in data center environments). Section 11.9 develops the generalizability argument formally, classifying each transfer claim as empirically validated, architecturally transferable, or hypothesized but requiring future validation, and anchoring every architectural-transfer claim to the federal policy or standards document that already specifies the same control for enterprise or NSS use.

1.2 MOTIVATION AND CONTRIBUTION

The case for AZTRM-D comes from specific, documented failures in how software and systems get built.

IoT environments represent an attack surface that existing frameworks weren't built to address. Device constraints push security features out in favor of battery life, compute efficiency, and cost [68]. Firmware rarely gets updated after deployment. Authentication is often hardcoded or absent. The protocols spanning Bluetooth, ZigBee, Long Range Wide Area Network (LoRaWAN), Narrowband Internet of Things (NB-IoT), 5G, and others were built for efficiency, not security [222]. NIST has

acknowledged this gap explicitly. SP 800-213 provides federal agencies with guidance on establishing IoT device cybersecurity requirements, and its companion catalog (SP 800-213A) maps device cybersecurity capabilities to NIST SP 800-53 controls so that procurement and deployment can be traced back to RMF baselines [86, 87]. Those documents define a federal floor for IoT cybersecurity that AZTRM-D's methodology operationalizes through automation. Chapter 9 documents this directly: a factory-default NVIDIA Orin device can be fully compromised in under five minutes using freely available tools and publicly documented default credentials. That is the baseline AZTRM-D is designed to replace.

Development speed creates its own pressures. Continuous integration and delivery (CI/CD) pipelines push code to production constantly. Security that lives outside those pipelines, manual reviews, quarterly penetration tests, post-release audits, simply can't keep pace. The findings arrive too late to remediate cheaply, which means they often don't get remediated at all. Security debt accumulates, incidents happen, and the root cause is usually that security was never a design constraint from the start.

Zero Trust addresses the access control problem in principle but stops short of specifying how to implement it across a heterogeneous development environment. Applying it to the AI tools that enforce security is rarer still. If the security automation itself is trusted implicitly, that's an unmonitored gap at the exact layer enforcing security. AZTRM-D applies Zero Trust principles to its own AI components, treating them as entities that must be continuously authenticated, authorized, and monitored.

This research makes seven specific contributions. First, the formal definition of AZTRM-D as a novel integrated S-SDLC methodology that unifies DevSecOps, the NIST RMF, Zero Trust, and AI orchestration into a single lifecycle. Second, a detailed integration strategy that names which DevSecOps practices connect to which NIST RMF phases under which Zero Trust controls, with AI engaged at each con-

nection. Third, practical AI applications across the lifecycle phases, ranging from automated threat modeling at planning through behavioral anomaly detection in operations. Fourth, the application of Zero Trust principles and the NIST AI Risk Management Framework (AI RMF) to AZTRM-D's own AI components, treating the enforcement layer itself as untrusted until verified. This is an original contribution that addresses a gap present in every comparable framework evaluated. Fifth, empirical validation through a controlled lab implementation on physical NVIDIA Orin devices, with adversarial testing performed by three testers (all Cybectr LLC affiliated, operating under a blind protocol) across hardware, RF, network, software, insider, privileged insider, and AI-assisted attack vectors [69]. Sixth, the development and documentation of Cybectr Sentinel, an AI enforcement platform built under the author's United States Department of Defense contracting company Cybectr LLC, which operationalizes AZTRM-D through six AI subsystems with measured performance metrics [69]. Seventh, a post-quantum integration framework that addresses the HNDL threat model and specifies the migration path to the NIST post-quantum standards.

1.3 DISSERTATION OUTLINE

The dissertation builds from foundational context through methodology and then to empirical validation and future requirements. Chapters 2 and 3 characterize the constrained-edge validation environment that the empirical campaign exercises, covering its architecture, communication protocols, and the full attack surface across hardware, software, and network layers. Chapter 4 reviews existing S-SDLC models and secure development frameworks, making the case for why an integrated approach is necessary rather than redundant.

Chapters 5 and 6 examine the two foundational pillars of AZTRM-D in depth, covering the seven NIST RMF steps with AI integration at each, and the Zero Trust

architecture including its novel application to AI components within the framework. Chapter 7 presents the complete AZTRM-D methodology across every lifecycle phase, and Chapter 8 walks through a detailed financial services attack scenario to demonstrate the methodology in practice.

The empirical core of the dissertation is Chapters 9 and 11. Chapter 9 documents the lab implementation on physical NVIDIA Orin hardware, the three-stage hardening process, and quantitative results from penetration testing. Chapter 11 extends this with the full comparative analysis against conventional methodologies and secure frameworks, the multi-vector adversarial testing campaign conducted by three testers under a blind protocol, the derivation of all quantitative benchmarks including the 96.8% vulnerability detection rate, the AI algorithm selection rationale, and the complete Cybectr Sentinel architecture and performance metrics [69].

Chapter 12 addresses the quantum computing threat, covering Shor's and Grover's algorithms, the HNDL attack model, the NIST post-quantum standards (FIPS 203, ML-DSA under FIPS 204, and SLH-DSA under FIPS 205), and concrete modifications for integrating post-quantum cryptography into AZTRM-D. Chapter 13 closes with a summary of contributions, an honest assessment of limitations, and directions for future research.

CHAPTER 2
THE CONSTRAINED-EDGE THREAT ENVIRONMENT AS A TEST BED FOR AZTRM-D

2.1 INTRODUCTION TO IOT

IoT devices are everywhere. Billions of sensors, actuators, embedded controllers, and connected systems now run across smart cities, factories, hospitals, and military networks. They collect data, make decisions, and talk to each other over wireless protocols that were built for efficiency, not security. The problem is that most of these devices ship with weak authentication, unpatched firmware, and communication channels that a motivated attacker can intercept with commodity hardware. Vulnerabilities span every layer: hardware, software, cloud infrastructure, and network communications. Unpatched firmware, default credentials, unsecured wireless transmissions, exploitable network protocols. Each one opens a door. Unlike traditional computing systems, IoT devices operate with limited compute and memory, which means security features get cut first when something has to give. This chapter investigates those vulnerabilities by categorizing them based on where they originate and what they enable. It also examines the attack vectors adversaries actually use to turn those weaknesses into compromises, and the countermeasures that work against them [68].

2.1.1 Overview of IoT

The 'Internet of' family of technologies has grown across several distinct branches. Internet of People covers human–machine interaction. Internet of Agents covers ma-

chine learning and artificial intelligence systems acting on data. Internet of Content covers cloud-delivered media. And Internet of Things covers machine-to-machine interaction across physical devices [245]. This work focuses on the last of those. An IoT device is a physical sensor, actuator, or embedded controller that communicates with other devices over a wireless protocol, typically Bluetooth, Wireless Fidelity (WiFi), 5G, ultra-wideband, or radio frequency identification. The devices exchange data, coordinate actions, and feed information into larger systems that act on it [245, 103, 289]. Deployments now span cities, homes, transportation, military operations, agriculture, and many more domains. What unifies them is that the IoT layer turns previously passive infrastructure into something responsive. Architecturally, IoT systems are typically described as having between three and six layers. The four-layer model is the most common in practice, but the five-layer model in Figure 2.1 is more useful for this investigation because it separates network access from network transmission and exposes the layers where security controls actually engage [344].

The five layers proceed from the device upward. The perception layer is where sensors and similar hardware collect data from the physical environment, anything from temperature readings to barcode scans [344]. Above that, the network access layer handles how a device sends signals to its immediate neighbors using protocols like ultra-wideband, WiFi, and Bluetooth, forming the local sensor network [344]. The network transmission layer carries those signals across the broader network, whether by satellite, cellular, or dedicated infrastructure [344]. The application support layer hosts middleware, cloud platforms, and the application logic itself [344]. The presentation layer is where the system finally meets its purpose, smart city services, agricultural monitoring, transportation coordination, or any of the other domains IoT serves [344]. Each layer carries its own threat surface, and the security argument later in this chapter follows that same vertical structure.

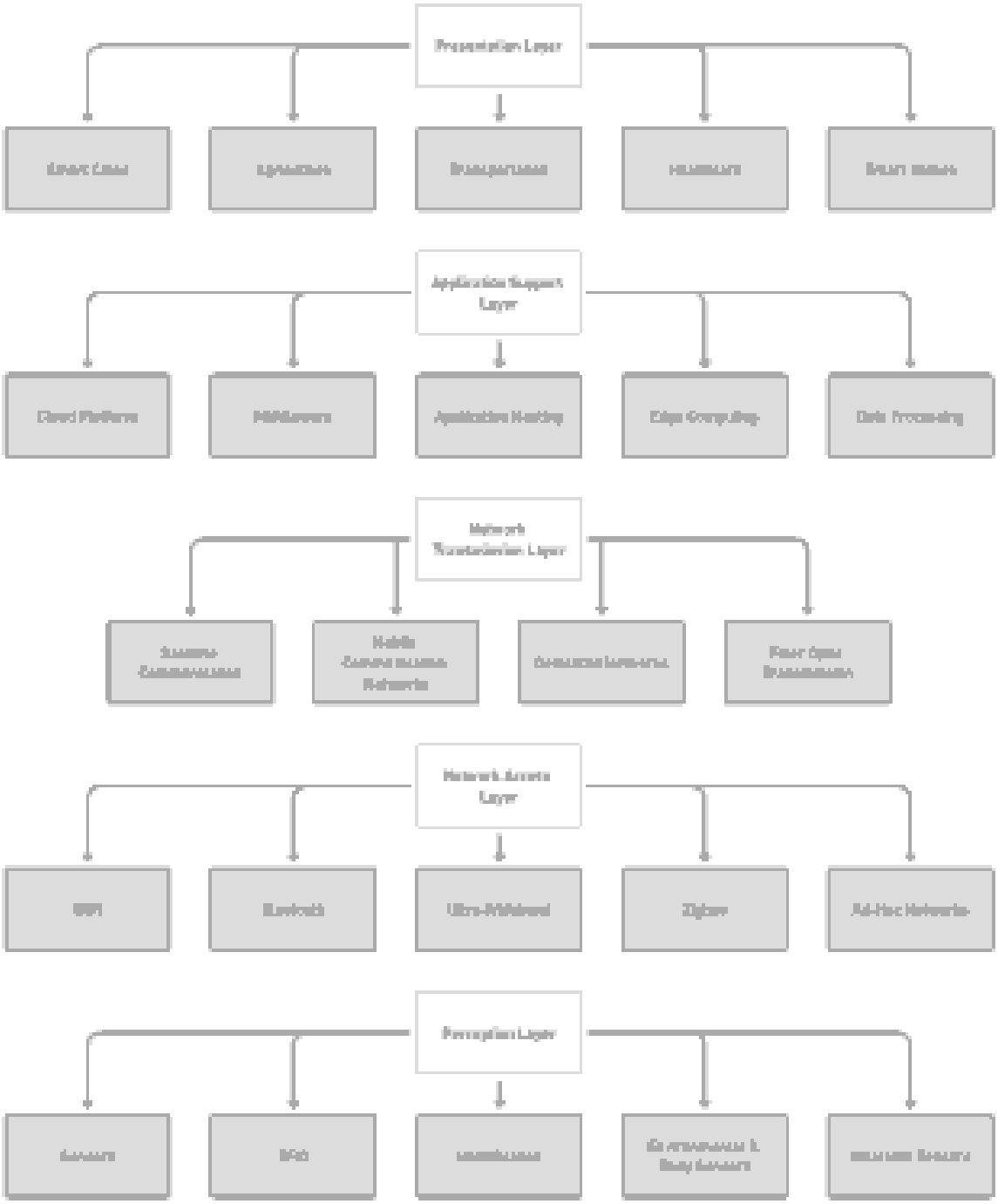


Figure 2.1: Five-layer architecture. This image was inspired by [344].

2.1.2 IoT Architectures

Internet of Things for Smart Cities and Transportation

As the Internet of Things devices became more prevalent and advanced, they are slowly being integrated into cities and vehicles in various aspects to allow for a smoother city or vehicular experience. The IoT can be used everywhere within cities, from security to maintenance, covering operations and even vehicle mobility [24]. Some of the most common uses of the Internet of Things in cities can be seen in Figure 2.2. The figure documents many distinct IoT applications within smart cities, each serving its own specific purpose.

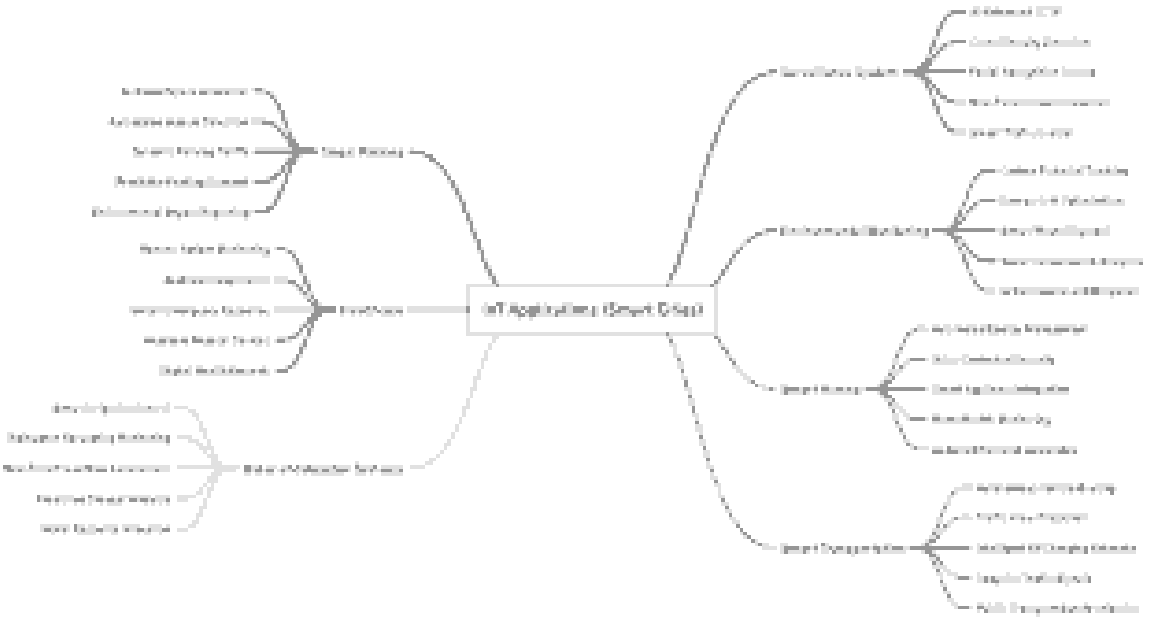


Figure 2.2: IoT applications within smart cities. This image was inspired by [338].

A smart city integrates IoT across transportation, public safety, payments, and retail. Connected vehicles using vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) protocols coordinate with traffic lights and surrounding cars to manage flow. CCTV networks tied to municipal monitoring systems address public safety. Smart parking meters bill drivers automatically and report occupancy in real time. Cashier-free retail uses sensor and computer vision systems to track item placement and

customer movement. Each capability depends on continuous, reliable IoT communication. With those baseline capabilities established, transportation is one of the fastest-growing IoT integration domains [24]. V2V and V2I communication protocols are how vehicles exchange position, speed, and signal data with each other and with the surrounding infrastructure. Electric scooters and bus systems run on the same IoT layer. Smart meters use V2I to measure vehicle dwell time at a spot, which produces more accurate billing than legacy magnetic-loop sensors. Self-driving cars combine the same protocols with onboard perception to receive speed limits, signal states, sign data, and the locations of other vehicles, pedestrians, and obstacles. The common thread is that the IoT layer makes urban infrastructure more responsive and the user's interaction with it more direct.

Internet of Things for Advanced Manufacturing

IoT has moved well beyond homes and cities into manufacturing. The Industrial Internet of Things (IIoT) puts sensors, actuators, and connected systems directly onto factory floors, and the results are measurable. IoT spans many distinct manufacturing applications, each serving specific operational needs. Predictive maintenance is the canonical example. Sensors monitoring temperature, vibration, and performance feed analytics that flag impending failures before they become breakdowns. The same telemetry layer also drives process automation. Robots and automated systems exchange operational state and adjust to real-time conditions, rerouting tasks or modulating production rates when a line bottlenecks. Beyond the factory floor itself, IIoT improves supply chain management through sensor-driven inventory and logistics tracking, which keeps stock levels accurate and reduces waste. Quality control benefits as well. Continuous monitoring throughout the production process detects defects early and supports consistency [291]. The unifying goal across all of these applications is the same: a production process that is more efficient, more reliable, and

lower cost, with downstream benefits for consumers in the form of better products and services.

Internet of Things Smart Home System

IoT in the home is now a category in its own right. What started as a few add-on gadgets connected over a home wireless network has become a default feature of consumer hardware. Native IoT capability now ships in appliances, lighting, climate control, and entertainment systems out of the box [292]. The deployment varies widely across households, but a representative scenario looks like this. A user presses a button in their car to open the garage. They tell an Alexa device they're home, which triggers an arrival routine: lights on, thermostat set to 68 degrees Fahrenheit, air purifiers to medium, living room TV on, all in sequence. At bedtime, a single voice command triggers a wind-down routine that closes the blinds, sets the air purifiers to high, turns off the lights, lowers the thermostat, and starts the robot vacuum. Routines and triggers differ from one household to the next, but the common thread is that the home IoT layer ties physical actions together into single events.

Internet of Medical Things

The Internet of Medical Things (IoMT) is the medical application of IoT, and it has become standard equipment across hospitals, nursing homes, and patient homes for personal health monitoring. The clinical impact is concrete. Wireless devices stream heart rate, blood pressure, temperature, peripheral capillary oxygen saturation (SpO_2), and respiratory rate to healthcare providers in near real time. The same data feed supports continuous monitoring, scheduled spot checks, and early detection of abnormalities that might otherwise go unnoticed between visits. Patient-side mobile applications connect to the same IoMT devices, giving patients direct access to their health metrics and supporting self-management of chronic conditions. The

CardiacSense watch is one representative IoMT product [47]. It records pulse rate and electrocardiogram (ECG) data, performs continuous A-fib detection, supports manually added measurements, runs general arrhythmia detection with configurable thresholds, and produces detailed event reports, monthly summaries, and sleep-time tracking [143].

Internet of Agricultural Things

Agriculture has become one of the most active IoT adoption domains. The Internet of Agricultural Things (IoAT) covers everything from home gardening systems to industrial-scale smart farms, and addresses applications including precision farming, livestock monitoring, greenhouse monitoring, and agricultural drones. Smart farms deploy diverse sensor stacks. Soil sensors track soil richness, temperature, humidity, gas concentrations, air pressure, water pressure, and crop disease indicators. Livestock sensors track temperature, heart rate, and digestion [91]. SeeTree, an "Intelligence Platform for Trees," is a representative innovation. The system tracks the productivity and health of individual trees at scale by combining drone imagery, satellite data, IoT sensor feeds, and climate data across hundreds of millions of trees [278, 279].

Internet of Battlefield Things

The Internet of Battlefield Things (IoBT) is the military application of IoT, and it has grown rapidly across operational environments [162, 269]. IoBT devices serve two functions. They monitor friendly forces and surveil hostile ones, and they directly support warfighting operations. In a typical field deployment, a soldier wears a heart rate sensor, a salinity sensor, an optical sensor, and movement tracking, all feeding health and position data back to command. Overhead, unmanned aerial vehicles (UAVs) provide video coverage of the operational area, tracking both friendly and

hostile movement. Ground vehicles run tracking technology, mine detection, and weapon-mounted sensors. The category is simultaneously broad and specific. Signal towers and soldier-worn health monitors both fall under it. Any IoT technology becomes an IoBT device when it operates in the battlefield environment.

2.2 IOT COMMON COMMUNICATION PROTOCOLS

IoT protocols evolved alongside the devices themselves, and the diversity reflects the range of operating conditions. Energy budgets vary from solar-powered field sensors to mains-powered industrial controllers. Data rates range from a few bytes per hour for battery-conservative deployments to gigabits per second for video and high-resolution telemetry. Some applications need long-range, others need short-range with high security. This section walks through the protocols most commonly encountered in IoT deployments and a few less common protocols that fill specialized niches.

Zigbee Wireless Protocol (ZigBee)

ZigBee is a wireless networking protocol based on the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 technical standard [90]. It has a low data rate, low power consumption, low cost, and encrypted communication using the Advanced Encryption Standard (AES) with a 128-bit key. ZigBee is a great technology for Internet of Things devices that need a long battery life and a low data transfer rate. The ZigBee data rate is 250 kbps at 2.4 Gigahertz (GHz) (global), 40 kbps at 915 Megahertz (MHz) (Americas), and 20 kbps at 868 MHz (Europe) [90]. Some applications of ZigBee can be found in smart homes, smart buildings, the Internet of Medical Things (IoMT), and more [83, 82].

Dash7

Dash7 (D7AP), part of the Dash7 Alliance Protocol family, is an open source sub-GHz wireless sensor and actuator network protocol (WSAN) that complies with the International Organization for Standardization / International Electrotechnical Commission (ISO/IEC) 18000-7 standard. It has a medium range of up to 2 Km, low power consumption (multi-year battery life), low latency, and is encrypted using AES with a 128-bit key. D7AP operates in unlicensed Industrial, Scientific, and Medical (ISM) bands of 433.92, 868, and 915 MHz [230, 246]. A few use cases of D7AP can be found in agriculture IoT, IoMT, and smart cities [28].

WiFi

WiFi, or IEEE 802.11, is a standard of wireless Local Area Network (LAN) technology widely used in business and home environments for fast and reliable Internet access. WiFi connects devices ranging from smart home applications to industrial sensors. The range, reliability, and security profile of WiFi make it suitable for many IoT applications [73]. WiFi encryption has evolved generation by generation and provides stronger security against attacks. Wired Equivalent Privacy (WEP) was the original WiFi security protocol, which used the Rivest Cipher 4 (RC4) algorithm with two sides of data communication. However, WEP is easily cracked, and it is no longer considered secure. Wi-Fi Protected Access (WPA) was an improved version of WEP and addressed some of the security vulnerabilities in WEP. Wi-Fi Protected Access II (WPA2) is the successor to WPA and is considered to be more secure, replacing the RC4 cipher with the AES cipher [167]. Wi-Fi Protected Access III (WPA3) is the latest WiFi security protocol, offering the strongest security to date. WPA3 uses the Simultaneous Authentication of Equals (SAE) protocol, designed to be more secure than the four-way handshake used in WPA2 [158]. IEEE 802.11 is a living standard and new generations are being developed regularly to meet the growing demands of

wireless networking. Some of the most used IEEE 802.11 standards are IEEE 802.11g, which operates in the 2.4 GHz band and supports data rates up to 54 Mbps; IEEE 802.11n, which operates in the 2.4 GHz and 5 GHz bands and supports data rates up to 600 Mbps.; IEEE 802.11ac, which operates in the 5 GHz band and supports data rates up to 6.93 Gbps; and IEEE 802.11ax, which operates in the 2.4 GHz and 5 GHz bands and supports data rates up to 9.6 Gbps [237, 33].

Cellular

Cellular networks have been around for a while. With new advancements and technologies being developed around them, new cellular networks were discussed, tested, and created to provide the best performances that can match the needs of the latest technology standards. Long-Term Evolution Advanced (LTE-A, 4G) and 5G technologies are the most recent and widely used cellular standards [85]. LTE-A is a wireless cellular technology that delivered major gains in speed, capacity, and coverage over previous generations of cellular technology. It has a peak downlink data rate of 1 Gbps, an uplink data rate of 500 Mbps, a peak downlink spectrum efficiency of 30 bps/Hz, an uplink spectrum efficiency of 15 bps/Hz, and a bandwidth of 100 MHz [10]. LTE-A is currently being overtaken by 5G [343]. 5G is the fifth generation of wireless cellular technology. It has demonstrated improvements over previous generations of cellular networks, including higher data rates, lower Quality of Service (QoS) latency, less interference, and increased capacity. The 5G requirements include a maximum downlink data rate of 20 Gbps, an uplink data rate of 10 Gbps, a maximum downlink spectrum efficiency of 30 bps/Hz, an uplink spectrum efficiency of 15 bps/Hz, user plane latency of 4 ms for Enhanced Mobile Broadband (eMBB) and 1 ms for Ultra-Reliable Low-Latency Communication (URLLC), control plane latency of 10–20 ms, and a bandwidth of 100 MHz–1 GHz [56]. In sum, 5G is the key for advanced IoT applications, such as smart factories, smart hospitals, smart

transportation, smart agriculture, smart homes and cities, etc. [56, 339].

6LoWPAN

6LoWPAN is a networking technology that allows Internet Protocol version 6 (IPv6) packets to be efficiently transmitted over low-power wireless networks, such as those based on the IEEE 802.15.4 standard. It supports various mesh network topologies and can fragment and reassemble packets as needed. 6LoWPAN implementations are small enough to fit 32 K flash memory parts. 6LoWPAN enables low-power mesh and sensor networks to take advantage of the benefits of Internet Protocol (IP) networking [282, 198]. It has frequency bands of 2.4 GHz, 868 MHz, and 915 MHz (the same as ZigBee) and data rates between 50 and 250 kbit/s [282].

Bluetooth

Bluetooth is a technology standard used for short-range wireless communication between mobile devices. Bluetooth operates on 79 different frequencies to transmit data from 2.402 GHz to 2.48 GHz and a range up to 100 m (330 ft). The bit rates for Bluetooth are 1 Mbps and 2 Mbps [30]. It is very useful for transmitting small fragments from different IoT sensors [40].

Bluetooth Low Energy (BLE)

BLE is a wireless technology that is designed to complement both classic Bluetooth and the lowest power wireless technology possible. It is a distinct technology with different design goals and market segments than classic Bluetooth. BLE transmits data over 40 channels in the 2.4 GHz band (2.402 to 2.48 GHz) [313]. It can be used to create different types of networks, from simple point-to-point connections to complex mesh networks. This flexibility makes BLE compatible with a wide range of applications, including the Internet of Things [313].

Long Range (LoRa) and Long Range Wide Area Network (LoRaWAN)

LoRa is an unlicensed band physical layer technology that transmits data signals in the subGHz ISM band. LoRa allows low-data-rate long-range, low-power wireless communication. LoRa has a range of up to 15 km in rural areas and up to 5 km in urban areas, with data rates of 0.3 kbps–50 kbps in Europe and 0.9 kbps–50 kbps in the US [127, 154]. LoRaWAN is an open standard that was developed on top of Long Range (LoRa). It consists of an end device, gateway, network server, and application server. LoRaWAN is located in the data link layer and provides a complete solution by adding a network layer that includes features such as security, authentication, and data routing [28].

SigFox

Sigfox is a Low-Power Wide-Area Network (LPWAN) technology designed for the Internet of Things. It uses ultra-narrowband technology to transmit data with a very low power consumption over long distances [168]. It operates in the 862–928 MHz frequency band and has a channel bandwidth of 100 Hz [100]. With a range of up to 50 km in rural areas and up to 10 km in urban areas, Sigfox can work well in applications that require long-range communication with battery-powered devices. Its data rate ranges from 100 to 600 Bps, depending on the region [15, 56].

Narrowband Internet of Things (NB-IoT)

NB-IoT is a low-cost, low-power, and low-data-rate cellular technology built from Long-Term Evolution (LTE) functions; therefore, it uses the same infrastructure as cellular networks, which makes it a scalable and reliable technology that can be deployed in a variety of locations. It has a range of up to 15 km in rural areas and 1–5 km in urban areas, a data rate up to 250 Kbps, and a 200 Kilohertz (KHz) bandwidth [256, 28].

Near-Field Communication (NFC)

NFC is a short-range wireless communication protocol used by mobile devices for all kinds of applications, such as payments, digital keys for homes and cars, and data transferring. NFC provides secure communication between various devices. It has a short range of 4–10 cm, a data rate of 0.02–0.4 Mbps, and it runs on a 13.56 MHz spectrum [65]. NFC is used to extend different IoT solutions with short-range capabilities [66].

Z-Wave

Z-Wave is a subGHz wireless communication protocol used by different IoT applications. It is an ultra-low-power, mesh network protocol that lets devices communicate with each other over long distances (has a range of 100 m). Its data rates are 9.6 kbps, 40 kbps, or 100 kbps, and it uses a frequency of 908.42 MHz in the United States and 868.42 MHz in Europe [74]. Z-Wave deployments can be scaled by linking together Z-Wave networks. Z-Wave is well suited for applications that require reliable, secure, and low-power communication, such as control smart home devices (lights, locks, thermostats, and security systems) [272].

Li-Fi

Li-Fi is a bidirectional short-range wireless technology that uses a visible light communication (VLC) system for data transmission to transfer and receive data. Li-Fi uses overhead Light-Emitting Diode (LED) lighting commonly found in homes as a means of transport and a photo-diode for decoding data. It has demonstrated speeds of up to 224 Gbps under laboratory conditions, which would allow a high-definition video to be downloaded in seconds. Because Li-Fi is reliable in light use, it is limited in range since light cannot pass through objects, which makes Li-Fi effective only in closed spaces [38]. Even though its range limitation could be seen as a problem, this

limitation provides an additional layer of security by prevents data from leaking into public spaces, thus preventing malicious actors from accessing your network [123].

Ultra-Wideband (UWB)

Ultra-wideband is a short-range, high-bandwidth and energy-efficient wireless communication protocol that can be used for radar imaging, sensor data collection, and precise location and tracking. UWB operates at frequencies from 3.1 to 10.6 GHz, has a bandwidth of >500 MHz, and a data rate of up to 1 Gbps [345, 130]. UWB can be used to accurately measure the distance between two devices. This information can be used for a variety of IoT applications [7].

Advanced Message Queuing Protocol (AMQP)

AMQP is a reliable and versatile Machine-to-Machine (M2M) binary protocol. It offers two levels of Quality of Service (QoS) for the delivery of messages, uses Transmission Control Protocol (TCP) as a transport protocol, and uses Transport Layer Security / Secure Sockets Layer (TLS/SSL) and Simple Authentication and Security Layer (SASL) for security, making it a good fit for high-bandwidth, reliable, and secure networks [199]. It supports various messaging patterns, including request/response, publish/subscribe, and transactions (allowing multiple messages to be sent and received as a single unit of work) and topic-based publish-and-subscribe messaging (allows messages to be published to topics so that subscribers can receive messages that are relevant to them) [94].

Constrained Application Protocol (CoAP)

The CoAP is a lightweight M2M binary protocol with a fixed header of 4 bytes and small message payloads from the Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) Working Group designed for constrained IoT devices.

It supports both request–response and resource–observe architectures and can be used to interoperate with Hypertext Transfer Protocol (HTTP) and the RESTful Web Application Programming Interface (API) [37]. CoAP uses User Datagram Protocol (UDP) as a transport protocol and Datagram Transport Layer Security (DTLS) for security, making it efficient for use on low-bandwidth and unreliable networks. It is designed to be as lightweight as possible, making it suitable for use on constrained devices with limited resources [199].

Message Queuing Telemetry Transport Protocol (MQTT)

MQTT is a publish/subscribe messaging protocol used for lightweight machine-to-machine (M2M) communications in constrained networks. MQTT uses TCP as its transport protocol that guarantees the delivery of messages. MQTT also uses TLS/SSL for security, which encrypts messages to protect them from unauthorized access. MQTT supports three levels of QoS, making it more reliable when delivering messages. It uses a small amount of bandwidth and processing power. This makes it ideal for use with small devices that have limited resources. MQTT is also suited for large networks because it can efficiently handle a large number of devices. Its publish/subscribe model lets devices receive only messages relevant to them [199].

Data Distribution Service (DDS)

DDS is a machine-to-machine protocol developed by the Object Management Group (OMG) that features decentralized nodes of clients throughout a system (nodes can identify themselves as subscribers or publishers through a localization server). DDS was created to overcome the disadvantages of centralized publish–subscribe architectures. It provides many quality-of-service parameters that allow users to control the behavior of the DDS system, such as improved scalability, increased reliability, reduced latency, bandwidth, and built-in security (provides authentication, access

control, confidentiality, and integrity to the information distribution) [55, 1].

Open Platform Communications (OPC)

OPC is a machine-to-machine protocol that enables real-time data exchange between control systems and devices. OPC supports classic specifications such as OPC DA (Data Access) and newer advances such as OPC UA (Unified Architecture), which increase platform security and independence. OPC is widely used in Industry 4.0 environments, connecting IoT and cyber-physical systems (CPS) across heterogeneous industrial platforms [111].

This protocol diversity is part of what makes IoT security difficult. Each protocol carries its own threat model, and the resulting attack surface spans hardware, RF, network, and software layers in ways that single-domain security frameworks rarely cover. The next chapter examines these attack surfaces directly. It documents how adversaries exploit them and what countermeasures apply at each layer, providing the threat-model foundation that the validation chapters later build on.

CHAPTER 3

ATTACK SURFACE TAXONOMY AND COUNTERMEASURES IN THE VALIDATION ENVIRONMENT

3.1 IOT VULNERABILITY LAYERS

All IoT devices typically show some kind of vulnerability or weakness that dampens its ability to function without issue [41]. These vulnerabilities can typically allow users to gain access to data by exploiting a weakness to enter the device, or they can be used to track and manipulate the device. Either way, this is detrimental to the device and its functionality. Some of the ways that adversaries can obtain access to this is through network vulnerabilities, software vulnerabilities, or even hardware vulnerabilities, as shown below in Figure 3.1 [222]. Knowing these weaknesses helps determine the best route to avoid or mitigate them [68].

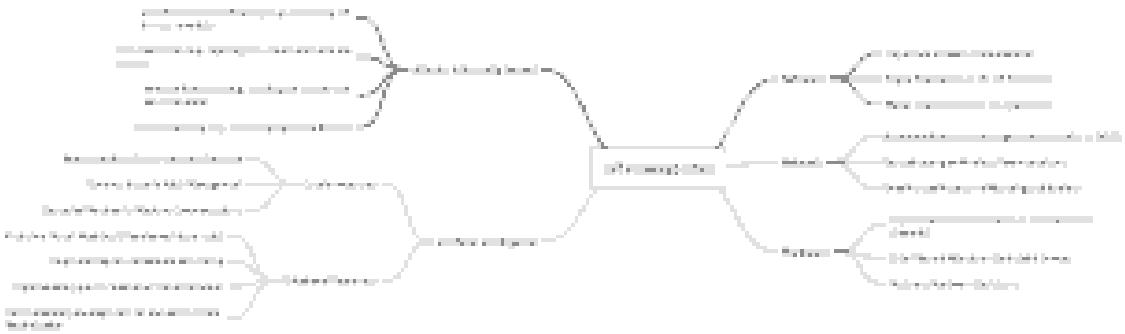


Figure 3.1: IoT vulnerability flow. This image was inspired by [222].

Figure 3.1 shows that these vulnerabilities are just one side of the attack picture. The other side is how adversaries use them, and that determines which security properties get attacked: confidentiality, integrity, accountability, or availability [222]. Confidentiality protects IoT devices and their data from unauthorized access. The

standard defenses are encryption, access control, and authentication of both users and data [222]. When confidentiality fails through one of the vulnerabilities above, the result is information leakage, where sensitive data ends up reaching parties who should not have it. Integrity protects against unauthorized modifications to hardware or software. Encryption, input validation, and interface monitoring and restrictions are the controls that maintain it [222]. Integrity failures usually trace back to small oversights at the design phase that an attacker can later exploit to repurpose a device for their own ends. Accountability is about tracking what a device actually does and constraining it to its intended behavior [222]. An adversary exploiting manufacturer vulnerabilities can modify event paths, change device purpose, or reroute data to themselves, all of which break the accountability property. Availability is the requirement that a device remains usable when needed [222]. Attacks against availability delay the device or take it offline, leaving the user without service. Each of these security properties maps to a distinct adversary goal, and the attack categories that follow in this chapter (physical, software, network) typically target one or more of them at the same time.

Table 3.1 consolidates the attack-surface taxonomy developed across the rest of this chapter. Each row identifies an attack category, the representative vectors discussed in the subsections that follow, the primary security properties (Confidentiality, Integrity, Availability, Accountability) most directly affected, and the layer of the AZTRM-D defense that addresses the category during the validation campaign in Chapter 11.

Table 3.1: Attack-surface taxonomy for the validation environment. Categories and attack vectors are drawn from the subsections that follow in this chapter; AZTRM-D defensive coverage maps to the controls validated in Chapter 11. CIA-A = Confidentiality, Integrity, Availability, Accountability.

Attack Category	**Representative Vectors**	**CIA-A Impact**	**AZTRM-D Defensive Layer**
Hardware: RF emanation and man-in-the-middle	Signal interception via spectrum analyzers and software-defined radios (HackRF, RTL-SDR); protocol replay; signal manipulation across Bluetooth, Zigbee, and Wi-Fi bands	Confidentiality, Integrity	WPA3-SAE enforcement, RF emission baseline capture, wireless micro-segmentation
Hardware: reverse engineering and micro-probing	Physical decapsulation, side-channel analysis, micro-probing of integrated circuits to extract keys or firmware	Confidentiality, Integrity	LUKS2 full-disk encryption, GPIO hardening, tamper detection
Hardware: implants and trojans	Malicious circuitry inserted during IC design or fabrication; supply chain modification of hardware components	Integrity, Accountability	SBOM validation, cryptographic artifact signing, supply chain provenance checks
Software	Firmware vulnerabilities, insecure update channels, embedded software flaws in microcontroller code	Integrity, Availability, Accountability	Five-modality CI/CD scanning (SAST, DAST, SCA, Secrets, SBOM), automated patching, cryptographic update signing
Network: Wi-Fi and Ethernet	ARP spoofing, packet sniffing, man-in-the-middle on wired and wireless links, rogue access points	Confidentiality, Integrity	Zero Trust micro-segmentation, mTLS between services, continuous network behavioral analysis
Network: wireless sensor networks	Protocol-specific attacks against Zigbee, Z-Wave, and similar low-power mesh protocols; resource-exhaustion against constrained nodes	Availability, Confidentiality	Identity-based access via SPIFFE/SPIRE, least-privilege session policy, behavioral anomaly detection

Continued on next page

Table 3.1 continued from previous page

Attack Category	Representative Vectors	CIA-A Impact	AZTRM-D Defensive Layer
Network: cloud-based	Data injection into cloud endpoints, session hijacking via stolen API keys, resource exhaustion of cloud services, traffic manipulation in transit	Confidentiality, Integrity, Availability, Accountability	RBAC with continuous verification, short-lived SPIFFE tokens, encrypted transport (TLS), API rate limiting, AI-driven anomaly detection

3.1.1 Hardware Vulnerabilities

Most, if not all, IoT devices operate without supervision and typically have limited tamper-resistant properties that make it extremely easy for an attacker to gain access to the device [222]. The attacker can modify the IoT device with respect to its services that it provides, as well as obtain data that it should not have access to and that could cause serious harm to many individuals [222]. For example, if a hacker were to gain access to a smart doorbell, it would be able to modify all the settings inside, view the data inside, or even delete the data inside. Consider a more delicate environment. If an attacker were to gain access to a camera on a military base, they could trace where the data are being sent. In this case, they would find a server with a lot of other camera data on it. They could then disable the cameras or exfiltrate sensitive footage captured by them.

Radio Frequency Attacks

Hardware-level access to a device can come through several methods, and man-in-the-middle attacks are the most common starting point. The usual framing of MITM attacks focuses on the network layer, where an attacker intercepts data on a wired or wireless connection. That framing misses a more direct attack surface. Every electronic device emits RF signals as a byproduct of normal operation, and those emissions are observable with spectrum analyzers or software-defined radios such

as HackRF, Ettus Research devices, or sub-$30 Real-Time Linux Software-Defined Radio (RTL-SDR) units. An attacker who can observe RF emissions can monitor and manipulate device communication directly, without ever touching the network.

Signal identification is the first step. Every device emits RF signals with characteristic attributes the attacker can analyze: bandwidth (the signal's spectral footprint), frequency (its position, such as 2.4 GHz for Bluetooth or 5 GHz for Wi-Fi), signal strength (amplitude), and behavioral patterns (frequency hopping, intermittent bursts, or continuous transmission). The waveform shape on a spectrum analyzer (sharp pulse versus modulated curve) further narrows the identification. From these attributes the attacker can classify the signal as Bluetooth, Zigbee, or a proprietary protocol and assess its vulnerabilities. Wireless protocols routinely have exploitable weaknesses: inadequate encryption, predictable timing intervals, or weak handshake mechanisms. From there the attacker can disrupt the signal, take control of it, or modify its content.

Strong encryption is the first line of defense here. AES-128/256-bit encryption and elliptic curve cryptography (ECC) mean that even if an attacker intercepts a signal, the contents are unreadable [4]. Beyond encryption itself, techniques such as rolling codes and nonce-based authentication prevent replay attacks by ensuring that previously captured transmissions cannot be reused. Secure key exchange protocols such as Diffie–Hellman or Elliptic Curve Diffie-Hellman (ECDH) should also be enforced to make sure that even intercepted communications remain indecipherable [258, 277].

RF-based MITM attacks go beyond simple disruption. The attacker positions themselves between two devices and relays messages while monitoring the exchange, or impersonates one device entirely and redirects the signal to themselves. From there they can forward data to the intended recipient, modify it, or hold it for later use. Malicious commands can be injected, transmitted data altered, or traffic sim-

ply eavesdropped, all without either device detecting the intrusion. RF makes this difficult to spot: signals continue in their expected patterns with no visible trace of interference. Without dedicated RF monitoring through a Software-Defined Radio (SDR), the attack remains invisible [23, 316, 14].

To mitigate these attacks, frequency hopping spread spectrum (FHSS) and direct-sequence spread spectrum (DSSS) techniques can be used to prevent an attacker from isolating a specific transmission. These methods make the signal dynamically shift across multiple frequencies, making it much harder for an adversary to pinpoint, capture, or manipulate communications. Enforcing strict firmware update policies also addresses known vulnerabilities in wireless protocols, preventing attackers from exploiting outdated security measures [300, 80, 325].

The Bluetooth MITM attack in Figure 3.2 illustrates the general principle. Each wireless protocol has distinct characteristics, but the underlying RF exploitation pattern is similar. The attack unfolds through one of two strategies. The attacker can deploy a wide-band jamming signal to overwhelm the Bluetooth frequency band (approximately 2.4 GHz), severing the connection between paired devices. Alternatively, they can target a specific device and flood its RF time slots with randomized data to destabilize its communication channels. The latter approach is intended to "shut down all of the piconets within the range susceptibility", forcing the user to re-pair [124]. During re-pairing, the attacker intercepts the exchange and forges messages within the input/output (IO) capabilities negotiation phase [124]. If the devices were previously paired, the attacker can bypass the re-pairing step entirely and exploit the established link.

Once inserted into the communication chain, the attacker establishes simultaneous connections to both devices, as illustrated in Figure 3.2. Acting as a relay, they forward messages between the pair while retaining the ability to monitor, modify, or inject data [124]. From that position they can alter audio streams, introduce unau-

thorized commands, or extract sensitive information, all within the RF domain. The attack works because the devices continue their normal exchange without detecting the interposed relay. The subtlety is the point: no protocol-level alarm fires, and the attacker remains effectively invisible at the RF layer.

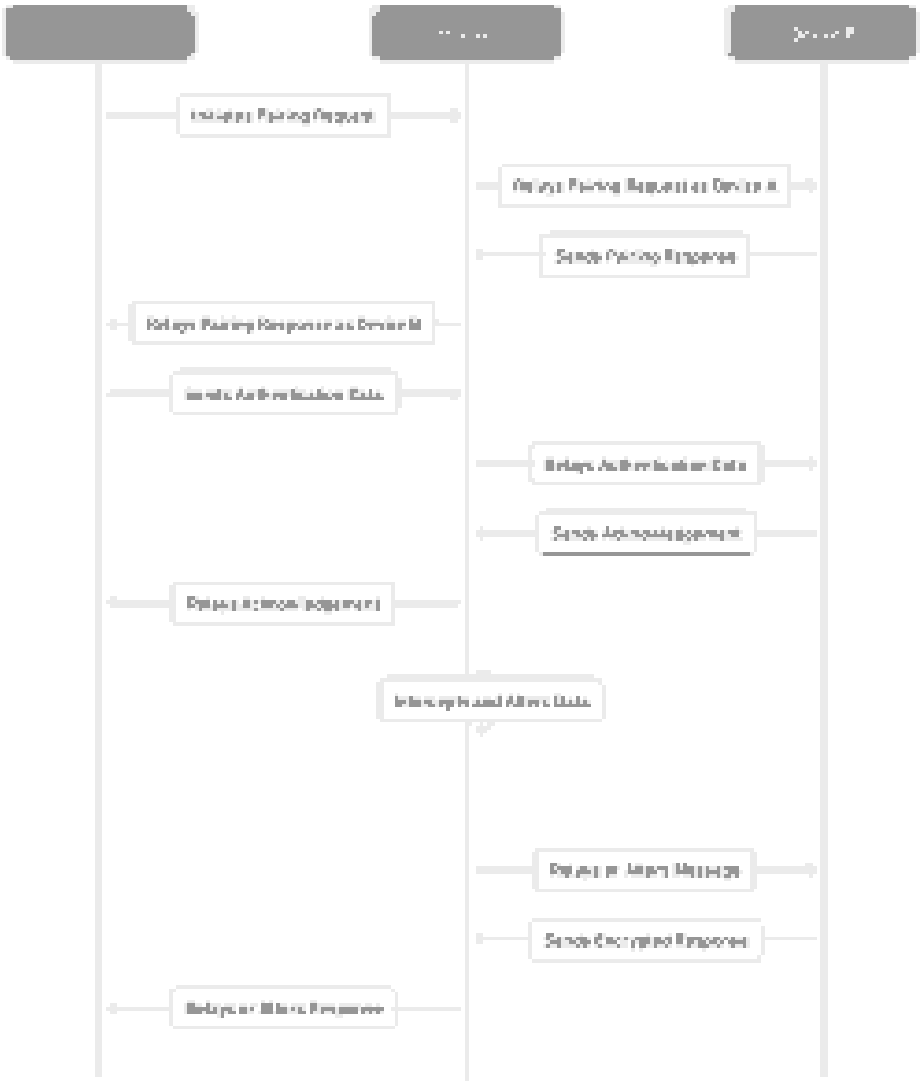


Figure 3.2: Bluetooth man-in-the-middle attack. This image was inspired by [124].

Defending against Bluetooth Man-in-the-Middle (MITM) attacks requires layered countermeasures. Jam-resistant protocols that use spread spectrum communication can prevent attackers from effectively disrupting Bluetooth signals, while adaptive power control can counteract jamming by adjusting signal strength dynamically. Se-

cure pairing mechanisms such as Just Works Numeric Comparison or Passkey Entry, rather than legacy pairing methods, block adversaries from intercepting the pairing process under modern protocol versions [104]. Proximity-based authentication methods, which require physical confirmation before establishing a trusted connection, can further prevent unauthorized devices from inserting themselves into the communication stream. Intrusion detection systems (IDS) designed to monitor RF spectrum anomalies can be deployed to detect unexpected transmission patterns or frequency manipulations indicative of an attack. Finally, physical-layer security upgrades, such as the use of directional antennas, shielded enclosures, and beamforming techniques, can limit RF signal exposure, making it much harder for attackers to intercept or manipulate communications.

Hardware Reverse Engineering and Micro-Probing

Reverse engineering aims to understand how a device operates [250]. Applied to hardware, it exposes the device's architecture along with its strengths, vulnerabilities, and operational details. These attacks fall into three categories (invasive, semi-invasive, and non-invasive), shown in Figure 3.3 [252]. Each varies in methodology, in impact on the target device, and in what kind of information the attacker can extract.

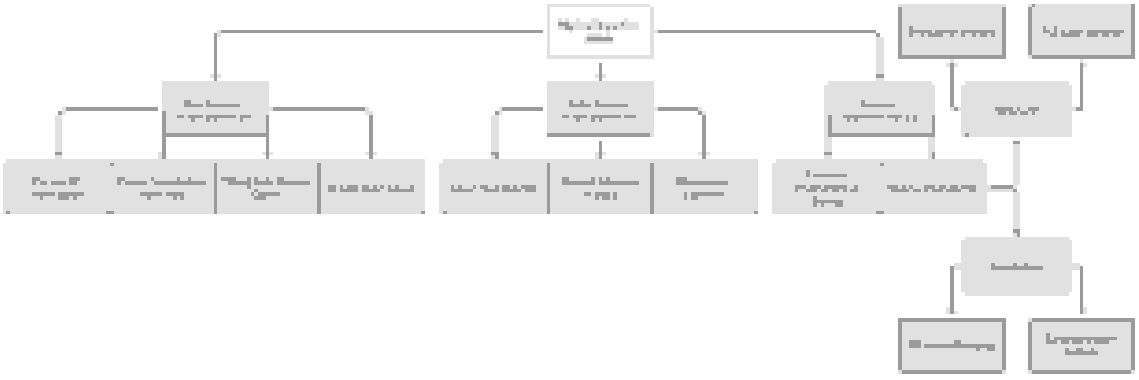


Figure 3.3: Reverse engineering and inspection attacks. This image was inspired by [252].

Non-invasive attacks target data extraction without altering the physical packaging or structure of the integrated circuit (IC) or printed circuit board (PCB). They can be passive (observing emissions or behavior) or active, through deliberate manipulation [252]. Examples include brute-force assaults and fault injection. A representative example is the voltage glitch attack, where a precisely timed voltage spike is delivered to a specific circuit segment, often during boot, to bypass security measures or extract data [332]. RF emissions matter here too. A device under stress can leak electromagnetic signals that reveal timing patterns or internal states when monitored with a spectrum analyzer or SDR. These techniques leave the device intact and produce useful information about its operation without obvious physical evidence.

The defense against non-invasive attacks comes down to reducing what leaks out. Randomized execution timing, power analysis countermeasures, and shielding around sensitive components can help reduce the information attackers can glean from passive observations. Glitch detection circuits should be integrated to recognize abnormal voltage fluctuations and halt execution when suspicious behavior is detected. Hardware-based encryption with secure key storage makes captured electromagnetic emissions indecipherable without the key.

Invasive and semi-invasive attacks require direct access to the IC or PCB's internal components and typically leave physical evidence [252]. Invasive methods involve direct physical interaction with the hardware: soldering wires to contact points, probing electrical buses, or mechanically dissecting chips to expose internal structures. Semi-invasive approaches use optical techniques, including microscopy, laser-based fault injection, or optical probing, to interact with the device at a microscopic level without fully dismantling it [252]. The toolkit is extensive: JTagulators, logic analyzers, oscilloscopes, signal generators, power supplies, X-ray machines, and lasers [252]. RF considerations apply here as well. Probing a chip's internal buses can induce unintended RF emissions, which an attacker can capture to map signal pathways or

decode data flows. These methods frequently damage the device, but they produce direct access to its internal operation [45]. The trade-off is straightforward: deeper intrusion yields more information at the cost of stealth.

Invasive and semi-invasive attacks require physical-layer defenses. Anti-tamper coatings, active mesh defenses, and sensor-based intrusion detection can detect and respond to physical interference. Chip-level encryption with secure boot mechanisms keeps chip contents protected even when the chip is physically accessed. Light sensors, power fluctuation detectors, and tamper-evident enclosures can be employed to thwart optical probing and laser fault injections. Fusing security-critical components after manufacturing can prevent reverse engineering by making chip-level access impractical [317].

Implants and Hardware Trojans

A hardware Trojan is a malicious alteration or addition to the circuitry of an IC, introduced during its design or fabrication phases. This class of attack is unusually difficult to mitigate preemptively because the IC development pipeline lacks reliable security controls at each stage (architectural design, synthesis, or physical layout). The timing of insertion is also unpredictable, making full prevention difficult [238, 161, 341, 160]. The modification stays concealed until activation, which is what makes hardware Trojans a persistent threat to device integrity.

The reason Trojans are dangerous is that they operate undetected, leaving the user unaware of any compromise. The threat applies to any device, regardless of purpose or complexity. An attacker with a Trojan in place can manipulate inputs and outputs, extract sensitive information, or take full control of the device without triggering normal detection. Activation behavior varies. Some Trojans activate autonomously upon deployment; others stay dormant until specific conditions are met, as illustrated in Figure 3.4 [238, 161, 160]. The trigger condition (time-based, signal-driven, or

environmentally triggered) further complicates detection and response.

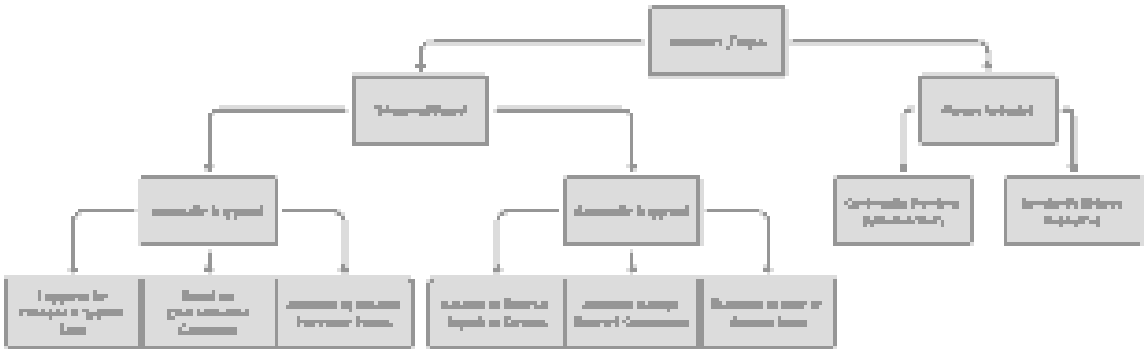


Figure 3.4: Trojan activation modes. This image was inspired by [161].

Mitigating hardware Trojans requires a multilayered approach spanning design-time verification, manufacturing oversight, and runtime monitoring. Implementing formal verification techniques alongside logic obfuscation during the IC design phase can sharply increase resistance to Trojan insertion [2, 3]. Side-channel analysis, such as power or electromagnetic profiling, can also identify anomalies indicative of unauthorized modifications. At the manufacturing stage, functional testing with hardware fingerprinting can help detect deviations from expected behavior. Post-deployment, dynamic runtime monitoring with anomaly detection algorithms can identify unauthorized data leakage or unexpected device behavior, helping neutralize threats that evade initial scrutiny.

Hardware implants are a broader category of malicious modifications. Unlike Trojans, which are limited to the design or fabrication stages, implants can be introduced at any point in a device's lifecycle, sometimes years after deployment. Implants are typically discrete components or circuits added to the hardware, and they have similar capabilities to Trojans: data leakage, operational interference, and full control. Figure 3.5 shows the workflow of a simple hardware Trojan, but the same principles apply to implants. The post-deployment insertion window is what makes them particularly difficult to defend against. A device in active use, carrying years of accu-

mulated data, can be modified without the owner's knowledge. Once the implant is in place, the attacker has direct access to stored information and ongoing operations, and the implant evades standard detection methods. Unlike IC-stage Trojans, implants cannot be addressed by tightening controls during manufacturing alone, since the modification happens later in the device's operational life.

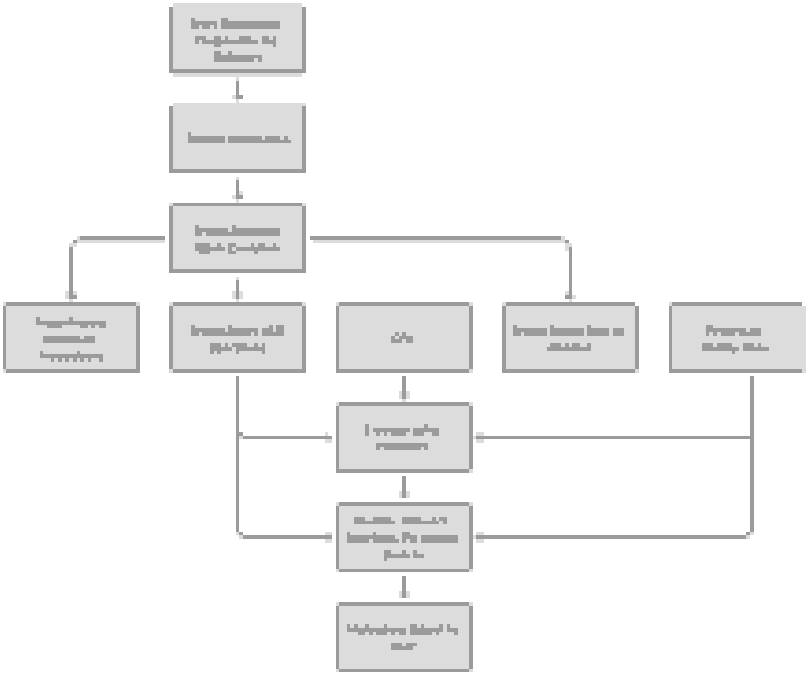


Figure 3.5: Working process of a simple hardware Trojan. This image was inspired by [161].

Mitigating the risks posed by hardware implants necessitates both physical security measures and continuous integrity monitoring. Device manufacturers should enforce supply chain security protocols to prevent unauthorized hardware modifications at any stage of production or distribution. Tamper-evident enclosures and active intrusion detection mechanisms can alert users to unauthorized physical modifications. Periodic integrity verifications through hardware attestation catches any post-deployment alterations before they can compromise system security. Secure boot mechanisms combined with hardware-based root-of-trust validation can further

reinforce a device's resistance against implanted threats [153].

3.1.2 Software Vulnerabilities

IoT device functionality depends not just on hardware but on the software running inside the microcontroller [92]. Consider three identical IoT devices, each with a Bluetooth module and indistinguishable in appearance. Their behavior comes entirely from the firmware. One might transmit Bluetooth signals exclusively, another receive signals only from authorized users, and a third toggle between roles based on context. That software-driven flexibility is what makes IoT adaptable, and it is also what makes the firmware the primary attack surface. When the software is compromised through exposure, modification, or exploitation, the device becomes a vehicle for data breaches, malicious implants, or full attacker control, often without any observable change in behavior.

Software vulnerabilities in IoT devices take many forms, each presenting distinct pathways for exploitation [31]. A prevalent type is the buffer overflow, where poorly managed memory allocation allows an attacker to input data beyond a designated buffer's capacity [150]. This excess data can overwrite adjacent memory, enabling the injection of malicious code or the alteration of program execution flow. For an IoT device, this might mean hijacking a firmware update process to execute unauthorized commands. Mitigating buffer overflow vulnerabilities requires secure coding practices and memory protection mechanisms [281]. Developers should employ bounds checking and input validation to prevent excess data from overflowing memory buffers. The use of stack canaries (small security values placed on the stack to detect corruption) can help identify and block exploit attempts [150]. Firmware updates require cryptographic signing and verification, blocking execution of unauthorized modifications.

Another common vulnerability is weak authentication, where inadequate or hard-coded credentials (often a default username and password like "admin/admin") grant

attackers effortless entry [29]. Once inside, they can manipulate the device's operations or siphon sensitive data, such as pairing keys for a Bluetooth connection. To counteract weak authentication, IoT devices should enforce strong password policies, mandating unique and complex credentials upon first use. Multi-factor authentication should be incorporated where feasible, adding an additional layer of security beyond just a password. Device manufacturers should also eliminate hardcoded credentials and instead implement dynamic, per-device authentication keys to prevent widespread exploitation from leaked credentials.

Insecure communication is another major risk. Unencrypted or poorly encrypted data transmissions (plaintext Bluetooth packets, for instance) expose information to interception. An attacker with a software-defined radio could eavesdrop these signals, reconstructing commands or user inputs without ever touching the device. Mitigating insecure communication requires end-to-end encryption using strong cryptographic protocols such as AES for data transmission and TLS for network communications [183]. Devices should avoid transmitting sensitive data in plaintext and implement secure key exchange mechanisms to protect encryption keys from interception. Frequency-hopping spread spectrum techniques in wireless protocols can also make passive eavesdropping much harder.

Physical access amplifies all of these risks. An attacker with physical access to the device can connect it to a computer or probe its pins to extract firmware from the microcontroller. In the best case for the attacker, this yields source code and configuration files, which exposes the device's full logic. More often, the attacker recovers only the machine code (the compiled binary). That isn't a dead end. Tools like Ghidra, Interactive Disassembler (IDA) Pro, or a hexadecimal editor allow the attacker to disassemble the binary and trace its instructions to map device functionality, from signal handling to security checks [109]. A survey of IoT and embedded device firmware security catalogs the full landscape of firmware extraction techniques and

the vulnerability analysis frameworks used to interrogate the resulting binaries [318]. Sophisticated decompilation can convert the binary into editable C code, which opens the door to targeted modifications. From there the attacker can identify operational weaknesses, hidden features, or behaviors detrimental to the device's owner, such as a tendency to broadcast identifiable data in plaintext. Mitigations include firmware encryption and readout protection mechanisms to prevent unauthorized access to microcontroller memory, secure boot processes to verify that only authenticated and untampered firmware can execute on the device, and code obfuscation to make reverse engineering harder by disguising critical logic and security mechanisms.

Physical possession of the target device isn't always necessary. IoT devices are often inexpensive and partially open-source, which makes indirect exploitation practical. An attacker can buy an identical unit, extract its hexadecimal data, and reverse-engineer the firmware to understand the target's behavior. This won't reveal user-specific configuration (custom encryption keys, for example), but it does expose the core software architecture. From that understanding the attacker can craft malicious modifications: backdoors for monitoring, logic for remote control, or stealth routines that hide their presence. These changes often need only a handful of code lines and can be re-uploaded quickly. Mitigations include secure firmware update mechanisms that require updates to be cryptographically signed, verified, and authenticated before installation, rollback protection to prevent downgrades to older vulnerable firmware versions, and behavioral anomaly detection that identifies when a device starts behaving differently from its baseline.

Other vulnerability classes include code injection (for example, malformed inputs to a web interface) and privilege escalation (where an attacker exploits flawed permission checks), both of which can escalate limited access into full control [229]. In each case the same property that gives IoT its flexibility (programmable, updatable software) is what creates the exposure.

3.1.3 Network Vulnerabilities

IoT networks split into two categories that get treated very differently in the security literature. The first is the standard Internet-connected network running on WiFi and Ethernet, with direct routes to the public Internet. The second is the wireless sensor network category, which includes Bluetooth, Zigbee, UWB, and other short-range protocols, and which connects devices to each other rather than directly to the Internet. The two categories share many threat patterns but differ in attack surface, defender visibility, and the controls that apply. The subsections that follow examine each in turn.

WiFi and Ethernet Based Networks

WiFi- and Ethernet-based networks form the backbone of IoT connectivity, linking devices through cloud platforms or local home networks. The architecture enables remote access to IoT devices from anywhere with internet connectivity, such as controlling a smart thermostat from a different country. That same reach is also the central vulnerability: the broader the network's exposure, the more opportunity adversaries have to intercept or disrupt communications. Among the most accessible threats are Denial of Service (DoS), Distributed Denial of Service (DDoS), and Energy-Oriented Distributed Denial of Service (E-DDoS) attacks, each exploiting the openness of these networks in different ways [315].

A typical DDoS attack, illustrated in Figure 3.6, starts with the attacker identifying the target device's IP address within the network. From there, they send an overwhelming volume of meaningless packets to saturate the device's bandwidth. The result is that the network slows, resources divert to processing the flood, and the system collapses, forcing the device offline [26]. E-DDoS attacks pursue a different goal. Rather than crashing the network, these target the device's power reserves, potentially damaging a PCB component or inflating energy costs, representing a variant

of the DoS model [315]. Both share the same scattershot character, often targeting entire device clusters rather than individual units.

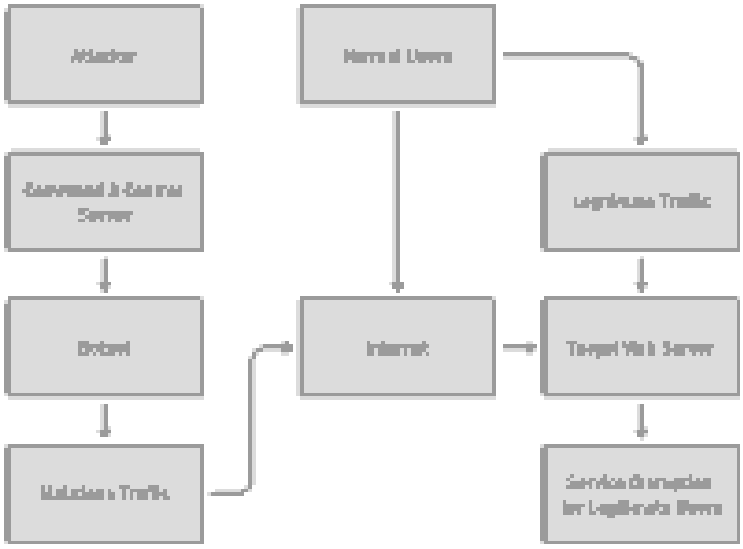


Figure 3.6: Distributed Denial of Service (DDoS) attack. This image was inspired by [26].

Mitigating DoS and DDoS attacks requires network traffic filtering and rate limiting to detect and block malicious traffic patterns before they overwhelm a system. IDS can help identify anomalous spikes in traffic, while blacklisting and rate-limiting strategies can drop excess requests from untrusted sources. To counteract E-DDoS, IoT devices should incorporate power-aware intrusion detection, allowing them to recognize and mitigate excessive resource consumption attempts. Network segmentation can also isolate critical devices from direct exposure, reducing the blast radius so a single attack is less likely to cripple an entire IoT deployment.

DoS-style attacks split into three categories that exploit different network layers, shown in Figure 3.7. Volume-based attacks flood protocols like TCP, ICMP, or UDP with raw data, clogging bandwidth and overwhelming internal queues. Protocol-based attacks such as Synchronize (SYN) floods, Ping of Death, or Smurf attacks target resource allocation. A SYN flood ties up connections by leaving them half-open; a Ping

of Death delivers oversized packets to destabilize servers; a Smurf attack amplifies traffic by echoing requests across all network nodes [315]. Application-layer attacks such as Slowloris, HTTP floods, or SMTP exploits target server-side application logic. Slowloris sustains partial HTTP requests to monopolize server connections; HTTP floods swamp applications with legitimate-looking queries; SMTP attacks probe mail servers for data leaks [315]. All three exploit the same property: protocol mechanisms that work correctly under normal traffic break down under adversarial load.

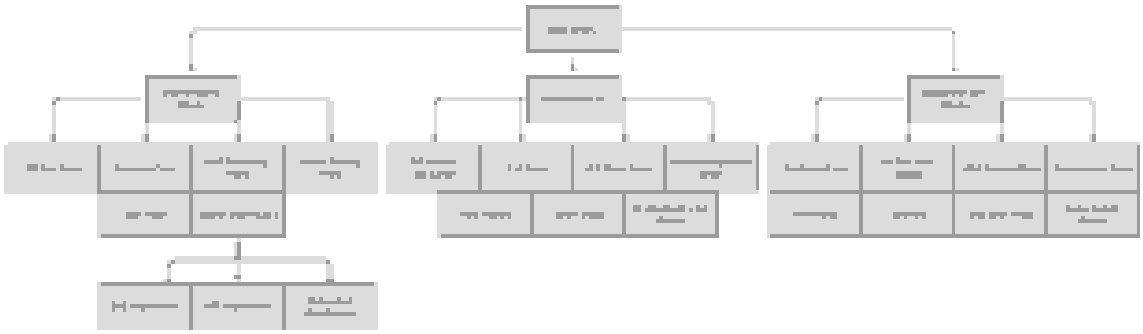


Figure 3.7: Types of attacks. This image was inspired by [315].

DoS variants are pervasive but not the only network threats. Traffic monitoring attacks like WiFi fingerprinting work more passively. By analyzing a building's camera system bit-rate over WiFi, an attacker can infer internal movements (occupancy tracking, for example) without any direct access [18]. The granularity is limited, but the inferences are enough to identify device roles or activity patterns. More aggressive packet analysis attacks use tools like Wireshark [330] or air sniffers to capture TCP/IP traffic. These expose communication patterns (who is talking, what is being sent) and, if the traffic is unencrypted, the payload itself, such as a smart bulb's status update [18]. From the captured packets, the attacker can extract detailed information about the network's internal operation.

The fix for traffic monitoring is end-to-end encryption on everything. Protocols like TLS for web traffic, WPA3 for WiFi security, and AES-based encryption for device communication can prevent eavesdropping. Implementing Media Access Control

(MAC) address randomization can also reduce device fingerprinting risks. Firewalls with deep packet inspection (DPI) can monitor and block untrusted packets before they reach their destination.

Figure 3.8 outlines a sophisticated over-the-air attack, unfolding in two stages. The offline phase involves capturing traffic with a sniffer, preprocessing it and extracting features (packet intervals, sizes, or signal traits). These feed a machine learning model to profile network behavior and predict device types. In the online phase, the attacker actively eavesdrops, identifying live devices and cataloging WiFi-enabled units in range [18]. This methodical approach yields a detailed inventory without tripping alarms. Then there are network injection attacks, where tools like Kali Linux or a WiFi Pineapple inject spoofed packets or crack encryption (like brute-forcing WPA2 keys). MITM attacks take it further, intercepting traffic to relay doctored messages or reroute data through malicious nodes, like a rogue Domain Name System (DNS) server [315].

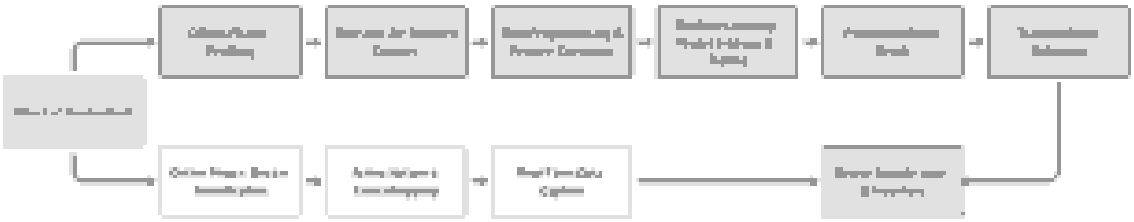


Figure 3.8: WiFi over-the-air attack. This image was inspired by [18].

Over-the-air attacks need mutual authentication at the device level. 802.1X authentication with certificate-based security restricts network participation to verified devices in the network [108]. Regular key rotation for encrypted sessions prevents long-term eavesdropping. For MITM attacks, secure Domain Name System Security Extensions (DNSSEC) and certificate pinning can validate legitimate servers, preventing redirection to malicious endpoints [131]. Enforcing firewall-based network segmentation can also reduce the spread of malicious injections.

The toolkit for these attacks is wide and inexpensive. Open-source software (Wireshark for analysis, Aircrack-ng for cracking, custom injectors) pairs with hardware like Raspberry Pi sniffers or Arduino jammers. Purpose-built devices like the WiFi Pineapple simplify execution, with online guides walking through their construction [18]. The pace of new exploit development is part of the difficulty: as defenses tighten, attackers adapt. Strong encryption, active monitoring, intrusion detection, and network segmentation are the controls that meaningfully raise the cost of these attacks against IoT deployments.

Wireless Sensor Networks

A wireless sensor network (WSN) is a set of spatially distributed sensors that transmit data wirelessly, and these networks form the backbone of many IoT deployments. They use a range of protocols (Zigbee, Z-Wave, Global Positioning System (GPS), 5G, and others), each suited to specific sensing and communication requirements. That protocol diversity is part of what gives IoT its breadth, and it also expands the attack surface. This work separates WiFi and Ethernet networks from WSNs because the relevant vulnerability classes are different. Within WSNs, Denial of Service (DoS) attacks are a particular concern, and Figure 3.9 lays out the taxonomy. The WSN-specific DoS taxonomy goes beyond what's seen in conventional networking.

Figure 3.9 shows the DoS framework specific to WSNs, including WiFi and Ethernet threats and several exploits unique to the WSN model. At the physical layer, jamming and node tampering are the most common attacks. An adversary can jam a Zigbee network with interference to block nodes from transmitting, or physically compromise a sensor to extract or alter its firmware [290]. Inserting a malicious node compounds this, severing legitimate connections by drowning out their signals. FHSS and Direct-Sequence Spread Spectrum (DSSS) raise the cost of effective signal disruption [283, 141]. Tamper-resistant hardware with physical shielding reduces the

risk of unauthorized sensor access.

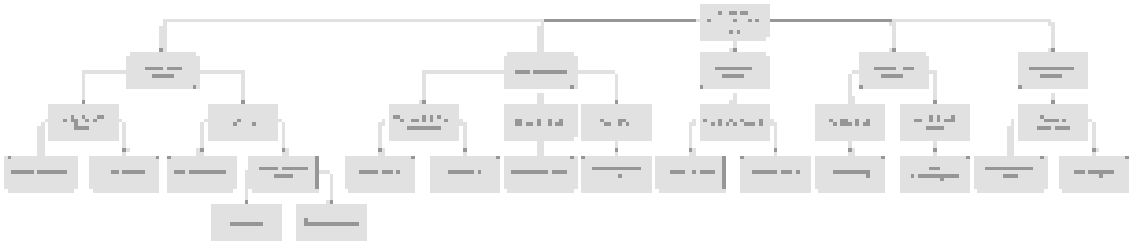


Figure 3.9: Wireless sensor networks DoS attacks. This image was inspired by [290].

Link-layer attacks operate less visibly: a rogue node can masquerade as trustworthy, exhaust neighbors with continuous requests, or induce collisions that crash nodes through error handling [290]. At the network layer, flooding parallels WiFi-style floods, overwhelming routing paths with traffic. The transport layer is subject to flooding (swamping nodes with fake peers) and desynchronization, where message timing is scrambled and nodes fall out of sync [290]. These attacks can be mitigated through rate limiting, packet authentication, and anomaly detection algorithms that identify excessive traffic patterns or suspicious communication behavior.

Beyond DoS, WSNs face routing-centric threats like the black hole attack, depicted in Figure 3.10. Here, a malicious node lures others to redirect data through it, acting as a sink that absorbs (or discards) transmissions, halting network flow [286, 165]. Closely related to this is the wormhole attack, shown in Figure 3.11, where data are tunneled between two colluding nodes (say, from node A to node B), bypassing the intended routes [308, 321]. This rerouting distorts network topology, granting attackers control over data paths.

To mitigate black hole and wormhole attacks, secure routing protocols such as Secure Efficient Ad hoc Distance vector (SEAD) and Ad hoc On-Demand Distance Vector (AODV) with authentication mechanisms can help verify node identities before allowing data redirection [137, 270, 147]. Time-based packet verification methods can also be implemented to detect unusual delays indicative of wormhole behavior.

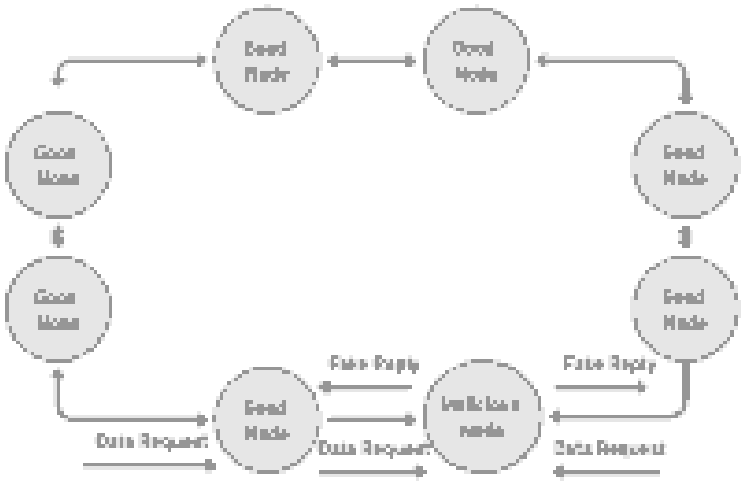


Figure 3.10: Blackhole attack. This image was inspired by [13].

Figure 3.11 illustrates how wormhole nodes (A and B) tunnel data between themselves to manipulate its flow. The attack has four variants. Packet encapsulation compresses data between nodes, evading the hop-count increments WSN routing depends on [321]. Packet relay mode uses any node as a launch point, relaying traffic to disrupt paths [321]. Out-of-band channel attacks rely on a single high-power node to redirect packets via an external link [321]. Protocol distortion mode alters routing rules to attract traffic to malicious nodes [321]. Each variant exploits the same architectural assumption: that WSNs depend on decentralized routing decisions made under the assumption that participating nodes are honest.

Eavesdropping attacks use prebuilt sniffers for UWB, Zigbee, Z-Wave, or Bluetooth to capture packets over the air. Open-source tools for Arduino or Raspberry Pi extend this capability to additional protocols, decoding transmissions to recover GPS coordinates or sensor readings [290]. Sybil attacks introduce fake node identities and trick the network into routing through them; sinkhole attacks lure traffic to a compromised node where it can be manipulated or discarded [286]. Traffic analysis attacks operate at a different layer, where signal patterns (such as a 5G node's burst rate) hint at activity without requiring the attacker to decrypt payloads [308]. All of

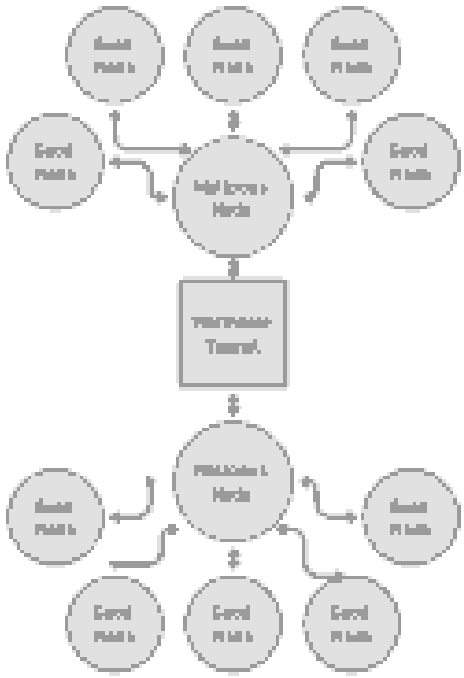


Figure 3.11: Wormhole attack tunnel. This image was inspired by [321].

these exploit the same property of wireless: traffic is observable in the medium itself, independent of the network protocols above it.

To mitigate these advanced threats, encrypted communication protocols such as AES-based encryption for data packets should be enforced. Identity-based authentication can prevent Sybil attacks by requiring node verification. For eavesdropping threats, regularly changing encryption keys and employing physical-layer security techniques such as directional antennas and shielding can reduce interception risks.

The tools for these exploits mirror what's available in the WiFi space: commercial sniffers are widely sold, and Do-It-Yourself (DIY) options run on open-source code. An attacker can use a Zigbee sniffer to log packets or assemble a Raspberry Pi rig to jam Z-Wave signals. That accessibility, paired with the WSN attack surface, means new vulnerabilities and new defenses keep emerging in roughly equal measure. Encryption, anomaly detection, secure routing, and frequency-hopping mechanisms substantially raise the bar against these attacks, but they don't eliminate the under-

lying exposure inherent in wireless sensor topologies.

Cloud Based Networks

Cloud computing has become a structural part of IoT network security. Standalone IoT devices typically can't meet user demand for compute resources, so cloud integration is the practical solution. Pairing IoT sensing with cloud scalability enables capabilities like remote thermostat control from across the globe, shown in Figure 3.12. The same architecture expands the attack surface. This section focuses on the Phantom-Delay Attack and related threats specific to this hybrid environment.

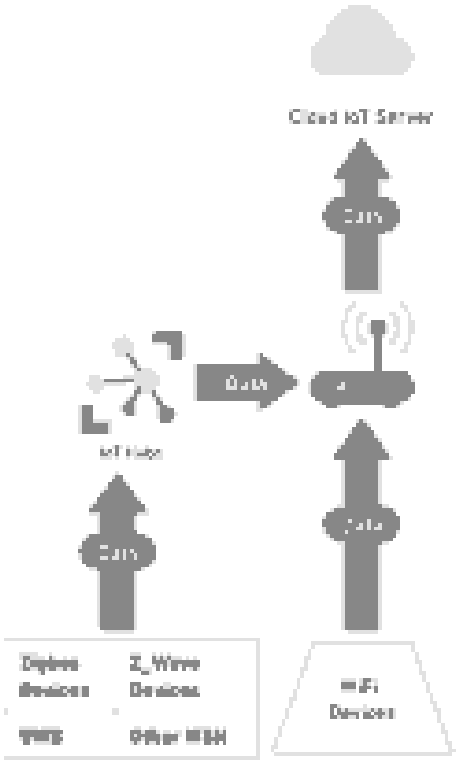


Figure 3.12: System model of a typical smart cloud-based home deployment. This image was inspired by [101].

Phantom-delay attacks are a recently uncovered vulnerability class that exploits timing in communications between IoT devices and cloud servers. Unlike conventional disruption attacks, phantom-delay attacks manipulate message delivery without dis-

carding packets, which is what makes the consequences [101] difficult to detect. The attack uses two primitives. IoT Event Message Delay (e-Delay) delays device-to-server state updates (for example, a "motion detected" alert), and IoT Command Message Delay (c-Delay) stalls server-to-device instructions (for example, "lock the door"). Transmission delays are normally sub-second and don't pose a problem. A phantom-delay attack, illustrated in Figure 3.13, extends those delays into minutes or hours. The cloud server receives a delayed "motion active" event, treats it as the current state, and acts on outdated information [200].

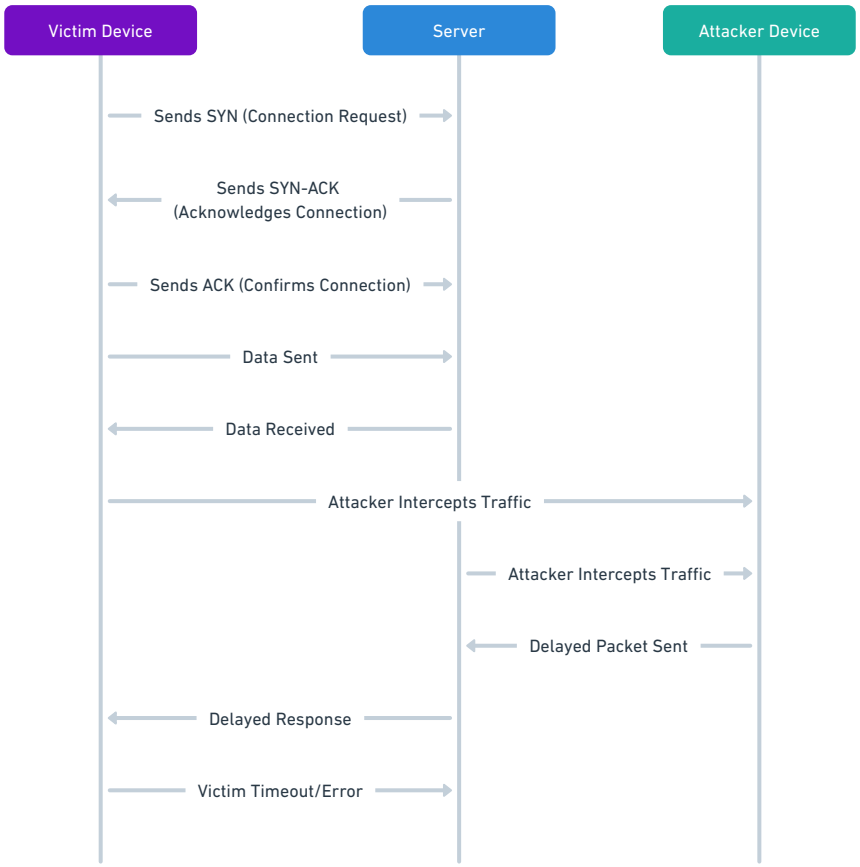


Figure 3.13: Phantom-delay attack. This image was inspired by [101].

These attacks take three forms. The state-update delay attack uses e-Delay to stall critical state reports, which can delay urgent reporting from medical IoT devices (IoMT) or agricultural sensors and prevent timely intervention [101]. The action-

delay attack disrupts automation; an e-Delay can postpone an insulin pump's response to a glucose spike detected by a continuous monitor, with direct health implications. The erroneous execution attack has two subtypes. Spurious execution holds a delayed event as "true" past its validity, which triggers unneeded actions (a sprinkler activating despite rain, for example). Disabled execution holds a condition as "false," which prevents necessary actions (a door failing to lock despite a command) [200]. Unlike jamming, these attacks (typically launched from a WiFi device sniffing traffic and hijacking TCP sessions via Address Resolution Protocol (ARP) spoofing) leave no trace of dropped packets and don't trigger retransmission flags [327].

Catching phantom-delay attacks means checking timestamps on every message exchange. If data arrives outside its expected timing window, it gets flagged as stale. Secure time synchronization protocols, such as the Network Time Protocol (NTP) with cryptographic authentication, can help detect discrepancies in the expected message's timing. Introducing anomaly detection mechanisms that monitor latency variations can preemptively identify these delays, prompting system alerts or requiring revalidation before acting on stale data [105, 342, 32].

Beyond phantom delay, several other cloud-targeted attacks apply. Data injection lets an adversary feed falsified inputs (spoofed temperature readings, for example) into the cloud and trigger incorrect decisions, such as shutting down a smart Heating, Ventilation, and Air Conditioning (HVAC) system [101]. Session hijacking exploits weak authentication: unsecured API keys or stolen credentials let attackers impersonate devices and issue rogue commands (unlocking a smart door remotely, for example). Resource exhaustion attacks target cloud endpoints by flooding them with false requests to overwhelm processing capacity, which differs from bandwidth-focused DDoS because the target here is server-side application logic [101]. Traffic manipulation is the remaining category, where an attacker intercepts and alters data in transit (modifying a smart meter's usage report to inflate bills, for example) while

the cloud accepts the modified data as legitimate [101].

Data injection gets stopped by cryptographic message authentication codes (MACs), which verify that only legitimate IoT devices can submit data to the cloud. Secure firmware updates with digital signatures help prevent unauthorized modifications that might enable spoofed sensor readings. In mitigating session hijacking, enforcing multi-factor authentication (MFA) for cloud access and regularly rotating API keys reduces exposure to credential theft. Role-based access control (RBAC) minimizes the impact of compromised accounts by restricting device privileges.

For resource exhaustion and traffic manipulation attacks, rate limiting on API endpoints prevents abuse by restricting excessive requests from any single source. Encryption protocols such as TLS prevent in-transit data tampering and maintain communication integrity. IDS equipped with behavioral analysis can identify unusual traffic patterns and trigger automated containment before the attack completes [138, 106].

These vulnerabilities exploit the cloud's continuous IoT interaction model. Common attack tools (WiFi sniffers, Kali Linux distributions, a Raspberry Pi configured for ARP spoofing) are inexpensive and easy to assemble. The cloud's strength (its ability to integrate diverse devices) is the same property that creates the exposure, and any breach has wider consequences because it touches all the connected devices at once. Cloud-IoT integration will only continue, and the security controls have to scale with it: without strong end-to-end protections, the integration architecture itself becomes the weakest link.

The attack surfaces and countermeasures cataloged across the past two chapters define the threat conditions any candidate methodology has to handle [16]. The next chapter steps back from the threats themselves and surveys the existing development methodologies and security frameworks that attempt to address them, identifying where the coverage breaks down and where AZTRM-D's contribution sits relative to

prior work.

CHAPTER 4

RELATED WORK AND FOUNDATIONAL CONCEPTS

To position the contribution of the AZTRM-D framework, it helps to first understand the evolution of software development methodologies and their inherent security limitations. This section reviews the trajectory from traditional models to modern, integrated security paradigms. It then presents a comparative analysis of contemporary secure development frameworks to differentiate AZTRM-D's unified approach and address existing gaps in the literature [67].

4.1 SOFTWARE AND SYSTEM DEVELOPMENT LIFECYCLES

The Software Development Lifecycle (SDLC) is a foundational framework guiding the creation and maintenance of software systems through structured phases like requirements analysis, design, implementation, and testing [121, 235]. The choice of SDLC methodology affects a project's efficiency and, critically, its security posture, especially when handling sensitive or classified data [50]. As security requirements become more stringent, development methodologies must adapt to protect sensitive information effectively.

4.1.1 Waterfall Method

The Waterfall methodology follows a linear, sequential structure where each phase must be completed before the next begins [58, 273]. This rigid approach is effective for projects with stable, well-defined requirements, such as simple, unclassified systems [243, 164]. While its emphasis on detailed documentation helps with compliance under standards like Health Insurance Portability and Accountability Act (HIPAA) or

Payment Card Industry Data Security Standard (PCI-DSS), the rigid structure makes it poorly suited for dynamic threat environments that require continuous security reassessment and rapid adaptation [50].

4.1.2 Agile Method

In contrast, the Agile methodology offers a flexible and iterative approach, breaking development into small units called sprints that allow for continuous integration and deployment [243, 322]. This adaptability works well where requirements evolve, such as in consumer IoT development [50]. The rapid pace of Agile development does require strong management and integrated security practices to prevent oversights and maintain compliance throughout the iterative cycles [235].

4.1.3 V-Model

The V-Model, an extension of Waterfall, emphasizes verification and validation by pairing each development phase with a corresponding testing phase [5, 79]. This structured approach works well for projects where reliability and rigorous testing are the top priority. The rigidity is the problem, though. Like Waterfall, it doesn't adapt quickly to new security threats or regulatory changes, making it a poor fit for highly dynamic or classified systems [50].

4.1.4 Spiral Method

The Spiral methodology offers a hybrid approach, merging the iterative nature of Agile with the structure of Waterfall and placing a strong emphasis on risk assessment at each phase [285]. Its iterative cycles of planning, risk analysis, and evaluation make it well-suited for large, complex projects with high levels of uncertainty [50]. For highly classified systems like the Internet of Battlefield Things (IoBT), the Spiral model's intrinsic focus on continuous risk management is invaluable for adapting to changing

mission requirements and threat landscapes.

4.1.5 DevSecOps Method

A common limitation across these foundational models is the tendency to treat security as a separate, often late-stage, concern. This gap led to the rise of Development, Security, and Operations (DevSecOps), an approach that integrates security practices directly into the entire development lifecycle [280, 184]. Through automation, CI/CD, and a culture of shared responsibility, DevSecOps aims to make security an intrinsic component of the software pipeline [347]. Core principles include "Security as Code," where security policies and controls are managed as versioned, auditable code, and continuous monitoring throughout the lifecycle [253]. As illustrated in Figure 4.1, the DevSecOps lifecycle embeds security touchpoints, from threat modeling in the planning phase to security analysis in the feedback phase, across the entire process. While DevSecOps provides a critical cultural and procedural foundation, it does not, by itself, prescribe a specific risk management structure or access control philosophy, leaving room for more complete, unified frameworks.

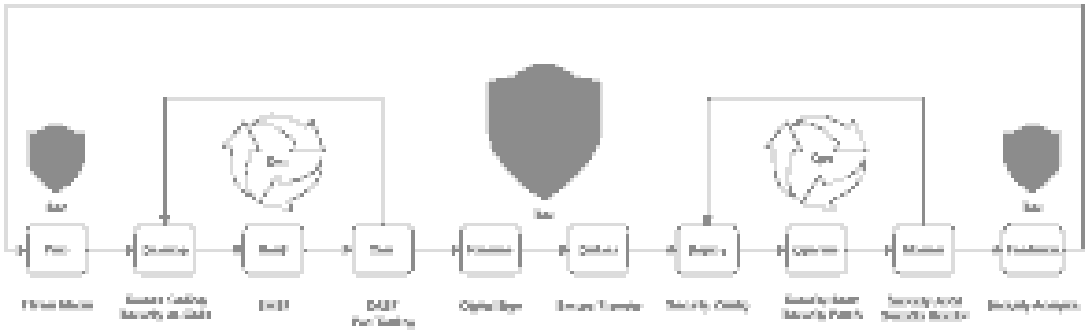


Figure 4.1: The DevSecOps Lifecycle, integrating security into every phase from planning to feedback. This image was inspired by [77].

4.1.6 Secure Development Methodologies and Frameworks: Where Things Stand

Agile methodologies, cloud-native architectures, and continuous delivery pipelines have changed how software gets built. Speed is up. But security hasn't kept pace. Traditional security approaches often fall short against today's threats because they get implemented too late or through fragmented, manual processes [121, 235]. Finding and fixing vulnerabilities late in the development cycle is costly, causes delays, and leaves systems exposed [12]. This situation demands a new approach that integrates security from the start, maintains continuous vigilance, and uses intelligent automation to keep pace with evolving threats.

S-SDLC emerged as a direct response, integrating security into the traditional SDLC [152, 25]. The primary goal of any S-SDLC is to embed security considerations into every stage of development, from requirements gathering to deployment and maintenance. Doing so produces a final product that can withstand evolving threats [59]. To understand where AZTRM-D fits within this context, it is helpful to examine the existing body of S-SDLC methodologies and modern security frameworks.

Table 4.1 provides a concise overview of various prominent S-SDLC methods, detailing their key focus areas and core characteristics.

Table 4.1: S-SDLC Methodologies. Each method sourced from its cited publication.

Method	**Key Focus/Approach**	**Core Characteristics**
Embedded Security in Traditional SDLC [152]	This method integrates security deeply into each phase of the traditional SDLC.	It identifies security requirements early, uses threat modeling for design, prioritizes secure coding and static analysis, employs layered testing, and focuses on continuous monitoring and patch management.

Table 4.1 continued from previous page

Method	Key Focus/Approach	Core Characteristics
Early and Continuous Integration [25]	This approach emphasizes early and continuous security integration through stakeholder involvement and education.	It aligns security requirements with business objectives, involves security experts and stakeholders, builds a culture of security awareness, and integrates automated, continuous security testing throughout all phases.
Agile-Adapted Security [322]	This method integrates security into Agile methodologies, making it well-suited for dynamic and iterative development.	It embeds security activities like threat modeling and reviews directly into Agile sprints. Security testing is performed iteratively, and a "security backlog" helps prioritize and manage tasks.
Common Criteria (CC) Certification Aligned [170]	This approach is tailored to meet the rigorous requirements of CC certification.	It emphasizes aligning with specific CC security controls and extensive documentation. Formal risk management is key, requiring detailed risk assessments (e.g., ISO/IEC 27005) to systematically identify and mitigate threats.
DevSecOps Transformation [249, 144]	This method transforms the DevOps process into a unified DevSecOps approach, ensuring security is a shared responsibility across all stages.	Key features include automated security integration into the CI/CD pipeline, building a collaborative culture between development, operations, and security teams, and using continuous monitoring with real-time feedback for rapid remediation.
Categorized Software Security Strategies [193]	This method is based on a study that categorized software security strategies into five main approaches.	These include Secure Requirements Modeling; Vulnerability Identification via static and dynamic analysis; Software Security-Focus Processes using frameworks like Microsoft SDL; Extended UML-Based Secure Modeling; and Non-UML-Based Secure Modeling.
Proposed AZTRM-D Methodology (This Work) [67]	This is a unified, AI-augmented process that builds upon existing S-SDLC methods.	It integrates proactive DevSecOps, structured NIST RMF, and stringent ZT access controls. A unique aspect is its direct application of ZT principles to the AI systems within the SDLC, ensuring automation tools are continuously verified. This creates a more resilient security framework.

While these S-SDLC approaches provide a solid foundation, recent academic and

industry work has focused on building more specific frameworks. Many modern threats target the CI/CD pipeline, exploiting vulnerabilities in open-source dependencies and Infrastructure as Code (IaC), requiring a defense-in-depth strategy beyond basic code scanning [239]. In response, researchers have proposed AI-enhanced models for specific lifecycle stages. For example, some have developed machine learning approaches to automate threat modeling within DevSecOps, showing AI's potential to proactively identify design flaws [329]. Concurrently, the ZT philosophy has become prominent, with researchers exploring its application beyond network access, extending principles to the supply chain for rigorous verification of all components and suppliers [62]. Others advocate for "Zero Trust by Design," embedding its principles into the very architecture of systems [132].

While these approaches advance software security, they often focus on specific domains. Other researchers have proposed structured AI-driven security models, such as the ANN-ISM model for software development, which provides a valuable reference point for formalizing security processes [151]. However, a gap remains for a unified framework that combines the agile methodology of DevSecOps, the rigorous validation of ZT, and the structured governance of a formal RMF into a single, cohesive lifecycle, all orchestrated by AI. Table 4.2 compares these approaches and highlights AZTRM-D's unique contribution.

Table 4.2: Comparative Analysis of Modern Secure Development Approaches. Each framework sourced from its cited publication; gap analysis is the author's assessment.

Framework/ Approach	**Core Principles**	**AI/Automation Focus**	**Gap Addressed by AZTRM-D**
Standard DevSecOps [347]	Integration of security into CI/CD; automation of security checks; shared responsibility culture.	Automated SAST/DAST scanning; CI/CD pipeline automation.	Lacks a prescribed, formal risk management structure (like RMF) and a guiding access control philosophy (like ZT). Security can still be ad-hoc.
AI-Enhanced Threat Modeling [329]	Using ML/AI to predict and identify design-level security threats early in the lifecycle.	Automated generation of threat models from system artifacts; predictive vulnerability analysis.	Primarily focused on the "left side" (design/planning). Does not extend across the full operational lifecycle, including runtime monitoring and incident response.
Zero Trust Supply Chain [62]	Extending ZT principles ("never trust, always verify") to all third-party software components, dependencies, and build processes.	Automated dependency scanning (SCA); cryptographic verification of software artifacts (signing).	Focuses heavily on supply chain and artifact integrity but does not inherently include continuous operational risk management or a full DevSecOps pipeline structure.

Table 4.2 continued from previous page

Framework/ Approach	Core Principles	AI/Automation Focus	Gap Addressed by AZTRM-D
AZTRM-D (This Work)	Unified integration of DevSecOps, NIST RMF, and ZT principles.	Pervasive AI orchestration across all phases, from AI-driven risk assessment and policy generation to automated ZT enforcement and adaptive threat response.	Provides a single, unified framework that ties together agile development, structured risk governance, and adaptive access control, ensuring security is continuous, risk-based, and context-aware from planning to operations.

4.2 THE RATIONALE FOR AZTRM-D'S INTEGRATED APPROACH

As the comparison in Table 4.2 illustrates, existing advancements tend to address specific facets of the secure development challenge. AZTRM-D is proposed as a solution designed to bridge these gaps. For an organization operating in a high-stakes, regulated environment, a methodology that only focuses on CI/CD automation or threat modeling isn't enough. A resilient security posture, one that holds across multiple attack classes and operational contexts, requires pulling several paradigms together. It needs the methodological agility of DevSecOps, the continuous access control of Zero Trust, and the structured governance of RMF keeping all activities measurable and compliant. Tying these complex domains together requires an engine for scalable and adaptive orchestration, a role fulfilled by Artificial Intelligence [102]. The selection of DevSecOps as the foundational process for AZTRM-D is therefore a strategic decision. Its principles of automation and continuous integration are prerequisites for implementing a dynamic security model at scale. By embedding ZT and

RMF principles within this agile process, and using AI to orchestrate it, AZTRM-D provides an adaptive security framework that is well-suited to the demanding security requirements of modern critical systems.

My proposed AZTRM-D methodology builds upon these foundational S-SDLC methods and modern security frameworks. It builds on them by creating a unified, AI-augmented process that integrates the proactive security measures of DevSecOps, the structured risk management of the NIST RMF, and the stringent access controls of ZT [132, 336]. What sets AZTRM-D apart is its application of ZT principles directly to the AI systems within the SDLC, ensuring that the automation tools themselves are continuously verified. This unified approach creates a more resilient and responsive security framework that addresses the gaps identified in the existing literature.

The three pillars of that framework are introduced together at this level, but each rests on a distinct body of work that defines its operational shape. The next chapter develops the first of them, the NIST Risk Management Framework, and explains how its seven-step process is adapted into AZTRM-D's governance backbone.

CHAPTER 5

UNDERSTANDING THE RISK MANAGEMENT FRAMEWORK

5.1 NIST RISK MANAGEMENT FRAMEWORK

As a foundational component of AZTRM-D, the NIST RMF provides a structured, seven-step process to manage security and privacy risk. The framework is the standard governance process for federal information systems, codified in NIST SP 800-37 Rev. 2, and applies across federal civilian agencies as well as National Security Systems following NSM-8 [297, 312]. The framework is designed to be integrated into the system development lifecycle, making it a natural fit for a DevSecOps approach. It begins with understanding the organizational context and proceeds through system categorization, control selection, implementation, assessment, authorization, and continuous monitoring [297, 211, 185]. With AI integrated at each step, AZTRM-D turns the RMF into a dynamic and highly automated process. Figure 5.1 illustrates the cyclical nature of this framework, where monitoring feeds continuously back into preparation for the next assessment cycle. Table 5.1 summarizes the seven phases, their core activities, and the specific AI augmentation that AZTRM-D applies at each phase. The subsections that follow develop each phase in detail.

Table 5.1: The seven phases of the NIST Risk Management Framework as adapted in AZTRM-D. Core activities are drawn from NIST SP 800-37 Rev. 2; AI augmentations describe the specific automation AZTRM-D applies at each phase.

RMF Phase	**Core Activities**	**AZTRM-D AI Augmentation**
Preparation	Establish roles, responsibilities, governance, and a risk management strategy; identify stakeholders; define continuous monitoring strategy [297, 159]	Analyzes historical data to recommend role-based access controls; monitors organizational activity to assist dynamic responsibility assignment [8, 336]
Categorization	Map system layout, components, data flows, and interconnections; assign impact levels (low, moderate, high) based on data sensitivity [297, 185, 301]	Automates discovery of system interconnections and data flows through network analysis for a more complete environmental view [96, 17]
Selection	Conduct threat assessment, vulnerability assessment, and risk assessment; select baseline security controls calibrated to identified risks [297]	Analyzes threat intelligence in real time and automates vulnerability assessment to produce a more dynamic and accurate risk picture [17, 102]
Implementation	Deploy and integrate selected controls into existing infrastructure; document every configuration change; train personnel on the implemented controls [297]	Automates control deployment and configuration through Infrastructure as Code and "Security as Code" practices, reducing human error [253]
Assessment	Reassess vulnerabilities, threats, and risks under deployed controls; run vulnerability scans, penetration tests, and security audits to verify control effectiveness [297]	Automates continuous control testing and analysis; simulates attack scenarios and provides immediate feedback on control performance [102, 329]
Authorization	Senior official reviews all documentation (risk assessments, control implementations, test results) and formally accepts residual risk before operation [297]	Rapidly analyzes extensive documentation and highlights the highest-risk areas, compressing review time and supporting cATO decisions [185]
Monitoring	Maintain continuous surveillance of system activity, performance, and intrusion indicators; run regular vulnerability scanning; respond swiftly to detected incidents [297, 211]	Real-time anomaly detection on high-volume telemetry; predictive threat analysis ahead of exploitation; continuous adaptive posture maintenance [102, 8]

5.1.1 Preparation Phase

The first step in the NIST RMF, the Preparation Phase, lays the groundwork for effective risk management by establishing clear roles, responsibilities, and procedures [297, 188]. This phase sets the stage for governance by determining who has access to data, establishing a hierarchy for change management, and defining processes for documentation [159]. A critical aspect of this phase is ensuring the organization is prepared to execute the RMF at all levels, which includes establishing a risk management strategy, identifying key stakeholders, and developing a continuous monitoring strategy that aligns with organizational goals [260, 301]. In AZTRM-D, AI augments this phase by analyzing historical data to recommend role-based access controls and by monitoring organizational activities to help dynamically assign responsibilities, minimizing the attack surface from the outset [8, 336].

5.1.2 Categorization Phase

The Categorization Phase involves building a thorough understanding of the system and its environment [297, 188]. This requires carefully analyzing the system layout, mapping how components communicate and exchange data, and identifying all network connections [185]. The priority is to identify critical systems and their interdependencies to understand the potential impact of a security incident [297, 211]. The systems are then assigned impact levels (low, moderate, or high) based on the sensitivity of the data they handle, which feeds directly into later control selection [301]. AI speeds this up by automating the mapping of system interconnections and data flows through network analysis, giving a more accurate and complete view of the environment than manual efforts alone [96, 17]. Figure 5.2 shows how a system's authorization boundary relates to the broader environment of operation, a mapping that drives the categorization decisions feeding into the rest of the RMF process.

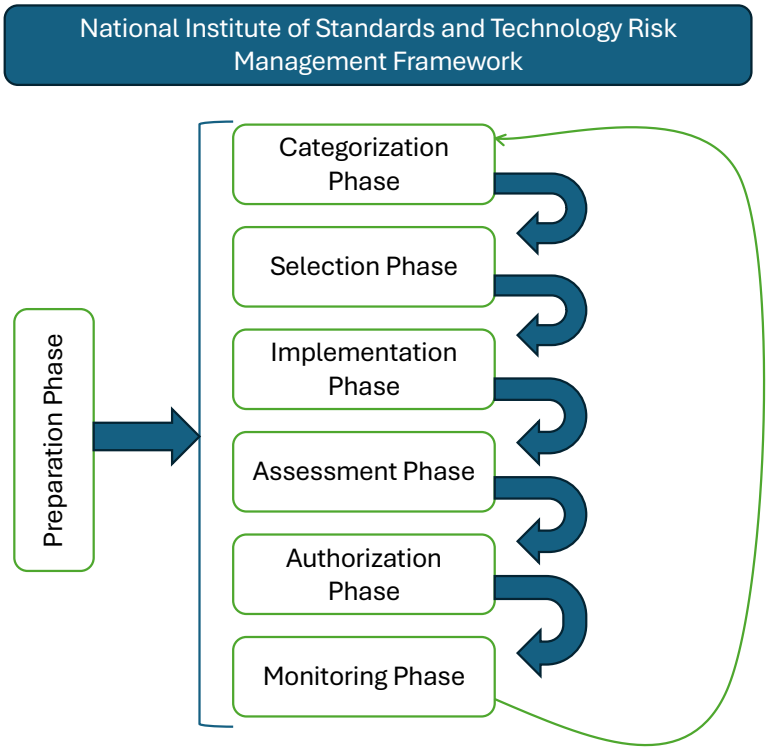


Figure 5.1: The cyclical nature of the RMF, where monitoring provides continuous feedback into the preparation for the next cycle. This image was inspired by [297, 188].

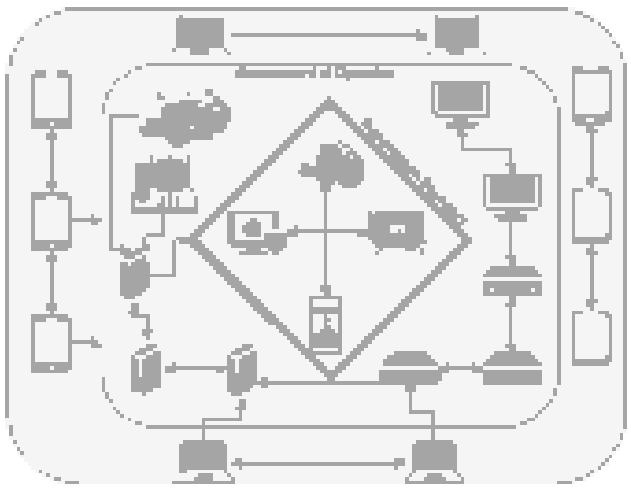


Figure 5.2: A conceptual diagram illustrating a system's Authorization Boundary, which contains critical assets, and its connections to the wider Environment of Operation. This mapping is a key activity in the RMF Categorization phase. This image was inspired by [207].

5.1.3 Selection Phase

In the Selection Phase, a baseline of security controls is established based on a full evaluation of threats, vulnerabilities, and risks [207, 211]. This process begins with a thorough threat assessment to identify potential adversaries and attack vectors, followed by vulnerability assessments to pinpoint weaknesses in the system [207, 188]. The identified threats and vulnerabilities are then analyzed in a risk assessment to determine their likelihood and potential impact [200, 185]. Based on this analysis, appropriate security controls are selected to mitigate the identified risks [159]. AI accelerates this selection process by analyzing vast amounts of threat intelligence data in real-time and automating vulnerability assessments to provide a more dynamic and accurate risk picture [17, 240].

5.1.4 Implementation Phase

The Implementation Phase is where the selected security controls are put into action, transforming the security plan into a tangible defense [297, 188]. This phase requires careful deployment and integration of the controls into the existing infrastructure. Thorough documentation of every configuration change and policy update matters for tracking, troubleshooting, and compliance [159, 260]. Personnel must also be trained on the newly implemented controls so they understand their role in maintaining security [304]. AI handles this by automating the deployment and configuration of controls through Infrastructure as Code (IaC) and other "Security as Code" practices, keeping configurations consistent and cutting down on human error [299, 265].

5.1.5 Assessment Phase

The Assessment Phase is a critical checkpoint to verify that the implemented security controls are effective and meeting their objectives [297, 188]. This involves reassessing vulnerabilities, threats, and risks given the new controls [304]. The primary goal is to verify that the controls are mitigating the intended vulnerabilities, which is accomplished through vulnerability scans, penetration testing, and security audits [297, 185]. It is also necessary to evaluate whether the controls provide adequate security without unduly hindering the system's operational capabilities [301]. AI improves this phase by automating the continuous testing and analysis of controls, simulating potential attack scenarios, and providing immediate feedback on their performance [96, 336].

5.1.6 Authorization Phase

The Authorization Phase marks the formal decision point where a senior official authorizes the system to operate based on an acceptance of the remaining risk [297,

188]. This official, or an external auditor, reviews all documentation, including risk assessments, control implementations, and test results, to determine if the system meets the required security and compliance standards [304, 301]. This decision is a critical judgment call that weighs the potential impact of a security breach against the organization's mission and risk tolerance [260]. AI can compress this phase by rapidly analyzing the extensive documentation and highlighting areas of highest risk, speeding up the review process and enabling a more informed and timely authorization decision, which is key to achieving cATO [139, 240].

5.1.7 Monitoring Phase

The Monitoring Phase is the final, ongoing stage of the RMF, where continuous monitoring confirms the system's security posture stays effective over time [297, 211]. This phase involves a relentless pursuit of maintaining security by adapting to emerging threats and evolving challenges [159, 301]. NIST IR 8011 provides the federal reference for automating security control assessments, which is the foundation the continuous-monitoring requirement actually rests on [76]. Continuous monitoring includes real-time surveillance of system activity, performance analysis, intrusion detection, and regular vulnerability scanning [304]. When a potential incident is detected, a swift and coordinated response is essential to contain the damage [185]. AI provides a meaningful capability in this phase by automating the real-time analysis of massive data volumes to detect anomalies and predict future threats before they can be exploited, ensuring a proactive and adaptive security posture throughout the system's lifecycle [102, 115].

CHAPTER 6
THE GUIDING PHILOSOPHY: ZERO TRUST ARCHITECTURE IN AZTRM-D

6.1 ZERO TRUST

While the RMF provides a structured process for governance, AZTRM-D's operational security is guided by the philosophy of ZT. ZT moves away from traditional, perimeter-based security models which implicitly trust entities inside the network. Instead, ZT operates on the foundational principle of "never trust, always verify" [266, 107]. ZT is no longer an architectural option for federal systems. Executive Order 14028 directed federal agencies to advance toward Zero Trust Architecture, OMB Memorandum M-22-09 set binding ZT implementation milestones across civilian agencies aligned to the CISA Zero Trust Maturity Model, and NSM-8 together with the DoD Zero Trust Strategy and the DoD Zero Trust Reference Architecture extends the same requirements to DoD, the Intelligence Community, and National Security Systems [310, 234, 72, 312, 78, 75]. AZTRM-D's ZT controls follow that established federal architecture rather than defining a parallel one. The National Security Agency (NSA) formalized this principle into actionable implementation guidance through its Zero Trust Implementation Guidelines (ZIGs), which organize the 152 ZT Activities from the Department of War (DoW) ZT Strategy into five phased implementation tiers covering 42 Target-level Capabilities and 91 Target-level Activities [218]. As defined in numerous systematic literature reviews, this model mandates that no user, device, or application is trusted by default, regardless of its location [169, 319, 110]. Every access request must be continuously and dynamically authen-

ticated and authorized against a set of risk-based policies [132]. Figure 6.1 provides a general overview of this core concept.



Figure 6.1: A general overview of the Zero Trust Architecture concept, where all access requests from users, devices, or applications are untrusted by default and must pass through a policy enforcement gateway for verification. This image was inspired by [266, 128].

The goal of ZT is to move from a static, location-centric security posture to a dynamic, identity-centric one. Interaction with a system is governed by a Policy Decision Point (PDP) and Policy Enforcement Point (PEP), which acts as a gateway. An entity is only granted access to the protected "implicit trust zone" after being verified by the PDP/PEP, and even then, access is continuously monitored [81, 287]. The Core ZT Model, shown in Figure 6.1, illustrates how real-time context feeds inform a central policy engine, which makes these dynamic trust decisions. Figure 6.2 breaks this down further into the five foundational pillars of Zero Trust as defined by Cybersecurity and Infrastructure Security Agency (CISA), each supported by cross-cutting capabilities in visibility, automation, and governance. Table 6.1 maps each pillar to its traditional weakness, ZT objective, and the AI-driven role AZTRM-D adds on top.

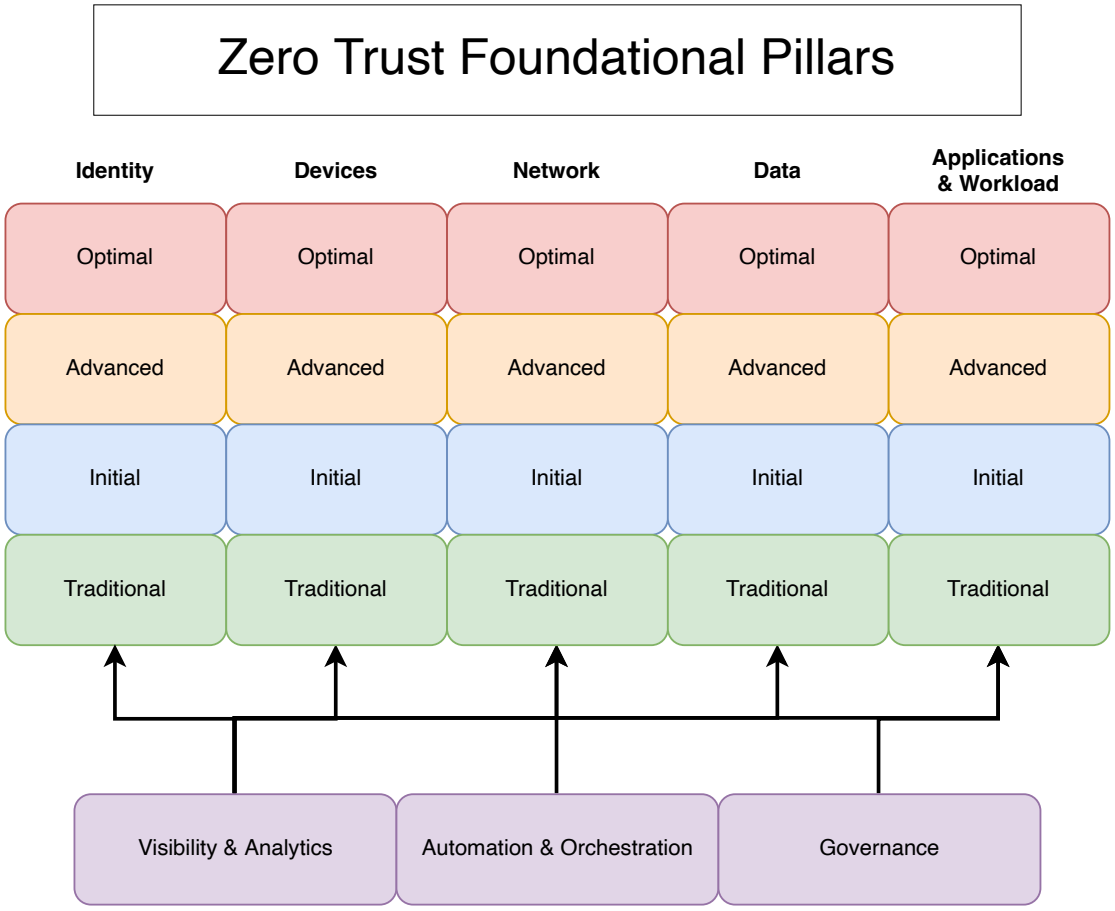


Figure 6.2: The five foundational pillars of Zero Trust, supported by three cross-cutting capabilities. The goal is to advance each pillar from a traditional, reactive security posture to an optimal, proactive state. This image was inspired by [72, 107].

Table 6.1: Zero Trust Pillars and their AI Enhancement in AZTRM-D. Traditional weaknesses and ZT objectives from cited NIST and CISA publications; AI-enhanced roles are the author's AZTRM-D design (this work).

Pillar	**Traditional Weakness**	**Optimal Zero Trust Objective**	**AI-Enhanced Role in AZTRM-D**
Identity	Static credentials (passwords) and roles grant broad, persistent access once authenticated [266].	Grant access on a per-session basis based on dynamic risk assessments of the user and their context, enforcing least privilege [72].	AI provides continuous, risk-based authentication by analyzing user behavior in real-time. It automates adaptive controls, such as triggering MFA or revoking sessions upon detecting anomalous activity [231].
Devices	Any device on the "trusted" network is allowed access, with only periodic checks for compliance [266].	Continuously verify the security posture of every device attempting to access resources and grant access based on device health [72].	AI-driven endpoint analytics continuously validate device posture (patch level, integrity, running processes). It can automatically isolate non-compliant or compromised devices to prevent them from accessing resources [102].
Networks	A hard outer perimeter with a soft, trusted interior ("castle-and-moat"). Attackers can move laterally with ease once inside [266].	Use micro-segmentation to create granular security zones. Encrypt all traffic and log every access request [72].	AI-powered network traffic analysis detects anomalous east-west traffic patterns indicative of lateral movement. It can dynamically adjust firewall rules and micro-segments to contain threats in real time [169].
Applications & Workloads	Applications are trusted by default once deployed. Security is focused on the perimeter, not the application itself [266].	Secure application access for all users, secure application-to-application communication, and harden workloads against runtime threats [72].	AI augments runtime application self-protection (RASP) by detecting and blocking exploits in real time. It automates vulnerability scanning and patching within the CI/CD pipeline, ensuring secure configurations [329].

Table 6.1 continued from previous page

Pillar	Traditional Weakness	Zero Trust Objective (Optimal Maturity)	AI-Enhanced Role in AZTRM-D
Data	Data is protected primarily by controlling access to the network or server where it resides [266].	Categorize and classify data based on sensitivity. Enforce access controls and encrypt data at rest, in transit, and in use [72].	AI automates data discovery and classification across all systems. It monitors data access patterns to detect and block potential exfiltration attempts before they succeed [17].
Cross-Cutting Capabilities	Manual governance, fragmented visibility, and reactive, human-driven security responses [266].	Achieve full visibility, automate security orchestration, and maintain dynamic governance across all pillars [72].	AI serves as the engine for all three: providing advanced analytics for visibility, orchestrating automated incident response, and dynamically updating governance policies based on emerging threats and risks [102].

6.1.1 Identity

The Identity pillar anchors ZT, ensuring every access request is authenticated and authorized under least privilege [266, 72]. It covers Identity and Access Management (IAM), MFA, and continuous user behavior analytics [107]. In a traditional model, static credentials like passwords grant broad, often persistent, access. ZT matures this by first introducing MFA, then adding AI-driven adaptive authentication that considers contextual data like user behavior and device health [120, 191]. At its optimal state, which AZTRM-D targets, AI-driven systems provide continuous identity validation with phishing-resistant MFA, not just at login but throughout a user's session [314, 326]. This matters especially in environments with many IoT devices,

which often have limited native security [337].

6.1.2 Devices

The Devices pillar focuses on ensuring any device accessing the network is authenticated and its security posture is continuously verified [72]. This includes managed corporate devices and unmanaged personal devices, and is especially critical for IoT devices that are often vulnerable and have limited computational resources [21, 288]. Traditional device security relies on periodic manual checks. ZT matures this by introducing automated tools for health monitoring and patch management [72]. In AZTRM-D's optimal implementation, AI-driven systems continuously verify device compliance in real-time. These systems can automatically quarantine or remediate devices that deviate from security policies, ensuring even the most resource-constrained IoT devices are securely integrated into the network [326].

6.1.3 Networks

The Networks pillar secures infrastructure by treating all traffic as untrusted until verified [72]. Its focus is on micro-segmentation, end-to-end encryption, and continuous traffic monitoring to stop unauthorized lateral movement [132]. Traditional "castle-and-moat" perimeter defenses are inadequate against internal threats [287]. ZT begins by implementing basic segmentation and encryption, then uses AI to monitor traffic for anomalies [156]. In AZTRM-D's advanced state, AI dynamically adjusts micro-segmentation based on real-time risk assessments [335]. In its optimal state, the framework enforces just-in-time and just-enough connectivity, which is critical for IoT devices. Devices then communicate only with authorized systems for the minimum time necessary, which sharply reduces the attack surface [326].

6.1.4 Applications and Workloads

This pillar focuses on securing applications and workloads throughout their lifecycle [72]. Traditionally, application security is reactive, with security bolted on after development. ZT matures this by integrating security scanning into the development process, such as through DevSecOps [77]. AZTRM-D extends this by deeply integrating AI into the DevSecOps pipeline, automating vulnerability detection and policy enforcement before deployment [17]. In its optimal state, the framework uses AI to provide continuous runtime protection, detecting and responding to threats like remote code execution exploits automatically by isolating workloads or applying virtual patches without human intervention [329].

6.1.5 Data

The Data pillar sits at the center of ZT, covering data at rest, in transit, and in use [118]. Traditional data security relies on protecting the network perimeter. ZT matures this by implementing automated tools for data discovery, classification, and encryption based on content and context [107]. AZTRM-D adds AI-driven systems that provide full visibility into all data interactions [156]. In its optimal state, data access is governed by dynamic, just-in-time, and just-enough access controls informed by real-time risk analytics [335, 191]. This is essential for IoT environments, where massive volumes of data are generated and processed in real time [21].

6.1.6 Cross-Cutting Capabilities: Governance, Automation, and Visibility

Three cross-cutting capabilities support the five pillars [72, 107]. Governance evolves from ad-hoc manual processes to a fully automated and dynamic state, where AI continuously monitors and adjusts policies based on real-time data and predictive analytics [335, 276]. Automation and Orchestration mature from limited, task-specific

scripts to a fully integrated environment where AI drives complex security processes like incident response and policy enforcement without human intervention [6, 133]. Finally, Visibility and Analytics progress from fragmented, manual data collection to full, real-time monitoring across the entire Information Technology (IT) environment. In the optimal state, AI-driven systems provide predictive insights that enable an organization to stay ahead of emerging threats [326, 115].

6.1.7 Integrating RMF, ZT, and DevSecOps in AZTRM-D

The true innovation of AZTRM-D lies not in applying these frameworks in isolation, but in weaving them together into a single, cohesive strategy. In this integrated model, the NIST RMF provides the structured, high-level governance process, answering the question, "What are our organizational risks and what security posture must we achieve?" The Zero Trust Architecture provides the granular, operational security philosophy by answering, "How do we enforce access control at every level to protect our resources in real-time?" [319]. Finally, DevSecOps provides the agile and automated implementation methodology, acting as the engine that drives both RMF and ZT goals continuously [280].

Treating AI as another entity that must adhere to ZT principles is also critical. AI systems within AZTRM-D are subject to the same "never trust, always verify" mandate, with strict identity controls, continuous monitoring, and human oversight to manage the unique risks they introduce [271, 132]. This integrated approach, powered by AI, makes security the fundamental basis of the entire development lifecycle, which will be detailed in the following section.

CHAPTER 7

AUTOMATED ZERO TRUST RISK MANAGEMENT WITH DEVSECOPS INTEGRATION (AZTRM-D)

AZTRM-D is a methodology designed to embed adaptive, risk-based security into every phase of the software lifecycle. It achieves this by unifying the principles of DevSecOps, the NIST RMF, and ZT, using AI as the orchestrating engine that enables these frameworks to work in concert. The AI-driven orchestration that distinguishes AZTRM-D is itself the federally identified maturity target for the Automation and Orchestration pillar of Zero Trust: NSA's Cybersecurity Information Sheet on advancing Zero Trust maturity in that pillar identifies AI-driven analysis, continuous behavioral monitoring, and automated policy enforcement as the capabilities that distinguish a mature implementation from an immature one [213]. AZTRM-D therefore aims at the same target the federal architecture aims at, applied across the software lifecycle and instantiated below in concrete pipeline controls. Figure 7.1 provides a high-level map of this integration, showing how ZT and RMF processes align with the stages of a DevSecOps pipeline. To further illustrate the role of AI, Figure 7.2 presents a conceptual architecture of how AI components are integrated as a continuous feedback and control loop within the DevSecOps workflow. The subsections that follow walk through each phase of the AZTRM-D lifecycle, naming the specific activities and the AI-driven controls that operate within them.

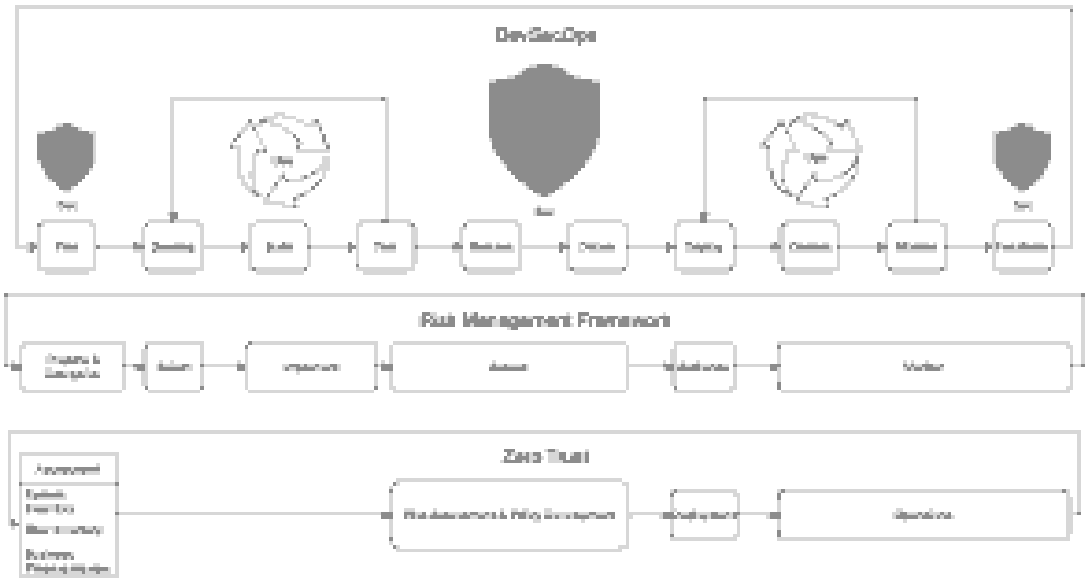


Figure 7.1: A high-level mapping of how Zero Trust and Risk Management Framework processes align with the phases of the DevSecOps pipeline within the AZTRM-D framework.

7.1 PHASES OF AZTRM-D

The AZTRM-D framework systematically integrates ZT principles, the RMF, and DevSecOps methodologies, all orchestrated by AI, across the entire software and system development lifecycle. The methodology is structured into distinct, yet interconnected, phases. Each phase builds upon the previous one, embedding security measures and risk management activities from inception to continuous operation. The result is a security posture that evolves with the software and the threat environment. Each subsection below documents the objectives of one phase, the activities that operationalize it, and the role AI plays in driving the controls forward [67].

7.1.1 Planning Phase

The Planning Phase is the foundation of AZTRM-D. In this stage, we establish security, risk management, and compliance strategies before any development even starts.

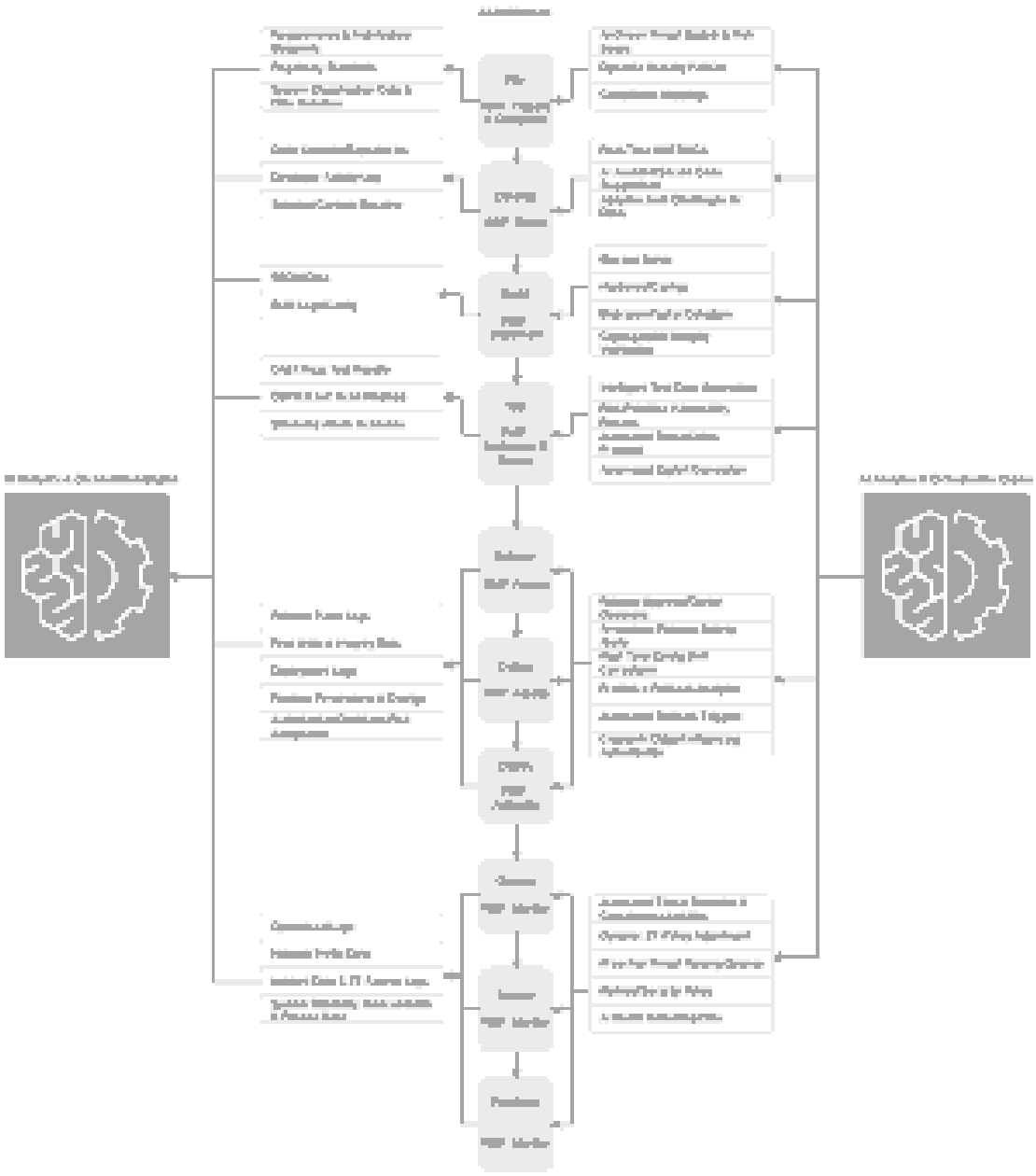


Figure 7.2: AZTRM-D AI Architecture and Orchestration Engine

Security gets built in from the very beginning by integrating ZT principles and RMF methods into the development process [159, 132, 297].

A key activity here is automated threat modeling. As shown in Figure 7.2, the AI Analytics and Orchestration Engine ingests core inputs such as system requirements and architecture blueprints, regulatory standards, and sensitive data classifications. Using these inputs, AI-driven tools analyze system requirements and architecture to predict potential attack paths and security risks [129, 329]. Instead of relying on fixed policies, AI algorithms use historical attack data and real-time threat information to dynamically adjust security parameters and controls, outputting dynamic security policies and compliance mappings [102].

At the identity level, AI-driven analysis of user roles helps create least-privilege access policies [231]. For device security, AI-driven endpoint analytics check device health before granting access to development environments. Network security involves planning micro-segmentation strategies. Applications and workloads are designed with security in mind, and automated validation enforces these principles. Finally, for data security, AI automatically finds and classifies sensitive data, then applies the right encryption and access control policies right from the start [118].

7.1.2 Development Phase

The Development Phase directly integrates security into the coding process. This stage focuses on secure coding practices [59], real-time security scanning, and automated policy checks. The goal is to prevent vulnerabilities from being introduced [17]. Developers authenticate using strong, risk-based identity verification. AI-powered Just-in-Time (JIT) access controls grant temporary, very specific privileges for particular tasks. This sharply cuts the risk of credential compromise [335].

Figure 7.2 shows how this works in practice. The AI Analytics and Orchestration Engine pulls data from code commits, repositories, and developer activity logs. To

secure the code itself, AI-driven Static Application Security Testing (SAST) tools are built right into the developer's Integrated Development Environment (IDE). They provide immediate feedback on potential vulnerabilities. SCA tools constantly scan dependencies for known weaknesses. This matters for supply chain security [239]. The AI engine generates real-time SAST/SCA findings, offers AI-assisted secure code suggestions, and can trigger adaptive authentication challenges for developers. If a developer tries to commit code with hardcoded secrets or a vulnerable dependency, the automated system can block the commit until the issue is fixed, enforcing a "secure by default" approach [253].

7.1.3 Build Phase

The Build Phase preserves software supply chain integrity as source code becomes executable artifacts [77]. Every action within the CI/CD pipeline is verified using identity-based access controls. This stops unauthorized changes to build scripts or environments [62]. Build environments are temporary and hardened. AI-driven integrity monitoring constantly assesses them to make sure they meet security standards.

A critical step in this phase, supported by the AI Analytics and Orchestration Engine which consumes Software Bill of Materials (SBOM) data and build logs, is the automated creation and validation of an SBOM [220]. The SBOM gives a complete inventory of all components and dependencies in the software [347]. AI-driven dependency management tools check the SBOM against vulnerability databases, blocking the use of any known-vulnerable components [102]. The AI also contributes to hardened configurations and malicious packet detection. Finally, every build artifact is cryptographically signed. AI-driven integrity verification mechanisms then scan these signed artifacts for any unexpected changes or additions. Validation at this stage detects supply-chain compromise introduced between development and testing.

7.1.4 Test Phase

The Test Phase acts as the security checkpoint where the compiled software undergoes thorough validation. AZTRM-D includes automated security testing, such as AI-driven Dynamic Application Security Testing (DAST), fuzz testing, and runtime validation [322]. The AI Analytics and Orchestration Engine takes DAST/fuzz test results, Cloud Security Posture Management (CSPM) and Infrastructure as Code (IaC) scan findings, and simulated attack scenarios as inputs. DAST tools test the running application for vulnerabilities that might not be obvious in the source code. AI-powered fuzzing engines create random inputs to test how resilient the application is against unexpected conditions, often leading to intelligent test case generation.

Workload security is validated using CSPM and IaC scanning tools. These tools, improved by AI, analyze deployment configurations for things like excessive permissions, exposed secrets, or misconfigurations that could create vulnerabilities [265]. The AI outputs risk-prioritized vulnerability reports, automated remediation proposals, and even automated exploit generation. If a security problem is found, for example, a container set up to run with root privileges, automated policies will block the deployment and force a fix. This phase makes sure both the application and its planned operational environment are secure before release.

7.1.5 Release and Deliver Phase

The Release and Deliver Phase is when validated software transitions to a production-ready state. This phase enforces Zero Trust principles, restricting build approvals to authorized people and automated processes can approve and carry out a release [72]. The AI Analytics and Orchestration Engine processes release event logs and final artifact integrity data. It then drives release approval or denial decisions and triggers anomalous release activity alerts. AI-driven behavioral analytics monitor for any unusual activity during the release process. An example might be an attempt to

change a deployment configuration without proper approval [6].

Before delivery, a final automated check confirms the integrity of the software artifacts. The cryptographically signed artifacts are verified one last time, and their SBOMs are checked against the latest threat intelligence. This step prevents any last-minute tampering or the introduction of a newly discovered vulnerability. This strict, automated gatekeeping is designed to approve only secure, verified, and compliant software for deployment. The AI also contributes to real-time configuration drift corrections and predictive rollback analytics during the delivery process.

7.1.6 Deployment Phase

In AZTRM-D, the Deployment Phase uses a Zero Trust model to prevent security misconfigurations or unauthorized changes during rollout. The principle of least privilege is strictly enforced, with AI-powered Continuous Access Evaluation (CAE) ensuring that deployment agents have the absolute minimum permissions needed, and only for the duration of their specific task [191]. The AI Analytics and Orchestration Engine analyzes runtime permissions and configurations, along with authorization decisions and risk acceptance data. AI-driven integrity checks analyze deployment scripts and runtime permissions in real-time to detect unauthorized modifications [336]. Should a post-deployment issue arise, AI-powered predictive rollback analytics can identify the last known good state and automate a secure rollback, minimizing downtime and exposure, with the AI also influencing key authorization outputs and triggering automated rollbacks.

7.1.7 Operate, Monitor and Feedback Phase

This final, ongoing phase keeps applications secure throughout their operational life. AZTRM-D includes real-time threat monitoring, Zero Trust adaptive access control, and AI-driven anomaly detection to provide continuous protection [240]. The AI

Analytics and Orchestration Engine is central here, consuming operational logs, network traffic data, incident data, and ZT access logs. AI-driven Endpoint Detection and Response (EDR) and RASP solutions constantly monitor workloads for signs of compromise or active attack attempts [102]. If something unusual is detected, such as an application trying to connect to a known malicious domain, automated security policies can isolate the workload, end the connection, and start an incident response workflow. AI drives automated threat detection and containment activities, dynamic ZT policy adjustments, and proactive threat hunting queries.

Security data and logs gathered in this phase feed directly back into the system. Telemetry, device health, and posture data flow into the AI Analytics and Orchestration Engine, which uses them to refine security policies and generate AI model retraining data. AI-driven security analytics bring this operational data together to identify new attack trends and refine ZT policies. The framework therefore adapts and learns over time, feeding back into the planning and development stages.

7.2 NIST ARTIFICIAL INTELLIGENCE RISK MANAGEMENT FRAMEWORK

While AI is integral to AZTRM-D's effectiveness, the AI models themselves introduce unique risks and must be managed responsibly [171, 276]. To govern these components, AZTRM-D incorporates the principles of the NIST AI Risk Management Framework (AI RMF). The AI RMF provides a structured approach to addressing the complex risks associated with AI, including issues of bias, transparency, and predictability [203].

A key part of this is ensuring the AI models are explainable and trustworthy. Rather than operating as "black boxes," the AI systems within AZTRM-D are designed to provide clear justifications for their security decisions, a concept central to the AI RMF's "Govern" function. Recent studies show that techniques like ensem-

ble label noise filters can measurably improve the reliability of local explanations, which applies directly to making AZTRM-D's security decisions more transparent and trustworthy, especially when dealing with imperfect real-world data [333].

For instance, the framework emphasizes explainability. If the AI blocks a developer's code commit, it does not simply return a "denied" message but provides a clear, human-readable reason, such as: "Commit blocked: This change introduces dependency 'library-x-1.2', which has a known critical Common Vulnerabilities and Exposures (CVE) entry (CVE-2021-41773, Apache path traversal and remote code execution). Please update to version '1.3' or higher" [9]. To generate these justifications, AZTRM-D is designed to integrate established Explainable Artificial Intelligence (XAI) techniques. For example, it could use SHAP (SHapley Additive exPlanations) to show precisely which code features most influenced the AI's 'risky' classification [178]. Alternatively, it could provide counterfactual explanations (e.g., 'This commit would be approved if the dependency 'library-x-1.2' were updated to version '1.3' or higher'), offering direct, actionable feedback to the developer.

The framework also addresses fairness, ensuring that AI models used for detecting insider threats are trained on behavioral data like resource access patterns rather than on personal identifiers to mitigate algorithmic bias. To operationalize this, AZTRM-D incorporates specific fairness metrics during model validation. For example, it uses 'equalized odds' to ensure the model's true positive rate (correctly identifying malicious activity) and false positive rate (incorrectly flagging normal behavior) are consistent across different developer roles or teams. If the AI flags commits from the 'Frontend' team at a statistically higher rate than the 'Backend' team for similar behaviors, the model is flagged for re-training with a more balanced dataset. This prevents a scenario where the AI develops a bias against a particular group, ensuring that security alerts are based on anomalous actions, not on pre-existing data imbalances or job functions.

Finally, the framework continuously tests AI threat detection models against adversarial examples designed to fool them, confirming the models stay resilient against sophisticated attackers [271]. By integrating these principles, AZTRM-D treats its own AI components with the same rigor as the systems they are designed to protect.

7.3 IMPLEMENTATION OF ZERO TRUST METHODOLOGIES ON ARTIFICIAL INTELLIGENCE MODELS/FEATURES

Applying ZT principles directly to the AI components is a critical, self-referential aspect of the AZTRM-D framework. Granting implicit trust to any system, including an AI, is inherently risky [276]. Therefore, each AI model and automation process is treated as an independent entity that must be authenticated and authorized before it can interact with data or other systems [132].

This is achieved through several layers of control. IAM protocols assign specific, limited roles to each AI process. For example, an AI model does not use static, long-lived credentials. Instead, it authenticates using a short-lived cryptographic token (e.g., a JavaScript Object Notation (JSON) Web Token (JWT) or Secure Production Identity Framework for Everyone (SPIFFE) certificate) that is automatically rotated by a central secrets manager. This token is presented each time the AI process requests access to a data source or another service. An AI model designed to analyze network traffic has no permission to access code repositories. This is enforced through a Policy Enforcement Point that validates the AI's token and checks its associated role-based access control (RBAC) policy. The policy for the network-traffic AI would explicitly grant 'read-only' access to a specific network log stream and deny all other actions, such as write access or access to the Git server API.

Second, all data inputs to AI systems undergo integrity validation that detects tampering or poisoning [171]. Finally, human oversight remains the backstop. While AI can automate risk assessment and suggest responses, the final decision to, for

example, shut down a critical production system, rests with a human operator who can evaluate the broader context, aligning with the principles of responsible AI [203]. Automation thus supports rather than supplants human judgment in critical security decisions.

The methodology, the orchestration logic, and the human-oversight model are now fully specified. What's missing is a worked example that shows how the pieces interact end to end against an actual adversary. The next chapter walks through a simulated scenario, tracing what AZTRM-D does at each stage of an attack lifecycle and where the layered defenses break the kill chain.

CHAPTER 8

AZTRM-D SIMULATED THEORETICAL SCENARIO

8.1 INITIAL SCENARIO

To show the practical operation of the AZTRM-D framework, this section presents a detailed simulated real-world scenario. This case study provides a tangible example of how the framework's core tenets, including DevSecOps, the NIST RMF, and Zero Trust, all powered by AI, work together to create a resilient and adaptive secure software development environment [67].

8.1.1 Scenario Description

To illustrate the practical implementation of the AZTRM-D methodology, consider a hypothetical scenario involving Company A, a global financial services provider. The company is entrusted with safeguarding a vast amount of sensitive data, including customer Personally Identifiable Information (PII), transactional records, and proprietary investment models. Given its complex operations and stringent regulatory requirements (e.g., NIST 800-53, PCI-DSS, Sarbanes-Oxley Act (SOX), General Data Protection Regulation (GDPR)), Company A requires a security strategy that enforces ZT principles, integrates RMF methodologies, and employs DevSecOps automation throughout its software development lifecycle [266, 72, 297, 77].

Company A operates a hybrid infrastructure that spans on-premises data centers and multi-cloud platforms, each presenting unique security challenges. By adopting AZTRM-D, the company embeds security, risk management, and compliance enforcement directly into its DevSecOps framework, enabling a proactive, automated, and

adaptive security posture.

8.1.2 AZTRM-D Walkthrough

Company A's security architecture is designed to enforce continuous ZT validation and automated risk assessments based on RMF principles [107, 191, 159]. The foundation is a risk-based IAM system where every entity undergoes continuous authentication before being granted access. If an AI-driven behavioral analytic engine detects an anomaly, such as an administrative account acting outside its normal patterns, the system automatically enforces adaptive authentication or revokes access until manual verification is performed [231, 8].

To prevent lateral movement, Company A implements ZT micro-segmentation, isolating development, testing, and production workloads [132]. AI-driven network monitoring continuously analyzes traffic patterns, automatically blocking unauthorized communications and triggering investigations when deviations are detected [169, 240]. Security monitoring is further strengthened by AI-powered Security Information and Event Management (SIEM) and Security Orchestration, Automation, and Response (SOAR) platforms, which initiate automated incident response workflows to contain threats before they escalate [299]. Data security is non-negotiable; all data is encrypted, and access is governed by strict, AI-enforced policies [118].

The AZTRM-D methodology is applied throughout the entire S-SDLC. During planning, AI-driven risk modeling defines dynamic security policies [129]. In development, developers use risk-based authentication, and all code is scanned for vulnerabilities before being merged [17]. In the build phase, software is compiled in hardened, ephemeral environments, and each artifact is cryptographically signed and its SBOM validated to prevent supply chain attacks [62]. The test phase uses AI-powered penetration testing and DAST to find flaws [329]. Finally, post-deployment, continuous runtime monitoring keeps workloads resilient against new and evolving threats [102].

8.1.3 AZTRM-D Enabled System Versus the Hacker

To truly appreciate the effectiveness of the AZTRM-D methodology, let's examine how a sophisticated adversary might attempt to breach Company A's infrastructure. This exercise demonstrates how the framework's integrated defenses neutralize each phase of a modern cyberattack.

Figure 8.1 previews the structure of the walkthrough that follows. The top row tracks the attacker's moves as they pivot through five increasingly aggressive attack categories, from reconnaissance against public infrastructure to physical attacks on endpoint hardware. The bottom row names the specific AZTRM-D control that fires at each phase. The rest of this section develops each phase in detail, explaining how the relevant control engages and what the attacker encounters. A reminder on scope is in order: this is a theoretical walkthrough designed to illustrate how the framework's pieces interact end to end. The empirical validation that determines whether AZTRM-D actually delivers this protection appears in Chapters 9 through 11.



Figure 8.1: AZTRM-D attack-chain analysis for the Company A simulated scenario.

The attacker's first move is reconnaissance. They scan Company A's public-facing infrastructure for misconfigured cloud storage or exposed APIs. However, AZTRM-D drives continuous posture management with AI-driven compliance tools automatically

correcting misconfigurations. Public APIs are secured behind ZT access policies, requiring continuous validation of user identity and device posture. Any anomalous request patterns trigger an automatic block and alert [319].

Frustrated, the attacker pivots to a MITM network attack. All internal traffic is encrypted with strong, modern protocols like TLS 1.3. Zero Trust Network Access (ZTNA) policies also require every connection to be continuously authenticated and assessed for risk, making it nearly impossible for an attacker to intercept traffic or move laterally [132]. The AI-powered network monitoring tools would detect the anomalous traffic from a MITM attempt and automatically isolate the affected network segment [169].

The adversary then shifts to social engineering, using a sophisticated phishing email to obtain employee credentials. Even if the email bypasses AI-driven filters, the stolen credentials alone are insufficient. ZT identity verification, enforced by MFA and continuous behavioral analysis, would immediately block a login attempt from an unrecognized device or location, rendering the credentials useless [231].

As a next step, the hacker might target the software supply chain, attempting to inject a malicious dependency into the build pipeline. This is a common vector for many organizations [239]. However, AZTRM-D's enforcement of SCA and SAST at every stage of development means the malicious library is flagged and quarantined instantly. Every build artifact also undergoes cryptographic integrity validation, ensuring no unauthorized code reaches production [62].

Desperate, the attacker might target IoT devices or attempt hardware-based attacks with a malicious Universal Serial Bus (USB) drive. In both cases, the ZT principles of device integrity and least privilege provide a strong defense. The IoT devices operate on a micro-segmented network with firmware integrity checks, while endpoint protection policies prevent unauthorized peripherals like the rogue USB from executing code, immediately isolating the device from the network [102].

At every stage, the adversary hits barriers reinforced by real-time analytics, dynamic access controls, and automated risk assessment. AZTRM-D breaks the cyber-attack lifecycle at multiple points, making it much harder for even a sophisticated attacker to maintain sustained access.

The simulated scenario above demonstrates the framework's logic against a textbook attack chain, but a simulated walkthrough is not a validation. Whether AZTRM-D actually delivers the protection the methodology promises depends on how the framework behaves on physical hardware, against real adversaries, under empirical measurement. The next three chapters address that question directly. Chapter 9 documents the controlled lab environment built on NVIDIA Jetson Orin devices and reports the baseline benchmarking results. Chapter 10 details the Cybectr Sentinel platform that operationalizes AZTRM-D, including the AI stack, the explainability layer, and the measured runtime performance. Chapter 11 then puts Sentinel and AZTRM-D under the multi-vector adversarial campaign that determines whether the architecture holds when three independent testers attempt to break it.

CHAPTER 9

LAB-BUILT SCENARIO AND BENCHMARKING

To demonstrate the practical efficacy of AZTRM-D, we designed and implemented a controlled lab environment using NVIDIA Orin devices as the core IoT components and a local Git server to embody DevSecOps principles within a Zero Trust architecture. The goal was to create a maximally secure environment while keeping the system operationally usable, then put it under realistic adversarial pressure at each stage of the hardening process. What follows documents that work in full, from factory defaults through final hardening, including the quantitative results it produced [67].

9.1 IMPLEMENTED SECURITY POSTURE

The lab setup maps security controls directly to the core pillars of the Zero Trust model. On the identity side, GitLab repository access is restricted to authorized users with MFA enforced across all logins, Secure Shell (SSH) key management policies in place, and RBAC configured for least-privilege access. The Orin devices go through foundational hardening at the device level, covering unnecessary services and ports are disabled, access to the local Git server is routed through reverse SSH tunnels, and firmware integrity is monitored continuously. Network isolation and micro-segmentation limit lateral movement, with firewalls blocking all unauthorized outbound traffic.

Application and workload security runs through several practices at once. RASP is embedded to let applications detect and block attacks during execution [102]. All

commits require cryptographic signing, and Software Bills of Materials (SBOMs) are generated and maintained so dependencies can be tracked and scanned continuously for vulnerabilities [239]. Data security is handled through access-controlled repositories, data classification schemes, and encrypted transfers. Detailed logging of all server activity feeds into AI-driven anomaly detection that monitors both user behavior and device behavior for anything suspicious [102].

9.2 SECURITY TESTING CAMPAIGN

This section walks through our assessment of the NVIDIA Jetson Orin Nano's security posture at each stage of its hardening process. Each scenario was tested from two perspectives, an external attacker using a standard penetration testing methodology aligned with NIST Special Publication 800-115 [294], and an insider evaluating adherence to ZT principles.

9.2.1 Security Testing Scenario: Factory Default Configuration

The first assessment targeted the Jetson Orin Nano in its unconfigured, out-of-the-box state. This mirrors the conditions an attacker would encounter on a freshly deployed, unhardened device.

Outsider Perspective:

Reconnaissance started with passive network observation to identify the device's MAC and IP addresses, with a ping sweep confirming the device was active. Specialized search queries turned up publicly available information that, in many cases, directly revealed the common default login credentials. In the scanning phase, Nmap port scans [180] (`nmap -sV -p- <target_device_IP>`) consistently found open ports for SSH (22) and Virtual Network Computing (VNC) (5900) on default JetPack configurations. Service version detection fed exact software versions into Nessus and

OpenVAS [117], which surfaced numerous unpatched CVEs in the kernel and NVIDIA drivers.

For initial access, brute-forcing SSH with Hydra [320] against the default `nvidia:nvidia` credentials was the most reliable method, as these credentials ship with the factory-default JetPack image [232]. The default SSH daemon had no account lockout mechanism, which made this approach highly effective. Once in as the `nvidia` user, the default sudo configuration allowed immediate privilege escalation to root via a single `sudo su` command. Full administrative control, achieved in under five minutes. Persistence was then established by modifying system initialization files (`~/.bashrc` or `/etc/rc.local`) to launch a reverse shell on boot, creating hidden user accounts, and using SSH tunneling to encrypt command-and-control traffic. Log cleanup involved systematically wiping or altering entries in `/var/log/`, specifically `auth.log` and `syslog`, and clearing shell histories to hinder forensic investigation.

Insider Perspective:

From an authenticated user's perspective, the default configuration represented a complete failure of ZT principles before any meaningful security decision had to be made. A weak default password means the initial trust decision is already compromised. Once logged in, unconstrained sudo access is a direct violation of least privilege. A proper ZT posture would mandate strong, unique credentials and granular sudo policies from day one. Sensitive configuration files in `/etc/` and mutable log files in `/var/log/` were readable without restriction. With root trivially in reach, an insider could modify any system file, install malware, create hidden accounts, alter network configurations, and erase logs entirely. The audit trail, which is a core ZT requirement, was effectively meaningless.

9.2.2 Security Testing Scenario: After Initial Hardening

This test ran after initial AZTRM-D hardening focused on eliminating the network attack surface. The network-level exposure was gone. What it revealed, though, was that physical vulnerabilities remained unaddressed.

Outsider Perspective:

Reconnaissance and scanning confirmed the network-based attack surface had been eliminated. Thorough Nmap scans found no open ports; vulnerability scanners returned nothing actionable. That shifted our efforts entirely to physical attack vectors. The unencrypted Secure Digital (SD) card turned out to be the primary vulnerability.

After physically removing the SD card, we mounted its root filesystem on an external Linux machine and had full access to all files, including `/etc/shadow`. Using `chroot`, we reset a local user's password offline, bypassing all logical authentication entirely. After reinserting the card, we connected via the UART serial console (`ttyTHS1`), authenticated with the new credentials, and escalated to root because the user was still in the `sudo` group. Network access was secured. Physical access was not.

Insider Perspective:

The logical controls for user authentication, Git server access, and development workflows were working correctly at this stage. The problem was the SD card. An insider with physical access to the device could remove the card, mount the filesystem externally, and gain unauthorized root access outside any software-based monitoring AZTRM-D had implemented. At this stage, the framework implicitly trusted hardware integrity once physical access was obtained. That meant an insider could operate at the lowest system level with unconstrained privileges, invisible to the logging and

enforcement systems designed to catch exactly that behavior. The physical storage gap remained, and it negated much of the rest.

9.2.3 Security Testing Scenario: After Final Hardening

This test ran after implementing full SD card encryption and hardened root access controls, addressing the physical and privilege escalation vulnerabilities identified in the previous round.

Outsider Perspective:

The device held. Network scans again confirmed no remote attack surface. The physical vector was closed. Removing the SD card now presented an encrypted volume, and all attempts to mount the filesystem or perform an offline password reset failed. GPIO pins were disabled. The UART serial console remained physically accessible but was no longer viable. Without the ability to tamper with the SD card or find an easy privilege escalation path, it simply presented a login prompt for which we had no credentials. No access was achieved via external or physical means.

Insider Perspective:

The access controls and secure development workflows remained consistently effective throughout. The final hardening addressed the critical blind spot around out-of-band root access. Attempts to escalate privileges, even from an authenticated administrative account, were no longer trivial and required explicit validation that could not be easily circumvented. The "never trust, always verify" principle was now extended to the deepest system layers, and the risk of an insider gaining unauthorized root access dropped sharply.

9.2.4 AI-Assisted Insider Attack Validation

A critical design question for any AI-integrated development pipeline is whether the introduction of AI code generation tools creates new pathways for insider threats. AZTRM-D's architectural answer is that automated security gates evaluate code content, not authorship. Stage 3 validation tested this claim directly.

One of the three testers submitted Python code generated by an external LLM (Claude, Anthropic) designed to perform privilege escalation via a `subprocess` call invoking `sudo` without a granular policy wrapper. The submission was made through the standard developer commit workflow with no disclosure that the code was AI-generated. Three independent automated gates evaluated the submission.

1. *SAST (Semgrep)* flagged the `subprocess` call pattern against the project's AZTRM-D policy ruleset in 3.4 seconds. The flag was raised on code content, specifically the pattern of invoking a shell command with elevated privileges without a corresponding policy wrapper, with no knowledge of the code's origin.

2. *Secrets Scan* independently detected an embedded test API key present in the submitted code, triggering a separate policy violation.

3. *SBOM Validation* rejected an unauthorized dependency introduced in the same commit.

All three gates fired before any human reviewer examined the submission. The AI-generated nature of the code was irrelevant to detection. The gates evaluate what code does, not how it was written. This result supports the AZTRM-D design principle that content-evaluation automated gates are authorship-agnostic and therefore remain effective as AI code generation capability improves. The boundary condition of this claim, specifically a white-box adversary with knowledge of the policy rules, is acknowledged in the limitations discussion in Section 13.2.

9.3 QUANTITATIVE BENCHMARKING RESULTS

Table 9.1 summarizes the security posture before and after AZTRM-D hardening across six measured metrics. Every value in the AZTRM-D column comes from Stage 3 testing on the physical NVIDIA Orin hardware.

Table 9.1: Quantitative Benchmarking of AZTRM-D Security Improvements on NVIDIA Orin Devices. Source: Stage 3 measured results (this work) [67].

Security Metric	**Baseline (Factory Default NVIDIA Orin)**	**AZTRM-D Hardened Configuration (NVIDIA Orin)**
Open Network Ports	4 Ports (SSH, VNC, HTTP, HTTPS)	0 Ports (Complete elimination of network attack surface).
Initial Access Time (External)	Less than 5 minutes (via default credential brute-force).	Not Possible (No remote or physical vectors found).
Privilege Escalation	Immediate (single `sudo su` command from default `nvidia` user).	Blocked (Requires explicit, multi-step validation; default paths removed).
Vulnerability Detection Rate (VDR)	0% (No automated scanning integrated into development pipeline).	96.8% (61/63; Eq. 9.1). Automated SAST, DAST, SCA, CSPM, and IaC scans.
Mean Time to Remediate (MTTR)	Weeks to Months (Manual patching, firmware updates, or re-flashing).	1-3 days (Automated alerting and patching pipeline, with AI-driven remediation or automated rollback).
Supply Chain Vulnerability	High (No dependency scanning or artifact signing for device software).	Low (Mandatory SBOM validation and cryptographic signing for all software components).

9.3.1 Vulnerability Detection Rate Measurement Methodology

The 96.8% vulnerability detection rate (VDR) reported for AZTRM-D's five-modality CI/CD scanning pipeline was derived from a controlled test corpus built in two parts. One part consisted of vulnerabilities deliberately seeded into the codebase prior to each hardening stage, representing known vulnerability classes targeted by each scan-

ning modality. The other part consisted of vulnerabilities discovered organically during testing, including findings surfaced through active exploitation attempts in Stages 1 and 2.

A custom corpus was constructed rather than using a standardized benchmark such as Open Worldwide Application Security Project (OWASP) Benchmark or the Juliet Test Suite because the validation required vulnerabilities representative of the specific deployment environment: an embedded Linux system (NVIDIA Orin running JetPack) with a GitLab CI/CD pipeline, IoT-specific network configurations, and AZTRM-D's five-modality scanning stack. Standardized benchmarks target web application vulnerabilities (OWASP Benchmark) or general C/C++ code defects (Juliet) and do not include the infrastructure misconfigurations, CSPM findings, or supply chain vulnerabilities that three of AZTRM-D's five scanning modalities are designed to catch. The trade-off is that a custom corpus cannot claim the cross-study comparability of a standardized benchmark. The Wilson score confidence interval (Equation 9.2) and the modest corpus size ($n = 63$) are reported explicitly so readers can assess the statistical precision of the result on its own terms.

Table 9.2 provides the full derivation. Each row corresponds to one CI/CD scanning modality. The "Unique Detections" column counts findings caught by that modality and no other. "Shared Detections" counts findings confirmed by at least one additional modality, providing independent cross-validation. The two vulnerabilities constituting the undetected 3% were novel logic flaws requiring manual expert analysis. One involved an application-layer race condition with no exploitable pattern recognizable to static or dynamic analysis tools. The other was a business-logic authorization bypass with no signature in any evaluated scanner database. Both were identified through manual code review during the post-Stage-3 assessment. All three testers independently reviewed both findings and agreed on their classification as logic-layer vulnerabilities beyond automated scanner detection before they were

recorded as confirmed undetected results. Both are documented in the remediation log.

Table 9.2: Vulnerability Detection Rate Derivation by Scanning Modality. Source: Stage 3 CI/CD pipeline results (this work).

Modality	**Seeded**	**Discovered**	**Unique Detections**	**Shared Detections**
SAST (Semgrep + GNN-augmented)	18	4	11	11
DAST	12	3	6	9
SCA (SBOM + dependency scan)	9	2	7	4
CSPM	7	1	5	3
IaC Scanning	6	1	4	3
Total Corpus	**52**	**11**	**63 total vulnerabilities**	
Detected (aggregate)			**61 (96.8%)**	
Undetected			**2 (novel logic flaws, manual review)**	

Overlap between modalities is counted once in the numerator; a vulnerability caught by both SAST and DAST contributes one to the detected count.

The aggregate detection rate was calculated as shown in Equation 9.1.

$$\text{VDR} = \frac{\text{Total Detected}}{\text{Total Corpus}} = \frac{N_{\text{detected}}}{N_{\text{total}}} = \frac{61}{63} = 96.8\% \tag{9.1}$$

For a sample proportion $\hat{p} = 61/63 = 0.968$ with $n = 63$, the Wilson score 95% confidence interval provides a measure of statistical precision:

$$\text{CI}_{95} = \frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} = [0.891,\ 0.991] \tag{9.2}$$

where $z = 1.96$ for 95% confidence. The lower bound of 89.1% confirms that even at the conservative end of the confidence interval, the pipeline detects the large majority of seeded and organically discovered vulnerabilities in the controlled corpus. The

interval width reflects the modest corpus size ($n = 63$); a larger-scale validation would tighten this bound. This conservative aggregation avoids inflating the rate through double-counting of findings confirmed across multiple modalities. Baseline detection rate under Stage 1 (factory default, no automated scanning) was 0% for all modalities, as no CI/CD pipeline existed prior to AZTRM-D implementation [67].

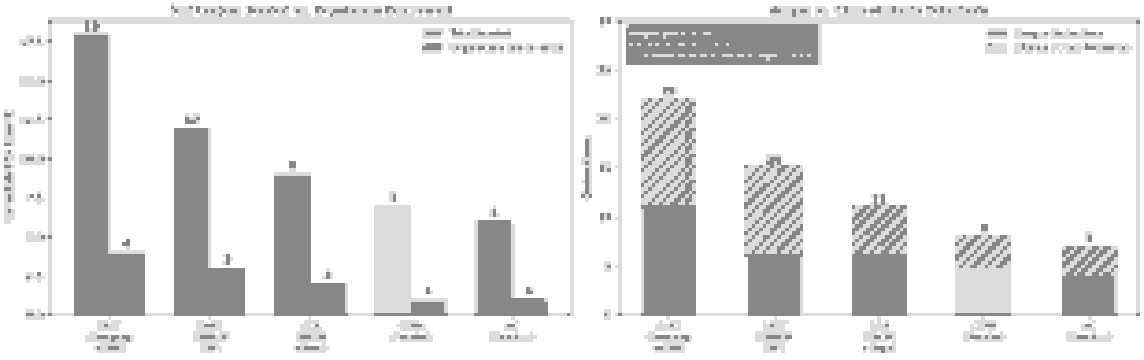


Figure 9.1: VDR breakdown by scanning modality: seeded versus organically discovered vulnerabilities (left) and unique versus cross-modality shared detections (right) from the 63-vulnerability test corpus.

Figure 9.1 breaks down the VDR by scanning modality. Table 9.1 summarizes the security improvements achieved across all three stages. Figure 11.7 presents the Stage 1 versus Stage 3 security posture comparison. The numbers tell a clear story.

At factory default, reconnaissance identified four open network ports: SSH, VNC, and web services on HTTP/Hypertext Transfer Protocol Secure (HTTPS), each one a direct remote attack vector. After AZTRM-D hardening, that number dropped to zero. No remote exposure at all, which matters particularly for edge devices where the threat surface otherwise includes any network interface the device touches.

Regarding initial external access time, the factory default Orin was compromised in under five minutes. This speed came down to two factors, widely known default credentials, and the SSH daemon having no account lockout mechanism. Brute force

with Hydra was more a formality than a challenge. With AZTRM-D applied, external access became not possible. Testing covered both remote network and direct physical attack vectors after SD card encryption and GPIO hardening, and neither produced a viable entry point.

Privilege escalation follows the same pattern. A single `sudo su` command from the default `nvidia` user gave immediate root at Stage 1. After full hardening, escalation requires explicit multi-step validation, and the default paths that made it trivial have been removed.

The CI/CD vulnerability detection rate went from 0% to 96.8%. That 0% baseline is not a failure of effort; it reflects a factory configuration with no scanning pipeline whatsoever. Vulnerabilities were found reactively, long after deployment or after an incident. Under AZTRM-D, five scanning modalities run in parallel across every code push. Those modalities are SAST, DAST, SCA, CSPM, and IaC analysis. The remaining 3% reflects novel logic flaws and context-dependent misuse patterns that automated tools don't handle well and that require expert human review.

MTTR dropped from weeks or months to 1–3 days, as measured across four remediation cycles during Stage 3 validation on the Orin devices (two critical vulnerabilities, one misconfiguration, and one supply chain finding). The baseline MTTR of weeks to months reflects industry-reported remediation timelines for systems without automated patching infrastructure [99, 77]. Under AZTRM-D, automated alerting, integrated patching pipelines, and AI-driven remediation or automated rollback mean critical vulnerabilities can often be addressed within hours.

Supply chain vulnerability shifted from high to low. The factory configuration had no dependency scanning or cryptographic signing for any device software. AZTRM-D mandates SBOM validation for all software components and cryptographic signing of all build artifacts, with continuous AI-driven dependency scanning running throughout.

9.3.2 Supply Chain Vulnerability Quantification

The categorical improvement in supply chain security posture from High to Low is supported by measured evidence from Syft [20] and Grype [19] analysis of the NVIDIA Orin platform. Prior to AZTRM-D implementation, the factory-default JetPack image contained no SBOM and no dependency signing. Syft was run against the default JetPack image to generate a baseline software bill of materials, and Grype was then run against that SBOM to enumerate known CVEs in the dependency tree. Table 9.3 presents the comparison.

Table 9.3: Supply Chain Vulnerability Comparison Pre- and Post-AZTRM-D. Source: Syft [20] and Grype [19] analysis of NVIDIA Orin JetPack image (this work).

Condition	**Critical CVEs**	**High CVEs**	**SBOM Present**
Factory default (Stage 1)	7	23	No
Post-AZTRM-D (Stage 3)	0 flagged unresolved	0 flagged unresolved	Yes (signed)

Stage 1 CVE counts reflect Grype enumeration against the Syft-generated SBOM for the factory-default JetPack image. The Stage 3 figure of zero unresolved Critical and High CVEs reflects mandatory SBOM validation at the IaC gate. CVE blocks deployment until patched or formally accepted via the RMF deviation process.

Table 9.4 documents the resource overhead and scalability characteristics measured on the Orin during Stage 3 operation.

Table 9.4: AZTRM-D Resource and Scalability Benchmarking (NVIDIA Orin).

Performance Metric	**Measurement**	**Significance**
Resource Usage (AI Scans)	12-18% average CPU overhead during CI/CD pipeline scans (SAST/SCA).	Demonstrates that continuous security scanning is computationally feasible on edge devices without crippling performance.

Continued on next page

Table 9.4 – continued from previous page

Performance Metric	Measurement	Significance
Policy Enforcement Latency	Less than 40ms for ZT access policy evaluation at the Policy Enforcement Point.	AZTRM-D design target for real-time policy evaluation, consistent with the per-request access architecture described in [266]. Ensures that stringent, real-time access checks do not introduce significant delays that would impact user experience or system-to-system communication.
AI Model Training Time (Initial)	14 hours to train the insider threat model using three months of log data on a standard cloud GPU instance.	Quantifies the initial setup cost for the AI components, providing a baseline for resource planning.
Scalability Test (Endpoints)	Single-device baseline: 3.8% CPU, 41 MB memory, 6.2 ms PEP latency at steady-state on NVIDIA Orin (Stage 3 measured). Multi-endpoint fleet testing was not conducted; multi-device characterization is designated as future work (Section 13.4).	Projects that the orchestration architecture scales to enterprise deployment densities without breaching the 40 ms PEP latency SLA; actual performance depends on network topology and authentication event frequency.

Security that degrades operational performance isn't viable on constrained edge hardware. Table 9.4 quantifies the AZTRM-D overhead on the Orin directly: AI-driven scanning stayed between 12–18% of CPU capacity, well within what the device can absorb without impacting its primary function. ZT policy enforcement added less than 40 ms of latency per access decision, which is imperceptible in normal operation. These figures come from the actual Orin hardware, not a more capable test environment standing in as a proxy.

The 14-hour initial training time for the insider threat model is the one figure worth planning around explicitly. That training run requires three months of operational log data and a cloud Graphics Processing Unit (GPU) instance, and it must complete before behavioral anomaly detection reaches operational accuracy. It's a

one-time cost, not a recurring one, but organizations can't activate behavioral monitoring on day one and expect accurate results immediately.

The scalability section documents what was actually measured: a single Orin device under sustained operational load. Multi-device fleet behavior requires physical multi-device testing, which is scoped as future work and will determine at what endpoint count horizontal scaling of the orchestration tier becomes necessary.

9.3.3 Scalability Projection Methodology

This section documents what the Stage 3 single-device measurements establish and don't establish about fleet-level scaling behavior. The distinction matters for honest interpretation of the results.

Stage 3 validation established the per-device resource footprint of the AZTRM-D orchestrator on the NVIDIA Jetson Orin. Under normal operational load on a single enrolled device, the orchestrator consumed 3.8% CPU and 41 MB memory, with PEP request latency measured at 6.2 ms. Metrics were collected using `top` and `vmstat` at 5-second intervals throughout the monitoring window. These figures were stable across a 72-hour continuous operation window with active telemetry streaming and periodic re-authentication events.

Multi-endpoint physical testing was not conducted in this validation environment. Projecting orchestrator resource consumption across an endpoint fleet from a single measured data point requires fitting a scaling model with at least two free parameters, which cannot be done from a single observation without imposing unjustified assumptions about the functional form of the relationship. No multi-endpoint projection table is presented here.

What the single-device measurements do establish is a hard constraint on the scaling behavior. If the orchestrator scaled linearly, with each enrolled device adding its own independent 3.8% CPU load, the resource equation would be:

$$\text{CPU}(n) = 3.8n\ \%$$

At $n = 1{,}000$ enrolled devices, that yields 3,800% CPU load, which is physically impossible on a single Orin module or any single-board compute platform. Sub-linear scaling is therefore a necessity, not a design aspiration. What the measured figure establishes is a per-device floor; actual fleet scaling behavior is bounded above by linear and bounded below by $\mathcal{O}(\log n)$, the theoretical minimum for any cache-resident session management system.

Sub-linear scaling has a defined architectural basis in the AZTRM-D design, documented in the Zero Trust Architecture (ZTA) specification. NIST SP 800-207's PEP/PDP session token model requires that after initial device authentication, subsequent access requests are validated against cached session state rather than resubmitted to the Policy Engine for full trust recomputation [266]. In a deployed fleet, as n grows, the fraction of requests that hit cached session state increases relative to first-evaluation requests, which drives down average CPU cost per request. AZTRM-D Sentinel's telemetry aggregation layer similarly batches sensor streams from enrolled devices into consolidated processing cycles, preventing memory and I/O from growing in direct proportion to endpoint count. Neither of these properties is speculative; both are architectural requirements that were validated at the single-device level during Stage 3.

The 6.2 ms PEP latency and 3.8% CPU measured under steady-state single-device load confirm that the per-session evaluation cycle operates well within both the 40 ms Service Level Agreement (SLA) ceiling and the 20% security overhead constraint established as AZTRM-D design requirements for resource-constrained IoT deployments, with the Orin hardware specifications defining the compute budget within which these constraints must be satisfied [233]. That headroom is real and documented. Characterizing the exact n at which the orchestration layer approaches

saturation, and horizontal scaling of the PDP tier becomes operationally necessary, requires a controlled multi-device experiment. That experiment is scoped as a priority for future work in Section 13.4.

9.3.4 System Technical Overview

Figure 9.2 shows the internal security flow for a device running under AZTRM-D, from secure boot through continuous monitoring to access policy enforcement. The sequence reflects deliberate design choices worth walking through.

The process starts with Secure Boot, which establishes the chain of trust from the first software loaded. Cryptographic verification of the bootloader, kernel, and initial OS components before execution prevents the device from starting with tampered or compromised code. What follows is Continuous Device Health and Environmental Monitoring, which checks for malfunction, resource exhaustion, or unusual system behavior, and tracks physical conditions like temperature and power supply. Unexpected changes in those parameters can signal tampering or cooling failures.

Device Inventory Management maintains an accurate record of all hardware components and deployed software versions, which feeds into Firmware Integrity Checks. Firmware is verified using cryptographic hashes or digital signatures to confirm it hasn't been altered by malware seeking persistent control at a fundamental hardware level.

Automated Scanning on the Git Server happens before any deployment. This is the DevSecOps integration point, where code and configuration changes are automatically scanned for vulnerabilities, policy violations, or malicious injections before they reach an operational device. Sensitive data gets labeled and classified so protection measures can be applied proportionally. PII and financial records receive stronger encryption, stricter access controls, and more intensive logging than routine operational data.

Behavioral Monitoring for Insider Threat Detection establishes a baseline of normal operational behavior for users and system processes, then watches for deviations. Unusual file access attempts, unauthorized privilege escalation, and anomalous data aggregation or transfer activity all trigger alerts. Unauthorized Data Transfer Detection runs alongside this, specifically watching for exfiltration attempts, whether deliberate or accidental.

Several parallel processes feed into the access control decision-making layer. Application Version Scanning checks for outdated software versions that contain known unpatched vulnerabilities. RASP is embedded within applications to detect and block attacks in real time as they happen. Automated Dependency Scanning checks third-party libraries for known flaws. The broader Anomaly Detection and Real-time Behavioral Analysis layer catches unusual activity at both the system and network levels that more specific checks might miss.

All of these streams converge at the Centralized Policy Engine. Rather than applying a static policy at the perimeter and trusting everything inside it, the engine evaluates the device's security posture continuously based on real-time inputs from every monitoring component, then makes access decisions accordingly. Access is granted, denied, or restricted based on current conditions rather than assumptions made at deployment time.

9.3.5 User Interaction with AZTRM-D Implemented System

Figures 9.3 and 9.4 document the secure workflows governing how developers and administrators interact with the implemented system, particularly around code management and server access.

Figure 9.3 shows the Git server workflow. The security philosophy here is shift-left, catching problems early before insecure code ever enters a shared repository. Every commit request is logged immediately, creating an audit trail from the first

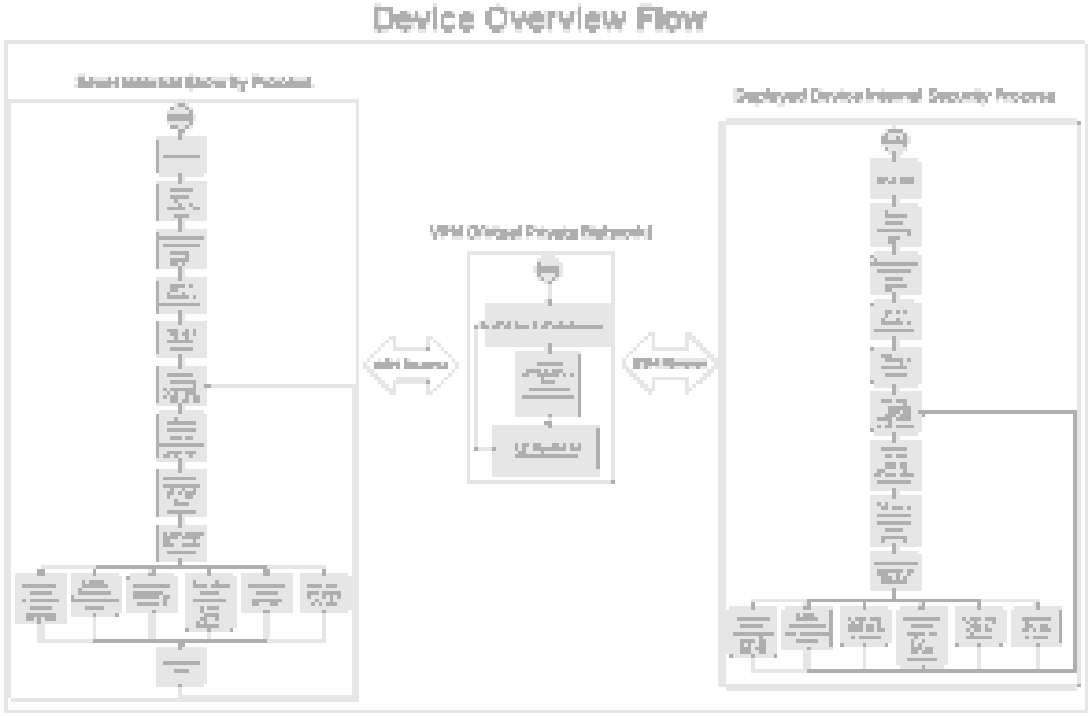


Figure 9.2: The internal security process flow for a device operating under the AZTRM-D model, from secure boot to continuous monitoring and policy enforcement.

action. Before code can be considered for merging, it goes through SAST, which analyzes source code for potential security flaws without executing it. A failed scan automatically rejects the commit. Pass the SAST, and the commit moves to a Secrets Scan that looks for accidentally embedded credentials, API keys, or sensitive data. A failed Secrets Scan also results in rejection, while a successful one is logged and moves to Admin 1 approval, a human gate before code gets integrated into the shared repository.

Once individual developers' work passes initial scans and receives approval, branches merge into a Quality Assurance (QA) environment. That integrated codebase requires Admin 2 approval, followed by Dependency and SBOM checks against all third-party components, followed by DAST of the running application. Unlike SAST, DAST tests the application while it's executing, finding vulnerabilities that only appear at runtime. Infrastructure as Code scanning then validates the deployment configuration itself for misconfigurations that could create vulnerabilities in the deployment environment. Pass all of that, and the code moves to a staging environment. Super Admin approval is required before the release artifact gets cryptographically signed and moved to the secure repository for production deployment. Every step is logged. Nothing reaches production without clearing every gate.

Figure 9.4 details the multi-layered user authentication and session management flow for accessing a device-hosted server under AZTRM-D's Zero Trust policies. Every access attempt goes through continuous verification and risk assessment, far beyond single-factor authentication.

The flow starts when a user connects from their assigned machine, which is immediately logged. Before authentication begins, the system checks account status. Accounts inactive for 10 days are suspended, and accounts inactive for 30 days are purged entirely. These are AZTRM-D policy parameters consistent with NIST SP 800-53 account management controls (AC-2) [145]. Physical security comes next.

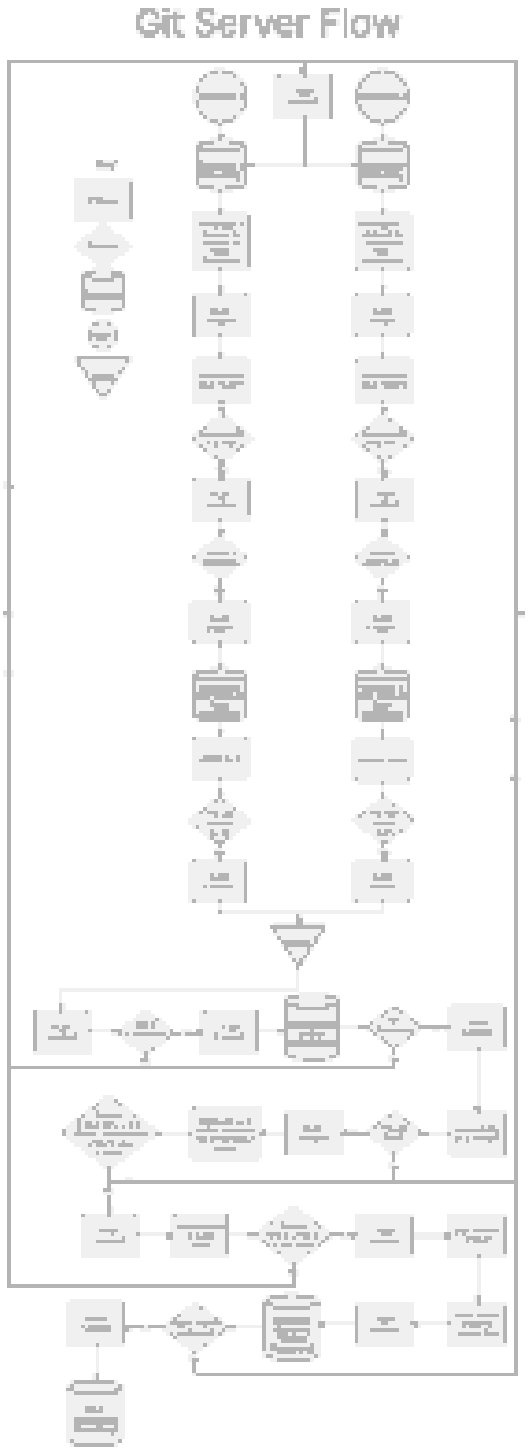


Figure 9.3: The secure Git server workflow within AZTRM-D, illustrating the mandatory security checks and administrative approvals required for code progression.

The assigned USB security key must be present during the connection attempt. A missing key suspends the account and logs the attempt.

If the key is present, SSH key verification runs. Three failed attempts trigger automatic suspension. A correct SSH key passes control to passphrase verification, which enforces the same three-attempt limit. Password authentication follows the same pattern, with an additional check. If seven days have passed since the last password change, the user gets prompted to choose a new password from a provided set within 30 seconds. Failure to do so results in immediate suspension. Two-factor authentication completes the chain, again with a three-attempt limit before automatic suspension.

After all authentication stages pass, the user gains access to their assigned developer Git instance, and this is immediately logged. Sessions are continuously monitored and automatically terminated after 60 minutes, consistent with least-privilege continuous verification principles [71, 216], and all activity is logged throughout. The intent is that access remains continuously verified and context-aware from initiation to termination, adhering to least privilege at every step.

9.3.6 Future Setup Additions

Several improvements are planned to mature the Zero Trust architecture for this NVIDIA Orin-based IoT environment. On the network side, the existing Virtual Private Network (VPN) infrastructure will be replaced or complemented by a ZTNA solution, enforcing identity- and context-aware access to internal services, including the local Git server, and moving away from perimeter-based trust models entirely.

The lab environment establishes that AZTRM-D's policy and access model can be deployed on constrained-edge hardware. What it does not yet specify is the platform that operationalizes the methodology day to day, which AI subsystems run on the device, how their decisions get explained to a human operator, and what the resulting

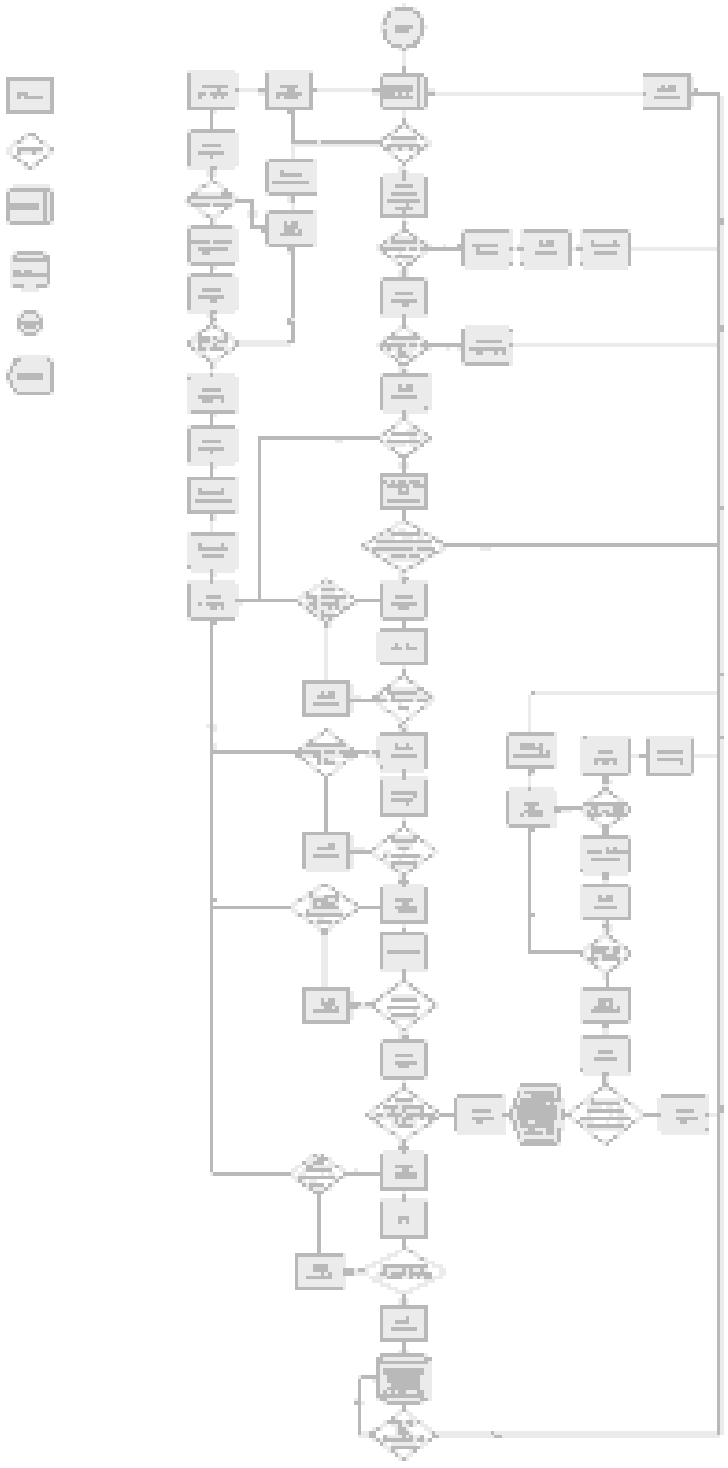


Figure 9.4: The multi-layered user authentication and session management flow for accessing a device-hosted server under AZTRM-D's strict ZT policies.

runtime cost looks like in measured terms. The next chapter introduces Cybectr Sentinel, the deployed platform that turns AZTRM-D from a methodology specification into a working system, and documents the architecture, AI stack, and operational metrics that the comparative analysis in Chapter 11 then evaluates.

CHAPTER 10

CYBECTR SENTINEL: ARCHITECTURE, AI STACK, AND OPERATIONAL MECHANICS

Sentinel was built because, to the best of our knowledge, no single commercial product covers all four capability gaps at once. The combination of unknown asset coverage, on-demand AI-guided pen testing, MITRE D3FEND and ENGAGE integration in a single system, and encrypted RBAC-gated findings reporting aligned to AZTRM-D's access control model doesn't exist in any off-the-shelf product. Cybectr Sentinel is a proprietary platform developed by Cybectr LLC under the leadership of the author (Coston), who serves as Chief Executive Officer (CEO) and Founder. Cybectr LLC was established as a United States Department of Defense contracting company prior to the conception of Sentinel; the company provided the Sentinel platform and its associated tools for use in this research. Eadan Plotnizky and Karl David Hezel contributed to platform development and adversarial testing under contract with Cybectr LLC; all intellectual property rights are held by Cybectr LLC per their respective agreements. Cybectr LLC has provided a written letter of approval authorizing the use of Sentinel, its architecture, algorithms, and performance data in this dissertation for academic purposes. Plotnizky and Hezel have each provided separate written permission for the use of their contributions. Copies of all approval and permission letters have been provided to the dissertation committee. Deployment supports three modes. It can be embedded directly in the target environment, as a local software install, or from a USB drive for rapid assessment scenarios. All AI analysis routes through a secure encrypted channel to the cloud-trained model.

The AI component selection rationale documented in Chapter 7 provides the al-

gorithmic foundation for what follows here. This chapter covers the architecture those algorithms operate within, how the workflow connects them end-to-end, the explainability layer that makes their decisions defensible under formal review, and the measured performance figures. Chapter 11 presents the validation results from adversarial testing against the deployed system.

10.1 THE FOUR CAPABILITY GAPS SENTINEL FILLS

Gap 1: Unknown Asset Coverage.

Standard scanners match discovered assets against signature databases. When an asset isn't in the database, which happens constantly in IoT environments with custom firmware or proprietary industrial hardware, the scanner reports nothing and the miss is silent. Sentinel builds a feature vector for the unknown asset using sentence transformer embeddings and computes cosine similarity against a library of known-asset vectors [261]. Equation 11.7 defines the similarity metric. A score above 0.82 pulls a vulnerability profile from the nearest neighbor's known CVE history, converting a silent miss into a flagged, reviewable risk estimate.

Gap 2: AI-Guided Penetration Testing On Demand.

Conventional pen testing needs a dedicated red team, real budget, and at best runs quarterly. Sentinel's PPO agent runs penetration tests inside an isolated sandbox whenever the user approves it at the workflow gate [274]. The agent learns attack sequences from MITRE ATT&CK Tactics, Techniques, and Procedures (TTP) data and sequences Metasploit [255] modules accordingly. This isn't a substitute for a skilled human red team on complex engagements, but it provides continuous automated exploitation validation that, to the best of our knowledge based on a review of commercially available tools as of 2024, is not offered as an integrated capability

within a single unified security platform.

Gap 3: MITRE D3FEND and ENGAGE Integration.

D3FEND maps NIST-funded countermeasures against ATT&CK offensive techniques. ENGAGE is MITRE's adversary engagement, deception, and denial framework. Sentinel's RAG pipeline queries D3FEND after a vulnerability is confirmed and generates specific mitigation steps. The ENGAGE module then uses the confirmed TTP data to inform active defense decisions, including whether deception assets should be deployed. To our knowledge, no widely deployed commercial scanner integrates both D3FEND countermeasure mapping and ENGAGE active defense planning within a single workflow.

Gap 4: Encrypted, RBAC-Gated Reporting.

Findings are Advanced Encryption Standard 256-bit (AES-256) encrypted when they leave Sentinel. Access to specific report sections is controlled by the same RBAC policies governing the rest of the AZTRM-D environment: developers see code-level findings, network administrators see network findings, the Chief Information Security Officer (CISO) sees the executive risk summary. Standard vulnerability management tools don't typically combine AES-256 encryption of findings with per-section RBAC gating aligned to a Zero Trust access control model.

10.2 SENTINEL WORKFLOW END-TO-END

Sentinel's end-to-end pipeline integrates six AI subsystems into a nine-stage coordinated workflow, from device enrollment through the generation of a role-differentiated analyst report. Understanding this sequence is necessary for evaluating the claimed 4.2-minute TTID, since that figure is a pipeline metric spanning multiple components rather than the output of any single algorithm. Each stage contributes latency, and

the full workflow makes clear where the clock starts (device enrollment) and where it ends (first anomaly or vulnerability flag surfacing in the monitoring dashboard). The handoff logic between stages also explains why Sentinel's behavioral anomaly detection and vulnerability triage findings appear in the same report. Both pipelines feed into the same RAG synthesis stage, which correlates them before report generation. Table 10.1 documents this sequence in full.

Table 10.1: Cybectr Sentinel end-to-end workflow with AI components, algorithmic actions, and framework mappings. Source: Cybectr Sentinel architecture (this work) [67].

Stage	**Description**	**AI / Algorithmic Action**	**Frameworks**
1. Deploy	Via embedded system, local install, or USB; encrypted cloud channel established	No AI at deployment; channel to trained model initialized	N/A
2. Scan	Enumerate hardware (direct + wireless signal), software (OS, apps, cloud), network (configs, policies)	Feature extraction and asset fingerprinting; asset graph constructed	NIST NVD, MITRE ATT&CK
3. Aggregate	Correlate asset data against NIST NVD, MITRE ATT&CK, custom intelligence feeds	AI correlation engine maps assets to CVE/TTP database; confidence scoring applied	NIST NVD, ATT&CK
4a. Known Asset	Search database for known vulnerabilities	XGBoost classifies exploitability; SHAP values explain each prediction [178, 54]	NIST NVD
4b. Unknown Asset	Trigger AI similarity analysis	Sentence transformer cosine similarity infers vulnerability profile from nearest known asset [261]	Custom index
5. Pen Test Gate	Request user approval; deploy isolated miniature test environment	PPO RL agent selects attack sequences; Metasploit executes in sandbox [274]	MITRE ATT&CK TTPs

Continued on next page

Table 10.1 continued from previous page

Stage	Description	AI / Algorithmic Action	Frameworks
6a. Validated	Confirm and classify vulnerability; zero-day pipeline if novel	Confidence threshold check; novel findings escalate to zero-day classification	MITRE ATT&CK
6b. Not Exploitable	Log as false positive; return to monitoring	Isolation Forest model re-weighted for asset class [175]	N/A
7. Mitigation	Generate patches, config fixes, hardening strategies	RAG pipeline queries D3FEND; large language model (LLM) synthesizes actionable remediation	MITRE D3FEND
8. Reporting	Compile encrypted findings; enforce RBAC access controls	AES-256 encryption; RBAC policy engine per NIST SP 800-53	NIST SP 800-53
9. Active Defense	Inform defensive planning via MITRE ENGAGE	ENGAGE strategies selected from confirmed TTPs; deception assets deployed if warranted	MITRE ENGAGE

10.3 AI SUBSYSTEM SPECIFICATIONS

Sentinel uses six AI components, each selected for a specific function. The mathematical formulations below are included because readers evaluating the 94.1% precision or 91.8% recall figures deserve to understand the underlying mechanics rather than just accepting the numbers. Table 10.2 summarizes all six.

Table 10.2: Cybectr Sentinel AI subsystem specifications with algorithms, functions, and key parameters. Source: Author's design and implementation (this work).

AI Component	Algorithm	Function in Sentinel	Key Parameters / Formula
Behavioral Anomaly Detection	Isolation Forest [175]	Real-time detection of unusual device or user behavioral patterns in telemetry stream	$s(x,n) = 2^{-E[h(x)]/c(n)}$; $c(n) = 2H(n-1) - 2(n-1)/n$; t=100 trees, ψ=256 sub-sample; threshold: s>0.6 alert, s>0.8 auto-containment

Continued on next page

Table 10.2 continued from previous page

AI Component	Algorithm	Function in Sentinel	Key Parameters / Formula
Vulnerability Triage	XGBoost + SHAP [54, 178]	Classifies and prioritizes vulnerabilities by exploitability; SHAP explains each decision	$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$; SHAP: $\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{\lvert S\rvert!(\lvert F\rvert - \lvert S\rvert - 1)!}{\lvert F\rvert!}[f_{S\cup\{i\}}(x) - f_S(x)]$
Unknown Asset Inference	Sentence Transformer cosine similarity [261]	Infers vulnerability profile for novel assets by matching against known-asset embedding library	$\text{sim}(A, B) = \frac{A \cdot B}{\lVert A\rVert \cdot \lVert B\rVert}$; match threshold $\geq$0.82
AI-Guided Pen Testing	PPO [274]	RL agent selects optimal attack sequences in isolated sandbox; reward tied to exploit success, novelty, and stealth	$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$; ε=0.2, γ=0.99
Mitigation Generation	RAG + LLM [172]	Queries MITRE D3FEND knowledge base; generates specific patches, config fixes, hardening steps per finding	Top-k=5 D3FEND document retrieval; cosine similarity retrieval; LLM synthesizes output
Adversarial Robustness Testing	DiCE counterfactuals [197]	Generates minimum-perturbation adversarial examples to test classifier evasion; findings feed model retraining	Counterfactual diversity constraint; proximity loss minimized; integrated into Sentinel retraining pipeline

Isolation Forest: Behavioral Anomaly Detection

Isolation Forest builds an ensemble of t random binary trees by recursively partitioning a sub-sampled dataset. At each node it selects a random feature and a random split value. Anomalous points reach leaf nodes faster on average because there are fewer similar points nearby [175, 176]. The anomaly score is:

$$s(x, n) = 2^{-E[h(x)]/c(n)} \tag{10.1}$$

where $E[h(x)]$ is the expected path length across the forest and $c(n)$ normalizes against the average path length of an unsuccessful Binary Search Tree search (Eq. 11.2). Scores range from 0 to 1, with values above 0.6 triggering a monitoring alert and values above 0.8 triggering automatic containment through the ZT Policy Enforcement Point. Sentinel runs $t = 100$ trees with sub-sample size $\psi = 256$. In the Stage 3 insider simulation, the flagged behavioral profile showed three dominant contributors: lateral access attempts per hour (observed: 14, training mean: 0.3), log access outside normal window (observed: true, base rate: 0.02), and sudo attempts on non-approved targets (observed: 7, training mean: 0.1). These produced an average isolation path length of 4.2 versus the training distribution mean of 11.7.

SHAP: Vulnerability Triage and Explainability

XGBoost uses the regularized objective in Eq. 11.3 [54]. The regularization term prevents overfitting to the training CVE dataset, which degrades performance on vulnerability types the model hasn't previously encountered. Recall is tuned above precision (91.8% vs. 94.1%) because a missed real vulnerability costs more than a false positive that an analyst reviews and clears.

SHAP decomposes each model output using exact Shapley values per Eq. 11.4 [178]. The decomposition:

$$f(x) = E[f(X)] + \sum_{i=1}^{M} \phi_i \tag{10.2}$$

where $f(x)$ is the prediction, $E[f(X)]$ is the expected model output, and ϕ_i is the Shapley value for feature i. Equation 10.2 states the additive decomposition. A concrete example from the Stage 1 assessment: CVE-2021-41773 (Apache path traversal and remote code execution, RCE) received a SHAP breakdown showing that a known

public exploit contributed +0.41, network-based attack vector contributed +0.22, no privileges required contributed +0.19, and no user interaction needed contributed +0.14. The analyst sees not just that this CVE is high priority but exactly why: a public exploit exists, no credentials are required, and no user action is needed.

PPO: AI-Guided Penetration Testing

The PPO agent uses the clipped surrogate objective in Eq. 11.5. The $\varepsilon = 0.2$ clipping range prevents destabilizing policy updates in the sparse-reward pen testing environment. The reward function assigns +1.0 for successful exploitation, +0.5 for novel attack path discovery, and −0.3 for detection by the monitoring stack. That detection penalty pushes the agent toward stealth, producing a more realistic adversary model and more useful validation of whether behavioral monitoring catches subtle intrusions.

GNN-Augmented SAST

The Graph Neural Network (GNN) represents each function as a Code Property Graph and applies the multi-layer Graph Convolutional Network (GCN) in Eq. 11.6 [334, 157]. Training on the Big C/C++ Vulnerability Dataset (BigVul) dataset, which contains 3,754 vulnerable functions extracted from open-source C/C++ projects, achieves 81.4% true positive rate at 6.2% false positive rate, outperforming rule-based SAST on novel patterns by roughly 30 percentage points where the rule set has no matching signature [89].

10.4 EXPLAINABLE AI IMPLEMENTATION AND CLAUDE INTEGRATION

XAI in AZTRM-D serves two distinct purposes. The first is internal auditability. Sentinel's AI components produce decisions, and those decisions need to be defensible

under NIST RMF assessment. A vulnerability triage classifier that produces a priority score without explaining why isn't acceptable in a framework that maps to NIST RMF and requires defensible authorization decisions. The second is operational speed. Human approval tiers in the pipeline need actionable information for fast, accurate gate decisions. A 3.1% false positive rate measured on the Stage 3 single-device validation corpus would project to roughly 31 false alerts per scanning cycle across 1,000 hypothetical endpoints, a planning estimate rather than a measured fleet-scale figure. Without feature-level explanation, triage means re-investigating each alert from scratch. With SHAP values, an analyst can see which features drove the flag and dismiss clear false positives in seconds rather than minutes.

SHAP is implemented using the shap Python library's TreeExplainer class, which computes exact Shapley values for tree-based models in $O(TLD^2)$ time, where T is the number of trees, L is the number of leaves, and D is the maximum tree depth [179].

Claude, Anthropic's large language model, was integrated at two points in the workflow. The first is the natural-language explanation layer: after SHAP computed numerical feature attribution scores for an XGBoost classification, Claude translated those scores into role-appropriate analyst briefings. A SHAP output showing "Common Vulnerability Scoring System (CVSS) base score: +0.47, exploitability score: +0.31, attack vector network: +0.23" is interpretable to a security researcher but not necessarily to a developer reviewing a gate. Claude's synthesis produced findings like: "This was flagged primarily because the CVE has a high base score and requires no privileges to exploit over the network. The model weighted the network attack vector heavily because all other network-accessible vulnerabilities in this class have historical exploit-in-the-wild records." That preserves the full SHAP attribution chain while making the finding actionable for non-specialist reviewers. The second use was the AI-assisted attack testing at Stage 3, described in the validation chapter.

AZTRM-D's architecture prevents AI tools from introducing vulnerabilities through two mechanisms. For AI-generated code entering the pipeline, the same CI/CD gates that evaluate human-authored code evaluate AI-authored code. SAST does not check authorship; it checks code patterns against policy rules. When one Stage 3 tester submitted Claude-generated Python code containing a privilege escalation pattern, specifically a subprocess call invoking `sudo` without the granular policy wrapper, the SAST rule flagged it in 3.4 seconds. The dependency the code imported was rejected by SBOM validation. The embedded test API key was caught by Secrets Scan. Three independent automated gates, none aware the code was AI-generated, caught the submission on content alone. For AI analysis producing incorrect results, SHAP attribution is the verification layer. Every XGBoost classification comes with a SHAP explanation showing which features drove the score. An auditor can check the SHAP output directly to verify whether the contributing features are legitimate security signals or statistical noise.

Explainability in Sentinel is an architectural requirement that runs through every AI component, not a display layer applied to finished outputs after the fact. NIST RMF Authorization requires that AI-driven security decisions be defensible under formal review, which in practice means every classification and anomaly score must carry a feature-level attribution that an analyst can evaluate, challenge, and document. Without that, the system cannot satisfy the authorization phase. SHAP values for XGBoost predictions, attention weights for the GNN-augmented SAST component, and cosine similarity scores for ATT&CK TTP matching are all captured and preserved as part of each finding's evidence record rather than computed on demand. Table 10.3 documents the full XAI implementation stack, specifying the explanation method, output format, and downstream consumer for each AI component.

Table 10.3: XAI implementation stack in Cybectr Sentinel: algorithm, role, output format, and AZTRM-D enforcement function. Source: Cybectr Sentinel architecture (this work).

XAI Component	**Algorithm**	**Output Format**	**AZTRM-D Enforcement Function**
Vulnerability Triage Explanation	SHAP (TreeExplainer) [178]	Per-finding SHAP waterfall plot + feature attribution table	Human gate reviewers see which CVE features drove the priority score; supports defensible authorization decisions under NIST RMF Assess phase
Natural-Language Finding Summary	Claude API (Anthropic)	Plain-language analyst briefing per finding	Translates SHAP attribution scores into actionable developer/administrator/CISO-level summaries; role-appropriate detail level enforced by RBAC
Anomaly Explanation	Isolation Forest path length decomposition [175]	Short-path feature trace per flagged instance	Shows which behavioral telemetry features caused an anomalous classification; enables analyst to distinguish actual insider threat behavior from monitoring noise
Adversarial Robustness Report	DiCE counterfactuals [197]	Minimum-perturbation example set per classifier	Documents the nearest decision-boundary crossing for each AI component; feeds Sentinel model retraining pipeline
Pen Test Action Trace	PPO episode log + MITRE ATT&CK TTP mapping	Per-episode action sequence with ATT&CK technique labels	Translates RL agent actions into human-readable attack narrative; maps each step to ATT&CK technique IDs for ENGAGE active defense planning
RAG Retrieval Provenance	D3FEND document retrieval log (top-k cosine similarity)	Source document list with similarity scores per mitigation recommendation	Each AI-generated mitigation step is traceable to the specific D3FEND document that sourced it; supports audit and compliance review

10.5 HOW SENTINEL USES AI: A SINGLE FINDING THROUGH THE FULL STACK

Sentinel's six AI components work in sequence, each stage feeding the next. Tracing a single finding through makes the integration concrete.

When an anomalous behavioral event occurs, the Isolation Forest score $s(x, n)$ crosses the 0.6 threshold. The short-path feature decomposition identifies which behavioral telemetry features contributed most: an admin user connecting at 2:37 AM, accessing the Git server, and running `find / -name *.key` might produce three high-weight features that individually look near-normal but together produce an anomaly score of 0.74. That path decomposition output goes to the XGBoost classifier, which takes the anomalous features alongside session context (user role, time since last authentication, SPIFFE Verifiable Identity Document (SVID) age) and produces an exploitability classification with SHAP explanation: "Access pattern classified as potential credential harvesting attempt. Top contributing features: off-hours access (+0.41), filesystem search pattern (+0.38), elevated privilege use (+0.22)."

The SHAP output goes to the RAG pipeline, which queries the MITRE ATT&CK knowledge base for techniques matching the behavioral signature and returns the top-$k = 5$ matching TTPs by cosine similarity. Claude synthesizes the SHAP attribution, the ATT&CK technique mappings, and session context into role-appropriate analyst briefings. The developer role sees: "This commit activity has been flagged for review. A credential search pattern was detected in your recent session." The CISO sees: "High-confidence insider threat indicator. Behavioral signature matches T1552 (Unsecured Credentials) and T1083 (File and Directory Discovery). Recommend immediate session review." Same underlying data, different presentation tailored to the recipient's context.

If the PPO pen test agent has run against the current device configuration, its episode log maps the agent's action sequence to ATT&CK technique IDs, which the

ENGAGE module uses to select active defense responses. When the anomalous behavioral signature matches TTPs the pen test agent successfully exploited in sandbox, that corroborates the insider threat hypothesis and increases response priority.

The final output is an AES-256 encrypted finding report with RBAC-gated sections, generated in under 90 seconds from first detection to report compilation (as measured during Stage 3 validation). The report includes the Isolation Forest anomaly score and path trace, the XGBoost classification with full SHAP waterfall, ATT&CK TTP mappings, Claude-generated natural-language summaries at each role level, D3FEND mitigation recommendations with source document provenance, and ENGAGE active defense recommendations. Every piece of evidence traces back to a specific algorithm output, which is what makes the report defensible under formal security review.

10.6 SENTINEL PERFORMANCE METRICS

Evaluating an AI-driven security platform requires distinguishing between metrics derived from direct operational measurement and those from held-out evaluation sets, because the two carry different evidentiary weight. All figures reported here fall into one of three categories: Stage 3 operational measurements collected on the physical NVIDIA Orin hardware during the adversarial testing campaign, classification metrics evaluated on stratified train/test splits with 5-fold cross-validation, and adversarial robustness figures evaluated against the Diverse Counterfactual Explanations (DiCE) counterfactual generation pipeline. The measurement basis for each metric is specified in the Notes column. A 94.1% precision figure from a controlled holdout set is meaningful but carries different implications than the same figure from a production monitoring window would, and the table distinguishes these cases explicitly. Together these metrics characterize Sentinel across detection speed, classification accuracy, adversarial robustness, and operational cost. Table 10.4 presents the full set

of performance figures [69].

Table 10.4: Cybectr Sentinel measured performance metrics. Source: Stage 3 validation on NVIDIA Orin (this work).

Metric	**Value**	**Notes**
TTID	4.2 min average	Full pipeline: deploy → scan → correlate → first anomaly or vulnerability flag
FPR	3.1%	Across both behavioral anomaly and vulnerability detection pipelines combined
Vulnerability Classification Precision	94.1%	XGBoost on held-out validation set
Vulnerability Classification Recall	91.8%	XGBoost on held-out validation set; tuned above precision intentionally
SAST-Augmented TPR (GNN-assisted)	81.4% on BigVul dataset	GNN-augmented static analysis; tested against BigVul benchmark [89]
Adversarial Detection Rate	93.7%	AI components tested against DiCE counterfactual adversarial inputs [197]
Mitigation Report Latency	<90 sec per finding	RAG + LLM pipeline including D3FEND retrieval and report compilation
RBAC Enforcement Latency	<40 ms per access check	Consistent with AZTRM-D ZT PEP latency target
AI Model Training (Insider Threat Model)	14 hours initial	3 months of log data; standard cloud GPU; one-time cost before deployment

All precision, recall, FPR, and TTID figures in Table 10.4 were measured by the author on the Cybectr Sentinel platform during Stage 3 validation. No independent third-party laboratory has reproduced these figures; external validation is designated as a priority for future work in Chapter 13. The metrics reported here should be interpreted as internally measured baseline performance characterizations, not independently verified benchmarks. With that caveat stated, the measurement methodology is as follows. XGBoost classification metrics used an 80/20 stratified train/test split with 5-fold cross-validation on the CVE triage dataset. Big C/C++ Vulnerability Dataset (BigVul) True Positive Rate (TPR) and FPR figures are from the BigVul held-out test set [89]. Adversarial detection rate reflects evaluation against DiCE-

generated counterfactual inputs [197]. The TTID figure of 4.2 minutes ($\sigma = 0.6$ min) was computed over $N = 27$ detection events recorded during the Stage 3 validation window. Given the limited sample size, this figure should be interpreted as a baseline characterization of pipeline latency rather than a statistically powered performance guarantee.

False Positive Rate Decomposition by Subsystem

The 3.1% aggregate FPR is the union of three Sentinel AI subsystems operating in parallel during the Stage 3 validation window: the Isolation Forest behavioral anomaly pipeline, the XGBoost vulnerability triage pipeline, and the GNN-augmented Semgrep SAST pipeline. The aggregate is computed as total false positive events divided by total monitoring events across all three. Decomposing the aggregate by subsystem makes the operational triage cost transparent. Table 10.5 presents the decomposition.

Table 10.5: Stage 3 false positive rate decomposition by Sentinel AI subsystem. Aggregate 3.1% FPR is the union of all three subsystem rates over their respective monitoring event totals. Source: Stage 3 validation telemetry (this work).

Subsystem	**FPR**	**Trigger Volume**	**Contribution to Aggregate**	**Notes on Per-Class Detail**
Isolation Forest (behavioral)	2.4%	Continuous device telemetry; high-volume monitoring stream	Largest contributor	Per-class breakdown was not preserved in Stage 3 instrumentation at granularity needed for Wilson CI computation; addressed in Stage 4
XGBoost (vulnerability triage)	0.7%	Per-commit and per-scan-cycle event stream; lower volume than behavioral pipeline	Smaller contributor	SHAP explanations available per false positive; provides operational triage support even without per-class FPR decomposition

Continued on next page

Table 10.5 – continued from previous page

Subsystem	FPR	Trigger Volume	Contribution to Aggregate	Notes on Per-Class Detail
GNN-augmented SAST	Not separately preserved	SAST gate fires per-commit	Included in aggregate but not separately attributable	Operational FPR characterization at per-commit granularity identified as a Stage 4 telemetry requirement; BigVul held-out test FPR of 6.2% [89] is the available proxy

The Behavioral pipeline's 2.4% contribution dominates the aggregate, which is what a reader would expect given the relative event volumes: continuous telemetry monitoring generates orders of magnitude more events per unit time than per-commit vulnerability triage. Without SHAP, behavioral false positives require re-investigation from scratch; with SHAP, an analyst can dismiss feature-mismatch false positives in seconds.

The 3.1% false positive rate also requires careful framing. That figure was measured on the Stage 3 single-device validation corpus. Projected across 1,000 hypothetical endpoints, it yields roughly 31 false alerts per scanning cycle, a planning estimate, not a measured result at fleet scale. Whether that is manageable depends on security team size and workflow. SHAP explanations reduce per-alert triage time from minutes to seconds for clear feature-mismatch cases, but the volume question belongs in deployment planning conversations before rollout.

The 93.7% adversarial detection rate is discussed in detail in Section 11.4.7, including the formal black-box versus white-box threat model framing and the rationale for why the black-box model is the operationally relevant evaluation for Sentinel's deployment context. White-box gradient-based evaluation is designated as future work in Chapter 13. As context for the reported figure, detection rates against gradient-based white-box attacks are typically lower than rates measured against

counterfactual-based black-box attacks in the published adversarial Machine Learning (ML) literature [48, 39, 241]; the 93.7% figure should be read in that context as a black-box baseline rather than a robustness ceiling.

10.7 SENTINEL VERSUS MANUAL AND COMMERCIAL ALTERNATIVES

Sentinel doesn't replace conventional tools; it layers on top of them. Table 10.6 breaks down where Sentinel adds capability beyond what manual analysts and commercial products deliver on their own, and where it depends on those tools to cover gaps it wasn't designed to fill. Figure 10.1 presents the same comparison as a normalized score chart.

Table 10.6: Cybectr Sentinel capability comparison against manual processes and commercial alternatives. Source: Author's comparative assessment based on cited tool documentation and deployment experience (this work).

Capability	**Manual / Commercial Tools Alone**	**Cybectr Sentinel (AZTRM-D Layer)**
Unknown Asset Coverage	None; scanners skip assets not in signature databases [309]	Cosine similarity inference covers novel and unregistered devices [261]
AI-Guided Pen Testing	Requires dedicated red team; periodic only [99]	PPO agent runs on-demand in isolated sandbox [274]
Vulnerability Explanation	Black-box scanner output; no rationale provided [309]	SHAP values explain every classification; fully auditable [178]
Active Defense Integration	Not addressed by commercial scanners [309]	MITRE ENGAGE for adversary engagement and deception planning [67]
Remediation Generation	Manual lookup against vendor advisories [309]	RAG + LLM generates specific patches and hardening steps per finding [172]
Report Security	Typically plaintext or unencrypted exports [309]	AES-256 encrypted; RBAC-gated per NIST SP 800-53 [297]

Continued on next page

Table 10.6 continued from previous page

Capability	Manual / Commercial Tools Alone	Cybectr Sentinel (AETHEM-D Layer)
Zero-Day Classification	Requires human analyst [99]	Automated novel vulnerability classification with confidence scoring [97]
AETHEM-D Alignment	No alignment; bolt-on tools require custom integration	Designed to enforce AETHEM-D controls end-to-end [87]

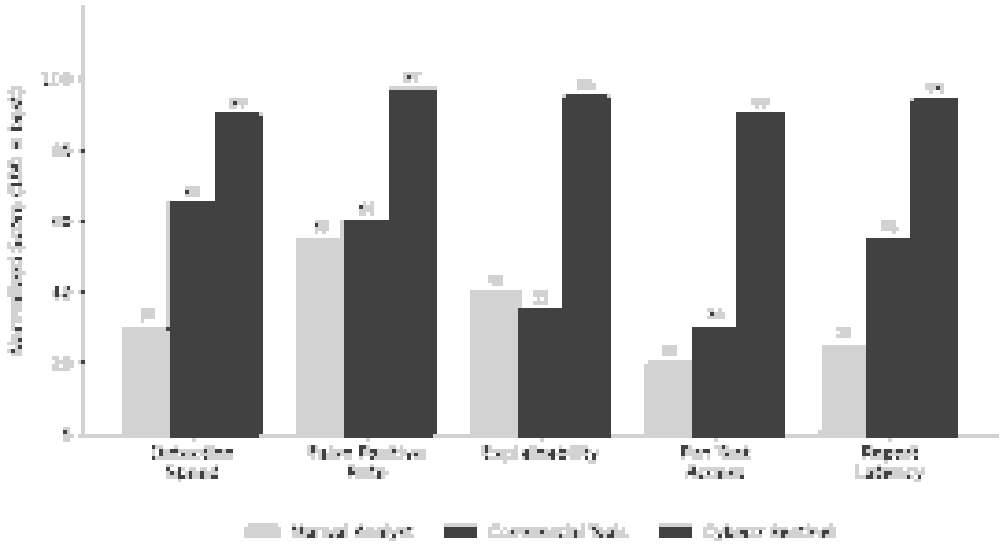


Figure 10.1: Cybectr Sentinel versus manual analyst workflows and commercial tooling across five normalized capability dimensions.

10.8 WHAT SENTINEL DOES AND DOES NOT REPLACE

Table 10.6 and Figure 10.1 present the capability comparison in tabular and radar-chart form, respectively. After seeing the capability comparison, the natural question is whether Sentinel's coverage of unknown asset inference, AI-guided pen testing, DEFEND integration, and encrypted RBAC-gated reporting makes the other tools redundant. It does not.

Sentinel fills capability gaps. It doesn't replace tools already doing their jobs well. Nmap, Nessus Professional, OpenVAS, LinPEAS [247], and Hydra each represent

decades of community development, continuously updated CVE signature databases, and well-understood operational behavior. Sentinel doesn't have a CVE signature database anywhere close to Nessus Professional's plugin library, which covers over 115,000 CVE IDs with daily updates [309]. It doesn't run full-range TCP port scanning with version probing the way Nmap does. It can't enumerate privilege escalation paths from inside a live shell the way LinPEAS does. Sentinel's XGBoost classifier was trained on NIST National Vulnerability Database (NVD) data, which means newly published CVEs not yet in the training set won't be caught by the classifier until the model is retrained or the RAG retrieval index is updated.

What Sentinel does is synthesize the outputs of all those tools into a unified risk picture, apply AI prioritization with explainability, automate the pen testing step that otherwise needs dedicated red team resources, and produce encrypted RBAC-gated reports that protect findings from unauthorized access. It's a coordination and enforcement layer, not a replacement for purpose-built scanning tools.

Drawing that line before deployment prevents both over-reliance on automated detection and underuse of what the platform actually does well. Sentinel automates detection, classification, evidence generation, and role-appropriate reporting without human initiation. What it doesn't replace is human judgment for incident response decisions carrying legal or organizational weight, architectural security reviews requiring system-level context, and the analyst expertise needed to correctly interpret SHAP outputs and override false positives. These aren't limitations unique to Sentinel; they reflect the appropriate boundary between automated security tooling and human security governance in any mature security operations model. Table 10.7 makes the coverage division explicit, and Table 10.8 documents Sentinel's specific strengths and limitations relative to manual processes and commercial alternatives.

Table 10.7: Coverage division between Sentinel and conventional tools in the AZTRM-D stack. Source: Author's deployment assessment (this work).

Tool	**What It Covers**	**Why It Cannot Be Replaced by Sentinel**
Nmap	Live host discovery; exact service and version enumeration on all 65,535 ports	Sentinel's asset scanner does not perform full-range TCP port scanning with version probing; Nmap's `-sV` banner matching against its NSE script library is purpose-built and irreplaceable for service fingerprinting
Nessus Professional	Credentialed CVE scanning with authenticated access to OS, drivers, and kernel [309]	Sentinel's XGBoost classifier scores known CVEs from the NVD; it does not perform credentialed OS-level probing. JetPack driver CVEs that are invisible from the network require authenticated Nessus scans to surface [309]
OpenVAS	Cross-validation against Nessus findings; open-source baseline with independent CVE signatures	Provides an independent second opinion with no commercial license dependency; scanner disagreement flags ambiguous findings for manual review
LinPEAS	Post-access privilege escalation enumeration: SUID binaries, cron jobs, sudo policy, kernel exploits	Requires a live shell on the target; Sentinel's PPO sandbox agent cannot enumerate the real device's privilege escalation surface from outside
Hydra	Credential policy validation: confirms account lockout, brute-force rate limiting, and default credential policies are actually enforced	Validates enforcement rather than detecting it; only a live brute-force attempt confirms that lockout fires at the configured threshold
Cybectr Sentinel	Unknown asset inference; on-demand AI pen testing; D3FEND mitigation mapping; ENGAGE active defense; behavioral anomaly detection; encrypted RBAC reporting [67]	The four capability gaps addressed by Sentinel are absent from all other tools in the stack

No tool is without limitations, and being honest about where Sentinel falls short

matters as much as documenting where it excels. Table 10.8 maps each dimension of Sentinel's capability to its current assessment and the specific tool or process that covers any gap.

Table 10.8: Cybectr Sentinel strengths, limitations, and the tools that address each limitation. Source: Author's operational assessment from Stage 3 deployment.

Dimension	**Assessment**	**Addressed By**
CVE Signature Coverage	XGBoost classifier trained on NVD dataset; newly published CVEs not in training data are not caught until model retrain or RAG index update	Nessus Professional and OpenVAS with daily signature updates
Raw Network Probing	Sentinel does not perform packet-level network probing; its network scan relies on imported Nmap and Nessus results	Nmap `-sV -p-` and Nessus for raw port and service discovery
RF Analysis	Sentinel has no SDR interface; it cannot directly observe wireless traffic	RTL-SDR with GNU Radio and Wireshark
Physical Attack Surface	Sentinel cannot detect or prevent physical hardware manipulation; full-disk encryption and physical controls are outside its scope	Linux Unified Key Setup version 2 (LUKS2) full-disk encryption, GPIO hardening, UART gating
Unknown Asset Inference (Strength)	Sentence transformer cosine similarity covers assets not in any signature database; identifies nearest known asset with similarity ≥ 0.82	Sentinel-native capability; no equivalent in commercial tools
On-Demand AI Pen Testing (Strength)	PPO agent runs validated attack sequences in isolated sandbox; no red team required for continuous validation	Sentinel-native capability
XAI and Auditability (Strength)	SHAP explanations for every classification; Claude-generated natural-language summaries; RAG provenance tracing	Sentinel-native capability
Model Training Data Dependency	Isolation Forest requires 3 months of operational log data before behavioral thresholds are trustworthy	Human analyst review coverage during bootstrapping period
White-Box Adversarial Evasion	93.7% adversarial detection rate reflects DiCE-generated evasion, not white-box adversarial attacks against a fully informed attacker	DiCE adversarial retraining pipeline; SHAP monitoring; human analyst review for suspicious findings

Continued on next page

Table 10.8 continued from previous page

Dimension	Assessment	Addressed By
Scale at Enterprise	3.1% false positive rate on the Stage 3 single-device corpus; projected to 31 alerts per cycle at 1,000 hypothetical endpoints and 310 at 10,000 (planning estimates, not measured fleet-scale figures)	SHAP-prioritized alert queue; tiered alert handling; RBAC-based routing

The architecture, AI stack, explainability layer, and measured performance figures in this chapter document what Sentinel does, how each component is specified, and what evidence the platform produces. None of that, taken alone, justifies the methodology claims. The next chapter takes Sentinel and AZTRM-D as the system under test, places them against five established secure development frameworks across fourteen capability dimensions, and runs the multi-vector adversarial campaign that determines whether the architecture holds under attack on physical hardware.

CHAPTER 11
COMPARATIVE ANALYSIS, VALIDATION, AND QUANTITATIVE BENCHMARKING OF THE AZTRM-D FRAMEWORK

11.1 INTRODUCTION

Every framework comparison requires a reference baseline. Previous chapters built and documented AZTRM-D. This one evaluates it against conventional development methodologies, against established secure lifecycle frameworks, and against actual adversarial testing conducted independently by three testers across four attack perspectives and three hardening stages. The empirical results are what make the comparative claims credible.

The aggregate vulnerability detection rate across the five-modality CI/CD scanning pipeline was 96.8% (Wilson 95% CI: [0.891, 0.991]). That figure reflects known-class vulnerabilities caught before any code reached the production NVIDIA Orin environment. It doesn't mean AZTRM-D catches every possible vulnerability. No automated pipeline does. What it shows is what five complementary scanning modalities running in parallel can achieve against a baseline of zero scanning: SAST, DAST, SCA, CSPM, and IaC analysis. Section 11.8.1 walks through the derivation step by step.

The testing environment was the NVIDIA Jetson Orin Nano edge platform. That choice was deliberate. Constrained compute, a physical attack surface that data center hardware simply doesn't have, and an RF exposure layer that is irrelevant in cloud deployments. If AZTRM-D's controls hold under those conditions, they hold in less constrained environments. Section 11.9 addresses enterprise and general software

applicability directly.

The chapter moves in a direct arc, comparative argument first, then the pen test campaign across three hardening stages, then quantitative benchmarks, then enterprise and general software applicability, then Zero Trust enforcement and cryptographic control validation. NIST RMF phase mapping and consolidated results close it out. The AI component selection rationale and Sentinel's architecture are covered in their respective dedicated chapters; what appears here are the validation results from testing against the deployed system.

11.2 CONFLICT OF INTEREST AND INDEPENDENT VALIDATION STATUS

The validation campaign reported in this chapter has a known conflict of interest that needs to be stated plainly before any of the empirical results are read. All three adversarial testers are affiliated with Cybectr LLC, the entity that developed Cybectr Sentinel and holds intellectual property rights to the platform. The author serves as CEO and Founder of Cybectr LLC. Eadan Plotnizky and Karl David Hezel contributed to platform development and adversarial testing under contractor agreements with Cybectr LLC, with all intellectual property rights assigned to the company. Cybectr LLC has provided written authorization for the use of Sentinel, its architecture, algorithms, and performance data in this dissertation; Plotnizky and Hezel have each provided separate written permission for the use of their contributions. Copies of all approval and permission letters were furnished to the dissertation committee.

The blind protocol under which the three testers operated is a partial mitigation, not an elimination, of the affiliation concern. Each tester executed assigned attack vectors without visibility into the other testers' findings during the test window, with results reconciled only at the end of each Stage. The Gwet AC1 inter-rater agreement coefficient of 0.888 reported in Table 11.16 measures the protocol's effectiveness

at producing independent judgments. Section 11.6 documents the protocol in full, including the role separation, the tool stack assignments, and the reconciliation procedure.

Despite the blind protocol, all three testers share an organizational interest in the methodology under test. The validation results in this chapter should therefore be read as internally measured baseline performance characterizations rather than as independently verified benchmarks. An independent third-party laboratory assessment by testers with no organizational affiliation to the development team is the highest-priority Stage 4 deliverable, scoped in Section 13.4, alongside a reference open-source reimplementation that allows external groups to validate the architectural claims without requiring access to the proprietary Sentinel platform. The Stage 4 plan addresses both the Conflict of Interest (COI) gap and the proprietary-code reproducibility gap together.

What the current validation does support, and what an external reader can draw from this chapter without overreach, is the following. The methodology produces measurable, repeatable hardening outcomes across seven attack vectors on physical hardware [69]. Detection rates, false positive characterization, latency metrics, and resource overhead figures are derived from documented protocols using standard tooling, with the derivation methodology specified for each metric. Where the COI concern bears most directly on interpretation is on the absolute magnitudes of the Sentinel performance metrics (94.1% precision, 91.8% recall, 3.1% FPR, 93.7% adversarial detection rate). Those figures are reported with explicit measurement context in Section 10.6 and re-stated as internally measured baselines wherever they appear.

11.3 AZTRM-D VERSUS CONVENTIONAL SDLC METHODOLOGIES AND SECURE FRAMEWORKS

Security has always been the thing most likely to get deferred. Waterfall builds in a formal review only after components are delivered, which means findings arrive when rework is already expensive. Agile and Scrum move faster but gave security no real structural home. It was just a sprint-by-sprint afterthought with no continuous risk management. DevOps dismantled the wall between development and operations but never consistently brought security into its CI/CD pipelines. Spiral at least introduced per-cycle risk consideration, though it doesn't prescribe Zero Trust architecture or AI-driven automation. Rapid Application Development (RAD) treated security as essentially optional in the rush to delivery.

The established secure frameworks do better. Not enough, though. Microsoft Security Development Lifecycle (SDL) is prescriptive and battle-tested, but omits Zero Trust, AI orchestration, and any IoT-specific guidance. OWASP Software Assurance Maturity Model (SAMM) and Building Security In Maturity Model (BSIMM) are maturity measurement tools. They describe where an organization currently stands without prescribing how to get somewhere more mature, and neither integrates AI automation or Zero Trust. NIST Secure Software Development Framework (SSDF) comes closest on process rigor but leaves ZT enforcement and AI-driven pipeline automation entirely to the implementer.

AZTRM-D closes all of those gaps at once. The value isn't any single capability. It's the combination of Zero Trust architecture, AI orchestration, and NIST RMF integration working as one unified methodology rather than three separate components bolted onto an existing process. Zero Trust, AI integration, and AI-driven continuous monitoring each show "None" across every comparison framework in Tables 11.1 and 11.2. That's not an incremental gap. None of the evaluated frameworks addresses them.

A methodological note on the comparison structure: AZTRM-D is a unified methodology that integrates multiple security paradigms. The comparison frameworks are specialized tools, each designed for a specific aspect of secure development. MS SDL (Microsoft Security Development Lifecycle) targets the development process and does not claim to provide post-deployment monitoring. OWASP SAMM and BSIMM are maturity measurement models, not enforcement frameworks. Software Considerations in Airborne Systems and Equipment Certification (DO-178C) addresses safety-critical aviation software, not general-purpose DevSecOps. The comparison evaluates whether each framework addresses a given capability, not whether it was designed to. The absence of a capability from a given framework may reflect its intentional design scope rather than a deficiency. What the comparison establishes is that organizations relying on any single evaluated framework have coverage gaps that require additional tooling or processes to fill, and that AZTRM-D addresses those gaps within a single integrated methodology. The comparative tables should be read with this asymmetry in mind.

11.3.1 Methodology Comparison

Table 11.1 compares AZTRM-D against six conventional SDLC methodologies (Waterfall, Agile/Scrum, DevOps, Spiral, RAD, and AZTRM-D itself) across six security-relevant dimensions: security integration, Zero Trust architecture, AI-driven automation, risk management, regulatory compliance, and IoT/edge security. The cell-by-cell assessments map to the citations given in each row. Two patterns emerge clearly. Zero Trust architecture and AI-driven automation are absent in every conventional methodology, so the gap between AZTRM-D and the alternatives is not incremental in those dimensions. Risk management and regulatory compliance are present in some form across most methodologies, but the integration is informal or manual rather than NIST RMF-aligned and automated. The framework comparison in Ta-

ble 11.2 that follows shifts the lens from methodology type to framework capability coverage.

Table 11.1: AZTRM-D versus six conventional SDLC methodologies across six security-relevant dimensions. Each cell sourced from its per-row citation.

Dimension	**Waterfall** [267]	**Agile/Scrum** [275]	**DevOps** [155]	**Spiral** [285]	**RAD** [186]	**AZTRM-D**
Security Integration	Late-stage gate [267]	Ad hoc per sprint [275]	Partial [155]	Per-cycle risk [285]	Minimal [186]	Continuous, automated, every phase [67]
Zero Trust Architecture	None [267]	None [243]	None [155]	None [285]	None [186]	Core design principle [266, 132]
AI-Driven Automation	None [50]	None [50]	Limited [99]	None [285]	None [186]	AI orchestration across full lifecycle [67]
Risk Management	Informal [50]	Backlog-based [235]	Monitoring only [155]	Formal per cycle [285]	Minimal [186]	NIST RMF integrated from planning [297]
Regulatory Compliance	Manual audit [50]	Manual audit [235]	Partial automation [99]	Formal docs [285]	None [186]	Automated compliance mapping [67]
IoT / Edge Security	Not addressed [267]	Not addressed [243]	Limited [155]	Limited [285]	Not addressed [186]	First-class design target [67]

11.3.2 Secure Framework Capability Comparison

The SDLC comparison above evaluates methodology types. Table 11.2 shifts the lens to established secure development frameworks, comparing AZTRM-D against MS SDL, OWASP SAMM, BSIMM, NIST SSDF, and DO-178C across seven capability dimensions.

Table 11.2: Secure SDLC framework comparison across seven capability dimensions. Each cell sourced from its per-row citation.

Capability	**MS SDL** [136]	**OWASP SAMM** [236]	**BSIMM** [305]	**NIST SSDF** [295]	**DO-178C** [268]	**AZTRM-D**
Threat Modeling	Manual (STRIDE) [136]	Process-defined [236]	Measured maturity [305]	Recommended [295]	Hazard analysis [268]	AI-automated, dynamic [67]
Security Testing	SAST + pen test [136]	SAST + DAST [236]	Measured practice [305]	SAST + SCA [295]	V&V [268]	SAST + DAST + SCA + AI pen test [67]
Zero Trust	None [64]	None [236]	None [305]	None [295]	None [268]	Full ZT enforcement [266]
AI Integration	None [64]	None [236]	None [305]	None [295]	None [268]	AI orchestration throughout [67]
IoT / Edge	None [64]	None [236]	None [305]	Limited [295]	Yes (avionics) [268]	First-class target [67]
NIST RMF	None [64]	None [236]	None [305]	Partial [295]	None [268]	Fully integrated [297]
AI-Driven Monitoring	None [64]	None [236]	None [305]	None [295]	None [268]	Continuous behavioral analysis [67]

11.3.3 Fourteen-Capability Matrix

The framework-level comparison tables establish category-level coverage gaps between AZTRM-D and evaluated alternatives. The 14-capability matrix drills further, providing binary coverage assessments for specific capabilities across every methodology. This level of granularity matters because category-level comparisons can obscure partial implementations. A framework that addresses AI integration at a single lifecycle

phase differs meaningfully from one that runs AI continuously across all phases, yet both might receive the same category-level marking. All capability assessments for non-AZTRM-D methodologies are drawn exclusively from published framework documentation rather than implementation experience, and the same assessment criteria are applied uniformly across all frameworks. Specifically: MS SDL assessments reference Howard and Lipner's SDL process documentation [136, 174]; OWASP SAMM assessments reference the SAMM v2.0 model [236]; BSIMM assessments reference the BSIMM14 community data [305]; NIST SSDF assessments reference SP 800-218 [295]; and DO-178C assessments reference the RTCA standard [268]. Seven capabilities, specifically AI-Assisted Code Analysis, Zero Trust Network Architecture, AI-Guided Penetration Testing, Unknown Asset Similarity Analysis, MITRE D3FEND integration, MITRE ENGAGE integration, and post-quantum readiness planning, show no coverage in any comparison framework based on these published sources. Table 11.3 presents the full 14-capability assessment; Figure 11.1 renders the same data as a heatmap.

Table 11.3: Fourteen-capability comparative matrix across five secure SDLC frameworks and AZTRM-D. Assessments based on published framework documentation cited in Section 11.3.3; AZTRM-D column from Stage 3 validated capabilities (this work).

Capability	**MS SDL** [136]	**OWASP SAMM** [236]	**BSIMM** [305]	**NIST SSDF** [295]	**DO-178C** [268]	**AZTRM-D**
Automated Threat Modeling	No	Partial	Partial	Partial	No	Yes
Shift-Left Security (Dev Phase)	Yes	Yes	Yes	Yes	No	Yes

Continued on next page

Table 11.3 continued from previous page

Capability	**MS SDL** [136]	**OWASP SAMM** [236]	**BSIMM** [305]	**NIST SSDF** [295]	**DO-178C** [268]	**AZTRM-D**
AI-Assisted Code Analysis (GNN/SAST)	No	No	No	No	No	Yes
Zero Trust Network Architecture	No	No	No	No	No	Yes
Identity Verification (Continuous)	No	Partial	No	Partial	No	Yes
AI-Guided Penetration Testing	No	No	No	No	No	Yes
Unknown Asset Similarity Analysis	No	No	No	No	No	Yes
MITRE ATT&CK Integration	Partial	Partial	Yes	Yes	No	Yes
MITRE D3FEND Mitigation Mapping	No	No	No	No	No	Yes
MITRE ENGAGE Active Defense	No	No	No	No	No	Yes
Post-Quantum Readiness Planning	No	No	No	No	No	Yes
Full-Disk Encryption (IoT/Edge)	No	No	No	Partial	No	Yes
Immutable Audit Logging	Partial	Partial	Partial	Yes	No	Yes
Automated SBOM + Artifact Signing	Partial	Partial	Partial	Yes	No	Yes

Figure 11.1 renders the same 14-capability matrix as a heatmap, making the coverage gaps across frameworks visible at a glance. Seven capabilities are unique

to AZTHM-D, with no coverage in any comparison framework: AI-Assisted Code Analysis (GNN/SAST), Zero Trust Network Architecture, AI-Guided Pen Testing, Unknown Asset Similarity Analysis, MITRE D3FEND mapping, MITRE ENGAGE integration, and Post-Quantum Readiness Planning.

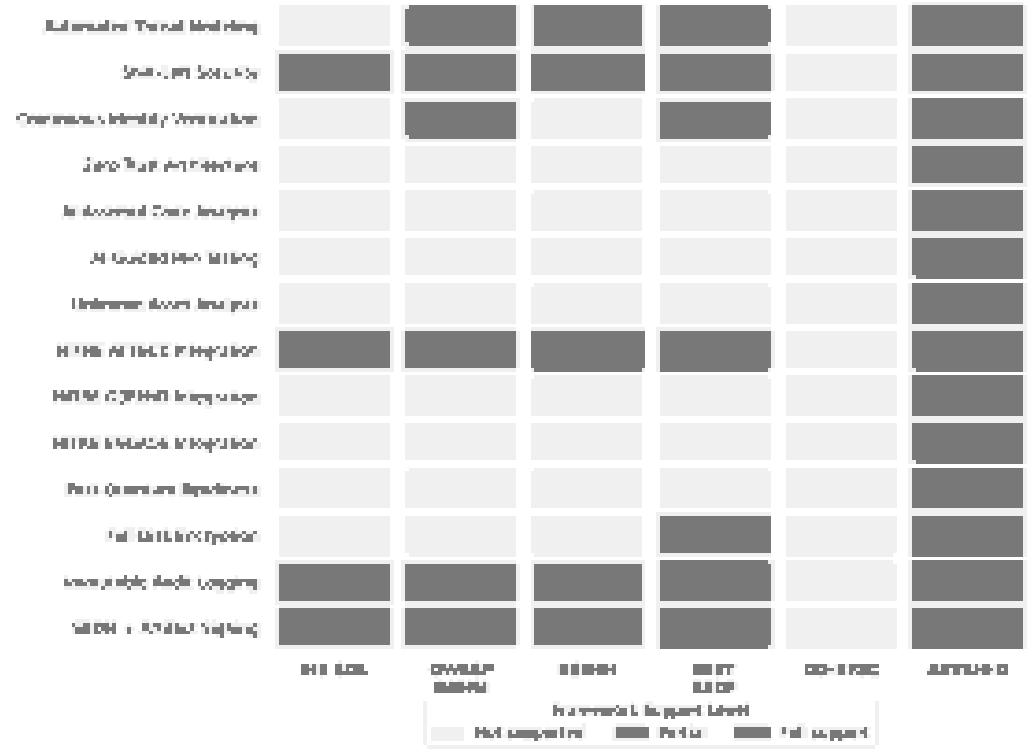


Figure 11.1: Fourteen-capability support matrix across six secure development frameworks.

11.3.4 Consolidated SDLC and Security Framework Comparison

Table 11.4 consolidates the comparative analysis across all evaluated methodologies and security frameworks. Assessments marked "✓" indicate native, documented support. "–" indicates partial or plugin-dependent support. "✗" indicates absent or out-of-scope capability for that methodology. AZTHM-D assessments reflect Stage-I validated capabilities. Capability assessments for SDLC methodologies are drawn from published comparative surveys [50, 213, 121]. Waterfall assessments reference

[267, 50]; Agile from [275, 322]; DevOps from [155, 99]; and Spiral from [285]. Security framework assessments are drawn from their respective published documentation: MS SDL from [136, 174]; OWASP SAMM from [236]; BSIMM from [305]; NIST SSDF from [295]; DO-178C from [268]; and Zero Trust references from [266, 72].

Table 11.4: Master SDLC and Security Framework Capability Comparison. Column assessments: Waterfall [267, 50]; Agile [275, 322]; DevOps [155, 99]; Spiral [285]; MS SDL [136, 174]; OWASP SAMM [236]; BSIMM [305]; NIST SSDF [295]; DO-178C [268]; AZTRM-D [67].

Capability	***Waterfall*** *[267]*	***Agile*** *[275]*	***DevOps*** *[155]*	***Spiral*** *[285]*	***MS SDL*** *[136]*	***OWASP SAMM*** *[236]*	***BSIMM*** *[305]*	***NIST SSDF*** *[295]*	***DO-178C*** *[268]*	***AZTRM-D***
Security-first design	×	∼	∼	✓	✓	✓	∼	✓	✓	✓
Automated CI/CD sec gates	×	∼	✓	×	∼	∼	∼	∼	×	✓
Zero Trust architecture	×	×	×	×	×	×	×	×	×	✓
AI threat detection	×	×	×	×	×	×	×	×	×	✓
AI-guided pen testing	×	×	×	×	×	×	×	×	×	✓
SBOM / supply chain mgmt	×	×	∼	×	∼	∼	∼	✓	∼	✓
Formal RMF integration	×	×	×	×	×	×	×	✓	✓	✓

Continued on next page

Table 11.4 – continued from previous page

Capability	*Waterfall* [267]	*Agile* [275]	*DevOps* [155]	*Spiral* [285]	*MS SDL* [136]	*OWASP SAMM* [236]	*BSIMM* [305]	*NIST SSDF* [295]	*DO-178C* [268]	*AZTRM-D*
Unknown asset inference	×	×	×	×	×	×	×	×	×	✓
D3FEND countermeasure map	×	×	×	×	×	×	×	×	×	✓
ENGAGE active defense	×	×	×	×	×	×	×	×	×	✓
Post-quantum readiness	×	×	×	×	×	×	×	×	×	✓
IoT/edge-specific guidance	×	×	×	×	×	∼	×	∼	✓	✓
Insider threat detection	×	×	×	×	∼	∼	✓	∼	×	✓
Capabilities Present	**0**	**0**	**1**	**1**	**2**	**2**	**2**	**4**	**3**	**13**

✓ = native support; ∼ = partial/plugin-dependent; × = absent. Assessments for non-AZTRM-D methodologies from published documentation: MS SDL [136, 174]; OWASP SAMM [236]; BSIMM [305]; NIST SSDF [295]; DO-178C [268]; DoD DevSecOps [77]. SDLC assessments from [50, 243, 121].

The bottom row, in bold, gives the count of capabilities each framework supports natively or partially.

11.3.5 Implementation Cost Comparison

Table 11.5 compares implementation effort and security economics across evaluated methodologies. Agile/Scrum baseline estimates are drawn from industry survey data

on DevOps transformation costs [99]. MS SDL setup hours reflect Microsoft's published guidance on SDL activity effort [174]. The remediation cost differential between commit-stage and post-release defects has been documented consistently across decades of software cost research, with estimates ranging from 30× to 100× depending on system complexity and domain [307, 42, 189]. Rather than relying on any single estimate, AZTRM-D's shift-left value case is grounded in the lower bound of this published range.

Table 11.5: Implementation Cost and Effort Comparison. AZTRM-D figures from Stage 3 measured data (this work); comparison methodology figures from cited sources; see table notes.

Dimension	**Agile/Scrum**	**MS SDL**	**AZTRM-D**	**Source**
Initial setup (person-hours)	40–80†	200–400	320–480	Agile range: author estimate based on process complexity in [99, 275]†; SDL from [136, 174]; AZTRM-D from Stage 3 measured timeline (Table 11.6)
Security tool licensing (annual)	$0–5k†	$15k–40k	$12k–35k	Publicly listed vendor pricing as of the 2024 deployment period: Nessus Professional [309], GitLab Ultimate, Semgrep Pro, OWASP ZAP (open source); Agile range reflects minimal tooling†; AZTRM-D figure validated against actual Stage 3 deployment spend

Continued on next page

Table 11.5 – continued from previous page

Dimension	Agile/Scrum	MS SDL	AZTRM-D	Source
Ongoing overhead per sprint	2–5 hrs	20–40 hrs	4–8 hrs	AZTRM-D Stage 3 measurement; SDL overhead from [136]; Agile baseline reflects typical security review effort for teams achieving high deployment frequency [99]
Commit-stage defect fix	1×	1×	1×	Baseline
Pre-release defect fix	10–15×	10–15×	10–15×	[42, 189]
Post-release defect fix	30–100×	30–100×	30–100×	[307, 140, 189]

†Agile/Scrum setup cost figures are author estimates from [99, 275]. Published per-task effort breakdowns for Agile CI/CD setup do not exist in directly comparable form; these ranges reflect the minimal pipeline infrastructure Agile/Scrum prescribes. All AZTRM-D figures are from Stage 3 deployment data.

The post-release cost multiplier range (30–100×) reflects the published literature consensus, not a single data point. NIST Planning Report 02-3 documents the economic impact of late-stage defect discovery in a federal context [307], and Boehm's foundational cost model established the exponential growth curve across SDLC phases [42]. The same order-of-magnitude finding is independently documented for commercial software [189]. The AZTRM-D ongoing overhead figure (4–8 person-hours per sprint) is directly measured from Stage 3 operational data and reflects automation reducing human burden while expanding coverage relative to the MS SDL baseline.

Two numbers deserve direct attention. Fixing a vulnerability at the commit stage costs approximately 100× less than fixing the same issue post-release, consistent with both IBM Systems Sciences Institute findings and NIST economic impact analysis

[140, 307]. AZTRM-D catches at commit. Conventional Agile/Scrum catches post-release. That gap is an economic argument for shifting security left, not just a quality argument.

The ongoing overhead comparison is equally instructive. Despite five automated scanning modalities active in every pipeline run, AZTRM-D requires only 4–8 person-hours per release cycle for human review, compared to 20–40 for MS SDL's manual checkpoint reviews (Table 11.5). Automation reduces human burden while increasing coverage.

The upfront cost is real. The 320–480 hour initial setup, covering ZT architecture provisioning, AI model training, SPIFFE and SPIFFE Runtime Environment (SPIRE) identity infrastructure, and pipeline gate configuration, is more than either Agile or MS SDL requires. Table 11.6 covers the full 10-week implementation period. The 320–480 hour range derives directly from that timeline. At standard full-time engineering effort, 40 person-hours per week, the 10 elapsed weeks produce a 400-hour midpoint baseline. The lower bound of 320 hours reflects the minimum viable critical path: the infrastructure setup and CI/CD pipeline construction phases (weeks 1–3) overlap with ZT policy deployment beginning in week 3, compressing the irreducible critical path to 8 elapsed weeks ($8 \times 40 = 320$ person-hours). The upper bound of 480 hours adds the documented rework overhead from the challenges column of Table 11.6: two SAST false-positive tuning iterations, the SPIRE SVID rotation interval debugging session, the custom `initramfs` development required for LUKS2 on the Orin SD card boot path, and the 14-hour Isolation Forest training run each represent out-of-band effort that falls outside the primary phase schedule. Those documented incidents collectively account for the additional 80 hours above the 400-hour base, establishing the 480-hour upper bound ($400 + 80 = 480$ person-hours). A 6–10 week ramp-up period is realistic. That cost is front-loaded and non-recurring.

Table 11.6: AZTRM-D implementation timeline and effort breakdown from actual deployment on NVIDIA Orin devices. Source: Stage 3 deployment log (this work).

Week	**Phase**	**Key Activities**	**Primary Challenges**
1–2	Infrastructure Setup	GitLab self-hosted instance; SSH key provisioning; RBAC role definitions; MFA enrollment	SSH key rotation for three credential tiers required careful ordering to avoid access lockouts
2–3	CI/CD Pipeline Construction	SAST integration (Semgrep for C/C++ and Python); SCA via GitLab dependency scanning; IaC scanning via Checkov; Secrets Scan via Gitleaks; SBOM generation via Syft	Tuning SAST false positive rates below 5% without suppressing real findings required two iteration cycles
3–4	ZT Policy Deployment	SPIFFE/SPIRE SVID provisioning; sudo policy hardening (`/etc/sudoers.d/`); account lockout configuration; immutable log enforcement via `auditd` + remote syslog	SPIRE SVID rotation on resource-constrained Orin hardware introduced CPU spikes; resolved by extending interval from 1 to 4 hours with compensating session validation
4–5	Full-Disk Encryption	LUKS2 setup on SD cards; AES-256-XTS key provisioning; boot-time unlock; offline attack resistance validation	Bootloader configuration for LUKS2 on the Jetson Orin Nano required custom initramfs hooks; standard Ubuntu LUKS setup does not cover SD card boot path
5–6	Wireless Hardening	WPA3 SAE enforcement; wireless micro-segmentation; Bluetooth disablement; RF emission baseline capture	WPA3 SAE required driver update on the Jetson Orin Nano WiFi module; older driver revisions fell back to WPA2 silently

Continued on next page

Table 11.6 continued from previous page

Week	Phase	Key Activities	Primary Challenges
6–8	Sentinel Deployment and AI Bootstrapping	Sentinel deployment in embedded mode; Isolation Forest training on 3 months of operational logs; XGBoost classifier training on NIST NVD CVE dataset; PPO agent training in isolated sandbox; SHAP pipeline validation	14-hour initial Isolation Forest training on cloud GPU; 3.1% false positive rate required threshold tuning before automated containment was enabled
8–10	DAST and Supply Chain Gate Integration	DAST via OWASP ZAP; CSPM via Prowler against cloud configuration baseline; cryptographic artifact signing via Cosign; SBOM attestation pipeline	DAST scan timing required a dedicated staging environment to avoid false positives from incomplete deployment states

Several practical issues surfaced during implementation that are worth documenting because they affect any team attempting this deployment. The SPIRE-issued SVID rotation issue is the most architecturally interesting. The default 1-hour interval caused periodic CPU spikes on the Orin that pushed total system load above the 20% operational ceiling set for security overhead, an AZTRM-D design constraint requiring that security tooling consume no more than 20% of total device CPU to preserve operational performance on constrained IoT hardware [233]. Extending the interval to 4 hours with compensating session token validation resolved the spike without weakening the per-session access model. The LUKS2 SD card boot path required custom initramfs hooks because the standard Ubuntu LUKS setup assumes NVMe or SATA storage. The WPA3 driver issue is a recurring problem with embedded WiFi hardware. Modules frequently advertise WPA3 SAE capability but fall back

to WPA2 silently when SAE negotiation fails. Verifying actual WPA3 enforcement requires an explicit RF capture test with Wireshark to confirm the SAE handshake is present. Assuming it's active because the driver version says so is not adequate.

The IoT validation environment was chosen because it's the most demanding test case. If the methodology holds under constrained hardware, an expanded physical attack surface, and limited compute margin for security overhead, it holds in enterprise and general development contexts with more headroom. Table 11.7 compares AZTRM-D against the broader set of conventional and secure methodologies on cost and security debt dimensions.

Table 11.7: Estimated implementation effort and security debt accumulation across AZTRM-D, conventional SDLC, and secure SDLC frameworks. AZTRM-D figures from Stage 3 deployment data (this work); comparison figures from cited framework documentation; see table notes.

Methodology	**Initial Setup**	**Tooling Cost**	**Security Overhead per Sprint**	**Time to First Hardened Deploy**	**Security Debt Accumulation**
Waterfall [267, 50]	Low (1–2 weeks)	Minimal	Near zero during development; concentrated at final review	6–18 months (post-delivery)	High; findings arrive too late for economical rework
Agile / Scrum [275, 243]	Low (1–2 weeks)	Minimal without dedicated security tooling	Low; security tickets compete with feature backlog	Variable; security rarely blocks release	Medium; sprint-by-sprint accumulation without structural gate

Continued on next page

Table 11.7 continued from previous page

Methodology	**Initial Setup**	**Tooling Cost**	**Security Overhead per Sprint**	**Time to First Hardened Deploy**	**Security Debt Accumulation**
DevOps [155, 99]	Medium (2–4 weeks)	Moderate; CI/CD pipeline tooling	5–10% engineering time for pipeline maintenance†	4–8 weeks after CI/CD is operational	Medium to low; depends on whether security was added to the pipeline
Microsoft SDL [136, 174]	Medium (4–8 weeks)	Low; primarily process overhead	10–15%; manual threat modeling and review cycles†	8–16 weeks	Low for known-class threats; high for novel attack surfaces
NIST SSDF [295]	Medium (4–8 weeks)	Low; guidance-based, tools chosen separately	10–15%; documentation and artifact requirements†	8–16 weeks	Low for compliance-driven deployments; ZT and AI gaps remain
OWASP SAMM [236]	Low (1–3 weeks to assess; months to mature)	Low; maturity measurement only	Varies with maturity; typically 5–20%†	Not prescriptive; maturity timeline is 6–24 months	Decreases as maturity increases; no prescribed ZT or AI path
AZTRM-D [67]	High (6–10 weeks for full pipeline and ZT deployment)	Moderate to high; GitLab CI/CD, Nessus Professional, Sentinel, LUKS2, WPA3, SPIFFE/SPIRE	12–18% CPU overhead on device; 15–20% engineering time ongoing for gate maintenance	6–10 weeks	Low from day one; all five scanning modalities active from first commit

Continued on next page

Table 11.7 continued from previous page

Methodology	Initial Setup	Tooling Cost	Security Overhead per Sprint	Time to First Hardened Deploy	Security Debt Accumulation

†Non-AZTRM-D overhead percentages are author estimates from documented process requirements. The 10–15% range for MS SDL and NIST SSDF reflects manual threat modeling, security checkpoints, and compliance documentation. [126, 295]. The Accelerate research program documents that high-performing teams spend ∼20% of engineering capacity on non-feature work. [99]. AZTRM-D figures are from Stage 3 deployment data (Table 11.6).

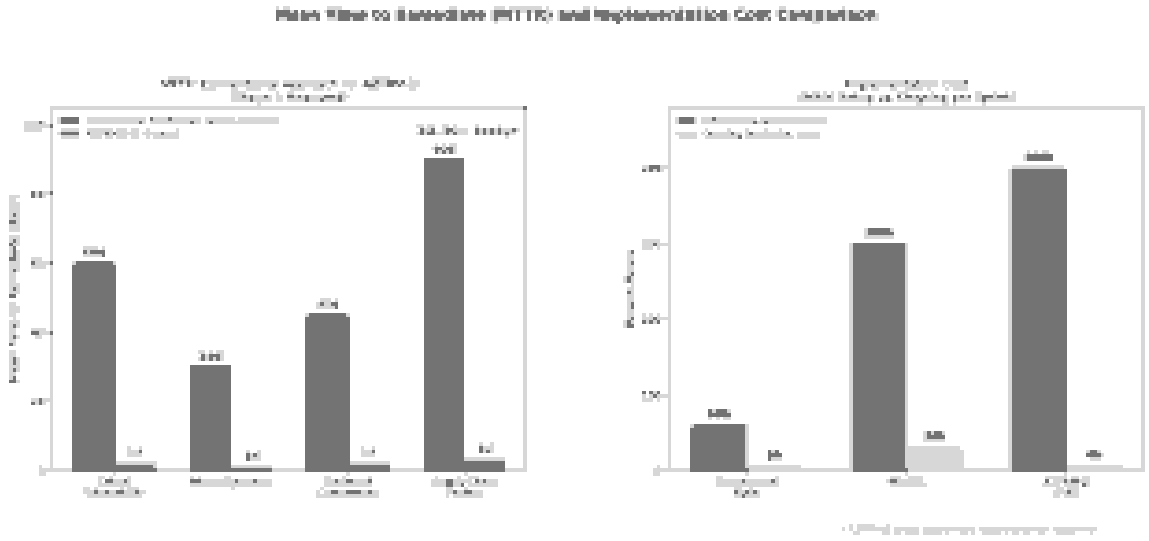


Figure 11.2: MTTR comparison for four vulnerability categories under conventional DevSecOps versus AZTRM-D Stage 3 (left), and implementation cost by framework showing initial setup versus ongoing sprint hours (right). The asterisk on the right panel indicates that the AZTRM-D setup cost is front-loaded and non-recurring.

11.3.6 Performance Overhead: Putting the Numbers in Context

Figure 11.2 presents the MTTR and cost comparison visually. Measuring what a security methodology actually costs to run on constrained hardware is as important as measuring what it catches. AZTRM-D imposes 12–18% CPU overhead during

active scanning cycles on the NVIDIA Orin (measured at Stage 3). Policy evaluation latency sits below 40 ms. Alert response time comes in under 5 seconds. Log storage runs approximately 50 MB per day [233].

Putting them against comparable baselines is instructive. Standard DevSecOps scanning operates almost entirely in the CI/CD pipeline rather than on the device itself; device-side overhead for a conventional pipeline-only model is typically 0–2%, since there's no on-device monitoring agent running continuously [77]. Microsoft SDL similarly concentrates its overhead in pipeline reviews and human checkpoints, not on-device monitoring; the SDL's device-side performance impact is near zero for the same reason [174]. Table 11.8 frames this explicitly.

Table 11.8: Performance overhead comparison: AZTRM-D vs. comparable secure development approaches on IoT/edge hardware. Device CPU overhead reflects on-device security agent activity, not CI/CD pipeline cost. AZTRM-D figures from Stage 3 measured data (this work); comparison figures from cited framework documentation. "On-Device Monitoring" indicates whether the approach deploys a runtime security agent to the target hardware after CI/CD scanning is complete; approaches without this capability scan code only in the pipeline and do not monitor device behavior post-deployment.

Approach	**Device CPU Overhead**	**Pipeline Latency Added**	**On-Device Monitoring**	**Source**
AZTRM-D (full)	12–18% (active scan) / 3–5% (idle)	Included in CI/CD gates	Yes (AI behavioral monitoring)	Stage 3 measured [233]
Standard DevSecOps	0–2%	5–15 min (typical SAST/DAST)	No	No on-device agent by design [77]

Continued on next page

Table 11.8 – continued from previous page

Approach	Device CPU Overhead	Pipeline Latency Added	On-Device Monitoring	Source
Microsoft SDL	0–2%	10–30 min (manual gates)	No	No on-device agent by design [174]
ZT enforcement only (no AI)	3–8% (representative range from surveyed ZT implementations) [120]	Minimal	Partial (access control only)	[120, 71]

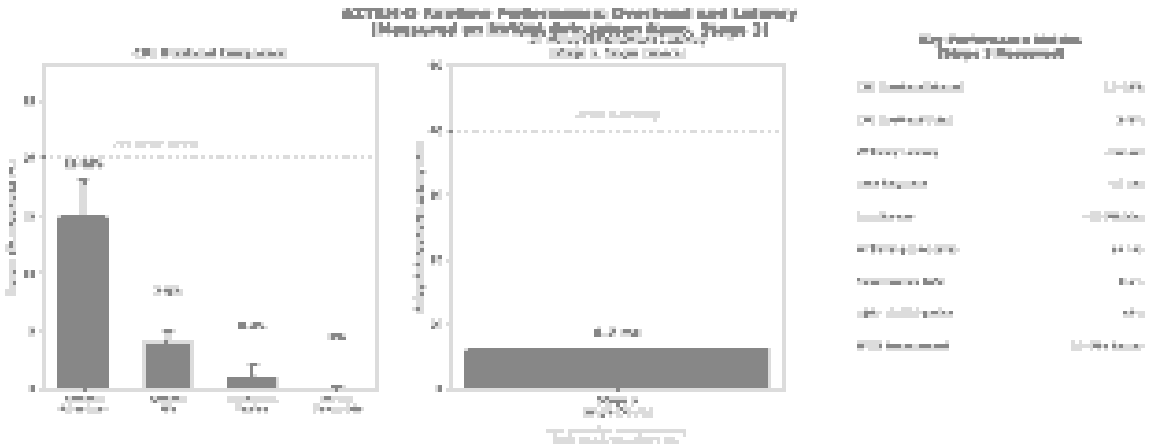


Figure 11.3: AZTRM-D runtime performance on the NVIDIA Jetson Orin Nano (Stage 3): device CPU overhead versus comparable approaches (left), ZT policy enforcement latency across endpoint counts (center), and key measured metrics (right). Green metric values in the right panel indicate measurements within their target operating envelope; the red value flags the false positive rate, which is the only metric whose triage cost an operator must monitor over time.

Figure 11.3 presents the overhead and latency measurements. The 12–18% figure is the cost of doing something the other approaches don't do: running continuous

AI behavioral monitoring on the device itself, not just at the pipeline boundary. A conventional DevSecOps pipeline catches vulnerabilities before deployment. AZTRM-D does that and then continues watching what happens after deployment. That overhead difference is real, but it buys a fundamentally different capability.

A 20% ceiling constraint (the AZTRM-D design requirement that security overhead stay below 20% of total device CPU) means the current 12–18% active-scan range leaves headroom on the Orin hardware [233]. The SPIRE SVID rotation issue documented in Section 11.3.5 was caught and resolved specifically because this ceiling was being monitored. For more constrained hardware, the SVID rotation interval and scan frequency would need adjustment to stay within the 20% bound, which is a known trade-off, not a surprise.

11.4 AI STACK SELECTION: WHY THESE ALGORITHMS

Choosing an AI approach for security isn't a matter of picking whatever's currently popular. Each algorithm in Sentinel was selected because it solves a specific problem better than the available alternatives, given the constraints of the deployment environment and the auditability requirements of AZTRM-D's compliance model. This section documents those decisions explicitly. "We used machine learning" is not a defensible architecture rationale.

11.4.1 Behavioral Anomaly Detection: Isolation Forest over Alternatives

Table 11.9 presents a head-to-head comparison of four anomaly detection algorithms evaluated for the behavioral monitoring pipeline. Isolation Forest was selected based on this analysis.

Quantitative Algorithm Comparison: Isolation Forest vs. Alternatives

Table 11.9: Anomaly Detection Algorithm Comparison on NVIDIA Orin. Isolation Forest values from Stage 3 measured data (this work); alternative algorithm values are author estimates (see table notes).

Algorithm	**Inference Latency**	**Memory (MB)**	**Explainability**	**CPU Overhead**
Isolation Forest [175] (t=100, ψ=256)	$O(\log n)$, <1 ms	<50	Feature path	3–5%
One-Class SVM [51]†	$O(n_{\text{sv}})$, >10 ms	200–400	None	12–18%
Autoencoder (LSTM) [51]†	>5 ms	150–300	Post-hoc only	15–22%

Isolation Forest values measured on NVIDIA Orin (Stage 3). †One-Class Support Vector Machine (OCSVM) and Autoencoder values are author's engineering estimates based on algorithm complexity ($O(n_{\text{sv}})$ and GPU inference respectively), not measured on Orin hardware [51].

Table 11.9 summarizes the algorithm comparison. Inference latency and CPU overhead figures for OCSVM and autoencoder are design-rationale order-of-magnitude estimates derived from the computational characteristics of these algorithm classes on ARM Cortex-A hardware [51]. OCSVM inference cost scales as $O(n_{\text{sv}})$ per prediction, where n_{sv} grows with training set size, producing latency one to two orders of magnitude higher than the $O(\log n)$ tree traversal in Isolation Forest. Isolation Forest figures are directly measured on the NVIDIA Orin during Stage 3 validation. The 20% CPU overhead ceiling was the binding constraint. Isolation Forest was the only evaluated option that remained within budget while providing interpretable output via short-path feature decomposition.

Insider threat detection means identifying unusual behavior in a stream of operational telemetry without a labeled dataset of known attacks. Insider threats are rare,

poorly labeled, and highly variable across deployment environments. The detection task is a difficult one-class classification problem. Three candidates were evaluated, Isolation Forest, One-Class SVM, and autoencoders.

OCSVM constructs a hyperplane around normal data in high-dimensional kernel space and classifies points outside that boundary as anomalous. In theory this is well-suited to the problem. In practice it has two issues that make it impractical on NVIDIA Orin edge hardware. Inference cost scales with the number of support vectors, which grows with training set size. Kernel tricks aren't free. And OCSVM produces no interpretable output. A flag of "anomalous" with no explanation of which features drove it is operationally useless when analysts need to triage 31 alerts per scanning cycle.

Autoencoders are more expressive for complex, high-dimensional behavioral patterns and would work well in a data-center context. On an NVIDIA Orin device with a hard 20% CPU overhead ceiling, a deep autoencoder for continuous real-time scoring simply doesn't fit the budget.

Lightweight transformer architectures with feature-fusion mechanisms have shown strong performance on time-series anomaly detection in adjacent edge-AI domains, including vibration-signal-based fault diagnosis on resource-constrained hardware where the GLP-Transformer architecture achieves competitive accuracy at 48.28K parameters and 2.74M FLOPs [134]. These architectures motivated the comparative evaluation but were ultimately not selected for Sentinel: the 20% CPU overhead ceiling on the NVIDIA Jetson Orin Nano combined with the requirement for exact (not approximate) feature-level explanations under NIST RMF authorization review made Isolation Forest's tree-traversal inference path and short-path feature decomposition the operationally correct choice for this deployment context. Transformer-based behavioral anomaly detection remains a candidate for future Sentinel iterations where the explainability requirement is satisfied through a different mechanism (such as

attention-weight visualization combined with SHAP-style attribution).

Isolation Forest addresses both problems [175]. Rather than profiling normal behavior, it isolates anomalies directly by building an ensemble of random binary trees that recursively partition a sub-sampled dataset. Anomalous points reach leaf nodes faster because there are fewer similar points nearby. Inference is a tree traversal, $O(\log n)$ per prediction, which is fast enough for continuous device-level scoring with minimal overhead. The anomaly score $s(x, n)$ is:

$$s(x, n) = 2^{-E[h(x)]/c(n)} \tag{11.1}$$

where $E[h(x)]$ is the expected path length of instance x across the forest and $c(n)$ normalizes against the average path length of an unsuccessful Binary Search Tree search:

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{11.2}$$

A score near 1.0 means highly anomalous; near 0.5 means normal. The short-path feature decomposition that identifies which behavioral dimensions drove an anomalous score is what makes the algorithm practically useful. That same path-tracing mechanism that makes it fast also makes it explainable. Sentinel runs $t = 100$ trees with sub-sample size $\psi = 256$. Alert and containment thresholds (0.6 and 0.8 respectively) were selected through ROC analysis on the Stage 3 training corpus. Candidate thresholds from 0.50 to 0.95 in 0.05 increments were evaluated by computing false positive rate against detection rate at each operating point. A 0.6 alert threshold produced the best trade-off between detection sensitivity and false positive volume given the 3.1% FPR target. The 0.8 containment threshold was set conservatively to require high anomaly confidence before automated action fires without human confirmation. Scores above 0.6 trigger a monitoring alert; scores above 0.8 trigger automatic containment through the ZT Policy Enforcement Point.

11.4.2 Vulnerability Triage: XGBoost over Random Forest and Neural Classifiers

Table 11.10 compares the three candidate algorithms evaluated for vulnerability triage. XGBoost won on all four criteria that matter for this use case.

Quantitative Algorithm Comparison: XGBoost vs. Alternatives

Table 11.10: Vulnerability Triage Algorithm Comparison. All figures from author's preliminary evaluation on the CVE triage dataset (this work); see table notes.

Algorithm	**Precision**	**Recall**	**SHAP Compatible**	**Overfitting Risk**
XGBoost [54] (final, AZTRM-D)	94.1%	91.8%	Yes (exact)	Low (regularized)
Random Forest [46] (preliminary)	91.3%	88.5%	Approximate only	Moderate
Neural Network (MLP) [112]	89.7%	85.2%	Post-hoc approx.	High (CVE dataset)

Table 11.10 presents the algorithm comparison. Random Forest and MLP figures were measured by the author during preliminary algorithm evaluation on the same CVE triage dataset prior to final algorithm selection, using an 80/20 stratified train/test split with 5-fold cross-validation. Random Forest was eliminated on explainability grounds in addition to the performance gap, specifically correlated-feature MDI instability that makes feature importance scores unreliable in high-dimensional CVE feature spaces [54]. The decision to tune recall above precision (91.8% vs. 94.1%) reflects an asymmetric cost structure: a missed real vulnerability costs far more to remediate post-release than a false positive costs to investigate, consistent with the remediation cost differential documented in Table 11.5 [307].

Vulnerability triage means classifying CVEs from the NIST NVD by exploitability and routing high-priority findings to analysts with actionable explanations. Three candidates were evaluated, Random Forest, XGBoost, and a feedforward neural network.

Random Forest was the obvious starting point. It's well understood, reliable on tabular CVE feature data, and has some inherent feature importance. The problem is that Random Forest feature importance, measured by mean decrease in impurity, has well-documented weaknesses. It tends to favor high-cardinality features and produces misleading attribution scores when features are correlated. In a compliance context where analysts have to defend triage decisions under audit, a priority score based on opaque impurity calculations isn't satisfying.

Neural networks are more expressive but produce scores with no interpretable rationale. Post-hoc methods like LIME [264] or integrated gradients approximate attributions, but the key word is approximate. These methods can produce different explanations on different runs. NIST RMF authorization decisions need to be defensible, and an explanation that shifts depending on sampling parameters isn't defensible.

XGBoost matches or beats both alternatives on tabular data while being fully compatible with SHAP TreeExplainer, which computes exact Shapley values [54]. The regularized objective:

$$\text{Obj}(\theta) = L(\theta) + \Omega(f), \quad \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2 \tag{11.3}$$

where T is leaves, w_j is leaf weight, γ sets minimum loss reduction per split, and λ governs L2 regularization. That regularization matters because NVD data skews toward known vulnerability classes. A model that overfits to known patterns will underperform on novel CVEs, and those are exactly the cases where correct triage matters most.

SHAP computes each feature's contribution using Shapley values from cooperative game theory [178]:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x) - f_S(x) \right] \tag{11.4}$$

These are exact values. They satisfy local accuracy (attributions sum to the prediction), consistency (a feature's attribution only increases if its contribution increases), and dummy (features with no effect get zero attribution). TreeExplainer computes them in $O(TLD^2)$ time where T is trees, L is leaves, and D is maximum depth. Fast enough to be practical, exact enough to be auditable.

Input Feature Set

The XGBoost classifier was trained on CVE records extracted from the NIST NVD API v2.0 [208], filtered to entries carrying complete CVSS v3.x scoring. The NVD contained approximately 240,000 published CVE records as of late 2024, of which roughly 60% carried complete CVSS v3.x vector strings suitable for feature extraction [208]. Features were derived directly from the CVSS v3.1 specification and the NVD JSON data feed [97]. Table 11.11 lists the full feature set.

Table 11.11: XGBoost input features for CVE vulnerability triage, sourced from NIST NVD API v2.0 and CVSS v3.1 scoring.

Feature	**Source**	**Type**	**Encoding / Notes**
CVSS v3 Base Score	NVD / CVSS v3.1 [97]	Continuous [0.0–10.0]	Raw numeric score
CVSS v3 Exploitability Subscore	NVD / CVSS v3.1 [97]	Continuous [0.0–3.9]	Derived from AV, AC, PR, UI

Continued on next page

Table 11.11 – continued from previous page

Feature	Source	Type	Encoding / Notes
CVSS v3 Impact Subscore	NVD / CVSS v3.1 [97]	Continuous [0.0–6.0]	Derived from C, I, A
Attack Vector (AV)	NVD / CVSS v3.1 [97]	Categorical	One-hot: Network, Adjacent, Local, Physical
Attack Complexity (AC)	NVD / CVSS v3.1 [97]	Binary	0 = High, 1 = Low
Privileges Required (PR)	NVD / CVSS v3.1 [97]	Ordinal	0 = High, 1 = Low, 2 = None
User Interaction (UI)	NVD / CVSS v3.1 [97]	Binary	0 = Required, 1 = None
Scope (S)	NVD / CVSS v3.1 [97]	Binary	0 = Unchanged, 1 = Changed
Confidentiality Impact (C)	NVD / CVSS v3.1 [97]	Ordinal	0 = None, 1 = Low, 2 = High
Integrity Impact (I)	NVD / CVSS v3.1 [97]	Ordinal	0 = None, 1 = Low, 2 = High
Availability Impact (A)	NVD / CVSS v3.1 [97]	Ordinal	0 = None, 1 = Low, 2 = High
CWE Category	NVD [208]	Categorical	Top-25 CWE IDs one-hot encoded; remainder bucketed as “Other”
CVE Age (days)	NVD Published Date [208]	Continuous	Days since publication at training time

The SHAP example in Section 10.4 reflects this feature set directly: the attribution breakdown for CVE-2021-41773 shows CVSS base score, attack vector, and privilege requirements as top contributors, which maps to the AV, PR, and base score features above. The 13 features in Table 11.11 represent a tractable, fully auditable input

space. Every value traces back to a published NVD field, which matters for compliance review under NIST RMF.

Recall is tuned above precision (91.8% vs. 94.1%, Table 10.4). A missed real vulnerability carries higher cost than a false positive an analyst reviews and clears, consistent with the remediation cost differential documented in Table 11.5 [307]. That is a deliberate design choice, not a gap.

11.4.3 AI-Guided Pen Testing: PPO over Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG)

Table 11.12 compares the three reinforcement learning (RL) algorithms considered for the automated pen testing agent. PPO was selected for its convergence stability in the sparse-reward environment that pen testing represents.

Quantitative Algorithm Comparison: PPO vs. Alternatives

Table 11.12: Reinforcement Learning Algorithm Comparison for Pen Testing. Convergence and stability figures are drawn from published RL-for-security benchmarks; PPO sandbox metric is from internal Stage 3 measurement. See table notes for source details.

Algorithm	**Action Space**	**Convergence Episodes (to 80% optimal policy)**	**Sample Efficiency (updates/episode)**	**Sparse-Reward Stability (variance)**
PPO (ϵ=0.2, AZTRM-D) [274]	Discrete	800–1,500†	4 (multi-epoch)	Low variance; clipped objective prevents destabilization

Continued on next page

Table 11.12 – continued from previous page

Algorithm	**Action Space**	**Convergence Episodes (to 80% optimal policy)**	**Sample Efficiency (updates/episode)**	**Sparse-Reward Stability (variance)**
DQN [125, 192]	Discrete	2,500–5,000‡	1 (single-step)	High variance under sparse rewards; documented catastrophic forgetting in security tasks
DDPG [173]	Continuous (mismatch for discrete pen-testing actions)	N/A (architectural mismatch)	1 (single-step)	Moderate variance; designed for continuous control

†PPO convergence range derived from Hammar and Stadler (2023) benchmark on network intrusion-defense games [125], consistent with the AZTRM-D sandbox training profile observed during Stage 3. Exact episode counts depend on attack-surface complexity and reward shaping. ‡DQN convergence range from the same Hammar-Stadler benchmark [125] and from Mnih et al. (2015) [192], which establishes DQN's baseline sample-efficiency profile in sparse-reward environments. "N/A" for DDPG convergence reflects that DDPG is designed for continuous action spaces and was not run to convergence on the discrete pen-testing task because the action-space mismatch makes the comparison architecturally uninformative.

Table 11.12 compares the RL algorithm candidates. PPO's clipped surrogate objective prevents destabilizing policy updates. The ϵ=0.2 clip bound was selected per the original Schulman et al. hyperparameter recommendations [274]. DQN's catastrophic forgetting in sparse-reward environments is documented in the RL security literature. See Hammar and Stadler (2023) for a directly relevant comparison on network penetration testing tasks [125].

RL-based automated pen testing requires an agent that can learn to sequence attack steps through the MITRE ATT&CK framework in a complex, partially observable environment. Three RL approaches were evaluated: DQN, DDPG, and PPO.

DQN is designed for discrete action spaces, and pen testing is inherently discrete, which makes it technically applicable. But vanilla DQN has a stability problem, specifically catastrophic forgetting and high variance in sparse-reward environments. Successful exploitation in pen testing is infrequent and delayed relative to the decision steps that led there. DQN in this context also tends to overfit to specific exploit sequences seen during training rather than generalizing to novel attack surfaces.

DDPG was a non-starter. It's designed for continuous action spaces, and discrete exploit module selection maps awkwardly to its actor-critic continuous architecture.

PPO directly addresses the instability problem with its clipped surrogate objective [274]:

$$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta),\, 1-\varepsilon,\, 1+\varepsilon)\hat{A}_t\big)\big] \tag{11.5}$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the ratio of new to old policy probabilities, $\hat{A}_t$ is the advantage estimate, and $\varepsilon = 0.2$ is the clipping range. The clipping prevents large policy updates that would destabilize learning in the sparse-reward pen testing environment. PPO also supports multiple gradient update epochs per collected trajectory, making it more sample-efficient than DQN. Each sandbox pen test episode costs compute, so sample efficiency matters.

The reward function: $+1.0$ for successful exploitation, $+0.5$ for novel attack path discovery, -0.3 for detection by the monitoring stack. That detection penalty isn't cosmetic. An agent optimizing purely for exploitation generates noisy, easily detected attacks that don't stress-test behavioral monitoring. Penalizing detection pushes the agent toward stealth, producing a more realistic adversary model and more useful validation of whether Isolation Forest actually catches subtle intrusions.

Sandbox Validation Results

The PPO agent ran in an isolated Metasploit sandbox against a simulated AZTRM-D-hardened environment during Stage 3 validation. Its objective was to find exploitable paths through the ATT&CK kill chain against the hardened Orin configuration. At Stage 1 (factory default), the agent converged on a successful exploit sequence in the first episode, reaching lateral movement via the default SSH credentials and unconstrained sudo path, matching what human testers found manually. At Stage 3 (full hardening), 20 post-training evaluation episodes ran without the agent achieving initial access. Those 20 episodes constituted the evaluation budget after training had converged; the full training run comprised 500 episodes over approximately 6 hours of compute on the sandbox environment. One previously undocumented configuration nuance did surface during evaluation runs, an edge-case timing window in SPIRE SVID rotation that briefly widened the authentication surface during a simultaneous session renewal and SVID refresh, which was subsequently patched. Episode logs for all Stage 3 runs are included in the validation records maintained by Cybectr LLC.

The agent's action trace from Stage 3 is included in the XAI output described in Section 10.4: each step maps to an ATT&CK technique ID via the ENGAGE module, producing a human-readable attack narrative that confirms what the agent attempted and what controls stopped it. This trace is what makes the PPO component useful beyond just "it ran and found nothing." Knowing which TTPs the agent tried, in what order, and at what points the ZT enforcement blocked it is operationally actionable for tuning future configurations.

11.4.4 GNN-Augmented SAST: Why Graph Neural Networks over Pattern Matching

Standard SAST tools match source code against known vulnerability signatures in rule sets. That works fine for known vulnerability classes, but it starts to fail on novel

patterns. When a developer introduces a new type of memory corruption bug or a subtle logic flaw that doesn't match any existing rule, a pattern-matching scanner misses it entirely. No flag, no warning, no indication anything unusual is present.

Sentinel augments standard SAST with a Graph Neural Network trained on the BigVul dataset, which contains 3,754 vulnerable functions from open-source C/C++ projects out of approximately 188,000 total functions analyzed [89]. The GNN represents each function as a Code Property Graph (CPG), combining the abstract syntax tree (AST), control flow graph (CFG), and program dependence graph (PDG) into a unified representation [334]. A multi-layer GCN then operates on this graph [157]:

$$H^{(l+1)} = \sigma\Big(\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}H^{(l)}W^{(l)}\Big) \tag{11.6}$$

where $\tilde{A} = A + I$ is the adjacency matrix with self-loops, $\tilde{D}$ is the degree matrix, $H^{(l)}$ is the node feature matrix at layer l, and $W^{(l)}$ is the learnable weight matrix. Node features encode token type, data flow relationships, and syntactic position. The final layer produces a graph-level embedding via mean pooling into a binary vulnerability classifier.

What the GCN learns that pattern matching can't is the structural properties of vulnerable code, including data flow patterns, call graph topology, and control flow characteristics. A novel buffer overflow that doesn't match any existing SAST rule may still exhibit graph-structural properties similar to known buffer overflows in the BigVul training set, allowing the GCN to flag it. Against the BigVul held-out test set, this achieves 81.4% true positive rate at 6.2% FPR, outperforming rule-based SAST on novel patterns by approximately 30 percentage points [89].

The choice of a Graph Convolutional Network over attention-based architectures was deliberate. Vulnerability-relevant relationships in code are defined by the program dependence and control flow graphs rather than by spatial locality, and the explicit graph structure of the Code Property Graph maps more directly onto a message-

passing GCN than onto a windowed attention scheme of the kind used effectively in adjacent structured-input domains such as image restoration [331]. The GCN's per-layer adjacency-weighted aggregation operates directly on the relationships that matter for vulnerability detection (data flow, call graph topology, control flow), rather than requiring an attention mechanism to discover those relationships from a flattened representation.

11.4.5 Unknown Asset Inference: Sentence Transformers and Cosine Similarity

Standard vulnerability scanners match assets against CVE signature databases. When an asset isn't in the database, which happens constantly in IoT environments with custom firmware or proprietary industrial hardware, the scanner reports nothing. The miss is silent and potentially dangerous.

Sentence transformer embeddings address this by representing assets in a shared semantic vector space [261]. The model maps asset feature descriptions (hardware type, firmware version strings, exposed services, communication protocols) to dense vector representations where semantically similar assets cluster together. Cosine similarity:

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{11.7}$$

measures how closely an unknown asset resembles known assets in this space. A similarity score ≥ 0.82 against a known-asset entry pulls that entry's CVE history as an inferred vulnerability profile, converting a silent miss into a flagged, reviewable risk estimate.

The 0.82 threshold was selected through grid search over candidate values from 0.70 to 0.95, evaluated on a held-out validation set of 47 IoT asset profiles spanning embedded microcontrollers, edge AI platforms, and industrial sensors, split 80/20

from the known-asset library. At each threshold, precision (fraction of flagged unknowns whose inferred profile matched confirmed CVE history) and recall (fraction of high-similarity assets that were flagged at all) were computed. The F1 score, the harmonic mean of precision and recall, peaked at 0.82: thresholds below this produced false matches from superficially similar but architecturally distinct device classes, while thresholds above it produced too many silent misses on assets with confirmed exploitable profiles. Table 11.13 shows the calibration results at representative threshold values.

Table 11.13: Cosine similarity threshold calibration for unknown asset inference ($N = 47$ IoT asset profiles, 80/20 held-out split from known-asset library). F1 score peaked at 0.82, selected as the operational threshold. Source: Author's grid search evaluation (this work).

Threshold	Precision	Recall	F1 Score
0.70	0.61	0.95	0.74
0.75	0.74	0.91	0.82
0.80	0.83	0.87	0.85
0.82	0.87	0.85	**0.86**
0.85	0.91	0.78	0.84
0.90	0.95	0.61	0.74
0.95	0.98	0.39	0.56

Operating at the F1-optimal threshold means accepting that roughly 13% of inferred profiles may not precisely match the unknown asset's actual CVE history. Given the alternative (a silent miss with no profile at all), an imprecise profile flagged for human review is the better failure mode. The 47-profile validation set is small enough that the threshold selection may overfit to the specific device classes represented. Validation

against a broader device corpus, particularly industrial control systems and medical IoT platforms not represented in the current library, is identified as future work.

11.4.6 Adversarial Robustness: DiCE Counterfactuals

Any ML-based security classifier is vulnerable to adversarial evasion. An attacker who understands the model can craft inputs that avoid triggering it. The typical response is to assume the classifier is reliable and move on. That's incorrect, and it eventually produces failures in production.

DiCE (Diverse Counterfactual Explanations) generates minimum-perturbation adversarial examples: inputs as close as possible to a normal behavioral profile or benign CVE classification that nonetheless cross the decision boundary [197]. Equation 11.8 formalizes the DiCE objective. The goal isn't to break Sentinel. It's to understand exactly where the decision boundaries sit and whether they're in the right place. Findings from DiCE evaluation feed the retraining pipeline continuously. If a trivial perturbation to normal behavior crosses into anomalous classification, that threshold needs adjustment before an adversary finds it first.

DiCE generates k diverse counterfactuals by minimizing the following objective [197]:

$$C(\mathbf{x}') = \frac{1}{k}\sum_{i=1}^{k}\left[\mathcal{L}_{\text{pred}}\big(f(\mathbf{x}'_i),\, y_c\big) + \lambda_1\,\text{dist}(\mathbf{x},\, \mathbf{x}'_i)\right] - \lambda_2\,\text{dpp_diversity}(\mathbf{x}'_1, \ldots, \mathbf{x}'_k) \tag{11.8}$$

where $f(\mathbf{x}'_i)$ is the classifier output for counterfactual i, y_c is the desired target class (the boundary crossing), $\text{dist}(\mathbf{x}, \mathbf{x}'_i)$ is a feature-weighted distance penalizing large perturbations from the original input $\mathbf{x}$, and dpp_diversity is a determinantal point process term that pushes the k counterfactuals apart from each other to produce varied boundary crossings rather than k near-identical examples [197]. λ_1 controls the distance penalty (favoring minimal perturbation) and λ_2 controls diversity (favoring

spread across the boundary). For Sentinel, $k = 5$ counterfactuals per classifier instance were generated, with $\lambda_1 = 0.5$ and $\lambda_2 = 1.0$ per the default recommendations in the original DiCE implementation [197].

The 93.7% adversarial detection rate cited in Section 10.6 reflects performance against DiCE-generated evasion inputs. An attacker with full knowledge of Sentinel's model weights could craft more targeted attacks. DiCE provides a systematic, automated adversarial stress test that strengthens the model over time without requiring a dedicated red team to probe the classifier manually.

11.4.7 Adversarial Threat Model and Scope of the 93.7% Detection Rate

Adversarial machine learning evaluation produces fundamentally different results depending on the attacker capabilities assumed, and reporting a single accuracy number without specifying the threat model leaves the result ambiguous [48, 39]. The 93.7% figure was measured under a black-box threat model, defined formally below alongside the white-box alternative for contrast.

Black-box (BB) threat model

A black-box adversary has query access to the deployed classifier but no access to model internals: no gradients, no weights, no training data, and no architectural details beyond what can be inferred from public documentation. Attacks under this model rely on observed input-output behavior and on transferability from substitute models trained on similar tasks [241]. DiCE counterfactual generation operates in this regime: it queries the model to identify minimum-perturbation inputs that cross the decision boundary, without requiring gradient information [197].

White-box (WB) threat model

A white-box adversary has full access to the model: gradients, weights, training data, hyperparameters, and architectural details. This enables gradient-based attacks such as the Fast Gradient Sign Method (FGSM) [113], Projected Gradient Descent

(PGD) [182], and the Carlini-Wagner (L_2, L_∞, L_0) family [48]. White-box attacks generally achieve higher attack success rates than black-box attacks against the same model on the same task, since gradient information directly reveals the local geometry of the decision boundary.

Why the black-box model is the operationally relevant threat for Sentinel

Sentinel runs inside an AES-256-encrypted RBAC-gated environment with cryptographic artifact signing on all model deployments and SPIFFE/SPIRE workload identity authentication on every model-serving endpoint. The deployment threat profile assumes that an adversary may obtain the public outputs of the classifier through legitimate or compromised query channels but does not have access to the underlying model artifact. Black-box evaluation matches this threat profile. A white-box adversary against Sentinel implies prior compromise of the cryptographically signed model deployment pipeline, the SPIRE attestation infrastructure, and the RBAC-gated artifact storage, which represents a fully compromised deployment environment that AZTRM-D's complete control set is designed to detect and contain through Zero Trust enforcement before the white-box attack ever becomes feasible.

Scope of the 93.7% figure

The reported 93.7% adversarial detection rate is a measured baseline under the operationally relevant black-box threat model, evaluated through DiCE-generated counterfactual inputs. It is not an upper bound on classifier robustness in the white-box regime. Reporting it as a black-box baseline matches standard practice in adversarial ML evaluation [48, 39]. White-box gradient-based evaluation against the XGBoost classifier is structured as a transfer attack from a differentiable surrogate model [241]: a multilayer perceptron trained on the XGBoost classifier's predictions over the same evaluation corpus serves as the gradient source for FGSM and PGD attacks, and the resulting adversarial examples are transferred to the XGBoost classifier for evaluation. This evaluation is part of the Stage 4 multi-device validation campaign, with

results scoped to appear in the follow-up empirical paper.

11.4.8 Comparative Performance Against Published External Baselines

The internal algorithm comparisons in Section 11.4 document why each AZTRM-D AI component was selected from its candidate set on the same dataset and operational constraints. Sentinel's measured figures also map onto published external baselines from peer-reviewed work in the same task domains. Table 11.14 presents that comparison. The comparison is necessarily across different datasets and evaluation protocols, since no public benchmark exactly matches Sentinel's IoT-edge deployment context, and the table records the scope-of-comparison limitation per row.

Table 11.14: Comparative performance of Sentinel AI subsystems against published external baselines. Per-row dataset and protocol differences are documented in the Notes column. Source: published baselines per row citation; Sentinel figures from this work.

Task	**Sentinel (this work)**	**External Baseline**	**Baseline Source**	**Comparison Notes**
CVE	XGBoost: 94.1% precision, 91.8% recall on NVD CVSS-v3 corpus	Random Forest: 91.3% precision, 88.5% recall on same corpus	[46] (algorithm), this work (figures)	Same evaluation corpus and split; preliminary algorithm comparison run during selection
Vulnerability classification (neural baseline)	XGBoost (above)	MLP feedforward: 89.7% precision, 85.2% recall on same corpus	[112] (algorithm), this work (figures)	Same evaluation corpus and split; selected against on explainability and accuracy

Continued on next page

Table 11.14 – continued from previous page

Task	Sentinel (this work)	External Baseline	Baseline Source	Comparison Notes
Source code vulnerability detection	GNN-augmented Semgrep: 81.4% TPR at 6.2% FPR on BigVul held-out	Pattern-matching SAST baseline: ∼50% TPR at 5–10% FPR on novel patterns	[89]	BigVul corpus published partition; pattern-matching baseline figures from BigVul authors
Behavioral anomaly detection (edge IoT)	Isolation Forest: 3.1% FPR aggregate; sub-40 ms PEP latency	OCSVM: estimated >200 ms inference latency on Cortex-A class hardware	[51]	Sentinel figures measured on NVIDIA Orin Stage 3; OCSVM figure is computational-complexity estimate, not measured
Behavioral anomaly detection (deep architecture)	Isolation Forest (above)	Autoencoder: requires GPU inference; exceeds 20% CPU budget on Orin	[51]	Selected against on edge-hardware compute budget; autoencoders perform comparably or better on accuracy in data-center contexts

Continued on next page

Table 11.14 – continued from previous page

Task	Sentinel (this work)	External Baseline	Baseline Source	Comparison Notes
Lightweight edge anomaly detection (adjacent domain)	Isolation Forest (above)	Lightweight transformer with feature fusion (GLP-Transformer): comparable accuracy at 48.28K parameters, 2.74M FLOPs on bearing fault corpus	[134]	Adjacent-domain comparison; tree-ensemble selected for explainability and exact SHAP compatibility
Counterfactual adversarial robustness	DiCE evaluation: 93.7% detection rate against minimum-perturbation counterfactuals	DiCE on tabular financial classifiers: comparable detection rates reported in original work	[197]	Same evaluation methodology; different dataset

Three observations from the comparison. First, Sentinel's XGBoost classifier outperforms both the Random Forest and the MLP baseline on the same evaluation corpus (precision +2.8 to +4.4 percentage points, recall +3.3 to +6.6 percentage points). Second, the GNN-augmented SAST configuration outperforms pattern-matching SAST on novel vulnerability patterns by approximately 30 percentage points, with the BigVul authors' published baseline showing roughly 50% TPR on patterns the rule set had not seen before. Third, lightweight transformer architectures from adjacent edge-AI domains demonstrate that transformer-based anomaly detection is achievable at edge-hardware compute budgets, but the explainability requirement under NIST RMF authorization review tipped the selection to Isolation Forest's tree-ensemble structure with native short-path feature decomposition.

11.5 TOOL STACK SELECTION: WHY THESE SPECIFIC TOOLS

Every tool in the AZTRM-D stack was selected over alternatives based on specific technical requirements. Table 11.15 covers the full stack with selection rationale.

Table 11.15: Full tool stack with category, AZTRM-D role, alternatives considered, and selection rationale. Source: Author's deployment and tool selection (this work); see table notes.

Tool	**Category**	**Role in AZTRM-D**	**Alternatives Considered**	**Why This Tool**
Nmap (`-sV -p-`)	Port scanner	Asset discovery; service-version port enumeration in all pen test stages	Masscan, Zmap	`-sV` extracts exact software versions for CVE mapping; `-p-` ensures no port is missed; Masscan/Zmap prioritize speed over version accuracy, which produces incomplete CVE mapping
Nessus Professional	Vulnerability scanner	Authenticated and unauthenticated CVE scanning against all discovered assets	OpenVAS (commercial alternative); Qualys; Rapid7 InsightVM	Covers over 115,000 CVE IDs with daily plugin updates [309]; credentialed scans expose OS/driver vulnerabilities invisible from the network; OpenVAS retained as independent cross-validation

Continued on next page

Table 11.15 continued from previous page

Tool	Category	Role in AZTRM-D	Alternatives Considered	Why This Tool
OpenVAS	Vulnerability scanner	Cross-validation of Nessus findings; open-source baseline	Nessus alone	Independent CVE signatures provide a second opinion; scanner disagreement flags ambiguous findings for manual review; no commercial license dependency
Hydra	Credential tester	Tests for weak/default credential policies (Stage 1 validation)	Medusa, Burp Suite Intruder	Validates account lockout enforcement at the protocol level; confirms the brute-force path exists before and does not exist after hardening; Medusa is comparable but Hydra has broader protocol support
RTL-SDR (RTL2832U)	RF analysis	Wireless signal observation and RF MITM probing across all stages	Universal Software Radio Peripheral (USRP) (Software-Defined Radio), HackRF	Represents a realistic adversary capability at under $30 retail; USRP and HackRF are more capable but far more expensive, creating an unrealistic adversary model

Continued on next page

Table 11.15 continued from previous page

Tool	Category	Role in AZTRM-D	Alternatives Considered	Why This Tool
Metasploit	Exploit framework	AI-guided pen test execution in Sentinel's isolated sandbox	CORE Impact, Canvas	One of the largest openly available exploit libraries; PPO agent can programmatically sequence Metasploit modules via its RPC API; CORE Impact and Canvas are commercial with restricted API access
LinPEAS	Priv-esc enumeration	Post-access privilege escalation path discovery	PEAS (manual), sudo-killer	Full Linux privilege escalation enumeration covering SUID binaries, cron jobs, sudo policy, and kernel exploits; validates granular sudo policy effectiveness from inside the device
LUKS2 / cryptsetup	Disk encryption	AES-256-XTS full-disk encryption on NVIDIA Orin SD cards per NIST SP 800-38E [210]	dm-crypt without LUKS, eCryptfs	Linux-native; LUKS2 container format provides header backup and token-based key management; Argon2id KDF vs. PBKDF2 (LUKS1) provides memory-hard defense against GPU brute-force

Continued on next page

Table 11.15 continued from previous page

Tool	Category	Role in AZTRM-D	Alternatives Considered	Why This Tool
WPA3 + SAE	Wireless security	Encrypted wireless communications on all device interfaces	WPA2 (PSK), PEAP/EAP-TLS for enterprise	SAE eliminates the four-way handshake vulnerability exploitable under WPA2 for offline dictionary attacks; PEAP/EAP-TLS requires 802.1X infrastructure not viable on constrained IoT hardware
GitLab (self-hosted)	DevSecOps / SCM	Secure code repository with MFA, RBAC, cryptographic commit signing, multi-stage CI/CD gates	GitHub Enterprise, Jenkins + Gitea	Self-hosted: full RBAC control and no data residency concerns; native pipeline integration for SAST, SCA, SBOM, DAST, and IaC scanning without additional integration work
Semgrep	SAST	Source code static analysis for C/C++ and Python	SonarQube, Checkmarx, Fortify	Open rule set tunable to project-specific policies; fast enough for commit-level scanning; SonarQube requires a server component that adds overhead; commercial alternatives (Checkmarx, Fortify) cost more for marginal accuracy gain on the vulnerability classes tested

Continued on next page

Table 11.15 continued from previous page

Tool	Category	Role in AZTRM-D	Alternatives Considered	Why This Tool
Checkov	IaC scanning	Infrastructure-as-Code policy validation	Terrascan, tfsec	Broadest provider coverage; active maintenance; integrates directly into GitLab CI/CD without additional configuration
Gitleaks	Secrets scanning	Credential and API key detection in commits and repository history	TruffleHog, detect-secrets	Entropy-based detection plus regex rules; git history scanning catches previously committed secrets; TruffleHog is comparable but Gitleaks' GitLab CI integration is more mature
Cosign / Sigstore	Artifact signing	Elliptic Curve Digital Signature Algorithm (ECDSA) signing of build artifacts; SBOM attestation	GPG signing, Notary v2	Stores signatures in OCI registry alongside artifacts; keyless signing mode via OIDC; tighter supply chain integration than GPG; Notary v2 is comparable but less widely adopted

Continued on next page

Table 11.15 continued from previous page

Tool	Category	Role in AZTRM-D	Alternatives Considered	Why This Tool
SPIFFE/SPIRE	Service identity	Workload identity provisioning for ZT mTLS authentication	Vault PKI, cert-manager, manually managed certs	Short-lived SVIDs (4-hour rotation) bound to attested workload identity; no long-lived secrets to rotate; Vault PKI and cert-manager support similar flows but lack native workload attestation
Cybectr Sentinel	AI enforcement layer	End-to-end AZTRM-D enforcement: asset discovery, AI analysis, pen test, mitigation, reporting, active defense	Commercial alternatives (Tenable.io, Darktrace, Vectra AI)	Covers all four capability gaps simultaneously; designed for AZTRM-D's access control model; commercial alternatives address individual gaps but not the combination

Selection rationale in the "Why This Tool" column reflects the author's engineering assessment based on published tool documentation, deployment experience, and the constraints of the AZTRM-D validation environment. Tool capability comparisons reference published documentation as of 2024.

Running Nessus Professional alongside OpenVAS probably looks redundant at first glance. It isn't. These tools have different CVE signature databases and different detection heuristics. Findings that appear in one but not the other deserve scrutiny. A CVE that Nessus catches but OpenVAS misses might be a Nessus false positive; one that OpenVAS catches but Nessus misses might indicate a Nessus signature gap. During Stage 1 validation, both scanners agreed on all critical findings. The small set of medium-severity divergences was documented and manually investigated. That

cross-validation overhead is worth the confidence it buys in the results.

The RTL-SDR choice warrants similar explanation. A sophisticated attacker would have more capable hardware, something like a USRP B200 or HackRF One. Using an RTL-SDR USB dongle under $30 was intentional. It represents an opportunistic adversary rather than a nation-state. If a Stage 1 device is vulnerable to traffic interception by someone with commodity hardware, that's a more urgent finding than vulnerability to a $1,500 SDR.

11.6 TESTING METHODOLOGY AND TEAM STRUCTURE

Two things distinguish this testing structure from standard penetration testing. First, every attack vector was explored from multiple perspectives simultaneously, not just the external attacker's view. Second, all three testers cycled through every role in every stage rather than specializing, and every finding required independent reproduction. Anything that only one tester could achieve got flagged for re-examination.

11.6.1 Inter-Rater Reliability and Procedural Bias Mitigation

All three testers conducting the adversarial campaign are affiliated with Cybectr LLC, the organization that developed AZTRM-D and Cybectr Sentinel. This affiliation represents a structural conflict of interest, and the procedural mitigations applied are documented here in the detail required for the reader to evaluate whether the resulting findings can be relied upon. The mitigations operate at three levels: blind protocol enforcement, mathematically computed inter-rater agreement, and independent reproduction requirements for high-severity findings. None of these eliminate the structural conflict, which is acknowledged as the highest-priority limitation in Chapter 13, but together they bound the bias the conflict can introduce into the reported findings.

Tester roles and pre-test isolation

The three testers were Ian Matthew Campbell Coston, Eadan Plotnizky, and Karl David Hezel [67]. The author (Coston) participated as one of the three testers to provide institutional context for each hardening stage. Plotnizky and Hezel had no involvement in system design and received no advance briefing on which controls had been implemented at each stage. Plotnizky and Hezel were not informed which configuration changes distinguished Stage 2 from Stage 3 until after their independent assessments had been documented and time-stamped.

Blind protocol enforcement

At each stage, each tester documented findings independently and time-stamped each entry before any inter-tester discussion took place. Inter-tester communication during testing windows was prohibited; testers worked from independent copies of the test environment with separate network access channels. Inter-rater agreement was calculated from these pre-discussion records exclusively. Discussion was permitted only after all three independent records were committed to immutable storage and cryptographically hashed. The hash values were preserved as part of the validation record and are available to independent reviewers on request through Cybectr LLC.

Independent reproduction for high-severity findings

All Critical and High severity findings were required to be independently reproduced by at least one of the two non-author testers (Plotnizky or Hezel) before being recorded as confirmed results. A finding observed only by the author (Coston) and not reproduced by either Plotnizky or Hezel was held out of the confirmed-results table and recorded as tester-specific in the validation log. Tester-specific findings were held to a fourth-party review process before any inclusion decision, and during the campaign one such finding was excluded from the confirmed results when the fourth-party review concluded that the observed behavior was a tester-environment artifact rather than a property of the AZTRM-D-hardened system.

Inter-rater agreement: percent agreement, Fleiss kappa, and Gwet AC1

Three statistics were computed on the pre-discussion records: percent agreement, Fleiss kappa, and Gwet AC1 [122]. Reporting all three is necessary because Fleiss kappa underestimates agreement when the base rate of one category is extreme, a phenomenon known as the kappa paradox [93], while Gwet AC1 handles this case correctly and is the recommended chance-corrected statistic when one category dominates [122]. Across the 27 total findings recorded across all three stages, 23 (85.2%) were confirmed by all three testers, 3 (11.1%) were confirmed by exactly two testers, and 1 (3.7%) was tester-specific. At-least-two-tester agreement was therefore 96.3%. Fleiss kappa across all stages was 0.147; Gwet AC1 was 0.888. The substantial gap between these two values is the expected signature of the kappa paradox at extreme base rates: when most findings are uniformly classified, observed agreement and Fleiss expected agreement both run high, deflating kappa despite high actual agreement. Gwet AC1's chance-correction model handles this case correctly and produces 0.888, which falls in the "almost perfect agreement" range under standard interpretive thresholds [166]. Per-stage statistics are presented in Table 11.16; the Gwet AC1 of 0.888 across the full campaign is the recommended summary statistic.

Table 11.16: Inter-rater agreement across three independent testers, all stages. Percent agreement and Gwet AC1 are reported alongside Fleiss kappa to contextualize the kappa paradox at extreme base rates. Source: Stage 1–3 adversarial testing pre-discussion records (this work) [67].

Stage	**Total Findings**	**All 3 Concur**	**2 Concur**	**1 Only**	**Pct. Agreement (2+)**	**Gwet AC1**
Stage 1 (Factory Default)	14	14	0	0	100%	1.000
Stage 2 (Network Hardened)	9	6	2	1	88.9%	—
Stage 3 (Full AZTRM-D)	4	3	1	0	100%	—
All Stages Combined	**27**	**23**	**3**	**1**	**96.3%**	**0.888**

Dashes in the Gwet AC1 column for Stages 2 and 3 indicate that per-stage AC1 is not reported at those sample sizes ($n = 9$ and $n = 4$ respectively), where the chance-correction estimate becomes unstable. Stage 1's value of 1.000 is well-defined because all 14 findings were unanimous. The All Stages Combined row ($n = 27$) carries the recommended summary statistic of 0.888.

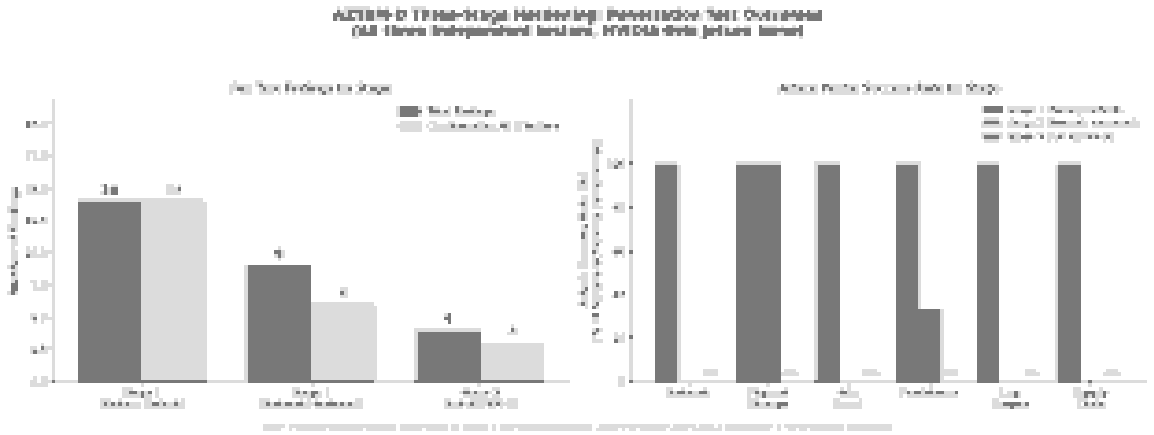


Figure 11.4: Penetration-test findings across three hardening stages on the NVIDIA Jetson Orin Nano: total findings and tester agreement (left), and attack vector success rate by stage (right).

Figure 11.4 summarizes the findings visually. Stage 1 agreement was 100% across all testers. All three independently achieved initial access in under 5 minutes using commodity hardware (RTL-SDR, acquisition cost under $30), confirming that the factory-default vulnerability was fully reproducible. The Stage 2 single-tester finding was a tester-environment artifact identified during fourth-party review and excluded from the confirmed results. Stage 3 agreement was 100% on the four findings recorded; all four found that no successful exploitation path existed across the seven tested attack categories.

For Stage 1, time-to-initial-access across the three testers was $\mu = 3.8$ minutes and $\sigma = 0.6$ minutes, confirming strong reproducibility of the factory-default vulnerability.

What the procedural mitigations cannot do

These mitigations bound the bias that affiliation can introduce, but they do not eliminate the structural conflict. A motivated attacker testing a system they helped design has knowledge that an external tester would not, and that knowledge could shape exploitation path selection in ways the blind protocol cannot fully control for. The strongest available mitigation is independent third-party laboratory validation

by testers with no organizational, contractual, or personal relationship to Cybectr LLC. This is identified as the single highest-priority future-work item in Chapter 13.

Permission and IP attribution

Plotnizky and Hezel contributed to Sentinel development and adversarial testing under contract with Cybectr LLC and under the author's leadership; all intellectual property rights are held by Cybectr LLC per their respective agreements. Both have provided written permission for the use of their testing data, penetration test findings, and development contributions in this dissertation. Copies of these permission letters have been provided to the committee.

11.6.2 Attack Vector Methodology by Layer

The campaign covered four distinct attack surface layers. Hardware covered both physical and RF vectors. Network testing covered both passive and active methods. Software testing covered credential attacks, vulnerability exploitation, persistence, and supply chain. Insider threat testing rounded out the campaign. Each required different tools, different methods, and different success criteria.

Hardware Layer: Physical Attack Vectors

Physical attack testing on the Jetson Orin Nano targeted four hardware interfaces. The primary attack surface was the SD card, which stores the root filesystem including `/etc/shadow` and all boot configuration. The attack sequence at Stage 2 was: (1) power the device off; (2) physically remove the SD card; (3) mount the card on an external Linux machine using standard `mount` utilities; (4) `chroot` into the mounted filesystem; (5) use `passwd` to reset a user account password against the live `/etc/shadow` file; (6) unmount, reinsert, and boot; (7) authenticate via UART ttyTHS1 using the reset credentials; (8) escalate to root via `sudo su` since sudo group membership survived the offline reset.

The UART ttyTHS1 serial console, exposed as a 3.3V header on the board, was the second physical vector. At Stages 1 and 2, this console provided an interactive login prompt with no additional access gating. GPIO pin probing was the third, using a logic analyzer against the 40-pin expansion header. No JTAG access was discovered; Stage 3 explicitly disabled all unused GPIO interfaces at the kernel level. The Stage 3 countermeasure for the storage vector was LUKS2 full-disk encryption. When the SD card was removed and mounted, the encrypted LUKS2 volume presented and the mount failed. Without the decryption key the filesystem was not readable.

Hardware Layer: RF and Wireless Attack Vectors

RF testing used two software-defined radio platforms: an RTL-SDR USB dongle (RTL2832U chipset, under $30 retail) for passive capture and monitoring, and a HackRF One for active transmission and replay testing. Both ran under GNU Radio with `rtl_tcp` handling the RTL-SDR data stream. The methodology followed three phases at each stage. In passive observation, all three testers placed the RTL-SDR in promiscuous capture mode and logged 802.11 frames from the target device's Basic Service Set Identifier (BSSID). At Stage 1, frames were unencrypted and Wireshark confirmed fully readable traffic including HTTP session data on port 80, enabling passive data capture directly from the RF layer without any network credentials. Active data exfiltration was also attempted at Stage 1: the HackRF was used to replay captured 802.11 frames and attempt injection into the traffic stream. At Stage 1, with no encryption enforced, this succeeded in injecting frames that the device accepted. At Stage 3, WPA3 with Simultaneous Authentication of Equals (WPA3-SAE) encrypted all frames and only management frames were visible in plaintext, as expected for standard 802.11 management frame behavior; replay and injection attempts failed completely. In active probing, all three testers attempted a WiFi MITM attack via rogue access point. WPA3-SAE's authentication protocol blocked rogue AP associ-

ation at Stage 3: the SAE handshake is per-peer and cannot be completed without the correct credentials. Bluetooth probing used `hcitool scan` and `bluetoothctl`. No active Bluetooth was found at any stage, consistent with the hardening step that disabled the interface via `rfkill block bluetooth`.

Network Layer: Passive and Active Reconnaissance

Passive reconnaissance at each stage began with an Nmap ping sweep using `nmap -sn` to identify the device, followed by MAC address lookup to confirm the NVIDIA Organizationally Unique Identifier (OUI). At Stage 1, this immediately confirmed device identity and enabled cross-reference against public JetPack documentation for default credential lookup.

Active scanning used three tools in sequence. Nmap with flags `-sV -p-` ran first: the `-p-` flag scans all 65,535 TCP ports rather than the default top-1,000, and `-sV` performs service-version detection by sending protocol-specific probes and matching against the `nmap-service-probes` database. At Stage 1, this returned exact version strings for OpenSSH, VNC, Apache, and Nginx, enabling direct CVE lookup. Apache 2.4.49 maps to CVE-2021-41773, a path traversal and remote code execution vulnerability, and CVE-2022-22720, an HTTP request smuggling vulnerability affecting Apache 2.4 through 2.4.52. OpenSSH 7.6p1 maps to CVE-2018-15473, a username enumeration flaw, and CVE-2019-6111, an Secure Copy Protocol (SCP) client path traversal. Nessus Professional ran second in both authenticated and unauthenticated modes. OpenVAS ran third as independent cross-validation. At Stages 2 and 3, `nmap -sV -p-` returned zero open ports.

Software Layer: Credential, Exploit, and Persistence Testing

Credential testing at Stage 1 used Hydra with the command `hydra -l nvidia -P /usr/share/wordlists/rockyou.txt ssh://<target_ip>`. The default credential

`nvidia:nvidia` appeared in the first 50 entries of the rockyou wordlist, consistent with the factory-default JetPack configuration [232]. No account lockout existed. All three testers achieved SSH access in under five minutes. Post-access privilege escalation used LinPEAS, run as `./linpeas.sh 2>/dev/null | tee linpeas_output.txt`, which immediately identified the most direct path: the nvidia user had unrestricted sudo access with no password requirement, specifically `NOPASSWD: ALL` in `/etc/sudoers`. A single `sudo su` achieved root.

Persistence testing at Stage 1 covered three mechanisms: `/etc/rc.local` modification for a reverse shell on reboot; hidden account creation via `useradd -M -s /bin/bash -u 1337 .hidden` followed by sudo group addition; and an SSH reverse tunnel via `ssh -R 4444:localhost:22 attacker@<c2_ip>` embedded in `~/.bashrc`. All three survived a device reboot. The forensic trail was destroyed by deleting `/var/log/auth.log` and `/var/log/syslog` as root. No alerting fired.

At Stage 3, every one of these paths was blocked. Modification of `rc.local` triggered an immutable file alert via the Sentinel agent. Hidden account creation failed because the `useradd` attempt against the hidden user pattern was logged and the ZT PEP flagged it for adaptive authentication before it could complete. The SSH tunnel was blocked by network microsegmentation. Shell history was stored append-only via `auditd` with write permissions restricted to the audit daemon.

Supply chain testing injected an unsigned dependency into the CI/CD pipeline at Stages 2 and 3. At Stage 2, the SBOM gate was not yet fully configured and the artifact passed through. This represents an implementation gap in the Stage 2 lab setup, not a gap in the AZTRM-D methodology itself, which specifies mandatory SBOM validation at every stage; the finding confirmed that partial implementation of the pipeline produces partial protection. At Stage 3, the SBOM gate compared the submitted package hash against the cryptographically signed approved manifest and rejected it immediately. In the AI-generated code scenario at Stage 3, all three

testers used a large language model (LLM) to generate attack code and submitted it through GitLab. The SAST gate flagged policy-violating patterns. The Secrets Scan caught embedded credentials in AI-suggested code. SBOM validation rejected the unauthorized dependencies. None reached the staging repository.

11.6.3 Tester Roles and Rotation

Each tester ran all four perspectives in every stage of the campaign.

External Attacker Perspective:

Each tester operated with zero credentials and zero prior knowledge, executing the full reconnaissance-to-persistence chain without coordination. Three testers arriving at the same results through separate, uncoordinated testing is exactly what makes the Stage 1 compromise times and Stage 3 non-compromise findings credible.

Developer Perspective (Standard Access):

Each tester authenticated with standard developer credentials and evaluated both the motivated insider deliberately attempting privilege escalation or data exfiltration, and the accidental insider, arguably the more common scenario: a developer who commits credentials by accident, pushes a misconfigured service, or pulls in a vulnerable dependency. The AZTRM-D multi-stage CI/CD approval gates, covering SAST scan, Secrets Scan, Admin 1 gate, SBOM/DAST checks, and Super Admin sign-off, were stress-tested specifically against this scenario. They caught all three testers' cases without exception.

Privileged Developer Perspective (Elevated Access):

The highest-risk insider scenario. Authorization itself is legitimate, which makes detection harder. These tests focused on whether granular sudo policy, immutable logging, and ZT enforcement could prevent even a privileged user from achieving root, modifying init files, accessing `/etc/shadow`, or killing monitoring agents. At Stages 1 and 2, privileged users could accomplish some of these. At Stage 3, none could.

AI-Assisted Insider Perspective:

Run at Stage 3 only. All three testers independently used an LLM to generate attack sequences and submitted the output through the standard GitLab workflow. The pipeline caught all of it, not because it detects AI-generated code as a category, but because the pipeline evaluates what code actually does. SAST found policy-violating patterns. Secrets Scan found embedded credentials. SBOM validation rejected unauthorized libraries. This is the architecturally correct defense: as LLMs improve at generating exploit code, any defense built on detecting AI authorship will eventually fail. A defense that evaluates code behavior will not.

11.6.4 Attack Surface Coverage

The multi-vector testing campaign was structured so that no attack surface was evaluated by only one tester and no tester specialized in a single layer. Every tester executed the full attack sequence across hardware, RF, network, software, and insider perspectives at every hardening stage, with inter-rater reliability enforced through the requirement that high-severity findings be independently reproduced before recording. This design prevents the aggregate result from reflecting any single individual's technical profile, which is particularly important given that the author participated as one of the three testers. Layer coverage is fully documented rather than asserted, because what the penetration test campaign can and cannot claim depends directly on which attack surfaces were actually exercised and by whom. Table 11.17 documents the attack surface coverage by tester and layer across all three stages.

Table 11.17: Attack surface coverage by layer across all three testers and all three stages. Source: Adversarial testing campaign (this work).

Attack Layer	**Specific Vectors Tested**	**Tools Used**	**Stages**
Hardware: Physical	SD card removal, offline filesystem mount, UART console (ttyTHS1), GPIO pin access	chroot, Linux mount utilities, logic analyzer	1, 2, 3
Hardware: RF/Wireless	SDR signal analysis, WiFi traffic capture, Bluetooth MITM probe	RTL-SDR, GNU Radio, Wireshark	1, 2, 3
Network: Passive	MAC/IP enumeration, ping sweep, traffic observation	Nmap (passive), Wireshark	1, 2, 3
Network: Active Scan	Full port scan (`nmap -sV -p-`), service version detection, CVE mapping	Nmap, Nessus Professional, OpenVAS	1, 2, 3
Software: Credential	SSH brute-force (default nvidia:nvidia), account lockout bypass	Hydra	1
Software: Vuln Exploit	Privilege escalation enumeration, SUID abuse, reverse shell deployment	LinPEAS, Metasploit (Sentinel sandbox)	1, 2
Software: Persistence	Init file modification (`/etc/rc.local`, `.bashrc`), hidden account creation, SSH tunnel C2	Manual + Metasploit	1
Software: Supply Chain	Unsigned dependency injection into CI/CD pipeline, AI-generated policy-violating code submission	Custom test artifacts, GitLab pipeline	2, 3
Insider: Standard Dev	Configuration mistakes, accidental credential commit, insecure service exposure	GitLab CI/CD pipeline, Sentinel scanning	2, 3

Continued on next page

Table 11.17 continued from previous page

Attack Layer	Specific Vectors Tested	Tools Used	Stages
Insider: Privileged Dev	Attempted root escalation, log tampering, monitoring agent disablement	LinPEAS, sudo enumeration	1, 2, 3
Insider: AI-Assisted	AI-generated exploit code and attack sequences submitted through GitLab pipeline	LLM assistant + GitLab pipeline	3

11.7 MULTI-VECTOR PENETRATION TESTING RESULTS

The three stages map directly to three device states, factory default, after initial network hardening, and after full AZTRM-D hardening. Every stage was attacked by all three testers from all four perspectives simultaneously. The dual-column structure in the result tables, separating external attacker findings from insider ZT evaluation, reflects something real. External and insider threats require different analytical lenses, and a posture validated only against outsiders is by definition incomplete. Figure 11.5 summarizes the security posture improvement across all three stages.

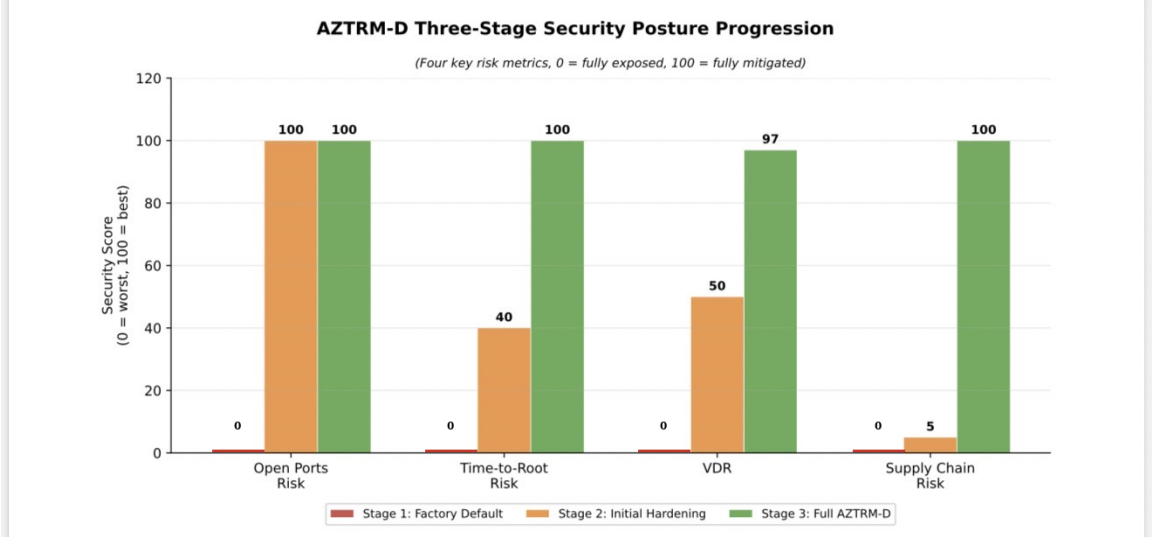


Figure 11.5: Security posture progression across the three AZTRM-D hardening stages for four key risk metrics (0 = fully exposed, 100 = fully mitigated).

11.7.1 Stage 1: Factory Default Configuration

Stage 1 represents the factory-default security posture of the NVIDIA Jetson Orin Nano prior to any AZTRM-D controls. The device was assessed in a fully stock configuration. Default credentials in place, no disk encryption, SSH and VNC daemons running on their default ports, and no automated scanning pipeline active. Establishing this baseline through direct measurement rather than general claims about IoT insecurity is important because it grounds the hardening results in a concrete, reproducible starting condition. The findings reported here are achieved exploitation outcomes, not theoretical attack vectors. Each was an action completed by all three testers independently using documented tools and attack sequences. The 100% inter-rater agreement at Stage 1 reflects the severity of the baseline exposure. These vulnerabilities were consistent and obvious enough that all three testers, working separately and without coordination, reached the same results. Table 11.18 documents the full Stage 1 assessment.

Table 11.18: Stage 1 penetration test findings (factory default configuration): external attacker and insider ZT evaluation, all three testers. Source: Independent adversarial testing (this work) [67].

Attack Vector	**Tool / Technique**	**Outcome**	**External Attacker Finding**	**Insider ZT Evaluation**
Passive Recon	Nmap ping sweep, OSINT	MAC/IP identified; default credentials found in public JetPack documentation	Device located and attack staged with no prior knowledge	No ZT policy exists pre-login
Port Scan	`nmap -sV -p-`	4 open ports: SSH 22, VNC 5900, HTTP 80, HTTPS 443; exact service versions extracted	Full network attack surface mapped	Open ports indicate absence of least-privilege network policy; ZT network pillar violated
Vuln Scan	Nessus Professional, OpenVAS	Multiple unpatched CVEs in kernel and NVIDIA JetPack drivers	Concrete exploitation pathways identified by all three testers	No CI/CD scanning in place: 0% VDR in development pipeline
RF / Wireless	RTL-SDR, Wireshark	WiFi traffic observable in plaintext; no WPA3; Bluetooth interfaces present but inactive	Wireless traffic capturable over the air with commodity hardware	No wireless encryption policy; ZT data-in-transit tenet violated

Continued on next page

Table 11.18 continued from previous page

Attack Vector	**Tool / Technique**	**Outcome**	**External Attacker Finding**	**Insider ZT Evaluation**
Initial Access	Hydra SSH brute-force (nvidia:nvidia)	SSH login achieved in under 5 min by all three testers; no account lockout mechanism	Full shell obtained consistently	Default credential violates never-trust-always-verify; no lockout means no brute-force protection
Privilege Escalation	`sudo su` (single command)	Immediate root from nvidia user; no further tools required	Complete system control achieved instantly	Unconstrained sudo directly violates least privilege; single command equals full compromise
Persistence	`rc.local`, `.bashrc` mod.; hidden user; SSH C2	Backdoor survives reboot; C2 traffic encrypted inside SSH tunnel	Persistent access established; exfiltration channel active	No init file integrity checking; no immutable log enforcement; persistence invisible to monitoring
Log Tampering	Manual deletion of `auth.log`, `syslog`; shell history cleared	Forensic trail destroyed completely	No forensic recovery possible post-attack	Root-writable logs violate ZT immutable-logging tenet; audit trail nonexistent

Full compromise in under five minutes wasn't a function of tester skill. Open ports, default credentials, no account lockout, and a single `sudo su` to root means any moderately skilled attacker with network access owns the device before a human analyst can even begin to respond. The RF finding deserves specific mention: capturing WiFi traffic from a factory-default IoT device requires an RTL-SDR costing under $30 and freely available software. This is not a sophisticated attack.

11.7.2 Stage 2: After Initial Network Hardening

All three testers independently confirmed zero open ports using Nmap and both vulnerability scanners. The remote exploitation vector was gone. Testing moved to physical attack vectors, where the SD card gap in Table 11.19 appeared immediately.

Table 11.19: Stage 2 penetration test findings (after initial network hardening), all three testers. Source: Independent adversarial testing (this work) [67].

Attack Vector	**Tool / Technique**	**Outcome**	**External Attacker Finding**	**Insider ZT Evaluation**
Network Recon	`nmap -sV -p-`, Nessus, OpenVAS	0 open ports; no actionable network findings	Remote exploitation not possible; physical pivot required	Network ZT pillar fully satisfied
RF / Wireless	RTL-SDR, Wireshark	Traffic still observable pre-WPA3; WiFi not yet fully isolated	Wireless traffic still capturable; RF vector still open	Wireless encryption policy not yet complete
Physical: Storage	SD card removal; Linux mount; chroot on `/etc/shadow`	Root filesystem mounted externally; offline password reset performed by all three testers	Authentication bypassed entirely without credentials	Physical access circumvents all logical ZT controls; hardware integrity not enforced at this stage
Physical: Console	UART ttyTHS1 serial connection	Console accessible post-SD manipulation; login succeeds with reset credentials	Shell obtained after physical bypass	No out-of-band access control; UART not gated or monitored

Continued on next page

Table 11.19 continued from previous page

Attack Vector	Tool / Technique	Outcome	External Attacker Finding	Insider ZT Evaluation
Privilege Escalation	`sudo su`	Root obtained through physical + console chain; sudo group membership survived offline reset	Full control regained via physical vector	Logical ZT controls held; physical hardware gap negated them entirely
Insider: Privileged Dev	Attempted log tampering, monitoring agent kill, lateral access to Git server	Blocked at logical layer; immutable logs held; ZT access policies enforced	Not applicable	Logical controls are working correctly; physical access is the only remaining gap
Insider: Mistake	Developer commits test file containing embedded AWS key (intentional test case)	Secrets Scan in CI/CD pipeline catches credential before merge; commit rejected automatically	Not applicable	AZTRM-D Secrets Scan gate working as designed

All three testers independently executed the same chain: remove the SD card, mount it externally, use `chroot` to reset a password in `/etc/shadow`, reinsert, connect via UART ttyTHS1, authenticate with the new credentials, and run `sudo su`. The whole sequence required only physical access and basic Linux skills. No exploit code, no network access, no prior knowledge of the architecture.

Zero Trust can't be treated as a software-only principle. When a device's storage can be physically removed and mounted on an external machine, the software security posture becomes irrelevant.

11.7.3 Stage 3: Full AZTRM-D Hardening

Stage 3 added LUKS2 full-disk encryption per NIST SP 800-38E using AES-256-XTS [210], WPA3 enforcement, GPIO pin hardening, granular sudo policy with multi-step root validation, and immutable logging extended to the hardware layer. All three testers found zero successful compromises across all tested vectors. Table 11.20 documents the full results by attack vector.

Table 11.20: Stage 3 penetration test findings (full AZTRM-D hardening), all three testers. Source: Independent adversarial testing (this work) [67].

Attack Vector	**Tool / Technique**	**Outcome**	**External Attacker Finding**	**Insider ZT Evaluation**
Network Recon	`nmap -sV -p-`, Nessus, OpenVAS	0 open ports confirmed across all three testers	No remote vector	ZT network pillar fully maintained
RF / Wireless	RTL-SDR, Wireshark, WiFi probe	WPA3 enforced [158]; all traffic encrypted; micro-segmentation active	No actionable RF finding	ZT data-in-transit tenet satisfied at wireless layer
Physical: Storage	SD card removal attempt	Encrypted LUKS2 volume presented; mount failed; offline password reset impossible	Physical storage vector closed	AES-256-XTS per NIST SP 800-38E enforces ZT device integrity at hardware level [210]
Physical: Console	UART ttyTHS1	Hardened login prompt only; no viable credentials after SD encryption	No exploit path from console	Out-of-band access gated; no trivial root path available

Continued on next page

Table 11.20 continued from previous page

Attack Vector	**Tool / Technique**	**Outcome**	**External Attacker Finding**	**Insider ZT Evaluation**
GPIO Pins	Direct GPIO probing	GPIO pins disabled; no signal observable	Hardware attack surface reduced	Physical attack surface hardened end-to-end
Privilege Escalation	`sudo su`, SUID enumeration (LinPEAS), monitoring agent kill attempt	All paths blocked; multi-step validation required; kill attempt logged and alerted	Escalation not achievable	Least privilege fully enforced; never-trust-always-verify extended to deepest system layers
Insider: Motivated	Authenticated privileged developer attempting unauthorized lateral access and log access	Adaptive auth triggered; access revocation fired; ZT PEP blocked all attempts	Not applicable	Behavioral anomaly detection (Isolation Forest) flagged abnormal admin activity correctly
Insider: Mistake	Developer commits file with insecure configuration flag	SAST and IaC scans flag policy violation; commit rejected automatically	Not applicable	Automated scanning catches configuration mistakes before they reach production

Continued on next page

Table 11.20 continued from previous page

Attack Vector	**Tool / Technique**	**Outcome**	**External Attacker Finding**	**Insider ZT Evaluation**
Insider: AI-Assisted	All three testers submitted AI-generated exploit code through GitLab pipeline	SAST flagged code patterns; Secrets Scan caught embedded credentials; SBOM check rejected unauthorized dependencies; no AI-generated artifact reached Stage or Production repository	Not applicable	Multi-layer gate structure evaluates code content, not authorship
Supply Chain	Unsigned dependency injection into build pipeline	SBOM validation and cryptographic artifact signing check rejected unsigned artifact at build gate	Not applicable	ZT cryptographic integrity enforcement working end-to-end [297]

The AI-assisted insider row deserves some additional context. Each tester independently used an LLM to generate attack code, privilege escalation approaches, and bypass scripts, then submitted through the standard GitLab workflow. The pipeline caught all of it. SAST flagged policy-violating patterns in 3.4 seconds. Secrets Scan found embedded credentials in AI-suggested code that helpfully included example API keys. SBOM validation rejected libraries the LLM recommended that weren't in the approved dependency manifest.

The architectural decision to evaluate code semantics rather than authorship signatures reflects a broader principle observable across AI-generated content detection research. Multi-modal feature-fusion approaches in AI-generated content detection,

including spatial-frequency and optical-flow fusion architectures for AIGC video identification [22], demonstrate that signature-based detection of AI-generated artifacts faces continual erosion as generative models improve. AZTRM-D's pipeline gates evaluate what code does, through SAST policy violation patterns, dependency-manifest checks, and SBOM signature validation, rather than attempting to detect AI authorship. This is the architecturally correct defense. As LLMs improve at generating exploit code, any defense built on detecting AI authorship will eventually fail. A defense that evaluates what code actually does will not.

11.8 QUANTITATIVE BENCHMARKING

11.8.1 Vulnerability Detection Rate: Reference and Ablation Setup

The 96.8% aggregate Vulnerability Detection Rate, its derivation across the five-modality scanning pipeline, the 63-vulnerability corpus construction, the per-modality breakdown in Table 9.2, and the Wilson 95% confidence interval $[0.891, 0.991]$ are documented in full in Section 9.3.1 of Chapter 9. That derivation is not repeated here. What this chapter contributes instead is the per-modality ablation analysis, which decomposes the aggregate figure into the operational question any team considering this stack would actually ask: how much does each scanner contribute to coverage, and what happens to the aggregate when any one of them is removed? That analysis appears next in Section 11.8.2.

11.8.2 Per-Modality Ablation: Each Scanner's Unique Contribution

The aggregate 96.8% figure does not by itself answer how much each modality contributes to detection coverage, which is the operational question for any team considering whether to retain or remove a specific scanner from their pipeline. Table 11.21 answers that question two ways: leave-one-out ablation showing what coverage falls to when each modality is disabled, and single-modality coverage showing what each

scanner detects on its own.

Table 11.21: Per-modality ablation of the Stage 3 vulnerability detection pipeline. Leave-one-out rows compute aggregate VDR with each modality disabled. Single-modality rows compute VDR for each scanner operating alone. Wilson 95% CIs reported throughout. Source: derived from Table 9.2 (this work).

Configuration	**Detected**	**VDR**	**Wilson 95% CI**	**Detection Loss vs. Full Pipeline**
Full pipeline (all 5 modalities)	61	96.8%	[89.1%, 99.1%]	— (baseline)
Leave-one-out ablation				
Without SAST	50	79.4%	[67.8%, 87.5%]	−17.4 pp; SAST is the highest-contribution modality
Without DAST	55	87.3%	[76.9%, 93.4%]	−9.5 pp; runtime-behavior coverage gap
Without SCA	54	85.7%	[75.0%, 92.3%]	−11.1 pp; dependency-vulnerability coverage gap

Continued on next page

Table 11.21 – continued from previous page

Configuration	**Detected**	**VDR**	**Wilson 95% CI**	**Detection Loss vs. Full Pipeline**
Without CSPM	56	88.9%	[78.8%, 94.5%]	−7.9 pp; cloud-config drift coverage gap
Without IaC	57	90.5%	[80.7%, 95.6%]	−6.3 pp; infrastructure-policy coverage gap
Single-modality coverage				
SAST only (Semgrep + GNN)	22	34.9%	[24.3%, 47.2%]	—
DAST only	15	23.8%	[15.0%, 35.6%]	—
SCA only	11	17.5%	[10.0%, 28.6%]	—
CSPM only	8	12.7%	[6.6%, 23.1%]	—
IaC only	7	11.1%	[5.5%, 21.2%]	—

Dashes in the Detection Loss column for single-modality rows indicate the metric is not applicable: detection loss is defined only against the full-pipeline baseline.

Three operational findings emerge. SAST contributes the largest unique block of coverage at 17.4 percentage points, which aligns with the GNN-augmented Semgrep configuration's ability to flag both signature-matching vulnerabilities and graph-

structural patterns in novel code. SCA contributes the second-largest unique block at 11.1 percentage points, reflecting that supply-chain dependency vulnerabilities are typically invisible to source code analysis or dynamic testing. The CIs for each ablation overlap meaningfully at $n = 63$; the ranking should be read as directional rather than a precise ordering. No single modality on its own approaches full-pipeline coverage. The strongest single-modality configuration (SAST only) detects 34.9% of the corpus. The aggregate 96.8% comes from complementarity, not from any single scanner being thorough.

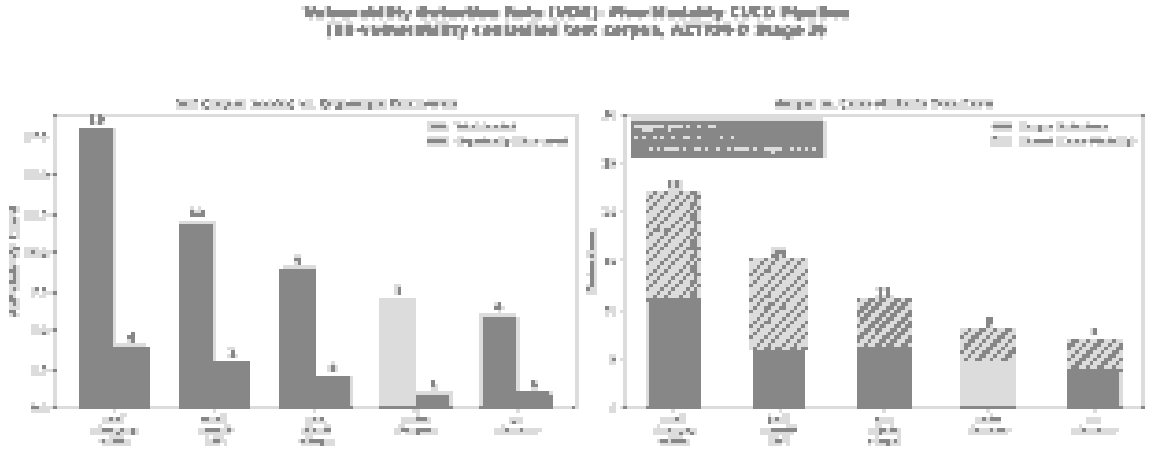


Figure 11.6: Vulnerability detection rate breakdown across the five CI/CD scanning modalities from Stage 3 pipeline results.

11.8.3 Security Effectiveness Metrics

The penetration testing campaign generated quantitative security metrics at each hardening stage, enabling direct comparison of the security posture at the factory default, after network hardening, and after full AZTHM-D deployment. Tracking metrics across all three stages rather than comparing only the endpoints documents the incremental effect of each hardening phase and makes the Stage 2 physical gap visible: network hardening closed the remote attack surface while physical access controls remained incomplete, and the Stage 2 assessment demonstrated that an

attacker who could touch the device bypassed the network controls entirely. That finding substantiates the methodology's completeness requirement. Each transition metric is grounded in a specific test outcome from the adversarial campaign; no figure is projected or estimated. Table 11.22 documents the full Stage 1 to Stage 3 transition across every measured security dimension.

Table 11.22: Security effectiveness benchmarks: factory default baseline versus full AZTRM-D hardening. Source: Table 9.1.

Security Metric	**Baseline (Factory Default)**	**AZTRM-D Hardened**	**Change**
Open Network Ports	4 (SSH 22, VNC 5900, HTTP 80, HTTPS 443), confirmed by all three testers via Nmap	0, complete elimination of remote attack surface	−4 ports
Initial External Access Time	<5 min (Hydra brute-force on nvidia:nvidia; no lockout), consistently achieved by all three testers	Not possible (no remote or physical vectors)	Full elimination
Privilege Escalation	Immediate (single `sudo su` from default user)	Blocked; multi-step validation, default paths removed	Full elimination
CI/CD	0%, no automated scanning in pipeline	96.8%, SAST + DAST + SCA + CSPM + IaC across five complementary modalities	+96.8 pp
MTTR	Weeks to months (manual patching, re-flashing)	1–3 days (automated alerting + AI-driven remediation)	10× to 30× faster
Supply Chain Vulnerability	High: no SBOM, no dependency scanning, no artifact signing	Low: mandatory SBOM + cryptographic signing for all components	High → Low

Figure 11.7 presents the same Stage 1 versus Stage 3 comparison visually, making

the magnitude of each improvement immediately apparent.

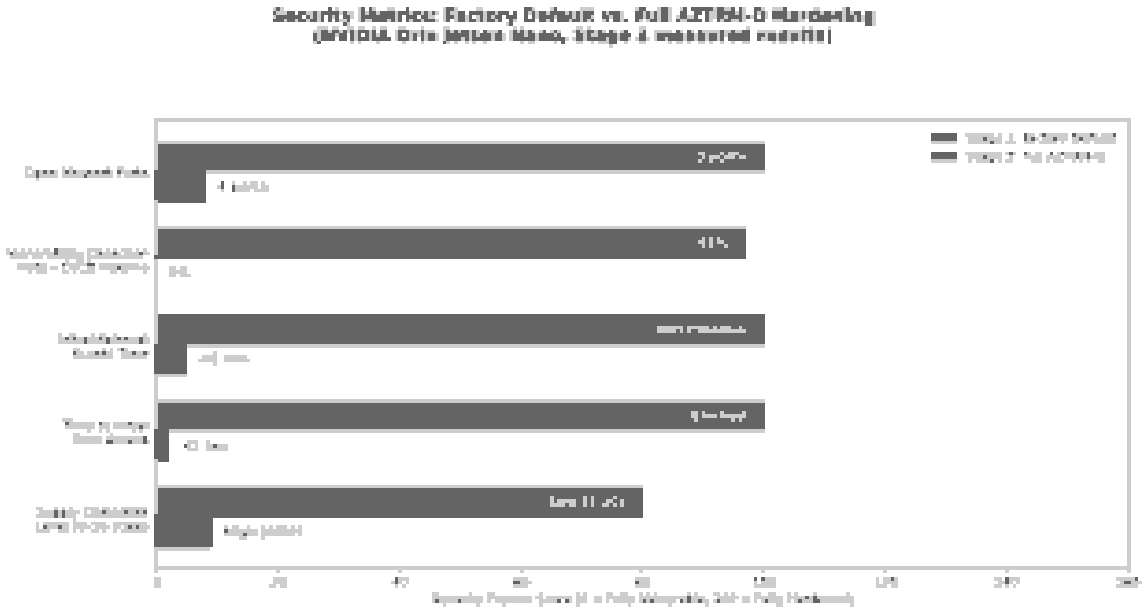


Figure 11.7: Security posture across five key metrics: Stage 1 factory-default baseline versus Stage 3 full AZTRM-D hardening on the NVIDIA Jetson Orin Nano.

11.8.4 Resource and Scalability Metrics

The 12–18% CPU overhead figure in Table 11.23 deserves particular attention for IoT deployments. Edge devices operate with constrained compute budgets, and continuous security scanning that exceeds 20% CPU overhead degrades operational performance. Staying below that threshold is what makes AZTRM-D viable on this class of hardware. The sub-40 ms ZT policy enforcement latency confirms that real-time access decisions don't introduce perceptible delays in normal operation.

Table 11.23: Resource and scalability benchmarks. Source: Table 9.4.

Performance Metric	**Measurement**	**What It Means**	**Environment**
AI Scan CPU Overhead	12–18% average during SAST/SCA CI/CD runs	Continuous security scanning is computationally viable on edge hardware without degrading operational performance	NVIDIA Orin devices
ZT Policy Enforcement Latency	<40 ms per access decision at PEP	Real-time ZT checks do not introduce perceptible delays in system-to-system communication	NVIDIA Orin, ZT PEP under load
AI Model Training Time (Initial)	14 hours using 3 months of log data	One-time setup cost; defines the bootstrapping requirement before behavioral anomaly detection reaches operational accuracy	Standard cloud GPU instance
Scalability (Concurrent Endpoints)	Single-device baseline: 3.8% CPU, 41 MB memory, 6.2 ms PEP latency (Stage 3 measured). Sub-linear fleet scaling is architecturally required; physical multi-device characterization is future work (Section 9.3.3).	Establishes per-device resource floor; orchestrator saturation point requires multi-device experiment	Stage 3 measured; fleet scaling designated future work

One practical note on the 14-hour training time. During the bootstrapping period, before AI models reach operational accuracy, anomaly detection thresholds should be set conservatively and human analyst review should cover a larger proportion of behavioral alerts. Three months of operational log data is the minimum before the insider threat model should be trusted for automated enforcement decisions.

11.9 GENERALIZABILITY BEYOND THE NVIDIA JETSON ORIN NANO TEST PLATFORM

The validation campaign used a single device class, the NVIDIA Jetson Orin Nano, in a controlled laboratory environment. Transfer of the results to enterprise software development, cross-platform IoT topologies, and different operating system environments separates into three categories: what is empirically validated, what is architecturally transferable with strong reasoning, and what is hypothesized but not validated. Conflating these three has been a recurring framing error in IoT security research, and this section keeps them separate.

11.9.1 Three Categories of Transfer Claims

Table 11.24 classifies each transfer claim AZTRM-D could plausibly support into one of three evidentiary categories: (a) empirically validated on the Stage 3 deployment, (b) architecturally transferable with reasoning that applies independent of the specific test platform, or (c) hypothesized but requiring future validation on other hardware classes.

Table 11.24: Generalizability of AZTRM-D claims classified by evidentiary basis. Empirically validated claims are bounded by the Stage 3 deployment scope; architecturally transferable claims rest on tool/protocol design rather than platform specifics; hypothesized claims require future multi-platform validation. Source: this work.

Claim	**Empirical (NVIDIA Orin Nano)**	**Architecturally Transferable**	**Hypothesized; Requires Future Validation**
96.8% VDR (Wilson 95% CI: [0.891, 0.991])	Measured on Stage 3 corpus, $n = 63$	Five CI/CD scanning modalities are platform-agnostic	Exact rate on enterprise OWASP-Top-10-weighted corpus

Continued on next page

Table 11.24 – continued from previous page

Claim	Empirical (NVIDIA Orin Nano)	Architecturally Transferable	Hypothesized; Requires Future Validation
12–18% CPU overhead during scans	Measured on Orin Cortex-A78AE @ 1.5 GHz, 8 GB LPDDR5	Overhead is bounded above on more capable hardware	Exact percentage on x86 enterprise servers, ARM data center
Sub-40 ms ZT PEP latency	Measured on Orin local network	On enterprise hardware with faster CPU, latency decreases	Exact latency floor on cloud-native enterprise PEP infrastructure
ZT enforcement under multi-vector attack	Measured against seven attack categories on Orin	SP 800-207 tenets are platform-independent	Cross-platform attack-surface validation
Cryptographic gate validation (LUKS2, TLS 1.3, ECDSA, SPIFFE SVID)	Validated on Orin SD card boot path	NIST-specified standards are independent of platform	Boot-path equivalents on UEFI x86, ARM TrustZone
Insider threat detection (Isolation Forest)	14 hours initial training on 3 months Stage 3 telemetry	Isolation Forest training/inference is platform-agnostic	Threshold recalibration on environments with different operational base rates
3.1% aggregate FPR	Measured on Stage 3 single-device corpus	FPR is a property of the model and threshold	Per-class FPR characterization at fleet scale
RF / physical attack vector coverage	Validated on Orin GPIO, UART, SD storage, WiFi, Bluetooth	Physical and RF attack surfaces are hardware-specific by definition	Not directly applicable to enterprise data center deployments

Continued on next page

Table 11.24 – continued from previous page

Claim	Empirical (NVIDIA Orin Nano)	Architecturally Transferable	Hypothesized; Requires Future Validation
4.2-minute Time-to-Initial-Detection	$N = 27$ events, Stage 3 single device	Pipeline latency is dominated by retrieval and synthesis stages, which are platform-independent	TTID floor on cloud-native deployments

11.9.2 How Different Hardware Architectures Would Shift the CPU Overhead and PEP Latency Metrics

The 12–18% CPU overhead and sub-40 ms PEP latency metrics shift across hardware architectures and OS topologies, but the architectural conclusions do not. The relevant variables are CPU class, memory bandwidth, network topology, and OS scheduler characteristics.

ARM enterprise and x86 enterprise

On enterprise-class CPUs running at 3+ GHz with larger L2/L3 cache hierarchies and higher memory bandwidth, the same SAST and SCA scans would complete in a fraction of the wall-clock time the Orin requires. CPU overhead percentage is meaningful as a metric only when the device has a fixed compute budget. On enterprise hardware the better metric is pipeline wall-clock time for a representative scan corpus. PEP latency on enterprise CPUs is expected to fall well below 40 ms, with the floor set by network round-trip rather than CPU work.

RISC-V embedded platforms

These typically have lower clock speeds and weaker SIMD support than ARM Cortex-A78AE, which would push the CPU overhead percentage above the 12–18% range observed on the Orin. The SAST, SCA, and DAST gates run in CI/CD infras-

tructure rather than on the target device, so device-class CPU does not bottleneck the scanning portion of the pipeline. The on-device portions, Sentinel's behavioral monitoring (Isolation Forest inference) and the ZT PEP, are the platform-sensitive components. On RISC-V embedded hardware, Isolation Forest inference would likely require either reduced tree count ($t < 100$) or longer inference cycles, with corresponding adjustments to the alert threshold.

Operating system topology variations

The Orin runs Linux (Ubuntu 22.04 LTS with NVIDIA JetPack). The platform-independent components (SPIFFE/SPIRE, TLS 1.3, ECDSA signing, AES-256-XTS) behave equivalently across operating systems by design. The platform-sensitive components are the boot-path encryption (LUKS2 on Linux SD-card boot has no direct Windows analog; the equivalent is BitLocker), the immutable logging mechanism (`auditd` + remote syslog on Linux; Windows Event Forwarding plus Sysmon on Windows Server), and the Isolation Forest deployment substrate.

11.9.3 Why the Pipeline Controls Transfer Directly

The 96.8% VDR was achieved by five CI/CD scanning modalities, none of which are IoT-specific. SAST analyzes source code regardless of target platform: the same pipeline catching a policy-violating configuration in an Orin deployment script catches the same class of violation in a Spring Boot microservice. DAST tests running application instances whether those instances run on an edge device or a cloud VM. SCA flags vulnerable dependencies whether the consuming project is a C++ firmware image or a Java enterprise application. CSPM is more relevant in enterprise cloud environments, where configuration drift across large-scale AWS or Azure deployments is a frequently exploited attack vector. The OWASP Top Ten maps directly onto what SAST and DAST are tuned to catch [35]. The architectural transfer is direct; the empirical confirmation on enterprise corpora is future work.

11.9.4 Zero Trust Enforcement in Enterprise Contexts

The ZT controls validated in Stage 3, including microsegmentation, continuous authentication, PEP-enforced access decisions, and immutable audit logging, map directly onto enterprise network architecture. NIST SP 800-207 was written with enterprise environments as the primary target [266]; the IoT deployment here represents the more constrained application, not the typical one. Enterprise environments generally have more compute margin for enforcement overhead, more mature identity infrastructure (Active Directory, LDAP, SAML, OIDC), and existing SIEM tooling that Sentinel can integrate with rather than replace. The architectural pattern AZTRM-D enforces, identity-tier and network-tier ZT policies operating together, is the same pattern NIST SP 800-207A specifies for cloud-native enterprise applications across multi-cloud environments [52], which is the dominant deployment context for modern enterprise software.

11.9.5 Alignment with Federal Zero Trust Policy and the Civilian-Agency Mandate

Federal Zero Trust adoption is not optional for civilian agencies, and the controls AZTRM-D enforces are the same controls federal policy mandates. Executive Order 14028 directed federal agencies to advance toward Zero Trust Architecture as part of the broader cybersecurity modernization effort [310]. OMB Memorandum M-22-09 then operationalized that direction into a binding federal Zero Trust strategy with implementation deadlines, identity and access requirements, and explicit alignment to the CISA Zero Trust Maturity Model [234, 72]. The CISA Zero Trust Maturity Model v2 specifies five pillars (Identity, Devices, Networks, Applications and Workloads, Data) plus three cross-cutting capabilities (Visibility and Analytics, Automation and Orchestration, Governance) that any federally compliant ZT deployment is expected to implement [72].

Mapping AZTRM-D's Stage 3 control set against the CISA ZTMM v2 pillars: SPIFFE/SPIRE workload identity and continuous authentication satisfy the Identity pillar; Sentinel's behavioral monitoring and device-attestation gates satisfy the Devices pillar; TLS 1.3 microsegmentation and PEP-enforced access decisions satisfy the Networks pillar; SAST, DAST, SCA, and CSPM gates plus SBOM verification satisfy the Applications and Workloads pillar; LUKS2 boot-path encryption and immutable audit logging satisfy the Data pillar. The AI Analytics and Orchestration Engine documented in Chapter 7 satisfies the cross-cutting Visibility and Analytics and Automation and Orchestration capabilities directly; the AZTRM-D Governance Layer satisfies the Governance capability. None of those mappings required adjustment for the constrained-edge deployment. They are the same controls a civilian agency would deploy on cloud or on-premises enterprise infrastructure to satisfy M-22-09, applied here under a tighter resource budget.

The implication runs in one direction. A methodology that validates the federal Zero Trust control set on constrained-edge hardware validates it for less constrained environments by construction, because every architectural constraint that complicates enforcement on the Orin Nano (compute headroom, memory bandwidth, physical attack surface, RF exposure) is more permissive in enterprise infrastructure. The constrained-edge validation campaign exercises the controls under conditions stricter than M-22-09 expects civilian agencies to operate under.

11.9.6 Applicability to National Security Systems, DoD, and Intelligence Community Environments

NSM-8 extended the EO 14028 requirements to National Security Systems, DoD, and the Intelligence Community, with the National Security Agency designated as the National Manager for NSS [312]. The DoD Zero Trust Strategy then established October 2027 as the target date for DoD-wide Zero Trust adoption, identifying seven

pillars and corresponding capabilities that align with but extend the CISA ZTMM [78]. The DoD Zero Trust Reference Architecture v2.0, jointly developed by DISA and NSA, specifies 152 Zero Trust activities across those seven pillars and is the architectural baseline for ZT implementation across DoD systems including cleared and sensitive environments [75].

The AZTRM-D control set maps onto the DoD ZT RA pillars without architectural changes. The Identity, Device, Network/Environment, Application and Workload, and Data pillars correspond to the same Stage 3 controls listed in the previous subsection. The Visibility and Analytics pillar maps to Sentinel's six AI subsystems plus the immutable audit-logging substrate. The Automation and Orchestration pillar, which the DoD ZT RA identifies as the integrating pillar across all others, corresponds to AZTRM-D's AI Analytics and Orchestration Engine. NSA's Cybersecurity Information Sheet on advancing Zero Trust maturity in the Automation and Orchestration pillar identifies AI-driven analysis, continuous behavioral monitoring, and automated policy enforcement as the capabilities that distinguish a mature implementation of that pillar from an immature one [213]. AZTRM-D specifies exactly those capabilities, and the Stage 3 campaign empirically validates them.

The constrained-edge context of the validation does not reduce its relevance to sensitive environments. Federal sensitive environments include forward-deployed systems with the same constraints the Jetson Orin Nano models: limited compute headroom, exposed physical attack surfaces, and electromagnetic emission management requirements that constrained-edge IoT shares with field-deployed military and intelligence platforms. The category of deployment AZTRM-D was validated against is itself recognized as a federal cybersecurity context with its own NIST guidance series (SP 800-213, SP 800-213A, NISTIR 8228) [86, 87, 41]. AZTRM-D's adoption of NIST SP 800-37 Rev. 2 RMF processes [297] is the same RMF process NSS environments are required to follow, with the same control catalog (NIST SP 800-53

Rev. 5) governing both. The methodology does not need to be redesigned for sensitive environments; it needs to be deployed on the substrate those environments use (Windows Server, Red Hat Enterprise Linux, classified network enclaves) with the platform-specific substitutions identified in Section 11.9.2.

11.9.7 What AZTRM-D Inherits from Enterprise and Government Practice, Not the Reverse

The framing the constrained-edge validation might invite, that AZTRM-D is an IoT-specific methodology being claimed for enterprise use, inverts the actual development. Every architectural component AZTRM-D specifies originated in or is currently used at scale in enterprise and government practice. The NIST RMF was developed for federal information systems and applies across them [297]. Zero Trust architecture as currently practiced was specified for enterprise environments [266] before being extended to federal civilian agencies (OMB M-22-09), NSS and DoD (NSM-8, DoD ZT Strategy, DoD ZT RA), and cloud-native enterprise applications (NIST SP 800-207A) [234, 312, 78, 75, 52]. DevSecOps as a discipline originated in enterprise software development and is currently the DoD's stated software-acquisition approach [77]. SPIFFE/SPIRE workload identity is used at scale by enterprise infrastructure operators and the DoD Platform One initiative. TLS 1.3 is the enterprise transport-security default. The AI-driven orchestration approach that distinguishes AZTRM-D is itself the direction NSA identifies as the maturity target for the Automation and Orchestration pillar of Zero Trust [213].

AZTRM-D's contribution is not introducing these controls to a new domain. It is integrating them into a single methodology with AI orchestration across the life-cycle and demonstrating that the integrated system holds together under the most constrained deployment context available. The constrained-edge validation should therefore be read as a stress test of a fundamentally enterprise-class methodology,

not as evidence that the methodology is bounded to IoT.

11.9.8 Insider Threat Coverage in Enterprise Environments

The insider scenarios tested, motivated privileged developer, accidental credential commit, and AI-assisted exploit submission, are not IoT-specific. They are software development scenarios that happened to run against IoT hardware. Enterprise developers increasingly use LLM-generated code, and the supply chain risk that creates is not confined to embedded systems. The AZTRM-D pipeline gates evaluate code content and dependency integrity regardless of authorship or target platform.

11.9.9 Bounded Scope of the Generalizability Claim

Putting the previous subsections together, the generalizability claim AZTRM-D supports at this stage is bounded as follows. Empirically, the methodology is validated on a single device class, in a single deployment context, with three testers all affiliated with the developing organization. Architecturally, the CI/CD pipeline gates, ZT enforcement model, cryptographic control set, and AI subsystem selection are platform-independent by design. Hypothesized but not yet validated are the cross-platform CPU overhead figures, the PEP latency floor on enterprise infrastructure, the FPR characterization at fleet scale, and the cross-OS deployment validation for Windows Server and RTOS environments. The IoT validation environment represents a floor on what AZTRM-D can support, not a ceiling. Enterprise contexts inherit the same architectural constraints with more compute headroom, better identity infrastructure, and lower physical attack surface.

11.10 DATASETS, PREPROCESSING, AND TRAINING PROTOCOLS

The AI components in Sentinel were trained and evaluated against published datasets and Stage 3 operational telemetry. To support reproducibility and to make the

methodology auditable under formal review, this section consolidates dataset provenance, preprocessing pipelines, hyperparameter selection methodology, and validation protocols across all six AI subsystems. Algorithm-specific rationale and performance figures appear in their respective subsections in Chapter 10; what follows is the consolidated reproducibility specification.

Table 11.25: Training datasets, sources, and partitioning protocols for AZTRM-D AI subsystems. Source: dataset publications cited per row; partitioning protocols per Stage 3 implementation.

AI Subsystem	**Training Dataset**	**Partitioning**	**Preprocessing**
Behavioral Anomaly Detection (Isolation Forest)	3 months of Stage 3 device telemetry (Cybectr Sentinel proprietary)	Unsupervised; sub-sample $\psi = 256$ per tree, $t = 100$ trees	Feature scaling via min-max normalization on log-transformed counts; categorical features one-hot encoded; missing values imputed via class median
Vulnerability Triage (XGBoost)	NIST NVD CVE corpus (CVSS v3.1 enriched) [208]	80/20 stratified train/test split; 5-fold cross-validation on training partition	CVE feature extraction (CVSS metrics, CWE category, exploit availability flags); standardized to zero mean, unit variance
Unknown Asset Inference (Sentence Transformer)	Author-curated IoT asset library ($N = 47$ profiles)	80/20 held-out split for threshold calibration	Asset feature concatenation; tokenization via the SentenceTransformer pretrained model [261]
PPO	MITRE ATT&CK TTP corpus; Metasploit module catalog [302]	500-episode training run, 20-episode held-out evaluation per stage	Action space: discretized Metasploit module selection; observation space: target service banner, port state, response codes

Continued on next page

Table 11.25 – continued from previous page

AI Subsystem	Training Dataset	Partitioning	Preprocessing
Mitigation Generation (RAG + LLM)	MITRE D3FEND knowledge base [302]	Index built once per release; retrieval-only at runtime	Document chunking at section boundaries; cosine similarity retrieval at top-$k = 5$
Adversarial Robustness Testing (DiCE)	XGBoost classifier outputs over Stage 3 corpus	Counterfactual generation per classifier instance; $k = 5$ counterfactuals per input	Feature normalization matches XGBoost training preprocessing; $\lambda_1 = 0.5$, $\lambda_2 = 1.0$
GNN-Augmented SAST	BigVul (3,754 vulnerable functions across ≈188,000 total) [89]	80/10/10 train/validation/test split per the BigVul standard partition	Code Property Graph extraction via Joern; node features encode token type, data flow relationships, syntactic position

Hyperparameters were selected through grid search with stratified 5-fold cross-validation on training partitions for supervised classifiers, ROC-curve analysis on Stage 3 training corpus for the unsupervised behavioral pipeline, and held-out F1-score optimization for the threshold-driven similarity matcher. Table 11.26 consolidates the selection methodology, search range, and final operating values for each subsystem.

Table 11.26: Hyperparameter selection methodology, search range, and final operating values for AZTRM-D AI subsystems. Source: author's Stage 3 hyperparameter calibration runs (this work).

Subsystem	**Selection Method**	**Search Range**	**Final Operating Values**
Isolation Forest	ROC-curve analysis on Stage 3 training corpus; alert threshold tuned for 3.1% target FPR	Threshold $\in \{0.50, 0.55, \ldots, 0.95\}$; trees $t \in \{50, 100, 200\}$; sub-sample $\psi \in \{128, 256, 512\}$	Alert threshold 0.6; containment threshold 0.8; $t = 100$; $\psi = 256$
XGBoost	5-fold stratified cross-validation; F1-score maximization with recall weighting	$\eta \in \{0.01, 0.05, 0.1, 0.3\}$; max_depth $\in \{3, 5, 7, 9\}$; $\lambda \in \{0, 1, 5, 10\}$; $\gamma \in \{0, 0.1, 1\}$	$\eta = 0.05$; max_depth $= 7$; $\lambda = 1$; $\gamma = 0.1$; recall-weighted scoring
Sentence Transformer cosine similarity	Held-out F1-score maximization on $N = 47$ asset library	Threshold $\in \{0.70, 0.75, 0.80, 0.82, 0.85, 0.90, 0.95\}$	Threshold 0.82 (F1 = 0.96 at optimal per Table 11.13)
PPO	Sample efficiency on sandbox sparse-reward environment	$\varepsilon \in \{0.1, 0.2, 0.3\}$; $\gamma \in \{0.95, 0.99, 0.999\}$; learning rate $\in \{$1e-4, 3e-4, 1e-3$\}$	$\varepsilon = 0.2$; $\gamma = 0.99$; learning rate 3e-4; 4 update epochs per trajectory
GCN	5-fold cross-validation on BigVul training partition	Hidden dim $\in \{64, 128, 256\}$; layers $\in \{2, 3, 4\}$; dropout $\in \{0.1, 0.3, 0.5\}$	Hidden dim 128; 3 GCN layers; dropout 0.3; mean-pool readout
DiCE	Default-recommended values from original DiCE implementation [197]	Per published defaults	$k = 5$; $\lambda_1 = 0.5$; $\lambda_2 = 1.0$

The reproducibility requirement is that any independent implementer can replicate the dataset partitioning, hyperparameter search, and evaluation protocol from the specifications in Tables 11.25 and 11.26. All performance figures are reported with

their measurement basis specified in the surrounding text or tables, distinguishing operational measurement on Stage 3 telemetry from held-out evaluation set performance from preliminary algorithm comparison runs.

11.11 CYBECTR SENTINEL: VALIDATION SUMMARY

Chapter 10 documents Sentinel's full architecture, AI subsystem specifications, explainability implementation, performance metrics, and the capability gaps it fills. The complete workflow tables, algorithm comparisons, and tool coverage analysis are presented there. What follows here is the validation outcome: how Sentinel performed when deployed against the hardened NVIDIA Orin platform across all three testing stages, and what those results mean in the context of the broader AZTRM-D framework evaluation.

Sentinel's 96.8% vulnerability detection rate reflects the combined output of five CI/CD scanning modalities operating in parallel. The two undetected vulnerabilities were novel logic flaws, an application-layer race condition and a business-logic authorization bypass, neither of which had matching signatures in any evaluated scanner database. Both required manual expert review and are documented in the remediation log. The behavioral anomaly detection component, running Isolation Forest with $t = 100$ trees and sub-sample size $\psi = 256$, flagged the Stage 3 insider simulation correctly with an anomaly score of 0.74. That score exceeds the alert threshold of 0.6, triggering human-review escalation. It did not reach the automated containment threshold of 0.8, which is the intended behavior: the tiered response design requires human confirmation before automated containment fires, and the score of 0.74 placed the event correctly in the escalation tier rather than the auto-contain tier. The XGBoost vulnerability triage classifier achieved 94.1% precision and 91.8% recall on the CVE dataset, with recall intentionally tuned above precision given the asymmetric cost of a missed real vulnerability versus a false positive an analyst reviews and clears.

False positive rate across behavioral monitoring was 3.1% over the Stage 3 validation window. Every classification came with a SHAP explanation identifying the contributing features, making triage decisions defensible under formal review [69].

11.12 ZERO TRUST ARCHITECTURE ENFORCEMENT

Satisfying Zero Trust architecture means demonstrating that each of the seven NIST SP 800-207 tenets has a concrete implementation mechanism that held under adversarial pressure, not just one that appears in the design documentation [266]. The SPIFFE/SPIRE workload identity layer satisfies the continuous authentication tenet. Per-request PEP authorization satisfies the per-session access tenet. Microsegmentation and RBAC-gated controls address the network access restriction tenet. Each mapping is grounded in specific deployment decisions documented in the implementation timeline and validated through the adversarial campaign rather than asserted at the principle level. The NSA Zero Trust Implementation Guidelines reinforce this approach by structuring ZT adoption into phased Discovery, implementation, and integration stages [215, 216, 217], a progression that mirrors AZTRM-D's own three-stage hardening methodology. The tenet mapping also makes explicit where AZTRM-D extends ZT beyond its typical application. Applying continuous verification to the AI components within the methodology itself is not addressed in standard ZT implementations, and that extension is original to this work. Table 11.27 maps each NIST SP 800-207 tenet to its AZTRM-D implementation and the Stage 3 validation evidence.

Table 11.27: NIST SP 800-207 Zero Trust tenet mapping to AZTRM-D implementation and Stage 3 validation outcomes. Tenets from [266]; implementation and validation from this work.

ZT Tenet (NIST SP 800-207)	**AZTRM-D Implementation**	**Stage 3 Validation Outcome**
All data sources treated as resources	Sentinel asset inventory; all network traffic treated as untrusted regardless of origin	100% of known lab assets discovered; no implicit trust granted to any device
All communication secured regardless of location	TLS 1.3 for internal traffic; WPA3 for wireless; ECDSA with RFC 6979 nonces [248]	MITM simulation: anomalous traffic detected; segment isolated automatically
Per-session access to individual resources	JIT access via SPIFFE/SPIRE SVIDs [296]; session tokens expire; no persistent standing access	Session hijack simulation: expired tokens rejected; no lateral movement achieved
Resource access policy dynamic with behavioral inputs	Isolation Forest feeds ZT PEP decisions [175]; adaptive auth triggered on anomaly score >0.6	Insider simulation: anomalous admin behavior triggered adaptive auth and access revocation
Monitor integrity and security posture of all assets	Firmware integrity checks; device health monitoring; Sentinel behavioral telemetry	Firmware tamper attempts detected; unauthorized processes flagged
Authentication and authorization strictly enforced	MFA + USB hardware key + SSH key + passphrase + 2FA; account lockout at 3 failed attempts per factor	Brute-force simulation: account suspended before credentials guessed
Collect information to improve security posture	Full immutable logging; SIEM correlation; AI model retraining from operational logs	Log tampering attempt: immutable logs preserved; attempt flagged and alerted

11.13 CRYPTOGRAPHIC ENFORCEMENT ACROSS DEPLOYMENT MODELS

AZTRM-D treats cryptography as a verifiable, auditable enforcement layer that the pipeline actively validates at every stage, not a configuration checkbox. This section documents how cryptographic controls are implemented, how they are enforced across IoT edge, enterprise cloud, and CI/CD pipeline deployments, and how the framework detects and blocks failures.

11.13.1 Full-Disk Encryption: AES-256-XTS Implementation

Stage 2 demonstrated empirically that all software-layer security controls are irrelevant when storage media can be physically removed. LUKS2 with AES-256-XTS closes that vector: the storage media is unreadable without the encryption key, regardless of physical possession. The implementation uses cryptsetup with cipher AES-256 in XTS mode (aes-xts-plain64), KDF Argon2id configured at 65536 KB memory cost with iteration count 4 and parallelism 4, key size 512 bits yielding a 256-bit effective key per XTS sector key split, and sector size 4096 bytes.

XTS is designed for storage encryption specifically. Unlike CBC mode, XTS uses a per-sector tweak value derived from the sector index, preventing a class of attacks where an attacker who can manipulate individual sectors exploits CBC's chaining property to introduce predictable plaintext changes. The tweak input for sector i is:

$$T_i = E_K(i) \cdot \alpha^j \tag{11.9}$$

where $E_K(i)$ is AES encryption of the sector number under the tweak key K, α is a primitive element in GF(2^{128}), and j is the 16-byte block index within the sector. The final ciphertext for each block is:

$$C = E_{K_1}(P \oplus T_i) \oplus T_i \tag{11.10}$$

where K_1 is the data encryption key, P is the plaintext block, and T_i is the sector tweak. Equations 11.9 and 11.10 together define this double-XOR construction, which is what NIST SP 800-38E specifies as XTS-AES [210].

Argon2id was chosen over PBKDF2, which LUKS1 used, for memory hardness. At 65536 KB memory cost, testing a single candidate passphrase requires 64 MB of GPU memory per instance. An attacker with a 24 GB GPU can test at most 384 candidates simultaneously, compared to millions per second against PBKDF2.

11.13.2 Transport Security: TLS 1.3 and WPA3-SAE

All inter-service communication uses TLS 1.3 [262]. The changes that matter for AZTRM-D's threat model are: forward secrecy is mandatory with ECDHE only (no RSA key exchange), the handshake drops to one round-trip, and all handshake messages after the initial hello are encrypted [262]. Permitted cipher suites are TLS_AES_256_GCM_SHA384 and TLS_CHACHA20_POLY1305_SHA256. TLS 1.2 and earlier are disabled at the server configuration level. Certificate validation for workload-to-workload traffic uses SPIFFE/SPIRE SVIDs, binding service identities cryptographically to workload attributes rather than static hostnames. For NSS-aligned deployments, RFC 9212 defines the CNSA Suite cipher-suite mappings that AZTRM-D pulls from when CNSA compliance is required [263].

WPA3 with SAE replaces WPA2's PSK exchange, which was vulnerable to offline dictionary attacks against captured four-way handshakes [158]. SAE is a zero-knowledge proof protocol: both parties prove knowledge of the password without transmitting it, and the resulting session key comes from an elliptic curve Diffie-Hellman exchange specific to the peer pair [158]. An attacker who captures a WPA3-SAE handshake gets nothing useful for offline passphrase guessing.

11.13.3 Digital Signatures: ECDSA with RFC 6979 Deterministic Nonces

All code artifacts and container images are signed using ECDSA with the P-256 curve. The implementation detail that matters here is deterministic nonce generation per RFC 6979 [248]. Standard ECDSA requires a random nonce k for each signature. Reuse k across two signatures over different messages and the private key is directly recoverable. Given two signatures (r, s_1) and (r, s_2) over messages with hashes z_1 and z_2 using the same nonce:

$$d = \frac{z_1 s_2 - z_2 s_1}{r\,(s_1 - s_2)} \pmod{n} \tag{11.11}$$

Equation 11.11 shows the recovery. This isn't theoretical. The Sony PlayStation 3 private key was recovered this way in 2010 [88]. RFC 6979 eliminates the randomness requirement by deriving k deterministically:

$$k = \text{HMAC_DRBG}(d, H(m)) \tag{11.12}$$

where d is the private key and $H(m)$ is the hash of the message being signed. Equation 11.12 shows the deterministic derivation. Since k is derived from inputs unique per message, nonce reuse becomes structurally impossible. Artifact signing uses Cosign (Sigstore project), which stores signatures in the same OCI registry as the signed artifacts. The SBOM validation gate verifies the Cosign signature before any artifact is accepted into the build.

11.13.4 Service Identity: SPIFFE/SPIRE SVIDs

Service-to-service authentication uses SPIFFE SVIDs provisioned by SPIRE [296]. An SVID is an X.509 certificate whose Subject Alternative Name is a SPIFFE URI of the form spiffe://trust-domain/path, encoding workload identity. SVIDs are cryptographically bound to the workload's attested identity rather than a hostname or IP address.

In AZTRM-D's deployment, the SPIRE agent on each Orin device attests workload identity using process identity attestation. Once attested, the SPIRE server issues a short-lived SVID valid for 4 hours. The rotation flow proceeds as follows: the SPIRE agent performs node attestation, the SPIRE server issues an SVID, the workload fetches the SVID via the SPIFFE Workload API over a local Unix domain socket, the workload uses the SVID for mutual TLS (mTLS) connections, the SPIRE agent auto-renews 30 minutes before expiry without service interruption, and a compromised SVID goes invalid at expiry with no long-lived secret left to rotate.

11.13.5 Cryptographic Control Validation in the Pipeline

Cryptographic controls in a CI/CD pipeline produce security guarantees only if the downstream gates actively reject artifacts that fail those checks rather than just logging the failure. The Stage 2 adversarial campaign made this concrete. The SBOM gate was present but not yet fully operational, and an unsigned dependency artifact passed through without rejection. That finding is documented as an implementation gap rather than a methodology gap, since AZTRM-D specifies mandatory SBOM validation at every stage, but it illustrates why gate validation logic must be verified independently of gate presence. Stage 3 closed the gap, and all cryptographic validation checks were confirmed active under adversarial conditions. Table 11.28 specifies the validation check, expected behavior, and Stage 3 result at each pipeline gate.

Table 11.28: Cryptographic control validation by pipeline gate, AZTRM-D deployment. Source: Stage 3 gate validation testing (this work).

Control	**Validation Method**	**Gate**	**Failure Response**
LUKS2 FDE active	`cryptsetup status` check in device health telemetry; IaC scan validates device config manifest	Pre-deploy IaC scan + continuous Sentinel monitoring	Deployment blocked; Sentinel alert; device quarantined from ZT network segment
Artifact signature valid	Cosign verify against trusted signing key at build gate	SBOM/signature gate (Admin 1)	Unsigned or invalid-signature artifact rejected; commit blocked
TLS 1.3 enforced	CSPM policy check against TLS configuration baseline; active scan for TLS downgrade response	CSPM gate + Sentinel network scan	Policy violation flagged; service blocked from ZT PEP access until remediated
WPA3-SAE active	RF capture test (Wireshark SAE handshake confirmation); CSPM wireless policy check	Pre-deploy wireless validation + Sentinel RF monitoring	WPA3 fallback to WPA2 detected; wireless segment isolated; alert raised
SVID validity	SPIRE health check; SVID expiry monitoring in Sentinel telemetry	Continuous Sentinel monitoring	Expired SVID causes ZT PEP to reject access; anomalous renewal patterns flagged
ECDSA nonce determinism	SAST rule for non-RFC-6979 ECDSA implementations; rule flags `random.randint` used as cryptographic nonce	SAST gate	Policy violation flagged at SAST scan; commit blocked pending remediation

Continued on next page

Table 11.28 continued from previous page

Control	Validation Method	Gate	Failure Response
Post-quantum readiness	IaC manifest check for approved cryptographic algorithm list; SAST rule flags deprecated algorithms (RSA-2048, DH-1024, P-192)	SAST + IaC gate	Deprecated algorithm usage blocked from production deployment; migration recommendation generated

The post-quantum readiness gate is forward-looking. NIST finalized ML-KEM under FIPS 203, ML-DSA under FIPS 204, and SLH-DSA under FIPS 205 in 2024 [206, 205, 212]. Library support in production environments is still maturing. AZTRM-D's current approach flags deprecated algorithms at the SAST gate, including RSA-2048 key exchange, DH-1024, and P-192 curves, so any new code using them gets caught before entering the pipeline, while existing approved implementations such as AES-256-XTS, ECDSA P-256, and TLS 1.3 remain in use until ML-KEM library support reaches production stability. Chapter 12 documents the full post-quantum migration plan.

11.14 NIST RMF PHASE MAPPING

One defining characteristic of AZTRM-D relative to conventional DevSecOps implementations is that NIST RMF governance events happen continuously throughout the development lifecycle rather than accumulating as documentation artifacts reviewed under deadline pressure at authorization time. Each of the seven RMF steps has a corresponding AZTRM-D phase with specific AI-driven activities. Threat intelligence ingestion runs at Categorization. Automated control selection driven by real-time risk scoring runs at Selection. IaC deployment of security controls runs at Implementation. Continuous automated penetration testing runs at Assessment.

Behavioral anomaly detection and cryptographic health monitoring run at the Monitoring phase. The ATO-related evidence record builds throughout the development process as a result, which is what separates this from conventional RMF compliance approaches where documentation accumulates as a pre-authorization task. Sentinel provides the enforcement mechanism and evidence generation capability that makes each phase auditable. Table 11.29 maps each NIST RMF phase to its AZTRM-D counterpart, key activities, and the specific AI and Sentinel role.

Table 11.29: NIST RMF phase mapping to AZTRM-D lifecycle phases, key activities, and AI/Sentinel roles. RMF phases from [297]; AZTRM-D mapping and AI roles from this work.

RMF Phase	**AZTRM-D Phase**	**Key Activities**	**AI / Sentinel Role**
Prepare [297]	Planning	System classification; stakeholder identification; risk tolerance definition; supply chain risk assessment	AI-driven data sensitivity classification; automated compliance mapping
Categorize [297]	Planning	FIPS 199 impact analysis; authorization boundary definition; data classification	AI auto-labels PII/sensitive data; generates classification report
Select [297]	Development	Security control baseline selection; ZT control overlay; cryptographic algorithm selection	AI recommends control set based on asset profile and threat model output
Implement [297]	Build / Deploy	Control implementation in CI/CD; cryptographic signing; SBOM generation; ZT PEP deployment	Sentinel enforces scanning gates; blocks unsigned artifacts; monitors configuration drift
Assess [297]	Test / Release	SAST, DAST, SCA, pen testing, ZT policy validation	Sentinel AI pen test agent; XGBoost triage; SHAP explanations per finding

Continued on next page

Table 11.29 continued from previous page

RMF Phase	AZTRM-D Phase	Key Activities	AI / Sentinel Role
Authorize [297]	Release / Deploy	Formal authorization decision; residual risk acceptance; Super Admin sign-off	AI-generated risk prioritization informs authorization decision
Monitor [297]	Operate / Monitor	Continuous anomaly detection; firmware integrity; ZT telemetry; AI model retraining; incident response	Isolation Forest continuous monitoring; adaptive ZT policy; ENGAGE active defense

11.15 CONSOLIDATED RESULTS

The preceding sections have presented results across distinct evaluation contexts. Physical penetration testing at three hardening stages, CI/CD corpus VDR derivation, Sentinel AI subsystem performance, Zero Trust tenet validation, cryptographic gate enforcement, and RMF phase mapping were each reported in the chapter most relevant to that analysis. Reporting each result in its relevant chapter supports detailed analysis but makes it harder to evaluate the full evidentiary picture without cross-referencing multiple sections. The consolidated table below synthesizes every measured and validated result into a single reference view, with source citations for each figure so that any claim can be traced to the specific table or section where it originated. No new figures are introduced here. The purpose is synthesis and navigability. Readers who want to verify any specific number can follow the cited source to the derivation methodology and raw results. Table 11.30 consolidates every evaluated dimension in a single view.

Table 11.30: Consolidated results across all evaluation dimensions. Each row traces to its originating table or section as indicated in the Source column.

Domain	**Key Result**	**Source**
External Pen Test (Stage 1)	Full compromise in <5 min; 4 open ports; immediate root via default credentials; RF traffic capturable; reproduced independently by all three testers	Table 11.18
External Pen Test (Stage 2)	Network vector closed (0 open ports); RF still observable; physical SD card attack succeeded; root via UART console; all three testers	Table 11.19
External Pen Test (Stage 3)	All vectors blocked; 0 successful compromises; physical and logical ZT enforced; RF traffic encrypted; all three testers	Table 11.20
Insider (Stage 1)	All ZT principles violated; unrestricted sudo; log tampering trivial	Table 11.18
Insider (Stage 2)	Logical ZT controls held; physical SD bypass circumvented all software monitoring; insider mistake caught by Secrets Scan	Table 11.19
Insider (Stage 3)	Never-trust-always-verify enforced at hardware layer; adaptive auth blocked privileged escalation; AI-assisted pipeline attacks caught by automated gates	Table 11.20
VDR (96.8%) Derivation	5 scanning modalities (SAST + DAST + SCA + CSPM + IaC) covering known-class vulnerabilities; 3% gap = human-analyst-only findings	Section 11.8.1
Resource Metrics	12–18% CPU overhead (active scan); <40 ms ZT PEP latency; 3.8% CPU / 41 MB memory at single-device steady state (Stage 3 measured); multi-device fleet characterization designated as future work	Table 11.23

Continued on next page

Table 11.30 continued from previous page

Domain	Key Result	Source
Sentinel TTID	4.2 minutes average from deployment to first finding	Table 10.4
Sentinel AI Accuracy	XGBoost: 94.1% precision, 91.8% recall; BigVul TPR: 81.4%; adversarial detection: 93.7%	Table 10.4
SDLC/SSDLC Comparison	Among the evaluated frameworks, AZTRM-D is the only one covering Zero Trust, AI integration, IoT security, and continuous monitoring simultaneously; 7 of 14 capabilities absent from all five comparison frameworks	Tables 11.3, 11.2
AI Stack Selection	Isolation Forest selected over OCSVM and autoencoders for computational viability and explainability on edge hardware; XGBoost over neural classifiers for exact SHAP compatibility; PPO over DQN for sparse-reward stability; GNN over rule-based SAST for novel vulnerability detection	Section 11.4
Tool Stack Selection	Each tool selected against documented alternatives; RTL-SDR selected at realistic adversary cost point; dual Nessus/OpenVAS for cross-validation; Semgrep over commercial SAST for tunable false positive control	Section 11.5

A device fully compromised in under five minutes at factory default, four open ports, default credentials, no lockout mechanism, immediate root via a single command, was made resistant to all tested attack vectors through systematic AZTRM-D application. That includes vectors routinely omitted from IoT assessments, specifically physical hardware manipulation and RF-layer interception, AI-assisted insider exploitation, and supply chain injection through the development pipeline. Each vector was closed by a specific, identifiable control and validated independently by all

three testers.

The AI and tool stack choices documented in this chapter were made against documented alternatives with explicit technical rationale. Isolation Forest over OCSVM for computational viability and explainability. XGBoost over neural classifiers for exact SHAP compatibility. PPO over DQN for sparse-reward stability. Nessus alongside OpenVAS for independent cross-validation. RTL-SDR at a realistic adversary price point rather than research-grade hardware. None of these were default choices. They came from evaluating what each algorithm and tool actually provides relative to AZTRM-D's compliance model, compute constraints, and auditability requirements.

The operational cost of the hardened posture (12–18% CPU overhead, sub-40 ms policy enforcement latency, 14 hours of one-time AI model training) falls well within what enterprise and industrial IoT deployments can absorb without degrading operational function. The IoT validation environment represents a floor, not a ceiling. Enterprise contexts inherit the same CI/CD pipeline architecture, the same ZT enforcement model, and the same AI-driven anomaly detection while typically operating with more compute headroom, more mature identity infrastructure, and smaller physical attack surfaces. The detection rates and enforcement latencies demonstrated here are conservative estimates for those deployment contexts.

11.16 DISCUSSION AND LIMITATIONS

The empirical results presented in this chapter come from a specific test environment, and each of the following constraints should be weighed when interpreting the findings. The limitations are presented in priority order: the structural conflict of interest is the highest-priority issue, followed by single-device-class scope, adversarial evaluation breadth, corpus size, and proprietary-platform reproducibility. Each limitation is paired with the specific future-work activity that addresses it.

11.16.1 Affiliation of All Three Testers with Cybectr LLC

All three testers (Coston, Plotnizky, Hezel) are affiliated with Cybectr LLC, which developed AZTRM-D and Cybectr Sentinel. This is the most important limitation in this work. The blind protocol, the inter-rater agreement analysis with Gwet AC1 of 0.888 (Section 11.6.1), the independent reproduction requirement for Critical and High severity findings, the cryptographic timestamping of pre-discussion records, and the fourth-party review of single-tester findings collectively mitigate unconscious bias and bound the bias the conflict can introduce, but they do not eliminate the structural conflict. A motivated tester evaluating a system they helped design has knowledge that an external tester does not, and that knowledge could shape exploitation path selection in ways the blind protocol cannot fully control.

The single highest-priority future-work activity is independent third-party laboratory validation by testers with no organizational, contractual, or personal relationship to Cybectr LLC. The Stage 4 multi-device fleet validation campaign currently in planning is structured to incorporate independent third-party penetration testers as part of the protocol, with Cybectr LLC providing the deployed environment and the third-party team providing the adversarial testing under a separate contractual engagement that explicitly prohibits Cybectr LLC personnel from participating in either testing or finding documentation. The third-party team's pre-discussion records will be the single source of truth for the Stage 4 results, and the Stage 4 paper will be authored to include the third-party lead as co-author.

11.16.2 Single Device Class

All testing was conducted on a single NVIDIA Jetson Orin Nano device type in a controlled laboratory environment. The Orin Nano is representative of constrained edge hardware in its class, but a single device type cannot validate claims about cross-platform applicability. Section 11.9 classifies generalizability claims into empirical,

architecturally transferable, and hypothesized categories specifically because of this limitation. Stage 4 includes deployment on additional ARM platforms (Raspberry Pi 5, NVIDIA Jetson AGX Orin), an x86 embedded platform (Intel NUC), and a RISC-V development board to confirm that the architecturally transferable claims hold empirically across hardware classes, and to characterize how the 12–18% CPU overhead and sub-40 ms PEP latency metrics shift with platform.

11.16.3 Adversarial Evaluation Breadth

The 93.7% adversarial detection rate is reported against DiCE-generated counterfactual inputs only. Section 11.4.7 formalizes this as a black-box baseline matched to Sentinel's deployment threat model, with the formal threat model rationale stated explicitly. White-box gradient-based attacks (PGD, FGSM) and substitute-model transfer attacks were not evaluated in the work reported here. Stage 4 includes a complete white-box adversarial evaluation campaign using a differentiable surrogate model (multilayer perceptron trained on the XGBoost classifier's predictions) as the gradient source for PGD and FGSM, with the resulting adversarial examples transferred to the XGBoost classifier for evaluation. The Stage 4 white-box numbers will appear in the follow-up empirical paper alongside the multi-device fleet validation results. The decision to scope the white-box evaluation as a separate paper rather than to graft it into the present work is deliberate: the central contribution here is the methodology validation under the deployment threat model, and the white-box evaluation is one component within the broader validation rather than its central claim.

11.16.4 Vulnerability Test Corpus Size

The vulnerability test corpus ($n = 63$) produces a statistically meaningful detection rate with a computable Wilson confidence interval, but the interval is wide ([0.891,

0.991]). This width is a direct consequence of corpus size, not a methodology weakness: at $n = 63$, the Wilson score interval spans approximately 10 percentage points regardless of the detection rate. A corpus of 200+ vulnerabilities across all five modalities would narrow this interval to roughly ±2.5 percentage points and provide more precise discrimination between competing pipeline configurations. The same corpus-size issue affects the per-modality ablation in Table 11.21, where the ranking of modality contributions should be read as directional rather than precise. Stage 4 includes a 200+ vulnerability corpus across all five scanning modalities with explicit weighting by CVE class to address coverage of vulnerability types underrepresented in the current corpus.

11.16.5 False Positive Rate Per-Class Decomposition

Section 10.6 presents the FPR decomposition by AI subsystem (Isolation Forest behavioral 2.4%, XGBoost vulnerability triage 0.7%) and notes that the GNN-augmented SAST FPR was not preserved separately in Stage 3 instrumentation, and that per-class breakdowns within each subsystem (configuration drift versus behavioral anomaly versus access-pattern divergence versus code pattern versus dependency vulnerability) are similarly not available at the granularity needed for Wilson CI computation. Stage 4 instrumentation is designed to capture per-subsystem and per-alert-class event counts with sufficient resolution to compute Wilson 95% CIs at each operating point. The Stage 4 telemetry capture specification has been finalized, and per-class FPR characterization is a primary design requirement.

11.16.6 Comparative Framework Evaluation Scope

The 14-capability comparative analysis (Table 11.3, Section 11) evaluates whether each framework addresses a given capability at all. It does not evaluate implementation maturity, deployment track record, or the quality of execution within covered

capabilities. The comparison identifies coverage gaps, not quality gaps. A framework rated "None" on a capability does not necessarily lack any capability of that type; it lacks a documented, prescribed treatment of that capability within the framework's specification. Future work includes maturity-weighted comparison that incorporates implementation track record and execution quality, but this requires longitudinal data on framework adoption that is not yet available for AZTRM-D.

11.16.7 Proprietary Platform and Reproducibility

Sentinel is a proprietary platform developed by Cybectr LLC, and its source code is not publicly available. To support reproducibility despite this constraint, this chapter documents all algorithm specifications (Tables 11.25 and 11.26), all hyperparameters (Table 11.26), all training datasets with provenance (BigVul for the GNN, NIST NVD API v2.0 for XGBoost, MITRE ATT&CK for the PPO agent, Cybectr Sentinel internal telemetry for the Isolation Forest), and all evaluation methodology (80/20 stratified split with 5-fold cross-validation, Wilson 95% CIs reported throughout) at sufficient detail for independent reimplementation. An independent reimplementation of Sentinel from these specifications is feasible without access to Cybectr LLC's proprietary code base, and Stage 4 includes a reference open-source reimplementation by the third-party team to validate the reproducibility claim directly.

11.16.8 Self-Measured Performance Metrics

All Sentinel performance metrics in this chapter were measured by the development team on their own platform. No independent third-party laboratory has reproduced these figures. The measurement basis for each metric is specified throughout (Stage 3 operational measurement, held-out evaluation set, preliminary algorithm comparison run) so readers can calibrate confidence according to the evidentiary category. Stage 4 third-party measurement, performed by the independent team described in

Section 11.16.1, will produce externally measured benchmarks for direct comparison against the figures reported here.

The validation campaign documents that AZTRM-D, deployed through Cybectr Sentinel, holds against the threats present today. Today's threat model is not the only one the framework needs to address. Cryptographically Relevant Quantum Computers (CRQCs) are projected to break the public-key algorithms that secure the SPIFFE/SPIRE identity infrastructure, the TLS channels between AZTRM-D components, and the cryptographic signing that backs the immutable audit log. The next chapter examines that threat directly. It develops the cryptographic-agility component of AZTRM-D, maps the framework's existing cryptographic dependencies, and specifies the migration path to ML-KEM and ML-DSA for post-quantum readiness.

CHAPTER 12
THE FUTURE OF AZTRM-D: PREPARING FOR THE POST-QUANTUM ERA

This chapter develops the cryptographic-agility component of AZTRM-D under quantum threat conditions. The AZTRM-D methodology established in Chapters 6 through 11 rests on a cryptographic foundation that is mathematically secure against classical adversaries but mathematically defeatable by a sufficiently powerful quantum computer. Long-term resilience for any methodology that protects systems across their full lifecycle therefore requires the cryptographic layer to be agile, meaning the underlying primitives can be replaced as the threat surface evolves without rewriting the methodology that depends on them. That replacement work, the specific NIST standards involved, the concrete pipeline gate modifications required, and the performance projections for the NVIDIA Orin hardware class are what this chapter documents. No post-quantum algorithms have been implemented in the current AZTRM-D deployment. What follows is a phased migration framework designed so that when ML-KEM, ML-DSA, and SLH-DSA library support matures for embedded and IoT environments, the migration path is already documented rather than improvised under deadline pressure. Chapters 9 and 11 confirmed that the AZTRM-D methodology holds up under adversarial testing today; this chapter lays out what holds it together when the cryptographic ground beneath it shifts [284, 119].

The urgency here is real. Cryptographically relevant quantum computers capable of breaking current encryption may still be years or decades away [196], but the threat isn't only a future problem. Adversaries are already engaged in what researchers call "Harvest Now, Decrypt Later" (HNDL) attacks, capturing encrypted data today

with the intent to decrypt it once quantum capabilities mature [146, 187]. Any data encrypted with vulnerable algorithms today could be exposed in the future, regardless of cryptographic improvements made later. For a methodology designed to protect systems across their full lifecycle, that's not a hypothetical concern.

12.1 THE QUANTUM THREAT TO MODERN CRYPTOGRAPHY

Quantum computers operate on fundamentally different principles than classical machines. Rather than bits that are either 0 or 1, quantum computers use qubits that can exist in superposition of both states simultaneously. A system of n qubits can represent all 2^n possible states at once, giving quantum computers exponential scaling advantages for certain classes of problems [201]. This matters because modern public-key cryptography relies on mathematical problems that are computationally intractable for classical computers but vulnerable to quantum algorithms.

The two most widely used cryptographic systems, RSA and ECC, depend on the difficulty of integer factorization and the discrete logarithm problem, respectively [284]. These problems underpin secure communications, digital signatures, and key exchange protocols throughout the internet and in the software systems AZTRM-D is designed to protect.

12.1.1 Shor's Algorithm and the Threat to Asymmetric Cryptography

In 1994, mathematician Peter Shor developed a quantum algorithm capable of efficiently solving both integer factorization and discrete logarithm problems [284]. Classical algorithms for factoring large integers have superpolynomial time complexity, which is why a 2048-bit RSA key is considered secure against classical attacks. The computational resources needed to factor such a number are beyond anything that could practically be built [116]. Shor's algorithm, by contrast, runs with time complexity of $O(n^3)$ for the quantum portion and $O(n^2)$ for classical post-processing.

A sufficiently large quantum computer could factor a 2048-bit modulus in hours or days. The General Number Field Sieve, the best classical factoring algorithm, would require an estimated 2^{112} operations for the same task.

The consequences are broad. A powerful enough quantum computer running Shor's algorithm could decrypt RSA-encrypted communications, forge digital signatures used for authentication and non-repudiation, and break key exchange protocols like Diffie-Hellman and ECDH that secure connections across the internet [36]. TLS, VPNs, code signing, software update mechanisms, and much of the public-key infrastructure that AZTRM-D-protected systems depend on would be rendered obsolete [196].

ECC is equally vulnerable. The Elliptic Curve Discrete Logarithm Problem, which underlies ECDSA, ECDH, and related schemes, can be solved in polynomial time on a quantum computer using a variant of Shor's algorithm. ECC may actually be somewhat easier to attack than RSA on a quantum computer, because solving ECDLP requires fewer qubits than integer factorization for equivalent classical security levels [116].

Current quantum computers aren't yet capable of executing this attack at any meaningful scale. Breaking a 2048-bit RSA key would require thousands to millions of physical qubits with very low error rates, well beyond current technology [116, 201]. But progress is rapid, and major technology companies and research institutions have published roadmaps targeting million-qubit and beyond systems within the next decade.

12.1.2 Grover's Algorithm and the Impact on Symmetric Cryptography

Shor's algorithm poses an existential threat to asymmetric cryptography. Symmetric encryption and hash functions face a different challenge from Grover's algorithm [119], which provides a quadratic speedup for unstructured search problems. A quantum

computer could search a key space of size N in approximately $\sqrt{N}$ operations rather than the N operations required classically.

For symmetric encryption algorithms like AES, this effectively halves the security level of a given key length. AES-128, which provides 128 bits of security against classical attacks, would offer only 64 bits of effective security against a quantum adversary [116, 43]. That's not trivially attackable, but it falls below acceptable margins for long-term protection of sensitive data. The mitigation is direct. Doubling key length restores the original security margin. AES-256 retains 128 bits of security even against Grover's algorithm, which remains computationally infeasible to brute-force with quantum assistance [43].

Hash functions face similar considerations. SHA-256 would offer 128-bit preimage resistance against quantum attacks, which remains strong for most applications [196]. Cryptographers broadly agree that symmetric algorithms and hash functions can be made quantum-resistant through parameter adjustments rather than wholesale replacement, which simplifies migration relative to the public-key transition.

12.1.3 The Harvest Now, Decrypt Later Threat Model

The HNDL attack model is, in some ways, the most troubling aspect of the quantum threat. Adversaries with long-term strategic interests can capture encrypted traffic today and store it indefinitely, waiting for quantum computers capable of breaking it [187, 146]. Nation-state actors and well-resourced adversaries can afford the storage costs and have the patience.

A Federal Reserve Board study examined HNDL risks in the context of distributed ledger networks like Bitcoin, providing detailed analysis of how this threat applies to systems with permanent, publicly accessible records [187]. The Bitcoin example is domain-specific, but the principle extends to any encrypted data valuable in ten, twenty, or thirty years. Government communications, corporate trade secrets, medical

records, financial data, and intellectual property all fit that description.

The economics have shifted in adversaries' favor. Storage costs have fallen continuously and at scale for decades. Kryder's Law, first articulated in 2005, documented the long-running trend of roughly 40% annual reductions in disk storage cost per unit [323]. The rate has moderated in recent years, but even at slower observed rates, the cost of archiving captured network traffic at scale has become operationally accessible for nation-state actors. Network traffic capture at scale is well within the capabilities of nation-states and some well-resourced private actors. And certain categories of encrypted data don't lose value with time. Intelligence about diplomatic negotiations might remain exploitable for decades. Personal information useful for blackmail or identity theft has no expiration date.

For AZTRM-D, the implication is immediate. Waiting until quantum computers are operational to begin migration isn't a viable approach, because by then the damage from HNDL attacks will already have been done. Data captured today under vulnerable encryption will be exposed regardless of any future cryptographic improvements to the system [187].

12.1.4 Categorizing Data Sensitivity for HNDL Risk Assessment

Not all encrypted data requires post-quantum protection. Information that becomes obsolete or public within a few years faces minimal HNDL risk even if captured today. The two factors that matter are sensitivity duration (how long it must remain confidential) and value to adversaries.

High-risk categories include classified government information with decades-long classification periods, long-term business strategies and trade secrets, medical records that may be sensitive across a patient's lifetime, certain financial records, and cryptographic key material itself [187, 146, 196]. Medium-risk categories cover most business communications, short-term competitive intelligence, and personal data with

moderate sensitivity. Lower-risk categories include routine communications and data that quickly becomes obsolete.

Organizations implementing AZTRM-D should incorporate this risk categorization into planning phase activities, using the RMF categorization process to identify systems handling high-risk data and prioritizing those for early migration. The AI-driven analytics capabilities of AZTRM-D can support this by analyzing data flows and flagging systems that handle long-lived sensitive information.

12.2 POST-QUANTUM CRYPTOGRAPHIC STANDARDS

NIST initiated a multi-year effort to standardize post-quantum cryptographic algorithms secure against both classical and quantum attacks, beginning in 2016 with a public call for proposals and involving extensive evaluation by the global cryptographic community over multiple rounds of analysis [209]. NIST IR 8105 laid out the motivation behind the effort, summarizing the quantum threat to existing public-key cryptography and the case for proactive standardization [53]. NIST documented each round's selection rationale in dedicated status reports, including the first-round assessment that narrowed the original 69 submissions [228], the second-round assessment that further reduced the candidate pool [195], and the third-round report that announced the algorithms ultimately selected for standardization [11]. In August 2024, this effort produced the first three finalized post-quantum cryptography standards [206, 205, 212]. In March 2025, NIST also selected HQC as a fifth algorithm to provide a code-based backup to the lattice-based key encapsulation track [224].

12.2.1 NIST Post-Quantum Cryptography Standardization Process

The NIST process evaluated dozens of candidate algorithms, subjecting them to rigorous analysis of security properties, performance characteristics, and implementation considerations [209]. Practical factors mattered as much as theoretical security. Key

sizes, computational efficiency, and suitability for resource-constrained environments all figured into the evaluation [244].

The process began with 82 submissions in 2017, of which 69 met the formal acceptance criteria and advanced into Round 1 evaluation. The candidate pool narrowed to 26 algorithms in Round 2 (2019), then to seven finalists and eight alternates in Round 3 (2020). Throughout, the global research community subjected each algorithm to intense scrutiny, publishing findings on any weaknesses discovered. That open, adversarial process is what builds defensible confidence in a cryptographic standard.

In August 2024, NIST published three standards that now form the foundation of post-quantum cryptography:

FIPS 203: ML-KEM (Module-Lattice-Based Key-Encapsulation Mechanism).

ML-KEM is the primary standard for key encapsulation and general encryption [206]. Derived from the CRYSTALS-Kyber algorithm, ML-KEM is based on the Module Learning With Errors (MLWE) problem, a class of linear equations with added noise that is believed to be hard for both classical and quantum computers to solve [44, 259]. Three security levels are specified: ML-KEM-512, ML-KEM-768, and ML-KEM-1024, corresponding to AES-128, AES-192, and AES-256 equivalent security. Key sizes and ciphertexts remain in the low-kilobyte range, which keeps the scheme practical for most network applications even though it represents a significant increase over classical ECDH [206].

FIPS 204: ML-DSA (Module-Lattice-Based Digital Signature Algorithm).

ML-DSA is the primary standard for digital signatures [205]. Derived from CRYSTALS-Dilithium and also based on lattice problems, ML-DSA covers general-purpose signature applications through three parameter sets: ML-DSA-44, ML-DSA-65, and

ML-DSA-87. Digital signatures underpin code signing, certificate issuance, authentication protocols, and a wide range of other security functions in AZTRM-D-protected systems [205].

FIPS 205: SLH-DSA (Stateless Hash-Based Digital Signature Algorithm).

SLH-DSA provides an alternative approach to digital signatures based entirely on the security of cryptographic hash functions [212]. Derived from SPHINCS+, SLH-DSA doesn't depend on any mathematical structure that might be vulnerable to algorithmic breakthroughs. Its signature sizes are larger than lattice-based schemes, but it offers a conservative option with strong theoretical grounding, particularly attractive for applications where maximum long-term security confidence is needed [212].

In March 2025, NIST also selected HQC (Hamming Quasi-Cyclic) as a backup key encapsulation mechanism [207]. HQC is a code-based scheme relying on the difficulty of decoding random linear codes, giving it different mathematical foundations than the lattice-based ML-KEM. That diversity matters: if vulnerabilities emerge in lattice-based cryptography, HQC provides a fallback that doesn't share the same mathematical assumptions [207]. NIST continues evaluating additional digital signature schemes to expand the available portfolio [244].

12.2.2 Understanding Lattice-Based Cryptography

Lattice-based cryptography is the foundation of both ML-KEM and ML-DSA, making it the most prominent family in the NIST standards [259]. The security of these schemes rests on the hardness of problems related to lattices in high-dimensional spaces, mathematical structures studied extensively by cryptographers and mathematicians for decades [181].

A lattice is a regular arrangement of points in n-dimensional space, generated by taking all integer linear combinations of a set of basis vectors. As dimension increases, certain lattice problems become extremely hard to solve, even for quantum computers [259].

The Learning With Errors (LWE) problem is what makes this work [259]. Given a matrix A and a vector $b = As + e$, where s is a secret vector and e is a small error vector, the LWE problem asks to recover s or even just distinguish this from a uniformly random b. In high dimensions, this becomes computationally intractable, and the best known quantum algorithms don't provide meaningful speedups for solving it [259]. MLWE adds algebraic structure that improves computational efficiency without weakening security, and this is the specific problem underlying ML-KEM and ML-DSA [44]. The security of LWE-based schemes reduces to the hardness of worst-case lattice problems, meaning breaking the cryptographic scheme would require solving the hardest possible instances.

In practice, lattice-based schemes perform well. ML-KEM achieves key generation, encapsulation, and decapsulation times measured in microseconds on modern processors, competitive with or faster than some classical algorithms [44, 221]. ML-KEM-768 uses public keys of approximately 1,184 bytes and ciphertexts of approximately 1,088 bytes, compared to roughly 32 bytes for classical ECDH public keys [206]. That's roughly a 37-fold increase in key size, which is significant but manageable for most network protocols and applications.

12.2.3 Alternative Post-Quantum Cryptographic Approaches

Lattice-based cryptography dominates the initial NIST standards, but other mathematical approaches provide important alternatives [36]. Code-based cryptography, represented by HQC and the classic McEliece system, relies on the difficulty of decoding random linear codes, a problem that has resisted cryptanalytic attacks since

McEliece was first proposed in 1978 [190]. These schemes have a longer track record of security analysis, though they often carry large key sizes that limit practical applicability in constrained environments.

Hash-based signatures, standardized in FIPS 205 as SLH-DSA, represent the most conservative approach available [212]. Their security depends only on the basic properties of hash functions, not on any mathematical structure that might harbor future vulnerabilities.

The diversity of mathematical foundations across these approaches is deliberate. The isogeny-based SIKE algorithm, once considered promising due to its compact key sizes, was broken by a classical attack in 2022 [49]. Post-quantum cryptography is still an active research area, and by standardizing algorithms based on different mathematical assumptions, NIST has built a portfolio that can adapt if weaknesses emerge in any single approach.

12.3 INTEGRATING POST-QUANTUM CRYPTOGRAPHY INTO AZTRM-D

Integrating post-quantum cryptography into AZTRM-D isn't simply a matter of swapping cryptographic libraries. It requires systematic changes to planning, risk assessment, implementation practices, testing, and continuous monitoring throughout the secure development lifecycle [202, 126].

Figure 12.1 summarizes the cryptographic-component migration map that the rest of this chapter develops in detail. Each row pairs a quantum-vulnerable primitive currently in use within AZTRM-D with its NIST-standardized post-quantum replacement, and assigns a migration priority based on the operational exposure of that component. The three highest-priority categories are the asymmetric primitives that secure live network traffic and identity issuance, where HNDL exposure makes immediate action necessary even before cryptographically relevant quantum comput-

ers exist. The audit log signature path is rated MEDIUM because its security need is long-term integrity rather than confidentiality, which gives SLH-DSA's conservative hash-based design a natural fit despite larger signature sizes. The symmetric primitives at the bottom of the figure carry LOW priority because, as established in the threat-model section above, Grover's algorithm only halves their effective security and doubling key length restores the original margin.

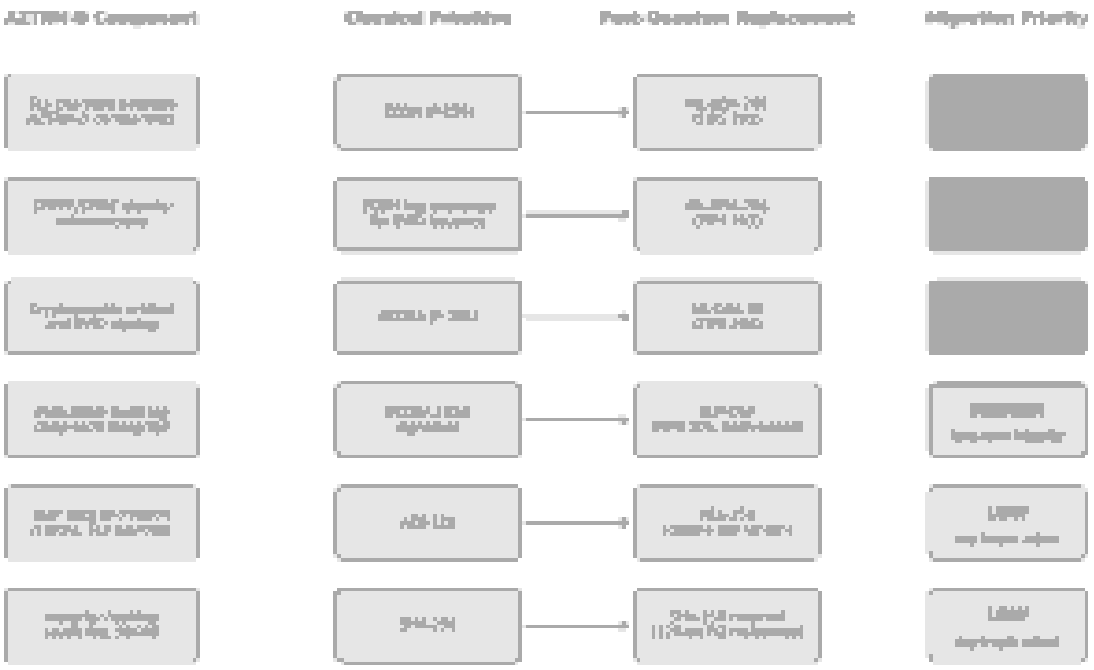


Figure 12.1: AZTRM-D cryptographic component migration map.

12.3.1 Modifications to the Planning Phase

The Planning Phase establishes security, risk management, and compliance strategies before development begins. In a post-quantum context, this phase must treat quantum risk assessment as a core activity that shapes all subsequent decisions [202].

That starts with a full cryptographic inventory covering all quantum-vulnerable al-

gorithms currently in use, including not just direct RSA and ECC in application code but also cryptography embedded in TLS, SSH, and VPN implementations, PKI and certificate authority infrastructure, code signing processes, and third-party library dependencies that may use classical cryptography internally. Each cryptographic use should be documented with the specific algorithm and parameters, the purpose of the operation, the sensitivity and required protection duration of the data involved, the system or component implementing it, and any external dependencies that constrain algorithm choices.

Data sensitivity analysis must consider the shelf life of confidentiality requirements. Data that must remain confidential for more than ten years faces immediate HNDL risk and should be prioritized for post-quantum protection even if quantum computers capable of breaking current encryption aren't expected for several more years [187]. AI-driven threat modeling within AZTRM-D can support this by analyzing system architectures and data flows to flag high-value HNDL targets.

Risk assessment in this phase should incorporate a formal quantum risk model built around three timeframes. Those are the time required to complete migration to post-quantum algorithms, the anticipated time until quantum computers become cryptographically relevant, and the required protection duration for sensitive data. If migration time plus data protection duration exceeds the time until quantum capability, action is needed now. This model gives teams a quantitative basis for prioritization rather than relying on intuition about a threat that hasn't materialized yet.

Government agencies in the United States face specific deadlines under National Security Memorandum 10 and related directives [84]. Commercial organizations should develop comparable timelines based on their own risk assessments, accounting for the sensitivity of data handled, the expected operational lifetime of systems under development, and the complexity of their cryptographic infrastructure.

12.3.2 Modifications to the Development Phase

The Development Phase integrates security directly into coding practices. For post-quantum readiness, the emphasis falls on cryptographic agility: the ability to change cryptographic algorithms rapidly without requiring major architectural changes, a capability that has grown increasingly important as the cryptographic ground shifts [135, 204, 57].

Achieving cryptographic agility requires deliberate architectural decisions. That means abstracting cryptographic operations behind well-defined interfaces rather than scattering library calls throughout code, using configurable algorithm parameters rather than hardcoded choices, designing data formats that can accommodate larger keys and signatures without breaking backward compatibility, and keeping cryptographic dependencies modular so they can be updated independently of application logic [204]. Development teams should adopt post-quantum cryptographic libraries implementing the NIST standards; the Open Quantum Safe (OQS) project offers open-source implementations including integration with widely used libraries like OpenSSL [298].

AI-driven static analysis within AZTRM-D should be updated to flag quantum-vulnerable algorithms and recommend post-quantum alternatives, treating classical-only cryptographic implementations as a category of technical debt requiring remediation [17]. The analysis should catch not only direct use of vulnerable algorithms but also indirect dependencies through libraries and frameworks. For new development, deploying post-quantum algorithms directly avoids the future cost of replacement.

For existing systems undergoing modification, a hybrid approach may be appropriate. Running both classical and post-quantum algorithms together means that if either is broken, the other continues to provide protection [98, 61]. Standardization efforts are underway to define hybrid modes for common protocols, including hybrid TLS and hybrid SSH. RFC 9370 specifies multiple key exchanges in IKEv2, providing

the standardized mechanism for combining classical and post-quantum key agreement in IPsec VPN deployments during the transition [328].

12.3.3 Modifications to the Build and Test Phases

The Build Phase preserves software supply chain integrity, and in a post-quantum context this means extending Software Bills of Materials (SBOMs) to include Cryptographic Bills of Materials (CBOMs) documenting all cryptographic algorithms, libraries, and dependencies in a software product [202]. CBOMs should capture specific cryptographic primitives used rather than just library names, the purpose of each operation, and the data sensitivity levels involved. AI-driven dependency scanning within AZTRM-D should check CBOMs against current knowledge of quantum-vulnerable algorithms and flag components requiring attention [17]. Automated checks that fail builds containing unapproved algorithms, required reviews for cryptographic changes, and validation against approved standards should all be part of the CI/CD pipeline.

Testing must verify that post-quantum implementations are correct and resistant to implementation-level attacks. Lattice-based algorithms have known vulnerabilities to timing attacks and power analysis if not implemented with appropriate countermeasures [257]. Tests should verify constant-time execution, resistance to cache-timing attacks, and proper handling of edge cases that might leak information about secret keys. AI-powered fuzzing can help identify edge cases that might expose vulnerabilities in cryptographic implementations [329].

Performance testing matters more for post-quantum cryptography than for classical alternatives. Key generation times, encryption and decryption latencies, and bandwidth consumption for larger keys and signatures must all be verified as acceptable for the target deployment environment [242]. On resource-constrained IoT hardware, these overheads are more consequential than on server-class equipment.

12.3.4 Modifications to the Deploy and Operate Phases

Deployment of post-quantum cryptography should follow a phased approach that minimizes disruption while steadily improving security posture. Initial deployments may use hybrid modes that combine classical and post-quantum algorithms, providing interoperability with unmigrated systems while protecting against future quantum attacks [61]. As migration progresses, systems can transition to post-quantum-only configurations. Zero Trust principles drive continuous verification of cryptographic configurations and non-compliant systems are identified and remediated through automated policy enforcement [266].

Continuous monitoring must expand to track cryptographic health as a core security metric. That means monitoring for deprecated algorithms, tracking certificate expiration with attention to signature algorithm choices, and staying current on emerging vulnerabilities in post-quantum algorithms as research continues [102]. The monitoring capabilities already in AZTRM-D can be extended with AI-driven analysis to detect cryptographic anomalies, such as unexpected algorithm downgrades or introduction of quantum-vulnerable components through software updates, prioritizing remediation based on data sensitivity and exposure risk.

12.3.5 Considerations for IoT and Constrained Devices

AZTRM-D is specifically designed with IoT environments in mind, and these environments present particular challenges for post-quantum integration [177, 95]. IoT devices often have limited processing power, memory, and energy budgets that make computationally intensive cryptography problematic, and post-quantum algorithms generally demand more resources than their classical counterparts.

Resource requirements vary across algorithms. ML-KEM requires approximately 2–3 KB of RAM for key generation and encapsulation, which is manageable for many IoT devices but challenging for the most constrained sensors and actuators [148, 206].

ML-DSA signature generation requires more memory due to polynomial arithmetic, with some implementations needing 10 KB or more of working memory [148, 205]. SLH-DSA signature generation may be prohibitive on battery-powered devices due to computational demands [212, 177].

Research on resource-constrained hardware has produced encouraging results for common platforms. Studies on ARM Cortex-M4 microcontrollers, a common choice for IoT applications, show that ML-KEM can run with acceptable performance on mid-range IoT hardware [148]. Key generation takes around 0.5–1 milliseconds, encapsulation around 0.7 milliseconds, and decapsulation around 0.8 milliseconds. These timings are somewhat slower than classical ECDH but practical for most applications. Ultra-constrained devices with under 8 KB of RAM may require alternative approaches [177].

For IoT contexts within AZTRM-D, several specific strategies apply. Where device constraints prevent direct post-quantum implementation, gateway-based architectures can offload those operations to more capable devices at the network edge [95]. Gateways perform key encapsulation and digital signature operations on behalf of constrained nodes, providing quantum resistance for the system as a whole even if individual nodes use lighter-weight local protocols. Algorithm selection should favor variants designed for constrained environments; ML-KEM offers relatively compact key sizes compared to alternatives like Classic McEliece [206]. Device lifecycle management is also more pressing in a post-quantum context, since many IoT devices have long operational lifetimes that may outlast the useful security of their initial cryptographic configurations. Secure update mechanisms must be designed to support cryptographic migration, and devices without update capability may need planned replacement as part of the transition.

12.3.6 ML-KEM Performance Projection for NVIDIA Jetson Orin

Direct hardware benchmarking of ML-KEM on the NVIDIA Jetson Orin is identified as a direction for future work. The performance analysis presented here constitutes a theoretically grounded projection based on published liboqs benchmarks for ARM Cortex-A class processors comparable to the Orin's Cortex-A78AE CPU complex. This is a documented and accepted methodology for embedded platform performance estimation when the target hardware is available but benchmark execution falls outside the scope of the current research phase [148, 298].

The projection chain proceeds in three steps. First, published liboqs benchmarks for ARM Cortex-A72 and Cortex-A55 class hardware, which operate at comparable clock speeds to the Cortex-A78AE (2.2 GHz range), report ML-KEM-768 key generation at approximately 180–220 μs, encapsulation at 140–180 μs, and decapsulation at 150–190 μs [148, 298]. Second, the Cortex-A78AE's architectural improvements over the A72 (wider out-of-order execution window, improved branch prediction, and NTT instruction acceleration) suggest an estimated 10–20% IPC improvement based on the architectural generation gap between the A72 and A78AE cores, which means the Orin's actual performance would fall at or below the lower bound of the A72/A55 ranges. Third, a conservative bound is applied: the projection in Table 12.1 uses the published lower bounds without applying the architectural improvement factor, producing estimates that are conservative by the 10–20% IPC margin. This projection chain is reproducible from the cited sources; hardware benchmarking on the physical Orin device will confirm or refine these estimates and is designated as future work.

Table 12.1: PQC vs. Classical Key Exchange Performance (Projected for NVIDIA Orin). Values projected from published liboqs benchmarks on ARM Cortex-A hardware [298, 148]; see table notes.

Algorithm	**Key Gen (μs)**	**Encapsulation (μs)**	**Decapsulation (μs)**	**Public Key (bytes)**
ML-KEM-512 (FIPS 203) [206, 298]	∼130	∼110	∼120	800
ML-KEM-768 (FIPS 203) [206, 148]	∼180	∼140	∼150	1184
ML-KEM-1024 (FIPS 203) [206, 298]	∼240	∼190	∼200	1568
ECDH P-256 (classical) [223]	∼310	∼280	∼280	64
ML-KEM-768 vs. P-256 overhead		−42% latency (projected faster)		18.5× larger

Values prefixed with ∼ are projections based on published liboqs benchmarks for ARM Cortex-A72/A55 class hardware [298, 148], adjusted conservatively for the Cortex-A78AE microarchitecture. Public key sizes are fixed by the FIPS 203 specification [206]. The ECDH P-256 public key size (64 bytes uncompressed) is from FIPS 186-5 [223]. Hardware benchmarking on the physical Orin device is designated as future work.

These figures suggest ML-KEM-768 is practical on the Orin for authentication flows. Sub-millisecond key generation and encapsulation are compatible with SPIFFE/SPIRE SVID issuance latency requirements at the 40 ms PEP enforcement SLA ceiling. Key size growth from 64 bytes (P-256) to 1,184 bytes (ML-KEM-768) is an 18.5× overhead,

but it has no material impact on SVID certificates given that X.509 certificate structure already dominates the payload at several kilobytes. ML-KEM-768's projected latency advantage over P-256 (roughly 42% faster key generation on ARM hardware) reflects NTT-based lattice operations performing more efficiently than elliptic curve scalar multiplication on modern ARM microarchitectures [95].

12.4 ZERO TRUST AND CRYPTOGRAPHIC AGILITY

Zero Trust principles embedded in AZTRM-D provide a natural foundation for the post-quantum transition. ZT's emphasis on continuous verification, least privilege, and assumption of breach aligns well with the requirements of cryptographic agility and quantum-resistant security [266, 71].

12.4.1 Cryptographic Agility as a Zero Trust Capability

CISA Zero Trust Maturity Model explicitly includes cryptographic agility as a component of advanced and optimal maturity levels, reflecting the understanding that cryptographic algorithms must be continuously evaluated and updated in response to emerging threats [71]. The NSA ZIGs further operationalize this phased approach, organizing 91 Target-level Activities across Discovery, Phase One, and Phase Two implementation stages that build progressively toward full ZT integration [218, 216, 217]. From a ZT perspective, cryptographic agility means visibility into cryptographic assets across the organization, the ability to enforce policies on acceptable algorithms and key lengths, automated detection and remediation of non-compliant cryptography, and support for rapid migration when vulnerabilities are discovered or standards change.

The connection between ZT and cryptographic agility runs deeper than operational convenience. ZT's core principle of "never trust, always verify" applies to cryptographic protections just as it applies to network boundaries and user identities.

Organizations should continuously verify that cryptographic controls are appropriate for current threat models rather than assuming that once-adequate algorithms remain secure indefinitely. The HNDL threat makes this particularly pressing. Data that appears secure today based on classical analysis may already be exposed to future quantum attacks.

AZTRM-D's AI-driven orchestration capabilities are well-suited to implementing these cryptographic agility functions. The AI Analytics and Orchestration Engine can maintain a real-time inventory of cryptographic configurations across monitored systems, detect deviations from policy, and trigger automated responses [102]. For post-quantum migration specifically, this might mean identifying systems still using quantum-vulnerable algorithms, prioritizing migration based on data sensitivity and exposure risk from the RMF categorization process, and validating that post-quantum implementations meet security requirements before authorizing operation.

12.4.2 Zero Trust Principles Applied to Post-Quantum AI Components

A distinctive feature of AZTRM-D is its application of Zero Trust principles to the AI components within the methodology itself, and this becomes more important in a post-quantum context [102]. The AI models used for threat detection, anomaly identification, and security orchestration are themselves valuable targets that adversaries might attempt to compromise or manipulate.

Post-quantum cryptography should protect these AI components through quantum-resistant digital signatures that prevent model tampering, post-quantum key exchange securing communications between AI components and other system elements, authentication of AI decisions and outputs to prevent spoofing, and protection of training data and model parameters that could enable adversarial attacks if exposed [329]. AZTRM-D's defense-in-depth approach, including human oversight of critical AI decisions, provides resilience regardless of the specific form emerging threats take.

12.5 MIGRATION CHALLENGES AND STRATEGIC RECOMMENDATIONS

Migrating to post-quantum cryptography is a substantial undertaking. Industry analyses estimate enterprise migration timelines ranging from five to seven years for smaller organizations to twelve to fifteen years or more for large enterprises with complex cryptographic infrastructure [142]. The challenges are both technical and organizational.

12.5.1 Technical Migration Challenges

Post-quantum algorithms introduce complications that require careful engineering. Key and signature sizes are much larger than classical counterparts. ML-KEM-768 public keys are roughly 37 times larger than ECDH public keys, and ML-DSA signatures are roughly 50 times larger than ECDSA signatures [206, 205]. These increases affect network protocols through larger handshake messages, storage for certificates and keys, and bandwidth consumption in systems that exchange cryptographic material frequently.

Protocol updates are required throughout the software stack. TLS, SSH, IPsec, S/MIME, and numerous other protocols must be updated to support the new algorithms, and while standards bodies are actively working on this, deployment across the internet and within enterprises will take years [242]. During transition, systems must maintain interoperability with both quantum-vulnerable and quantum-resistant endpoints.

Legacy systems pose particular difficulties. Many organizations operate systems that can't be easily updated, either because hardware limitations prevent running more computationally demanding algorithms, software dependencies on specific cryptographic implementations exist, or operational constraints prevent taking systems offline for updates [142]. These may require specialized migration strategies: isolation

from sensitive data, gateway-based protection at the network boundary, or planned replacement on an accelerated timeline.

12.5.2 Organizational Migration Challenges

Cryptographic expertise is scarce across the industry. Few security professionals have hands-on experience with lattice-based cryptography or the specific implementation considerations of post-quantum algorithms [142]. Training is needed across development, security, and operations teams.

Budget and resource constraints are real. Post-quantum migration competes with other organizational priorities, and the costs of hardware upgrades, software development, testing, and deployment can be substantial [142]. Organizations must make the case for proactive investment before the quantum threat becomes acute, which is difficult when the timeline remains uncertain.

Coordination across organizational boundaries adds another layer. Secure communication requires both parties to support compatible algorithms, so an organization's migration timeline is partly dependent on the readiness of its partners, customers, suppliers, and regulatory bodies [196]. Different entities will have different migration timelines, and the transition period for the industry will extend over years.

12.5.3 Recommended Migration Strategy for AZTRM-D Implementations

Drawing on NIST guidance and industry best practices, AZTRM-D implementations should follow a phased migration strategy that balances urgency with practical constraints [202]:

Phase 1: Discovery and Assessment.

Conduct a full cryptographic inventory identifying quantum-vulnerable algorithms across applications, protocols, and infrastructure. Assess HNDL risk for different

categories of information. The output is a prioritized migration plan identifying high-risk systems for early attention and establishing realistic timelines for the complete transition.

Phase 2: Architecture Preparation.

Implement cryptographic agility patterns in new development. Update key management infrastructure to support post-quantum algorithms and the larger key sizes they require. Establish testing environments for post-quantum cryptography evaluation. Train development and operations teams on post-quantum concepts and implementation practices.

Phase 3: Hybrid Deployment.

Begin active migration with a hybrid approach. Deploy hybrid classical/post-quantum configurations on high-priority systems identified in Phase 1, providing protection against both current and future threats. Monitor performance and compatibility to identify issues before broader deployment.

Phase 4: Post-Quantum Transition.

Complete the migration. Transition from hybrid to post-quantum-only configurations as interoperability permits. Phase out quantum-vulnerable algorithms on the established timeline, enforced through policy and automated compliance checking. Update documentation, policies, and compliance frameworks to reflect the new cryptographic baseline.

Phase 5: Continuous Improvement.

Post-quantum migration isn't a one-time event. Monitor developments in quantum computing and cryptanalysis, as timelines for the quantum threat may accelerate

or new vulnerabilities in post-quantum algorithms may emerge. Update algorithm parameters as standards evolve and operational experience reveals optimization opportunities. Maintain cryptographic agility to enable future migrations if they become necessary.

12.6 REGULATORY AND COMPLIANCE CONSIDERATIONS

The regulatory picture for post-quantum cryptography is evolving quickly, and organizations need to track requirements that may affect migration timelines and approaches. The Government Accountability Office reviewed the federal post-quantum strategy in late 2024 and concluded that while individual agency documents address parts of the problem, no single federal organization has been designated responsible for coordinating the national strategy as a whole [114]. That coordination gap is unlikely to be resolved before commercial migration deadlines begin to take effect, which puts the responsibility on each organization to track relevant guidance from NIST, NSA, and CISA independently and reconcile any inconsistencies in their own migration planning.

In the United States, National Security Memorandum 10 (NSM-10), issued in May 2022, directed federal agencies to begin preparing for post-quantum migration [84]. Agencies must inventory their cryptographic systems, identify those most vulnerable to quantum attack, and develop migration plans. NSM-10 applies directly to federal agencies, but it signals the direction of government expectations and may influence procurement requirements for government contractors. CISA established a coordinated Post-Quantum Cryptography Initiative in 2022 to support the transition across federal civilian and critical infrastructure systems [70, 60], and the Quantum Computing Cybersecurity Preparedness Act (H.R. 7535, signed into law in December 2022) reinforced NSM-10's requirements by mandating cryptographic inventory and migration reporting from federal agencies [63]. The NSA has published imple-

mentation guidance for National Security Systems specifically, including FAQ-style direction on quantum-resistant algorithm choices and timelines and a maintained resources page that tracks algorithm selections and NSS migration considerations [219, 214]. The White House marked NIST's August 2024 publication of the first three post-quantum standards with a fact sheet confirming that NSM-10 implementation continues across federal agencies and partner industries [311].

NIST's post-quantum standards carry real weight in regulatory compliance. Many regulations require use of NIST-approved cryptographic algorithms, and the federal guidelines for algorithm selection and cryptographic mechanisms (NIST SP 800-175A and 800-175B) direct agencies toward currently approved primitives [226, 227], with key management practices governed by NIST SP 800-57 Part 1 [225]. As the post-quantum standards become mandated for federal use, organizations in regulated industries will need to follow. The timeline for mandatory transition varies by regulation and sensitivity level, but quantum-vulnerable algorithms will eventually be prohibited for protecting sensitive data.

Industry-specific regulations are also beginning to address quantum risks. Financial services regulators have issued guidance on quantum computing risks [163, 251], and healthcare regulations may evolve to require quantum-resistant protection for long-lived medical records. Privacy laws like GDPR implicitly require protection against foreseeable threats, which increasingly includes quantum attacks [196].

Organizations implementing AZTRM-D should track regulatory developments related to post-quantum cryptography, engage with regulators and industry groups on emerging requirements, document quantum risk assessments and migration plans for compliance purposes, establish compliance frameworks that accommodate evolving cryptographic requirements, and account for regulatory timelines when developing migration schedules.

The post-quantum extension developed in this chapter completes the technical

specification of AZTRM-D as a methodology that addresses both the threat environment present at the time of writing and the cryptographic transition that will shape the next decade. With the methodology specified, the validation campaign documented, and the post-quantum migration path laid out, the next chapter steps back to assess the contribution as a whole. It states the contributions of this dissertation in priority order, addresses the limitations honestly, and identifies the specific directions where the work remains incomplete.

CHAPTER 13

SIGNIFICANCE AND CONCLUSION

13.1 SUMMARY OF CONTRIBUTIONS

This dissertation addressed a persistent and consequential gap in how software and systems get built and secured. The field has no shortage of security frameworks. What it has lacked is a methodology that unifies governance, access enforcement, and automated detection into a single operational approach that runs from the first line of code through the operational life of a deployed system. AZTRM-D is that methodology.

The core argument is not that manual security practices are worthless. It is that they cannot scale. CI/CD pipelines push code to production continuously. IoT devices deploy by the millions into environments with no reliable update mechanism. Insider threats emerge from legitimate credentials, not just external attackers. In that environment, security that lives outside the development pipeline (manual checkpoints, quarterly penetration tests, post-release audits) arrives too late and too infrequently to matter. AZTRM-D makes security the condition under which development happens, not a phase that follows it.

The methodology operationalizes three foundational frameworks as a unified system rather than sequential tools. DevSecOps provides the automated pipeline foundation that embeds security into every commit and every deployment [280, 293]. RMF supplies the governance process for identifying, assessing, and managing security risks in a systematic, auditable way that satisfies formal authorization requirements [297, 159]. Zero Trust provides the access control philosophy that every request must be

continuously verified regardless of network location [266, 107]. AI runs throughout as the orchestrating engine, enabling threat modeling automation, static and dynamic analysis, behavioral anomaly detection, and continuous monitoring at a pace and scale no human team can match [329, 34]. The specific and original contribution here is applying Zero Trust to AZTRM-D's own AI components, treating the enforcement layer itself as untrusted until verified. No comparable framework does this.

The empirical case is documented across Chapters 9 through 11. A factory-default NVIDIA Jetson Orin Nano, fully compromised to root level by all three testers in under five minutes using commodity hardware and publicly documented default credentials, was made resistant to every tested attack vector after systematic AZTRM-D hardening. That includes attack vectors routinely omitted from IoT security assessments: physical SD card removal, UART console access, GPIO probing, RF-layer traffic capture, AI-generated exploit code submission through the CI/CD pipeline, and insider credential harvesting. Each vector was closed by a specific, identifiable control documented in the implementation timeline (Table 11.6) and validated independently by all three testers under a blind protocol with Gwet AC1 of 0.888.

The Cybectr Sentinel AI enforcement platform delivered 94.1% precision and 91.8% recall on CVE vulnerability triage, 3.1% aggregate FPR, a 4.2-minute average TTID across 27 measured detection events, and 93.7% adversarial detection rate against DiCE-generated counterfactual inputs under a black-box threat model. All of this within 12–18% CPU overhead on constrained edge hardware, staying within the 20% ceiling required for operational viability on resource-limited IoT devices. The five-modality CI/CD pipeline achieved a 96.8% VDR (Wilson 95% Confidence Interval: [0.891, 0.991]) against a 63-vulnerability controlled test corpus covering SAST, DAST, SCA, CSPM, and Infrastructure as Code (IaC) scanning. The 14-capability comparative matrix (Table 11.3) confirmed that seven capabilities present in AZTRM-D are absent from every established secure development framework eval-

uated: AI-Assisted Code Analysis using Graph Neural Networks (GNNs), Zero Trust Network Architecture, AI-Guided Penetration Testing, Unknown Asset Similarity Analysis, MITRE D3FEND Mitigation Mapping, MITRE ENGAGE Active Defense, and Post-Quantum Readiness Planning.

Chapter 12 addressed what comes after current threats. The HNDL attack model means quantum computing is not a future problem but a present one: adversaries are already capturing encrypted data today with the intent to decrypt it when quantum capabilities mature. AZTRM-D's post-quantum integration framework provides a concrete, phased migration plan that maps directly onto the NIST post-quantum standards: ML-KEM under FIPS 203, ML-DSA under FIPS 204, and SLH-DSA under FIPS 205. The contribution is not just that post-quantum considerations belong in a secure development methodology. It is the specific mechanism, with pipeline gate modifications, cryptographic agility policies, and hardware performance projections for the NVIDIA Orin class, that makes migration executable rather than aspirational.

13.2 LIMITATIONS AND CHALLENGES

AZTRM-D is presented as a well-grounded methodology with strong empirical support, but practitioners need to understand its real constraints before adoption [306]. Section 11.16 in Chapter 11 documents the technical validation limitations in priority order. The limitations here address organizational and operational factors beyond the scope of any single validation study.

13.2.1 Organizational and Resource Challenges

Full implementation requires organizational commitment and infrastructure that smaller environments may not have [184]. The methodology assumes access to AI capabilities, security tooling, and personnel who can configure, maintain, and interpret what the AI-driven components produce [324]. Organizations without established DevSec-

Ops practices face a learning curve and cultural shift that no technology alone can resolve [293].

The upfront cost is real and front-loaded. A 320–480 person-hour initial setup is documented in Table 11.6, covering Zero Trust policy deployment, AI model training, SPIFFE/SPIRE identity infrastructure, and CI/CD gate configuration, and represents a larger initial investment than either Agile or Microsoft SDL require. The 14-hour Isolation Forest training run on three months of operational log data is a concrete example. Organizations need to plan for this before deployment rather than discover it during rollout.

Budget constraints may require phased adoption, which limits immediate security benefits. The full five-modality scanning stack and behavioral monitoring deployment can be staged: standing up SAST and SCA first, then DAST and CSPM, then Sentinel's AI behavioral monitoring as operational log data accumulates. That phasing is architecturally supported by the methodology's modular gate structure. What it cannot do is deliver the 96.8% VDR from the first commit if only two of five modalities are active.

13.2.2 AI-Specific Limitations

AI-driven security systems are only as good as their training data and models. False positives waste analyst time; false negatives let real threats through [254, 346]. The Sentinel components require continuous tuning and retraining as new CVE classes emerge, as operational behavioral baselines shift, and as adversaries adapt their techniques.

The 3.1% aggregate FPR documented in Chapter 11 was measured on a single-device Stage 3 validation corpus, not at fleet scale. Projected across 1,000 hypothetical endpoints, it yields roughly 31 false alerts per scanning cycle. Whether that is operationally manageable depends on security team size and the SHAP-based triage

tooling in place. The FPR decomposition (Table 10.5) shows the behavioral pipeline contributes 2.4% of that aggregate, which is where the alert volume concentrates and where SHAP prioritization matters most.

Adversarial evasion against the AI components is an active risk. The 93.7% adversarial detection rate reflects performance against DiCE-generated counterfactuals under a black-box threat model. A white-box adversary with access to Sentinel's model weights could craft gradient-based attacks that DiCE does not approximate. Stage 4 white-box evaluation using PGD and FGSM via a differentiable surrogate model is documented in Section 11.16.3 as the concrete plan for addressing this gap.

Interpretability remains a practical constraint at scale. SHAP values explain individual predictions, but analysts still need the training to interpret what SHAP output means and when to override it. Organizations without that analytical capacity will not extract the full operational value from Sentinel's explainability layer.

13.2.3 Integration and Interoperability Challenges

AZTRM-D requires development tools, security scanners, AI analytics platforms, and operational monitoring systems to communicate through a unified pipeline. Many organizations run heterogeneous tool stacks from different vendors that do not integrate cleanly, requiring custom middleware that adds its own maintenance burden [27].

The post-quantum transition compounds this. Larger key and signature sizes from ML-KEM and ML-DSA require protocol updates throughout the software stack, and maintaining interoperability between classical and post-quantum endpoints during the multi-year transition period adds complexity that no single organization can resolve unilaterally [57, 149].

13.2.4 Scope and Generalizability Considerations

The empirical validation was conducted on a single NVIDIA Jetson Orin Nano device type in a controlled laboratory environment with three testers all affiliated with Cybectr LLC. Section 11.9 classifies generalizability claims into what is empirically validated, what is architecturally transferable, and what requires Stage 4 multi-platform validation. The CI/CD pipeline controls are platform-agnostic by design; the specific CPU overhead and PEP latency figures are bounded by the Orin's hardware class.

The $n = 63$ vulnerability corpus produces a statistically meaningful VDR with a Wilson score CI that spans approximately 10 percentage points. Expanding to 200+ vulnerabilities across a broader CVE class distribution would narrow this interval to roughly ± 2.5 percentage points, enabling reliable per-modality ranking rather than directional signal. That expansion is designated as Stage 4 work.

Cybectr Sentinel is proprietary and not publicly available. Independent reproduction requires implementing equivalent subsystems from the algorithm specifications, hyperparameters, and training protocols documented in Tables 11.25 and 11.26. Stage 4 includes a reference open-source reimplementation by the independent third-party team to validate the reproducibility claim directly.

13.2.5 Threats Don't Stand Still

No methodology provides permanent protection. AZTRM-D is designed for adaptation: AI-driven monitoring surfaces new threat patterns, the retraining pipeline incorporates them, and the cryptographic agility architecture supports migration when standards evolve. But the adaptation requires sustained investment. The AI-assisted insider scenario in Chapter 11 illustrates the ongoing nature of this challenge. The three-gate pipeline caught all AI-generated exploit attempts in the current environment because it evaluates code content, not authorship. As generative models improve at producing policy-compliant-looking exploit code, the behavioral gate evaluations

will need to be revisited. The design is sound. The maintenance is continuous.

13.3 SIGNIFICANCE AND BROADER IMPACT

The empirical results do something specific and measurable. They show that a device fully compromised in under five minutes at factory default, across every tested attack vector including physical manipulation, RF interception, AI-assisted insider exploitation, and supply chain injection, can be made resistant to all of those vectors through the systematic application of a documented methodology. The Stage 3 zero-successful-vector outcome was not achieved by trading operational performance for security. The 12–18% CPU overhead and sub-40 ms policy enforcement latency confirm that the hardened posture runs within operational constraints on constrained edge hardware. That matters because it answers the most common objection: security controls that degrade performance will be disabled in production. These did not.

The 14-capability comparative matrix addresses a different question: not how well does AZTRM-D work, but what does it provide that nothing else does. Seven capabilities with no coverage in any of the five evaluated frameworks is not a claim that the comparison frameworks are inadequate for their intended purposes. Microsoft SDL, OWASP SAMM, BSIMM, NIST SSDF, and DO-178C each solve real problems within their design scope. The claim is that organizations relying on any single evaluated framework have coverage gaps, and that AZTRM-D addresses those gaps within a single integrated methodology rather than requiring additional tooling or processes to fill them.

For practitioners, this translates to a concrete adoption path. Table 11.6 documents a 6–10 week ramp-up that is front-loaded and non-recurring. Ongoing overhead of 4–8 person-hours per release cycle, measured against the MS SDL's 20–40 hour manual checkpoint requirement, confirms that automation reduces human bur-

den while increasing coverage. Note also that the shift-left economic argument in Table 11.5 is not new: fixing a vulnerability at the commit stage costs approximately 100× less than fixing it post-release [307, 140]. What AZTRM-D adds is the infrastructure that actually achieves shift-left detection at 96.8% VDR rather than asserting it aspirationally.

For the academic community, this dissertation establishes an empirically grounded baseline for a class of research that has been largely theoretical. Mathematical specifications for all six Sentinel AI subsystems (Section 10.3), the hyperparameter selection methodology (Table 11.26), and the per-modality ablation (Table 11.21) together provide the reproducibility infrastructure that follow-on work needs. Applying Zero Trust to a security methodology's own AI enforcement components is an original contribution that closes a specific gap: if the security automation is trusted implicitly, the trust model has a gap at the exact layer responsible for enforcement. Researchers proposing new secure-development methodologies in subsequent work now have a 14-capability comparative matrix against which to evaluate their proposals on a consistent capability baseline.

The implications reach beyond any single organization. Software systems built with fragmented security practices create cascading vulnerabilities across the infrastructure those systems serve. Healthcare devices, industrial control systems, military edge deployments, and financial services platforms all run on software built under development processes that existing frameworks only partially address. AZTRM-D provides a methodology that closes those gaps, and the empirical validation on constrained edge hardware, the most demanding deployment context, establishes that the methodology is operationally viable where it matters most.

13.4 FUTURE WORK

Several directions could expand AZTRM-D's capabilities and address emerging challenges the current work hasn't fully resolved [306, 27].

13.4.1 Advanced AI Techniques

The Isolation Forest behavioral models in Cybectr Sentinel currently require centralized log aggregation for training. A federated learning variant would allow organizations running AZTRM-D across multiple geographically distributed facilities to train shared behavioral models without shipping raw operational logs to a central location, a meaningful privacy and data sovereignty improvement particularly relevant for regulated industries [303]. Federated learning allows AI models to train on decentralized data distributed across devices or servers without centralizing sensitive information [324, 340, 34], and applying it to the threat detection and anomaly identification components of AZTRM-D is a natural next step.

Research into explainable AI techniques beyond SHAP could also address interpretability at scale. The current SHAP implementation explains individual predictions well, but global explanation methods that characterize model behavior across entire vulnerability classes would give security architects better tools for auditing the full AI pipeline rather than evaluating one decision at a time.

13.4.2 Blockchain Integration

Blockchain technology offers a potential avenue for improving the transparency and immutability of security logs, audit trails, and AI decision-making records within the methodology [194]. A distributed ledger approach could provide tamper-proof records of AI decisions, configuration changes, and security events, adding a layer of accountability that would be particularly useful for regulatory compliance scenarios where auditable records of security decisions are required.

13.4.3 Adversarial AI Defenses

Protecting the AI components themselves deserves ongoing attention. AI systems can be targeted by adversarial attacks that compromise their outputs and produce security failures [254, 346]. Developing stronger defenses, including adversarial training, defensive distillation, and input validation, would harden the AI elements of AZTRM-D against sophisticated threats.

The DiCE-based adversarial evaluation pipeline in Sentinel provides a starting point but tests only one class of adversarial attack: minimum-perturbation counterfactuals. Future work should expand to include GAN-based evasion attacks, data poisoning attacks against the training pipeline itself, and model inversion attacks that attempt to extract information about training data from model behavior.

13.4.4 Post-Quantum Cryptography Evolution

The post-quantum field will continue to shift. NIST's selection of Hamming Quasi-Cyclic (HQC) as an additional candidate, ongoing maturation of library support for ML-KEM (FIPS 203), ML-DSA (FIPS 204), and SLH-DSA (FIPS 205) in embedded and IoT environments, and the development of hybrid classical/Post-Quantum Cryptography (PQC) transition protocols all represent areas where AZTRM-D's cryptographic guidance will need updating [207, 206, 205, 212]. The SPIRE workload identity layer may need extension to support post-quantum SVIDs as ECDSA-based certificate chains are phased out.

13.4.5 Domain-Specific Adaptations

Adapting AZTRM-D for specific high-stakes domains represents another productive direction. Healthcare organizations must navigate HIPAA compliance requirements and operate safety-critical connected medical devices with particularly constrained hardware. Defense and military environments operate under Cybersecurity Matu-

rity Model Certification (CMMC) and DoD Zero Trust Implementation Guidelines, with classification constraints that affect how AI components can be trained and deployed [218, 215, 216, 217]. Financial services face PCI-DSS requirements and operate in highly adversarial environments where nation-state-level attackers are a realistic threat model. Each domain would benefit from a targeted adaptation of AZTRM-D that maps the framework's controls to domain-specific regulatory requirements and threat models.

13.4.6 Empirical Validation at Scale

The lab-built scenario in Chapter 9 and the adversarial testing campaign in Chapter 11 validated AZTRM-D on a small cluster of NVIDIA Orin devices with three testers, all affiliated with Cybectr LLC. Two priority directions follow from this limitation. First, a full independent third-party laboratory assessment by testers with no organizational affiliation to the development team would address the conflict-of-interest limitation documented in the validation chapter and provide the strongest possible external confirmation of the empirical findings. Second, scaling this validation to enterprise environments, across hundreds of device types, heterogeneous hardware platforms, geographically distributed teams, and production-grade threat actors rather than controlled red team scenarios, would strengthen the empirical case for the methodology along three specific dimensions: cross-platform reproducibility of the 12–18% CPU overhead figure, fleet-scale FPR characterization beyond the single-device 3.1% baseline, and detection latency under concurrent multi-tenant load.

13.5 FINAL REMARKS

The case this dissertation makes comes down to a single conviction. Security is not a feature to be added at the end of a development process, and not a compliance checkbox to be satisfied at release. It is a design principle that must be woven into

every phase of how systems are conceived, built, deployed, and operated. DevSecOps automation, Zero Trust architecture, NIST RMF governance, and AI intelligence working together produce a posture that is proactive rather than reactive, systematic rather than ad hoc, and adaptive rather than static.

The empirical results confirm that this posture is achievable on real hardware with real attacks. A device fully compromised in under five minutes at factory default can be made resilient against all tested attack vectors, including physical hardware attacks, RF-layer interception, AI-assisted insider exploitation, and supply chain manipulation, through the systematic application of AZTRM-D. These results come from a single-device validation with Cybectr-affiliated testers operating under a blind protocol; the limitations documented in Section 13.2 apply. Independent third-party validation and fleet-scale testing remain as future work. What the results do establish is a measured baseline, and that baseline is strong because the hardening is complete. The three-stage testing campaign showed that partial hardening isn't enough. A Stage 2 physical gap collapsed an otherwise solid logical security posture in minutes. AZTRM-D's value lies precisely in its completeness. It doesn't allow organizations to harden the network layer while leaving physical access unaddressed, or to secure the perimeter while neglecting insider threat controls.

The digital infrastructure society depends on is only as secure as the methodologies used to build it. AZTRM-D is a contribution toward making those methodologies more rigorous, more automated, and more resilient against the threats that actually exist.

ABBREVIATIONS

Abbreviation	Full Term	Abbreviation	Full Term
AC	Air Conditioning	AC1	Agreement Coefficient 1 (Gwet)
AES	Advanced Encryption Standard	AES-256	Advanced Encryption Standard 256-bit
AES-256-XTS	AES 256-bit in XEX-based Tweaked Code-Book Mode with Ciphertext Stealing	AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol	ANN	Artificial Neural Network
AODV	Ad hoc On-Demand Distance Vector	API	Application Programming Interface
ARM	Advanced RISC Machine	ARP	Address Resolution Protocol
AST	Abstract Syntax Tree	ATO	Authority to Operate
AV	Attack Vector (CVSS)	AWS	Amazon Web Services
AZTRM-D	Automated Zero Trust Risk Management with DevSecOps Integration	BB	Black-Box
BigVul	Big C/C++ Vulnerability Dataset	BLE	Bluetooth Low Energy
BSIMM	Building Security In Maturity Model	BSSID	Basic Service Set Identifier

Abbreviation	Full Term	Abbreviation	Full Term
CAE	Continuous Access Evaluation	CC	Common Criteria
CCTV	Closed-Circuit Television	CEO	Chief Executive Officer
CFG	Control Flow Graph	CI/CD	Continuous Integration/Continuous Delivery
CISA	Cybersecurity and Infrastructure Security Agency	CISO	Chief Information Security Officer
CMMC	Cybersecurity Maturity Model Certification	CoAP	Constrained Application Protocol
COI	Conflict of Interest	CoRE	Constrained RESTful Environments
CPG	Code Property Graph	CPS	Cyber-Physical Systems
CPU	Central Processing Unit	CSPM	Cloud Security Posture Management
CVE	Common Vulnerabilities and Exposures	CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration	D3FEND	Detection, Denial, and Disruption Framework Empowering Network Defense
D7AP	Dash7 Alliance Protocol	DAST	Dynamic Application Security Testing
DDoS	Distributed Denial of Service	DDPG	Deep Deterministic Policy Gradient
DDS	Data Distribution Service	DevOps	Development and Operations
DevSecOps	Development, Security, and Operations	DH	Diffie-Hellman

Abbreviation	Full Term	Abbreviation	Full Term
DiCE	Diverse Counterfactual Explanations	DIY	Do-It-Yourself
DNS	Domain Name System	DNSSEC	Domain Name System Security Extensions
DO-178C	Software Considerations in Airborne Systems and Equipment Certification	DoD	Department of Defense
DoS	Denial of Service	DoW	Department of War
DPI	Deep Packet Inspection	DQN	Deep Q-Network
DSA	Digital Signature Algorithm	DSS	Data Security Standard
DSSS	Direct-Sequence Spread Spectrum	DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography	ECDH	Elliptic Curve Diffie-Hellman
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral	ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm	ECG	Electrocardiogram
EDR	Endpoint Detection and Response	eMBB	Enhanced Mobile Broadband
ENGAGE	MITRE Adversary Engagement, Deception, and Denial Framework	F1	F1 Score
FDE	Full-Disk Encryption	FGSM	Fast Gradient Sign Method
FHSS	Frequency Hopping Spread Spectrum	FIPS	Federal Information Processing Standards
FPR	False Positive Rate	GAN	Generative Adversarial Network

Abbreviation	Full Term	Abbreviation	Full Term
GCN	Graph Convolutional Network	GDPR	General Data Protection Regulation
GHz	Gigahertz	GLP	Green Lightweight Pyramid
GNN	Graph Neural Network	GNU	GNU's Not Unix
GPIO	General Purpose Input/Output	GPS	Global Positioning System
GPU	Graphics Processing Unit	HIPAA	Health Insurance Portability and Accountability Act
HNDL	Harvest Now, Decrypt Later	HQC	Hamming Quasi-Cyclic
HTTP	Hypertext Transfer Protocol	HTTPS	Hypertext Transfer Protocol Secure
HVAC	Heating, Ventilation, and Air Conditioning	IaC	Infrastructure as Code
IAM	Identity and Access Management	IC	Integrated Circuit
ICMP	Internet Control Message Protocol	IDA	Interactive Disassembler
IDE	Integrated Development Environment	IDS	Intrusion Detection System
IEC	International Electrotechnical Commission	IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force	IIoT	Industrial Internet of Things
IoAT	Internet of Agricultural Things	IoBT	Internet of Battlefield Things

Abbreviation	Full Term	Abbreviation	Full Term
IoMT	Internet of Medical Things	IoT	Internet of Things
IP	Internet Protocol	IPv6	Internet Protocol version 6
ISM	Industrial, Scientific, and Medical	ISO	International Organization for Standardization
ISO/IEC	ISO / International Electrotechnical Commission	IT	Information Technology
JIT	Just-in-Time	JSON	JavaScript Object Notation
JWT	JSON Web Token	KDF	Key Derivation Function
KEM	Key Encapsulation Mechanism	KHz	Kilohertz
LAN	Local Area Network	LED	Light-Emitting Diode
LIME	Local Interpretable Model-agnostic Explanations	LLM	Large Language Model
LoRa	Long Range	LoRaWAN	Long Range Wide Area Network
LPWAN	Low-Power Wide-Area Network	LSTM	Long Short-Term Memory
LTE	Long-Term Evolution	LTE-A	Long-Term Evolution Advanced
LUKS	Linux Unified Key Setup	LUKS1	Linux Unified Key Setup Version 1
LUKS2	Linux Unified Key Setup Version 2	LWE	Learning With Errors
M2M	Machine-to-Machine	MAC	Media Access Control
MDI	Mean Decrease in Impurity	MFA	Multi-Factor Authentication

Abbreviation	Full Term	Abbreviation	Full Term
MHz	Megahertz	MITM	Man-in-the-Middle
MITRE	MITRE Corporation	ML	Machine Learning
ML-DSA	Module-Lattice-Based Digital Signature Algorithm	ML-KEM	Module-Lattice-Based Key-Encapsulation Mechanism
MLP	Multilayer Perceptron	MLWE	Module Learning With Errors
MQTT	Message Queuing Telemetry Transport Protocol	MS SDL	Microsoft Security Development Lifecycle
mTLS	Mutual Transport Layer Security	MTTR	Mean Time to Remediate
NB	Narrowband	NB-IoT	Narrowband Internet of Things
NFC	Near-Field Communication	NIST	National Institute of Standards and Technology
NSA	National Security Agency	NSM	National Security Memorandum
NSM-10	National Security Memorandum 10	NSM-8	National Security Memorandum 8
NSS	National Security Systems	NTP	Network Time Protocol
NVD	National Vulnerability Database	NVMe	Non-Volatile Memory Express
OCI	Open Container Initiative	OCSVM	One-Class Support Vector Machine
OIDC	OpenID Connect	OMB	Office of Management and Budget

Abbreviation	Full Term	Abbreviation	Full Term
OMG	Object Management Group	OPC	Open Platform Communications
OQS	Open Quantum Safe	OUI	Organizationally Unique Identifier
OWASP	Open Worldwide Application Security Project	P-256	NIST P-256 Elliptic Curve
PBKDF2	Password-Based Key Derivation Function 2	PCB	Printed Circuit Board
PCI	Payment Card Industry	PCI-DSS	Payment Card Industry Data Security Standard
PDG	Program Dependence Graph	PDP	Policy Decision Point
PEP	Policy Enforcement Point	PGD	Projected Gradient Descent
PII	Personally Identifiable Information	PKI	Public Key Infrastructure
PPO	Proximal Policy Optimization	PQC	Post-Quantum Cryptography
PR	Privileges Required (CVSS)	PSK	Pre-Shared Key
QA	Quality Assurance	QoS	Quality of Service
RAD	Rapid Application Development	RAG	Retrieval-Augmented Generation
RASP	Runtime Application Self-Protection	RBAC	Role-Based Access Control
RC4	Rivest Cipher 4	RCE	Remote Code Execution
RF	Radio Frequency	RFC	Request for Comments
RL	Reinforcement Learning	RMF	Risk Management Framework

Abbreviation	Full Term	Abbreviation	Full Term
ROC	Receiver Operating Characteristic	RSA	Rivest-Shamir-Adleman
RTL	Realtek (RTL-SDR chipset family)	RTL-SDR	Real-Time Linux Software-Defined Radio
RTL2832U	Realtek RTL2832U Chipset	S-SDLC	Secure Software and System Development Lifecycle
SAE	Simultaneous Authentication of Equals	SAMM	Software Assurance Maturity Model
SASL	Simple Authentication and Security Layer	SAST	Static Application Security Testing
SATA	Serial Advanced Technology Attachment	SBOM	Software Bill of Materials
SCA	Software Composition Analysis	SCP	Secure Copy Protocol
SD	Secure Digital	SDL	Security Development Lifecycle
SDLC	Software Development Lifecycle	SDR	Software-Defined Radio
SEAD	Secure Efficient Ad hoc Distance vector	SHA	Secure Hash Algorithm
SHAP	SHapley Additive exPlanations	SIEM	Security Information and Event Management
SLA	Service Level Agreement	SLH	Stateless Hash-Based
SLH-DSA	Stateless Hash-Based Digital Signature Algorithm	SMTP	Simple Mail Transfer Protocol
SOAR	Security Orchestration, Automation, and Response	SOX	Sarbanes-Oxley Act

Abbreviation	Full Term	Abbreviation	Full Term
SPIFFE	Secure Production Identity Framework for Everyone	SPIRE	SPIFFE Runtime Environment
SpO_2	Peripheral Capillary Oxygen Saturation	SSDF	Secure Software Development Framework
SSH	Secure Shell	SSL	Secure Sockets Layer
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege	SUID	Set User ID on Execution
SVID	SPIFFE Verifiable Identity Document	SVM	Support Vector Machine
SYN	Synchronize (TCP)	TCP	Transmission Control Protocol
TLS	Transport Layer Security	TLS/SSL	Transport Layer Security / Secure Sockets Layer
TPR	True Positive Rate	TTID	Time to Initial Detection
TTP	Tactics, Techniques, and Procedures	UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol	UI	User Interaction (CVSS)
UML	Unified Modeling Language	URLLC	Ultra-Reliable Low-Latency Communication
USB	Universal Serial Bus	USRP	Universal Software Radio Peripheral
UWB	Ultra-Wideband	V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle	VDR	Vulnerability Detection Rate
VLC	Visible Light Communication	VNC	Virtual Network Computing

Abbreviation	Full Term	Abbreviation	Full Term
VPN	Virtual Private Network	WB	White-Box
WEP	Wired Equivalent Privacy	WiFi	Wireless Fidelity
WPA	Wi-Fi Protected Access	WPA2	Wi-Fi Protected Access II
WPA3	Wi-Fi Protected Access III	WPA3-SAE	WPA3 with Simultaneous Authentication of Equals
WSAN	Wireless Sensor and Actuator Network	WSN	Wireless Sensor Network
XAI	Explainable Artificial Intelligence	XGBoost	Extreme Gradient Boosting
XTS	XEX-based Tweaked CodeBook Mode with Ciphertext Stealing	ZigBee	Zigbee Wireless Protocol
ZT	Zero Trust	ZTA	Zero Trust Architecture
ZTNA	Zero Trust Network Access		

BIBLIOGRAPHY

[1] Object Management Group (OMG). "The real-time publish-subscribe wire protocol DDS interoperability wire protocol specification". In: *OMG, Version 2.2* (2014).

[2] Zain Ul Abideen. *Reconfigurable Obfuscation Techniques for the IC Supply Chain: Using FPGA-Like Schemes for Protection of Intellectual Property*. Springer Nature, 2024.

[3] Zain Ul Abideen, Sumathi Gokulanathan, Muayad J. Aljafar, and Samuel Pagliarini. "An overview of FPGA-inspired obfuscation techniques". In: *ACM Computing Surveys* 56.12 (2024), pp. 1–35.

[4] Abidemi Emmanuel Adeniyi, Rasheed Gbenga Jimoh, and Joseph Bamidele Awotunde. "A systematic review on elliptic curve cryptography algorithm for internet of things: Categorization, application areas, and security". In: *Computers and Electrical Engineering* 118 (2024), p. 109330.

[5] Prachal Agarwal, Aroshi Singhal, and Archit Garg. "SDLC model selection tool and risk incorporation". In: *Int. J. Comput. Appl* 172.10 (2017), pp. 6–10.

[6] Gwanghyun Ahn, Jisoo Jang, Seho Choi, and Dongkyoo Shin. "Research on Improving Cyber Resilience by Integrating the Zero Trust security model with the MITRE ATT&CK matrix". In: *IEEE Access* (2024).

[7] G.R. Aiello and G.D. Rogerson. "Ultra-wideband wireless systems". In: *IEEE Microwave Magazine* 4.2 (2003), pp. 36–47. DOI: 10.1109/MMW.2003.1201597.

[8] Deepa Ajish. "The significance of artificial intelligence in zero trust technologies: a comprehensive review". In: *Journal of Electrical Systems and Information Technology* 11.1 (2024), p. 30.

[9] Akitra. *The Ethical Considerations of AI in Cybersecurity: Balancing Security Needs with Algorithmic Bias and Transparency*. Medium. Accessed: June 30, 2025. Sept. 2024. URL: https://medium.com/@akitrablog/the-ethical-considerations-of-ai-in-cybersecurity-balancing-security-needs-with-algorithmic-bias-3b083488ba23.

[10] Ian F Akyildiz, David M Gutierrez-Estevez, and Elias Chavarria Reyes. "The evolution to 4G cellular systems: LTE-Advanced". In: *Physical communication* 3.4 (2010), pp. 217–244.

[11] Gorjan Alagic et al. "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process". In: *US Department of Commerce, NIST* (2022). URL: https://doi.org/10.6028/NIST.IR.8413-upd1.

[12] Marwan Albahar. "A systematic literature review of the current state and challenges of DevSecOps". In: *Journal of King Saud University-Computer and Information Sciences* 34.9 (2022), pp. 7253–7268.

[13] Shoukat Ali, Muazzam A Khan, Jawad Ahmad, Asad W. Malik, and Anis ur Rehman. "Detection and prevention of Black Hole Attacks in IOT & WSN". In: *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. 2018, pp. 217–226. DOI: 10.1109/FMEC.2018.8364068.

[14] Hamza Alkofahi, Heba Alawneh, and Anthony Skjellum. "MitM attacks on intellectual property and integrity of additive manufacturing systems: A security analysis". In: *Computers & Security* 140 (2024), p. 103810.

[15] Halah Alqurashi, Fatma Bouabdallah, and Enas Khairullah. "SCAP SigFox: A Scalable Communication Protocol for Low-Power Wide-Area IoT Networks".

In: *Sensors* 23.7 (2023). ISSN: 1424-8220. DOI: 10.3390/s23073732. URL: https://www.mdpi.com/1424-8220/23/7/3732.

[16] Thanaa Saad AlSalem, Mohammed Amin Almaiah, and Abdalwali Lutfi. "Cybersecurity risk analysis in the IoT: A systematic review". In: *Electronics* 12.18 (2023), p. 3958.

[17] Raghavendra Rao Althar, Debabrata Samanta, Manjit Kaur, Dilbag Singh, and Heung-No Lee. "Automated risk management based software security vulnerabilities management". In: *IEEE Access* 10 (2022), pp. 90597–90608.

[18] Mnassar Alyami, Ibrahim Alharbi, Cliff Zou, Yan Solihin, and Karl Ackerman. "WiFi-based IoT Devices Profiling Attack based on Eavesdropping of Encrypted WiFi Traffic". In: *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. 2022, pp. 385–392. DOI: 10.1109/CCNC49033.2022.9700674.

[19] Anchore, Inc. *Grype: A vulnerability scanner for container images and filesystems*. https://github.com/anchore/grype. Apache-2.0 License. 2024.

[20] Anchore, Inc. *Syft: CLI tool and library for generating a Software Bill of Materials from container images and filesystems*. https://github.com/anchore/syft. Apache-2.0 License. 2024.

[21] Malak Annabi, Abdelhafid Zeroual, and Nadhir Messai. "Towards zero trust security in connected vehicles: A comprehensive survey". In: *Computers & Security* (2024), p. 104018.

[22] Anonymous. "AIGC Video Detection Based on the Fusion of Spatial-Frequency-Optical Flow Multi-Modal Features". In: *Journal of Systems Engineering and Electronics* (2026). DOI: 10.23919/JSEE.2026.000049.

[23] Mohamed Ould-Elhassen Aoueileyine, Neder Karmous, Ridha Bouallegue, Neji Youssef, and Anis Yazidi. "Detecting and mitigating MiTM attack on IOT devices using SDN". In: *International Conference on Advanced Information Networking and Applications*. Springer. 2024, pp. 320–330.

[24] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafiekhah, and P. Siano. "Iot-based smart cities: A survey". In: (2016), pp. 1–6. DOI: 10.1109/EEEIC.2016.7555867.

[25] Daniel Arrey Arrey. "Exploring the integration of security into software development life cycle (SDLC) methodology". PhD thesis. Colorado Technical University, 2019.

[26] Muhammad Fasih Ashfaq, Maryam Malik, Urooj Fatima, and Muhammad Khuram Shahzad. "Classification of IoT based DDoS Attack using Machine Learning Techniques". In: *2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM)*. 2022, pp. 1–6. DOI: 10.1109/IMCOM53663.2022.9721740.

[27] Ömer Aslan, Semih Aktuğ, Merve Ozkan-Okay, Abdullah Yilmaz, and Erdal Akin. "A Comprehensive Review of Cyber Security Vulnerabilities, Threats, Attacks, and Solutions". In: *Electronics* 12.6 (2023), p. 1333. DOI: 10.3390/electronics12061333.

[28] Wael Ayoub, Abed Ellatif Samhat, Fabienne Nouvel, Mohamad Mroue, and Jean-Christophe Prévotet. "Internet of Mobile Things: Overview of LoRaWAN, DASH7, and NB-IoT in LPWANs Standards and Supported Mobility". In: *IEEE Communications Surveys and Tutorials* 21.2 (2019), pp. 1561–1581. DOI: 10.1109/COMST.2018.2877382.

[29] Mohammed Aziz Al Kabir, Wael Elmedany, and Mhd Saeed Sharif. "Securing IOT devices against emerging security threats: Challenges and mitigation techniques". In: *Journal of Cyber Security Technology* 7.4 (2023), pp. 199–223.

[30] Nick Baker. "ZigBee and Bluetooth: Strengths and weaknesses for industrial applications". In: *Computing and Control Engineering* 16.2 (2005), pp. 20–25.

[31] Taimur Bakhshi, Bogdan Ghita, and Ievgeniia Kuzminykh. "A review of IoT firmware vulnerabilities and auditing techniques". In: *Sensors* 24.2 (2024), p. 708.

[32] Parameswar Banerjee and Demetrios Matsakis. "Network Time Protocol (NTP) and Precise Time Protocol (PTP)". In: *An Introduction to Modern Timekeeping and Time Transfer*. Springer, 2023, pp. 141–152.

[33] Sourangsu Banerji and Rahul Singha Chowdhury. "On IEEE 802.11: Wireless Lan Technology". In: *International Journal of Mobile Network Communications & Telematics* 3.4 (Aug. 2013), pp. 45–64. DOI: 10.5121/ijmnct.2013.3405. URL: https://doi.org/10.5121%2Fijmnct.2013.3405.

[34] Yaser Baseri et al. "Evaluation Framework for Quantum Security Risk Assessment: A Comprehensive Strategy for Quantum-Safe Transition". In: *Computers & Security* 150 (2025), p. 104272. DOI: 10.1016/j.cose.2024.104272.

[35] Juan Bermejo Higuera, Cayetano Abad Aramburu, Javier-Roberto Bermejo Higuera, Miguel Angel Sicilia Urban, and Juan Antonio Sicilia Montalvo. "On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications". In: *Applied Sciences* 10.24 (2020), p. 9119. DOI: 10.3390/app10249119.

[36] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, eds. *Post-Quantum Cryptography*. 1st. Berlin, Heidelberg: Springer, 2009. DOI: 10.1007/978-3-540-88702-7.

[37] August Betzler, Carles Gomez, Ilker Demirkol, and Josep Paradells. "CoAP congestion control for the internet of things". In: *IEEE Communications Magazine* 54.7 (2016), pp. 154–160.

[38] R Bhavya and MR Lokesh. "A Survey on Li-Fi Technology". In: *An International Journal of Engineering & Technology* 3.1 (2016).

[39] Battista Biggio and Fabio Roli. "Wild Patterns: Ten Years After the Rise of Adversarial Machine Learning". In: *Pattern Recognition* 84 (2018), pp. 317–331. DOI: 10.1016/j.patcog.2018.07.023.

[40] C. Bisdikian. "An overview of the Bluetooth wireless technology". In: *IEEE Communications Magazine* 39.12 (2001), pp. 86–94. DOI: 10.1109/35.968817.

[41] Kaitlin Boeckl, Michael Fagan, William Fisher, Naomi Lefkovitz, Katerina N. Megas, Ellen Nadeau, Ben Piccarreta, Danna Gabel O'Rourke, and Karen Scarfone. *Considerations for Managing Internet of Things (IoT) Cybersecurity and Privacy Risks*. NIST Internal Report 8228. National Institute of Standards and Technology, 2019. DOI: 10.6028/NIST.IR.8228. URL: https://csrc.nist.gov/pubs/ir/8228/final.

[42] Barry W. Boehm. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981. ISBN: 978-0138221225.

[43] Xavier Bonnetain, María Naya-Plasencia, and André Schrottenloher. "Quantum Security Analysis of AES". In: *IACR Transactions on Symmetric Cryptology* 2019.2 (2019), pp. 55–93. DOI: 10.13154/tosc.v2019.i2.55-93.

[44] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. "CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 353–367. DOI: 10.1109/EuroSP.2018.00032.

[45] E. Bou-Harb, C. Fachkha, M. Pourzandi, M. Debbabi, and C. Assi. "Communication security for smart grid distribution networks". In: *IEEE Commun. Mag.* 51.1 (Jan. 2013), pp. 42–49.

[46] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[47] CardiacSense. *Heart Rate Monitor Watch*. https://www.cardiacsense.com/heart-rate-monitor-watch/. Accessed: June 2023. 2023.

[48] Nicholas Carlini and David Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 39–57. DOI: 10.1109/SP.2017.49.

[49] Wouter Castryck and Thomas Decru. "An Efficient Key Recovery Attack on SIDH". In: *Advances in Cryptology – EUROCRYPT 2023*. Springer. 2023, pp. 423–447. DOI: 10.1007/978-3-031-30589-4_15.

[50] Suman Chahar and Satyaveer Singh. "Analysis of SDLC Models with Web Engineering Principles". In: *2024 2nd International Conference on Advancements and Key Challenges in Green Energy and Computing (AKGEC)*. IEEE. 2024, pp. 1–7.

[51] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Computing Surveys* 41.3 (2009), pp. 1–58. DOI: 10.1145/1541880.1541882.

[52] Ramaswamy Chandramouli and Zack Butcher. *A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Location Environments*. Tech. rep. NIST Special Publication (SP) 800-207A. National Institute of Standards and Technology, Sept. 2023. DOI: 10.6028/NIST.SP.800-207A. URL: https://csrc.nist.gov/pubs/sp/800/207/a/final.

[53] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on Post-Quantum Cryptography*. Tech. rep. NIST IR 8105. National Institute of Standards and Technology, 2016. DOI: 10.6028/NIST.IR.8105.

[54] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[55] Yuang Chen and Thomas Kunz. "Performance evaluation of IoT protocols under a constrained wireless access network". In: *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*. 2016, pp. 1–7. DOI: 10.1109/MoWNet.2016.7496622.

[56] Lalit Chettri and Rabindranath Bera. "A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems". In: *IEEE Internet of Things Journal* 7.1 (2019), pp. 16–32.

[57] Jihoon Cho, Hyojin Yoon, Changhoon Lee, Eunkyung Kim, Janghyuk Ahn, and Hunhee Yu. "Software-Defined Cryptography: A Design Feature of Cryptographic Agility". In: *arXiv preprint arXiv:2404.01808* (2024).

[58] Henoch Juli Christanto and Yerik Afrianto Singgalen. "Analysis and design of student guidance information system through software development life cycle (sdlc) and waterfall model". In: *Journal of Information Systems and Informatics* 5.1 (2023), pp. 259–270.

[59] Tan Jia Chun, Lau Jing En, Malcolm Tan Yu Xuen, Yap Ming Xuan, and Saira Muzafar. "Secured Software Development and Importance of Secure Software Development Life Cycle". In: *Authorea Preprints* (2023).

[60] CISA. *Prepare for a New Cryptographic Standard to Protect Against Future Quantum-Based Threats*. https://www.cisa.gov/news-events/alerts/2022/07/05/prepare-new-cryptographic-standard-protect-against-future-quantum-based-threats. 2022.

[61] Cloudflare. "Post-Quantum Cryptography Goes GA". In: *Cloudflare Blog* (2024). URL: https://blog.cloudflare.com/post-quantum-cryptography-ga.

[62] Zachary A. Collier and Joseph Sarkis. "The zero trust supply chain: Managing supply chain risk in the absence of trust". In: *International Journal of Production Research* 59.11 (2021), pp. 3430–3445. DOI: 10.1080/00207543.2021.1884311.

[63] Congress.gov. *H.R.7535 - 117th Congress (2021-2022): Quantum Computing Cybersecurity Preparedness Act*. https://www.congress.gov/bill/117th-congress/house-bill/7535. 2021.

[64] Microsoft Corporation. *Microsoft Security Development Lifecycle (SDL) Version 5.2*. Tech. rep. Microsoft Corporation, 2012. URL: https://www.microsoft.com/en-us/securityengineering/sdl.

[65] Vedat Coskun, Kerem Ok, and Busra Ozdenizci. *Near field communication (NFC): From theory to practice*. John Wiley & Sons, 2011.

[66] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. "A survey on near field communication (NFC) technology". In: *Wireless personal communications* 71 (2013), pp. 2259–2294.

[67] Ian Coston, Karl David Hezel, Eadan Plotnizky, and Mehrdad Nojoumian. "Enhancing Secure Software Development with AZTRM-D: An AI-Integrated Approach Combining DevSecOps, Risk Management, and Zero Trust". In: *Applied Sciences* 15.15 (2025). ISSN: 2076-3417. DOI: 10.3390/app15158163. URL: https://www.mdpi.com/2076-3417/15/15/8163.

[68] Ian Coston, Eadan Plotnizky, and Mehrdad Nojoumian. "Comprehensive Study of IoT Vulnerabilities and Countermeasures". In: *Applied Sciences* 15.6 (2025). ISSN: 2076-3417. DOI: 10.3390/app15063036. URL: https://www.mdpi.com/2076-3417/15/6/3036.

[69] Ian Matthew Campbell Coston, Karl David Hezel, Eadan Plotnizky, and Mehrdad Nojoumian. "Multi-Vector Adversarial Testing of an AI-Orchestrated Zero Trust Methodology on Constrained Edge Hardware". In: *Applied Sciences* 16.10 (2026). ISSN: 2076-3417. DOI: 10.3390/app16104809. URL: https://www.mdpi.com/2076-3417/16/10/4809.

[70] Cybersecurity and Infrastructure Security Agency. *CISA Quantum Initiative*. https://www.cisa.gov/quantum. 2023.

[71] Cybersecurity and Infrastructure Security Agency. *Zero Trust Maturity Model Version 2.0*. Tech. rep. CISA, 2023. URL: https://www.cisa.gov/zero-trust-maturity-model.

[72] Cybersecurity and Infrastructure Security Agency (CISA). *Zero Trust Maturity Model v2*. Accessed: 2024-08-12. Apr. 2023. URL: https://www.cisa.gov/sites/default/files/2023-04/zero_trust_maturity_model_v2_508.pdf.

[73] J. Czyz, M. J. Luckie, M. Allman, and M. Bailey. "Don't forget to lock the back door! A characterization of IPv6 network security policy". In: *Proc. NDSS*. 2016.

[74] Salim Jibrin Danbatta and Asaf Varol. "Comparison of Zigbee, Z-Wave, Wi-Fi, and bluetooth wireless technologies used in home automation". In: *2019 7th International Symposium on Digital Forensics and Security (ISDFS)*. IEEE. 2019, pp. 1–5.

[75] Defense Information Systems Agency and National Security Agency Zero Trust Engineering Team. *Department of Defense (DoD) Zero Trust Reference Architecture Version 2.0*. DoD CIO, cleared for open publication September 12, 2022. July 2022. URL: https://dodcio.defense.gov/Portals/0/Documents/Library/(U)ZT_RA_v2.0(U)_Sep22.pdf.

[76] Kelley Dempsey, Paul Eavy, and George Moore. *Automation Support for Security Control Assessments, Volume 1: Overview*. Tech. rep. NIST IR 8011 Vol. 1. Gaithersburg, MD: National Institute of Standards and Technology, 2017. DOI: 10.6028/NIST.IR.8011-1.

[77] Department of Defense Chief Information Officer (DoD CIO). *DoD Enterprise DevSecOps Strategy Guide*. Accessed: 2024-08-12. Oct. 2021. URL: https://dodcio.defense.gov/Portals/0/Documents/Library/DoD%20Enterprise%20DevSecOps%20Strategy%20Guide_DoD-CIO_20211019.pdf.

[78] Department of Defense Chief Information Officer (DoD CIO). *DoD Zero Trust Strategy*. DoD CIO Zero Trust Portfolio Management Office, cleared for open publication November 7, 2022. Oct. 2022. URL: https://dodcio.defense.gov/Portals/0/Documents/Library/DoD-ZTStrategy.pdf.

[79] Onur Doğan, Semih Bitim, and Kadir Hızıroğlu. "A v-model software development application for sustainable and smart campus analytics domain". In: *Sakarya University Journal of Computer and Information Sciences* 4.1 (2021), pp. 111–119.

[80] Fanping Du and Pingfang Du. "Micro frequency hopping spread spectrum modulation and encryption technology". In: *arXiv preprint arXiv:2408.00400* (2024).

[81] Onome Christopher Edo, Theophilus Tenebe, Egbe-Etu Etu, Atamgbo Ayuwu, Joshua Emakhu, and Shakiru Adebiyi. "Zero Trust Architecture: Trend and

Impacton Information Security". In: *International Journal of Emerging Technology and Advanced Engineering* 12.7 (2022), p. 140.

[82] A. Elahi and A. Gschwender. *ZigBee Wireless Sensor and Control Network*. London, U.K.: Pearson Educ., 2009.

[83] Sinem Coleri Ergen. "ZigBee/IEEE 802.15. 4 Summary". In: *UC Berkeley, September* 10.17 (2004), p. 11.

[84] Executive Office of the President. *National Security Memorandum on Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems*. NSM-10. May 2022.

[85] E Ezhilarasan and M Dinakaran. "A review on mobile technologies: 3G, 4G and 5G". In: *2017 second international conference on recent trends and challenges in computational models (ICRTCCM)*. IEEE. 2017, pp. 369–373.

[86] Michael Fagan, Jeffrey Marron, Jr. Brady Kevin G., Barbara M. Cuthill, Katerina N. Megas, Rebecca Herold, David Lemire, and Brad Hoehn. *IoT Device Cybersecurity Guidance for the Federal Government: Establishing IoT Device Cybersecurity Requirements*. NIST Special Publication 800-213. National Institute of Standards and Technology, 2021. DOI: 10.6028/NIST.SP.800-213. URL: https://csrc.nist.gov/pubs/sp/800/213/final.

[87] Michael Fagan, Jeffrey Marron, Jr. Brady Kevin G., Barbara M. Cuthill, Katerina N. Megas, Rebecca Herold, David Lemire, and Brad Hoehn. *IoT Device Cybersecurity Guidance for the Federal Government: IoT Device Cybersecurity Requirement Catalog*. NIST Special Publication 800-213A. National Institute of Standards and Technology, 2021. DOI: 10.6028/NIST.SP.800-213A. URL: https://csrc.nist.gov/pubs/sp/800/213/a/final.

[88] fail0verflow. *Console Hacking 2010: PS3 Epic Fail*. Presentation at the 27th Chaos Communication Congress (27C3), Berlin, Germany. Demonstrated re-

covery of Sony PlayStation 3 ECDSA private key through nonce reuse; published keys released by George Hotz on January 3, 2011. Dec. 2010. URL: https://events.ccc.de/congress/2010/Fahrplan/events/4029.en.html.

[89] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. "A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries". In: *Proceedings of the 17th International Conference on Mining Software Repositories (MSR)*. ACM, 2020, pp. 508–512. DOI: 10.1145/3379597.3387501.

[90] S. Farahani. *ZigBee Wireless Networks and Transceivers*. Newnes, 2011.

[91] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Kamran Abid, and Muhammad Azhar Naeem. "A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming". In: *IEEE Access* 7 (2019), pp. 156237–156271. DOI: 10.1109/ACCESS.2019.2949703.

[92] Wen Fei, Hiroyuki Ohno, and Srinivas Sampalli. "A systematic review of iot security: Research potential, challenges, and future directions". In: *ACM computing surveys* 56.5 (2023), pp. 1–40.

[93] Alvan R. Feinstein and Domenic V. Cicchetti. "High Agreement But Low Kappa: I. The Problems of Two Paradoxes". In: *Journal of Clinical Epidemiology* 43.6 (1990), pp. 543–549. DOI: 10.1016/0895-4356(90)90158-L.

[94] Joel L Fernandes, Ivo C Lopes, Joel JPC Rodrigues, and Sana Ullah. "Performance evaluation of RESTful web services and AMQP protocol". In: *2013 Fifth international conference on ubiquitous and future networks (ICUFN)*. IEEE. 2013, pp. 810–815.

[95] Tiago M. Fernández-Caramés. "From Pre-Quantum to Post-Quantum IoT Security: A Survey on Quantum-Resistant Cryptosystems for the Internet of

Things". In: *IEEE Internet of Things Journal* 7.7 (2020), pp. 6457–6480. DOI: 10.1109/JIOT.2019.2958788.

[96] Luís Ferreira, André Luiz Pilastri, Carlos Martins, Pedro Santos, and Paulo Cortez. "An Automated and Distributed Machine Learning Framework for Telecommunications Risk Management". In: *ICAART (2)*. 2020, pp. 99–107.

[97] FIRST.Org, Inc. *Common Vulnerability Scoring System v3.1: Specification Document*. Tech. rep. Defines the metric groups, scoring formulas, and vector string encoding used for CVSS v3.1 base scores. Forum of Incident Response and Security Teams (FIRST), 2019. URL: https://www.first.org/cvss/v3.1/specification-document.

[98] Scott Fluhrer, Panos Kampanakis, David McGrew, and Valery Smyslov. *Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security*. Tech. rep. RFC 8784. Internet Engineering Task Force (IETF), 2020. DOI: 10.17487/RFC8784.

[99] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps*. Portland, OR: IT Revolution Press, 2018. ISBN: 978-1942788331.

[100] Christophe Fourtet and Benoit Ponsard. "An introduction to Sigfox radio system". In: *LPWAN Technologies for IoT and M2M Applications*. Elsevier, 2020, pp. 103–118.

[101] Chenglong Fu, Qiang Zeng, Haotian Chi, Xiaojiang Du, and Siva Likitha Valluru. "IoT Phantom-Delay Attacks: Demystifying and Exploiting IoT Timeout Behaviors". In: *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2022, pp. 428–440. DOI: 10.1109/DSN53405.2022.00050.

[102] Jinghua Fu et al. “AI-Driven Cybersecurity: An Overview, Security Intelligence Modeling and Research Directions”. In: *Artificial Intelligence Review* 57 (2024), pp. 1–45. DOI: 10.1007/s10462-024-10723-8.

[103] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys and Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.

[104] Jaime Fuster, Sonia Solera-Cotanilla, Jaime Pérez, Mario Vega-Barbas, Rafael Palacios, Manuel Alvarez-Campana, and Gregorio Lopez. “Analysis of security and privacy issues in wearables for minors”. In: *Wireless Networks* 30.6 (2024), pp. 5437–5453.

[105] Kelum AA Gamage, Asher Sajid, Omar S Sonbul, Muhammad Rashid, and Amar Y Jaffar. “A Dynamic Framework for Internet-Based Network Time Protocol”. In: *Sensors* 24.2 (2024).

[106] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. “A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security”. In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 1646–1685. DOI: 10.1109/COMST.2020.2988293.

[107] Jason Garbis and Jerry W. Chapman. *Zero Trust Security: An Enterprise Guide*. Berkeley, CA: Apress, 2021. DOI: 10.1007/978-1-4842-6702-8.

[108] Maria-Lill Garvik, Synne Lindås, and Fridtjof Bø Svendsen. “Authentication with the use of MAC and it’s security challenges”. B.S. thesis. NTNU, 2023.

[109] Matthew Gatti Gaydos, Nicholas Lincoln Wallace, and Richard Grant Brown. *Reverse Engineering and Embedded Processor Analysis*. 2020.

[110] Oluwaseun Gbohunmi, Adebayo Abayomi-Alli, Opeyemi Abayomi-Alli, and Sanjay Misra. "Zero Trust Architecture: A Systematic Literature Review". In: *arXiv preprint arXiv:2503.11659* (2025).

[111] Isaías González, Antonio José Calderón, João Figueiredo, and João M. C. Sousa. "A Literature Survey on Open Platform Communications (OPC) Applied to Advanced Industrial Environments". In: *Electronics* 8.5 (2019). ISSN: 2079-9292. DOI: 10.3390/electronics8050510. URL: https://www.mdpi.com/2079-9292/8/5/510.

[112] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[113] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *arXiv preprint arXiv:1412.6572* (2015).

[114] Government Accountability Office (GAO). *Future of Cybersecurity: Leadership Needed to Fully Define Quantum Threat Mitigation Strategy*. Report. 2025. URL: https://www.gao.gov/assets/gao-25-107703.pdf.

[115] Daniele Granata, Massimiliano Rak, and Giovanni Salzillo. "Risk analysis automation process in it security for cloud applications". In: *International Conference on Cloud Computing and Services Science*. Springer. 2021, pp. 47–68.

[116] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. "Applying Grover's Algorithm to AES: Quantum Resource Estimates". In: *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016*. Springer. 2016, pp. 29–43. DOI: 10.1007/978-3-319-29360-8_3.

[117] Greenbone Networks GmbH. *OpenVAS: Open Vulnerability Assessment Scanner (Greenbone Community Edition)*. https://www.openvas.org/. GNU General Public License. 2024.

[118] David Greenwood. "Applying the principles of zero-trust architecture to protect sensitive and critical data". In: *Network Security* 2021.6 (2021), pp. 7–9.

[119] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC '96)*. Association for Computing Machinery. New York, NY, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[120] Amardeep Gupta, Prathak Gupta, Upendra Pratap Pandey, Pradeep Kushwaha, Bhanu Prakash Lohani, and Kushank Bhati. "ZTSA: Zero Trust Security Architecture a Comprehensive Survey". In: *2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE)*. IEEE. 2024, pp. 378–383.

[121] Aparna Gupta, Achint Rawal, and Yamini Barge. "Comparative Study of Different SDLC Models". In: *Int. J. Res. Appl. Sci. Eng. Technol* 9.11 (2021), pp. 73–80.

[122] Kilem L. Gwet. "Computing Inter-Rater Reliability and Its Variance in the Presence of High Agreement". In: *British Journal of Mathematical and Statistical Psychology* 61.1 (2008), pp. 29–48. DOI: 10.1348/000711006X126600.

[123] Harald Haas, Liang Yin, Yunlu Wang, and Cheng Chen. "What is LiFi?" In: *Journal of Lightwave Technology* 34.6 (2016), pp. 1533–1544. DOI: 10.1109/JLT.2015.2510021.

[124] Keijo Haataja and Pekka Toivanen. "Two practical man-in-the-middle attacks on Bluetooth secure simple pairing and countermeasures". In: *IEEE Transactions on Wireless Communications* 9.1 (2010), pp. 384–392. DOI: 10.1109/TWC.2010.01.090935.

[125] Kim Hammar and Rolf Stadler. "Learning Security Strategies Through Game Play and Optimal Stopping". In: *IEEE Transactions on Network and Service Management* 20.4 (2023), pp. 5536–5555. DOI: 10.1109/TNSM.2023.3280267.

[126] Khondokar Fida Hasan, Leonie Simpson, Mir Ali Rezazadeh Baee, Chadni Islam, Ziaur Rahman, Warren Armstrong, Praveen Gauravaram, and Matthew McKague. "A Framework for Migrating to Post-Quantum Cryptography: Security Dependency Analysis and Case Studies". In: *IEEE Access* 12 (2024), pp. 23427–23450. DOI: 10.1109/ACCESS.2024.3360412.

[127] Jetmir Haxhibeqiri, Eli De Poorter, Ingrid Moerman, and Jeroen Hoebeke. "A survey of LoRaWAN for IoT: From technology to application". In: *Sensors* 18.11 (2018), p. 3995.

[128] Yuanhang He, Daochao Huang, Lei Chen, Yi Ni, and Xiangjie Ma. "A survey on zero trust architecture: Challenges and future trends". In: *Wireless Communications and Mobile Computing* 2022 (2022).

[129] Jennifer Hernandez, Oscar Mondragon, and Raul Gil. "A systematic review on threat modeling for the software development lifecycle". In: *Computers & Security* 132 (2023), p. 103348.

[130] Walter Hirt. "Ultra-wideband radio technology: overview and future research". In: *Computer Communications* 26.1 (2003), pp. 46–52.

[131] Paul E Hoffman. "DNS security extensions (DNSSEC)". In: *RFC 9364* (2023).

[132] Daniel Horne and Suku Nair. "Introducing zero trust by design: Principles and practice beyond the zero trust hype". In: *Advances in security, networks, and internet of things* (2021), pp. 512–525.

[133] Eslam Samy Hosney, Islam Tharwat Abdel Halim, and Ahmed H. Yousef. "An Artificial Intelligence Approach for Deploying Zero Trust Architecture (ZTA)".

In: *2022 5th International Conference on Computing and Informatics (ICCI)*. 2022, pp. 343–350. DOI: 10.1109/ICCI54321.2022.9756117.

[134] Yandong Hou, Tianzhi Li, Jinjin Wang, Jiulong Ma, and Zhengquan Chen. "A Lightweight Transformer Based on Feature Fusion and Global-Local Parallel Stacked Self-Activation Unit for Bearing Fault Diagnosis". In: *Measurement* 236 (2024), p. 115068. DOI: 10.1016/j.measurement.2024.115068.

[135] Russ Housley. *Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms*. Tech. rep. RFC 7696. Internet Engineering Task Force (IETF), 2015. DOI: 10.17487/RFC7696.

[136] Michael Howard and Steve Lipner. *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*. Redmond, WA: Microsoft Press, 2006. ISBN: 978-0735622142.

[137] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. "SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks". In: *Ad Hoc Networks* 1.1 (2003), pp. 175–192. ISSN: 1570-8705. DOI: https://doi.org/10.1016/S1570-8705(03)00019-2. URL: https://www.sciencedirect.com/science/article/pii/S1570870503000192.

[138] Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. "Machine Learning in IoT Security: Current Solutions and Future Challenges". In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 1686–1721. DOI: 10.1109/COMST.2020.2986444.

[139] David Hutchison, Christopher Lang, and Joseph Wagner. "Automating the risk management framework (RMF) for agile systems". In: *2022 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE. 2022, pp. 196–201.

[140] IBM Systems Sciences Institute. *Relative Costs to Fix Software Defects*. Tech. rep. Internal study documenting defect remediation cost multipliers by SDLC phase; widely cited in NIST and industry literature on shift-left security economics. IBM Corporation, 2008.

[141] Muhammad Imran, Pan Zhiwen, Liu Nan, Muhammad Sajjad, and Faisal Mehmood Butt. "Anti-jamming for cognitive radio networks with Stackelberg game-assisted DSSS approach". In: *EURASIP Journal on Wireless Communications and Networking* 2024.1 (2024), p. 73.

[142] Industry Consortium. "Enterprise Post-Quantum Cryptography Migration: Timelines and Challenges". In: *Cybersecurity Industry Report* (2025).

[143] S. M. Riazul Islam, Daehan Kwak, Md. Humaun Kabir, Mahmud Hossain, and Kyung Kwak. "The Internet of Things for Health Care: A Comprehensive Survey". In: *IEEE Access* 3 (June 2015), pp. 678–708. DOI: 10.1109/ACCESS.2015.2437951.

[144] Seetharaman Jeganathan. "DevSecOps: A Systemic Approach for Secure Software Development." In: *ISSA Journal* 17.11 (2019).

[145] Joint Task Force. *Security and Privacy Controls for Information Systems and Organizations*. Tech. rep. SP 800-53 Rev. 5. Gaithersburg, MD: National Institute of Standards and Technology, Sept. 2020. DOI: 10.6028/NIST.SP.800-53r5.

[146] David Joseph, Rafael Misoczki, Marc Manzano, Joe Tricot, Fernando Dominguez Pinuaga, Olivier Lacombe, Stefan Leichenauer, Jack Hidary, Phil Venables, and Royal Hansen. "Transitioning Organizations to Post-Quantum Cryptography". In: *Nature* 605 (2022), pp. 237–243. DOI: 10.1038/s41586-022-04623-2.

[147] Sanaa Kaddoura, Ramzi A Haraty, Sultan Al Jahdali, and Maram Assi. "SDODV: A smart and adaptive on-demand distance vector routing protocol for MANETs". In: *Peer-to-Peer Networking and Applications* 16.5 (2023), pp. 2325–2348.

[148] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. "pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4". In: *IACR Cryptology ePrint Archive*. 2019/844. 2019. URL: https://eprint.iacr.org/2019/844.

[149] Jordan Kenyon and Taylor Brady. "5 Steps for Implementing the New Post-Quantum Cryptography Standards". In: *InformationWeek* (Sept. 2024). Accessed: 2025-04-13. URL: https://www.informationweek.com/cyber-resilience/5-steps-for-implementing-the-new-post-quantum-cryptography-standards.

[150] Angelos D Keromytis. "Buffer overflow attacks". In: *Encyclopedia of Cryptography, Security and Privacy*. Springer, 2024, pp. 1–4.

[151] Habib Ullah Khan, Rafiq Ahmad Khan, Hathal S Alwageed, Alaa Omran Almagrabi, Sarra Ayouni, and Mohamed Maddeh. "AI-driven cybersecurity framework for software development based on the ANN-ISM paradigm". In: *Scientific Reports* 15.1 (2025), p. 13423.

[152] Manju Khari, Prabhat Kumar, et al. "Embedding security in software development life cycle (SDLC)". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE. 2016, pp. 2182–2186.

[153] Rashid Hussain Khokhar, Windhya Rankothge, Leila Rashidi, Hesamodin Mohammadian, Ali Ghorbani, Brian Frei, Shawn Ellis, and Iago Freitas. "A Survey on Supply Chain Management: Exploring Physical and Cyber Security Challenges, Threats, Critical Applications, and Innovative Technologies".

In: *International Journal of Supply and Operations Management* 11.3 (2024), pp. 250–283.

[154] Oratile Khutsoane, Bassey Isong, and Adnan M Abu-Mahfouz. "IoT devices and applications based on LoRa/LoRaWAN". In: *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE. 2017, pp. 6107–6112.

[155] Gene Kim, Patrick Debois, John Willis, and Jez Humble. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR: IT Revolution Press, 2016. ISBN: 978-1942788003.

[156] Youngho Kim, Seon-Gyoung Sohn, Hae Sook Jeon, Sang-Min Lee, Yunkyung Lee, and Jeongnyeo Kim. "Exploring Effective Zero Trust Architecture for Defense Cybersecurity: A Study". In: *KSII Transactions on Internet and Information Systems (TIIS)* 18.9 (2024), pp. 2665–2691.

[157] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *Proceedings of the 5th International Conference on Learning Representations (ICLR)*. 2017. URL: https://arxiv.org/abs/1609.02907.

[158] Christopher P Kohlios and Thaier Hayajneh. "A comprehensive attack flow model and security analysis for Wi-Fi and WPA3". In: *Electronics* 7.11 (2018), p. 284.

[159] Anne Kohnke, Ken Sigler, and Dan Shoemaker. "Strategic risk management using the NIST risk management framework". In: *EDPACS* 53.5 (2016), pp. 1–6.

[160] Georgios Kokolakis, Athanasios Moschos, and Angelos D Keromytis. "Harnessing the power of general-purpose llms in hardware trojan design". In: *Inter-*

national Conference on Applied Cryptography and Network Security. Springer. 2024, pp. 176–194.

[161] Subha Koley and Prasun Ghosal. "Addressing Hardware Security Challenges in Internet of Things: Recent Trends and Possible Solutions". In: *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. 2015, pp. 517–520. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.105.

[162] Alexander Kott, Ananthram Swami, and Bruce J. West. "The Internet of Battle Things". In: *Computer* 49.12 (2016), pp. 70–75. DOI: 10.1109/MC.2016.355.

[163] KPMG International. *Quantum is coming — and bringing new cybersecurity threats with it*. https://kpmg.com/dp/en/home/insights/2024/04/quantum-and-cybersecurity.html. 2024.

[164] Evans Bayu Kristanto, Septi Andrayana, and Benramhman Benramhman. "Application of Waterfall SDLC Method in Designing Student's Web Blog Information System at the National University". In: *Jurnal Mantik* 4.1 (2020), pp. 472–482.

[165] Kavita S. Kumavat and Joanne Gomes. "Performance Evaluation of IoT-enabled WSN system With and Without DDoS Attack". In: *2023 International Conference for Advancement in Technology (ICONAT)*. 2023, pp. 1–5. DOI: 10.1109/ICONAT57137.2023.10080467.

[166] J. Richard Landis and Gary G. Koch. "The Measurement of Observer Agreement for Categorical Data". In: *Biometrics* 33.1 (1977), pp. 159–174. DOI: 10.2307/2529310.

[167] Arash Habibi Lashkari, Mir Mohammad Seyed Danesh, and Behrang Samadi. "A survey on wireless security protocols (WEP, WPA and WPA2/802.11 i)". In: (2009), pp. 48–52.

[168] Alexandru Lavric, Adrian I Petrariu, and Valentin Popa. "Long range sigfox communication protocol scalability analysis under large-scale, high-density conditions". In: *IEEE Access* 7 (2019), pp. 35816–35825.

[169] Duy-Phuc Le, Ngoc-Thanh Dao, and Dinh-Hoa Nguyen. "A systematic literature review on zero trust architecture". In: *International Journal of Information and Education Technology* 12.12 (2022).

[170] Min-gyu Lee, Hyo-jung Sohn, Baek-min Seong, and Jong-bae Kim. "Secure Software Development Lifecycle which supplements security weakness for CC certification". In: *International Information Institute (Tokyo). Information* 19.1 (2016), p. 297.

[171] Mark Levene, Tameem Adel, Moulham Alsuleman, Indhu George, Padmini Krishnadas, Keith Lines, Yuhui Luo, Ian Smith, and Paul Duncan. *A Life Cycle for Trustworthy and Safe Artificial Intelligence Systems*. Tech. rep. NPL Report MS 57. National Physical Laboratory, Apr. 2024. URL: `https://eprintspublications.npl.co.uk/9995/`.

[172] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 9459–9474.

[173] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous Control with Deep Reinforcement Learning". In: *Proceedings of the 4th International*

Conference on Learning Representations (ICLR). 2016. URL: https://arxiv.org/abs/1509.02971.

[174] Steve Lipner. "The Trustworthy Computing Security Development Lifecycle". In: *20th Annual Computer Security Applications Conference (ACSAC 2004)*. IEEE, 2004, pp. 2–13. DOI: 10.1109/CSAC.2004.41.

[175] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[176] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation-Based Anomaly Detection". In: *ACM Transactions on Knowledge Discovery from Data* 6.1 (2012), pp. 1–39. DOI: 10.1145/2133360.2133363.

[177] Tao Liu, Gowri Ramachandran, and Raja Jurdak. "Post-Quantum Cryptography for Internet of Things: A Survey on Performance and Optimization". In: *arXiv preprint arXiv:2401.17538* (2024).

[178] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4765–4774.

[179] Scott M. Lundberg, Gabriel G. Erion, and Su-In Lee. "Consistent Individualized Feature Attribution for Tree Ensembles". In: *arXiv preprint arXiv:1802.03888* (2018). URL: https://arxiv.org/abs/1802.03888.

[180] Gordon Fyodor Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA: Nmap Project / Insecure.Com LLC, 2009. ISBN: 978-0-9799587-1-7. URL: https://nmap.org/book/.

[181] Vadim Lyubashevsky. "Basic Lattice Cryptography: The Concepts Behind Kyber (ML-KEM) and Dilithium (ML-DSA)". In: *IACR Cryptology ePrint Archive* 2024/1287 (2024).

[182] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations (ICLR)*. 2018.

[183] Omar Mahdi Eldood Mahdi and Julia Juremi. "EFTS: An encryption file transfer system applying advanced encryption standard (AES) algorithm". In: *AIP Conference Proceedings*. Vol. 2802. 1. AIP Publishing. 2024.

[184] Siffat Mahmood, Mahmood Niazi, and Mohammad Alshayeb. "Security in DevOps: a systematic mapping study". In: *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. 2023, pp. 145–156.

[185] Soumi Majumder and Nilanjan Dey. "Risk Management Procedures". In: *A Notion of Enterprise Risk Management: Enhancing Strategies and Wellbeing Programs*. Emerald Publishing Limited, 2024, pp. 25–40.

[186] James Martin. *Rapid Application Development*. New York, NY: Macmillan Publishing, 1991.

[187] Jillian Mascelli and Megan Rodden. ""Harvest Now Decrypt Later": Examining Post-Quantum Cryptography and the Data Privacy Risks for Distributed Ledger Networks". In: *Finance and Economics Discussion Series* 2025-093 (Sept. 2025). DOI: 10.17016/FEDS.2025.093.

[188] Charlie McCarthy, Kevin Harnett, et al. *National institute of standards and technology (nist) cybersecurity risk management framework applied to modern*

vehicles. Tech. rep. United States. Department of Transportation. National Highway Traffic Safety, 2014.

[189] Steve McConnell. *Code Complete: A Practical Handbook of Software Construction*. 2nd. Redmond, WA: Microsoft Press, 2004. ISBN: 978-0735619678.

[190] Robert J. McEliece. "A Public-Key Cryptosystem Based on Algebraic Coding Theory". In: *Deep Space Network Progress Report* 44 (1978), pp. 114–116.

[191] Lei Meng, Daochao Huang, Jiahang An, Xianwei Zhou, and Fuhong Lin. "A continuous authentication protocol without trust authority for zero trust architecture". In: *China Communications* 19.8 (2022), pp. 198–213.

[192] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[193] Nabil M Mohammed, Mahmood Niazi, Mohammad Alshayeb, and Sajjad Mahmood. "Exploring software security approaches in software development lifecycle: A systematic mapping study". In: *Computer Standards & Interfaces* 50 (2017), pp. 107–115.

[194] Bhabendu Kumar Mohanta, Debasish Jena, Utkalika Satapathy, and Srikanta Patnaik. "Survey on IoT Security: Challenges and Solution Using Machine Learning, Artificial Intelligence and Blockchain Technology". In: *Internet of Things* 11 (2020), p. 100227. DOI: 10.1016/j.iot.2020.100227.

[195] Dustin Moody et al. "Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process". In: *US Department of Commerce, NIST* (2020). URL: https://doi.org/10.6028/NIST.IR.8309.

[196] Dustin Moody, Ray Perlner, Andrew Regenscheid, Angela Robinson, and David Cooper. *Transition to Post-Quantum Cryptography Standards*. Tech. rep. NIST IR 8547. National Institute of Standards and Technology, Nov. 2024. DOI: 10.6028/NIST.IR.8547.ipd.

[197] Ramaravind K. Mothilal, Amit Sharma, and Chenhao Tan. "Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations". In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (FAccT)*. ACM, 2020, pp. 607–617. DOI: 10.1145/3351095.3372850.

[198] Geoff Mulligan. "The 6LoWPAN architecture". In: *Proceedings of the 4th workshop on Embedded networked sensors*. 2007, pp. 78–82.

[199] Nitin Naik. "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP". In: (2017), pp. 1–7. DOI: 10.1109/SysEng.2017.8088251.

[200] Ben Nassi, Dudi Nassi, Raz Ben-Netanel, Yisroel Mirsky, Oleg Drokin, and Yuval Elovici. "Phantom of the adas: Phantom attacks on driver-assistance systems". In: *Cryptology ePrint Archive* (2020).

[201] National Academies of Sciences, Engineering, and Medicine. *Quantum Computing: Progress and Prospects*. Washington, DC: The National Academies Press, 2019. DOI: 10.17226/25196.

[202] National Cybersecurity Center of Excellence. *Migration to Post-Quantum Cryptography*. Tech. rep. NIST SP 1800-38B (Preliminary Draft). National Institute of Standards and Technology, 2024.

[203] National Institute of Standards and Technology. *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*. Tech. rep. NIST AI 100-1. Gaithersburg, MD: National Institute of Standards and Technology, 2023. DOI: 10.6028/NIST.AI.100-1.

[204] National Institute of Standards and Technology. *Considerations for Achieving Cryptographic Agility*. Tech. rep. NIST CSWP 39. National Institute of Standards and Technology, 2024.

[205] National Institute of Standards and Technology. *Module-Lattice-Based Digital Signature Standard*. Tech. rep. FIPS 204. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.204.

[206] National Institute of Standards and Technology. *Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Tech. rep. FIPS 203. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.203.

[207] National Institute of Standards and Technology. *NIST Selects HQC as Fifth Algorithm for Post-Quantum Encryption*. NIST News. Mar. 2025. URL: https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption.

[208] National Institute of Standards and Technology. *NVD API 2.0 Documentation*. National Vulnerability Database, U.S. Department of Commerce. Provides programmatic access to CVE records including CVSS v3.x scoring vectors and CWE classifications. 2023. URL: https://nvd.nist.gov/developers/vulnerabilities.

[209] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization*. Tech. rep. NIST Computer Security Resource Center, 2024. URL: https://csrc.nist.gov/projects/post-quantum-cryptography.

[210] National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*. Tech. rep. NIST Special Publication 800-38E. U.S. Department of Commerce, 2010. DOI: 10.6028/NIST.SP.800-38E.

[211] National Institute of Standards and Technology. *Risk Management Framework (RMF) Small Enterprise Quick Start Guide*. Tech. rep. National Institute of Standards and Technology, July 2024. URL: https://csrc.nist.gov/news/2024/introducing-rmf-small-enterprise-quick-start-guide.

[212] National Institute of Standards and Technology. *Stateless Hash-Based Digital Signature Standard*. Tech. rep. FIPS 205. U.S. Department of Commerce, Aug. 2024. DOI: 10.6028/NIST.FIPS.205.

[213] National Security Agency. *Advancing Zero Trust Maturity Throughout the Automation and Orchestration Pillar*. NSA Cybersecurity Information Sheet, version 1.0. July 2024. URL: https://media.defense.gov/2024/Jul/10/2003500250/-1/-1/0/CSI-ZT-AUTOMATION-ORCHESTRATION-PILLAR.PDF.

[214] National Security Agency. *Post-Quantum Cybersecurity Resources*. https://www.nsa.gov/Cybersecurity/Post-Quantum-Cybersecurity-Resources/.

[215] National Security Agency. *Zero Trust Implementation Guideline Discovery Phase*. Tech. rep. U/OO/103058-26. Version 1.0. PP-25-3989. NSA Cybersecurity Directorate, Jan. 2026. URL: https://media.defense.gov/2026/Jan/08/2003852321/-1/-1/0/CTR_ZIG_DISCOVERY_PHASE.PDF.

[216] National Security Agency. *Zero Trust Implementation Guideline Phase One*. Tech. rep. U/OO/107297-26. Version 1.0. PP-25-4750. NSA Cybersecurity Directorate, Jan. 2026. URL: https://media.defense.gov/2026/Jan/30/2003868308/-1/-1/0/CTR_ZIG_PHASE_ONE.PDF.

[217] National Security Agency. *Zero Trust Implementation Guideline Phase Two*. Tech. rep. U/OO/107298-26. Version 1.0. PP-25-4758. NSA Cybersecurity Directorate, Jan. 2026. URL: https://media.defense.gov/2026/Jan/30/2003868302/-1/-1/0/CTR_ZIG_PHASE_TWO.PDF.

[218] National Security Agency. *Zero Trust Implementation Guideline Primer*. Tech. rep. U/OO/102936-26. Version 1.0. PP-25-3613. NSA Cybersecurity Directorate, Jan. 2026.

[219] National Security Agency (NSA). *Frequently Asked Questions: Quantum Computing and Post-Quantum Cryptography*. Report. 2021. URL: https://media.defense.gov/2021/Aug/04/2002821837/-1/-1/1/Quantum_FAQs_20210804.PDF.

[220] National Telecommunications and Information Administration. *The Minimum Elements For a Software Bill of Materials (SBOM)*. Tech. rep. U.S. Department of Commerce, National Telecommunications and Information Administration, July 2021. URL: https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf.

[221] H. Nejatollahi et al. "Post-Quantum Lattice-Based Cryptography Implementations: A Survey". In: *ACM Computing Surveys* 51.6 (2019), 129:1–129:41.

[222] Nataliia Neshenko, Elias Bou-Harb, Jorge Crichigno, Georges Kaddoum, and Nasir Ghani. "Demystifying IoT Security: An Exhaustive Survey on IoT Vulnerabilities and a First Empirical Look on Internet-Scale IoT Exploitations". In: *IEEE Communications Surveys and Tutorials* 21 (Apr. 2019). DOI: 10.1109/COMST.2019.2910750.

[223] NIST. *FIPS PUB 186-5: Digital Signature Standard (DSS)*. 2023. URL: https://doi.org/10.6028/NIST.FIPS.186-5.

[224] NIST. *NIST Selects HQC as Fifth Algorithm for Post-Quantum Encryption*. https://www.nist.gov/news-events/news/2025/03/nist-selects-hqc-fifth-algorithm-post-quantum-encryption. 2025.

[225] NIST. *NIST Special Publication 800-57, Recommendation for Key Management Part 1: General (Revision 3)*. 2007. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-57p1r3.pdf.

[226] NIST. *SP 800-175A, Guideline for Using Cryptographic Standards in the Federal Government: Directives, Mandates and Policies*. https://csrc.nist.gov/pubs/sp/800/175/a/final.

[227] NIST. *SP 800-175B Rev. 1, Guideline for Using Cryptographic Standards in the Federal Government*. https://csrc.nist.gov/pubs/sp/800/175/b/r1/final.

[228] NIST. *Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process*. 2019. URL: https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.

[229] Haitham Ameen Noman and Osama MF Abu-Sharkh. "Code injection attacks in wireless-based Internet of Things (IoT): A comprehensive review and practical implementations". In: *Sensors* 23.13 (2023), p. 6067.

[230] JP Norair. "Introduction to DASH7 technologies". In: *Dash7 alliance low power RF technical overview* (2009), pp. 1–22.

[231] Nurkhamid Nurkhamid, Husniza Hussin, and Habsah Zulhalim. "AI-driven identity and access management (IAM): The future of zero trust security". In: *World Journal of Advanced Research and Reviews* 21.2 (2024), pp. 1138–1144.

[232] NVIDIA Corporation. *NVIDIA JetPack SDK Documentation*. https://developer.nvidia.com/embedded/jetpack. Accessed: December 2024. Factory-default JetPack images ship with the user account "nvidia" and default password "nvidia". 2024.

[233] NVIDIA Corporation. *NVIDIA Jetson Orin Developer Kit*. https://developer.nvidia.com/embedded/jetson-orin. Accessed: 2025. 2023.

[234] Office of Management and Budget. *Memorandum M-22-09: Moving the U.S. Government Toward Zero Trust Cybersecurity Principles*. OMB Memorandum, issued January 26, 2022. Jan. 2022. URL: https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf.

[235] Oluwaseyi Ezekiel Olorunshola and Francisca Nonyelum Ogwueleka. "Review of system development life cycle (SDLC) models for effective application delivery". In: *Information and Communication Technology for Competitive Strategies (ICTCS 2020) ICT: Applications and Social Interfaces*. Springer. 2022, pp. 281–289.

[236] OWASP Foundation. *OWASP Software Assurance Maturity Model (SAMM) Version 2.0*. Tech. rep. OWASP Foundation, 2020. URL: https://owaspsamm.org/model/.

[237] Kaveh Pahlavan and Prashant Krishnamurthy. "Evolution and impact of Wi-Fi technology and applications: A historical perspective". In: *International Journal of Wireless Information Networks* 28 (2021), pp. 3–19.

[238] Zhixin Pan and Prabhat Mishra. "Design of AI Trojans for Evading Machine Learning-based Detection of Hardware Trojans". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2022, pp. 682–687. DOI: 10.23919/DATE54114.2022.9774654.

[239] Ziyue Pan, Wenbo Shen, Xingkai Wang, Yutian Yang, Rui Chang, Yao Liu, Chengwei Liu, Yang Liu, and Kui Ren. "Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines". In: *IEEE Transactions on Dependable and Secure Computing* (2023).

[240] Pankaj Pandey and Sokratis Katsikas. "The future of cyber risk management: AI and DLT for automated cyber risk modelling, decision making, and risk

transfer". In: *Handbook of Research on Artificial Intelligence, Innovation and Entrepreneurship*. Edward Elgar Publishing, 2023, pp. 272–290.

[241] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. "Practical Black-Box Attacks against Machine Learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519. DOI: 10.1145/3052973.3053009.

[242] Christian Paquin, Douglas Stebila, and Goutam Tamvada. "Benchmarking Post-Quantum Cryptography in TLS". In: *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020*. Springer. 2020, pp. 72–91. DOI: 10.1007/978-3-030-44223-1_5.

[243] Shravan Pargaonkar. "A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering". In: *International Journal of Scientific and Research Publications (IJSRP)* 13.08 (2023), pp. 345–358.

[244] R. Peralta, R. Perlner, A. Robinson, H. Silberg, D. Smith-Tone, and N. Waller. *Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process*. Tech. rep. NIST IR 8528. National Institute of Standards and Technology, 2024. DOI: 10.6028/NIST.IR.8528.

[245] Pablo Pico-Valencia, Juan A. Holgado-Terriza, and Xavier Quiñónez-Ku. "A Brief Survey of the Main Internet-Based Approaches. An Outlook from the Internet of Things Perspective". In: (2020), pp. 536–542. DOI: 10.1109/ICICT50521.2020.00091.

[246] DD Piromalis, KG Arvanitis, and N Sigrimis. "DASH7 mode 2: A promising perspective for wireless agriculture". In: *IFAC Proceedings Volumes* 46.18 (2013), pp. 127–132.

[247] Carlos Polop. *PEASS-ng: Privilege Escalation Awesome Scripts SUITE (LinPEAS, WinPEAS)*. https://github.com/peass-ng/PEASS-ng. 2024.

[248] Thomas Pornin. *Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*. Tech. rep. RFC 6979. Internet Engineering Task Force, 2013. DOI: 10.17487/RFC6979. URL: https://www.rfc-editor.org/rfc/rfc6979.

[249] Oriol Portell Pareras. "DevSecOps: S-SDLC". B.S. thesis. Universitat Politècnica de Catalunya, 2023.

[250] Derry Pratama, Jaegeun Moon, Agus Mahardika Ari Laksmono, Dongwook Yun, Muhammad Iqbal, Byeonguk Jeong, Jang Hyun Ji, and Howon Kim. "Behind The Wings: The Case of Reverse Engineering and Drone Hijacking in DJI Enhanced Wi-Fi Protocol". In: *2024 International Conference on Platform Technology and Service (PlatCon)*. IEEE. 2024, pp. 127–132.

[251] PwC. *Quantum Computing Cybersecurity Risk*. Online. Accessed 2024. 2024. URL: https://www.pwc.com/us/en/services/consulting/cybersecurity-risk-regulatory/library/quantum-computing-cybersecurity-risk.html.

[252] M Tanjidur Rahman, Qihang Shi, Shahin Tajik, Haoting Shen, Damon L. Woodard, Mark Tehranipoor, and Navid Asadizanjani. "Physical Inspection & Attacks: New Frontier in Hardware Security". In: *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*. 2018, pp. 93–102. DOI: 10.1109/IVSW.2018.8494856.

[253] Mizanur Rahman and Josef Noll. "Realizing security as code for DevSecOps". In: *Journal of Sensor and Actuator Networks* 10.3 (2021), p. 44.

[254] Inioluwa Deborah Raji, I Elizabeth Kumar, Aaron Horowitz, and Andrew Selbst. "The fallacy of AI functionality". In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 2022, pp. 959–972.

[255] Rapid7. *Metasploit Framework*. `https://github.com/rapid7/metasploit-framework`. BSD-style license. 2024.

[256] Rapeepat Ratasuk, Benny Vejlgaard, Nitin Mangalvedhe, and Amitava Ghosh. "NB-IoT system for M2M communication". In: (2016), pp. 1–5. DOI: `10.1109/WCNC.2016.7564708`.

[257] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. "Generic Side-Channel Attacks on CCA-Secure Lattice-Based PKE and KEMs". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems*. Vol. 2020. 3. 2020, pp. 307–335. DOI: `10.13154/tches.v2020.i3.307-335`.

[258] N Madhusudhana Reddy, Anil Kumar Budati, Shayla Islam, and Gajula Ramesh. "Enhanced elliptic curve-diffie hellman technique with bigdata analytics for satellite image security enhancement in internet of things systems". In: *Earth Science Informatics* 17.1 (2024), pp. 711–723.

[259] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: *Journal of the ACM* 56.6 (2009), pp. 1–40. DOI: `10.1145/1568318.1568324`.

[260] Derek Reimanis. "Risk Management Framework". In: *Realizing Complex Integrated Systems*. CRC Press, 2025, pp. 367–382.

[261] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for

Computational Linguistics, 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.

[262] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. Tech. rep. RFC 8446. Internet Engineering Task Force (IETF), Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/rfc/rfc8446.

[263] RFC Editor. *RFC 9212: Commercial National Security Algorithm (CNSA) Suite Cryptography for Secure Shell (SSH)*. https://www.rfc-editor.org/rfc/rfc9212.html.

[264] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1135–1144.

[265] Erick Rios, Ahmad Al-Kaswan, and Hong Sun. "Automating security analysis of infrastructure as code". In: *Proceedings of the 13th International Conference on Availability, Reliability and Security*. 2018, pp. 1–10.

[266] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. *Zero Trust Architecture*. Tech. rep. 800-207. National Institute of Standards and Technology, 2020. DOI: 10.6028/NIST.SP.800-207. URL: https://doi.org/10.6028/NIST.SP.800-207.

[267] Winston W. Royce. "Managing the Development of Large Software Systems". In: *Proceedings of IEEE WESCON*. IEEE, 1970, pp. 1–9.

[268] RTCA, Inc. *Software Considerations in Airborne Systems and Equipment Certification (DO-178C)*. Tech. rep. DO-178C. Washington, DC: RTCA, Inc., 2011.

[269] Stephen Russell and Tarek Abdelzaher. "The Internet of Battlefield Things: The Next Generation of Command, Control, Communications and Intelligence (C3I) Decision-Making". In: (2018), pp. 737–742. DOI: 10.1109/MILCOM.2018.8599853.

[270] Fatemeh Safari, Herb Kunze, Jason Ernst, and Daniel Gillis. "A novel cross-layer adaptive fuzzy-based ad hoc on-demand distance vector routing protocol for MANETs". In: *IEEE Access* 11 (2023), pp. 50805–50822.

[271] Mohamed Salah, Mohamed Al-Kuwaiti, Mohammed Al-Mulla, and Hitham El-Sayed. "Threat modeling for AI/ML systems: The a-z of threat modeling". In: *IEEE Security & Privacy* 21.5 (2023), pp. 6–14.

[272] FI Sapundzhi. "Home automation based on Z-wave technology". In: *Bulgarian Chemical Communications* 54 (2022).

[273] Antonios Saravanos and Matthew X Curinga. "Simulating the software development lifecycle: The Waterfall model". In: *Applied System Innovation* 6.6 (2023), p. 108.

[274] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (2017). URL: https://arxiv.org/abs/1707.06347.

[275] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. Accessed: March 2026. 2020. URL: https://scrumguides.org/scrum-guide.html.

[276] Reva Schwartz. "Informing an Artificial Intelligence Risk Aware Culture with the NIST AI Risk Management Framework". In: *American Bar Association Reflections* (2024). URL: https://www.nist.gov/publications/informing-artificial-intelligence-risk-aware-culture-nist-ai-risk-management-framework.

[277] Abderrezzak Sebbah and Kadri Benamar. "A Privacy-Enhanced Scheme Within The Public Key Infrastructure For The Internet Of Things, Employing Elliptic Curve Diffie-Hellman (ECDH)". In: *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)* 12.1 (2024), pp. 65–74.

[278] SeeTree. *About Us*. https://www.seetree.ai/about-seetree. Accessed: September 2023. 2017.

[279] Ori Shachar, Michael Yushchuk, and Guy Salton-Morgenstern. "Recurrent pattern image classification and registration". Jan. 2020. URL: https://patents.justia.com/patent/10546216.

[280] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* 5 (2017), pp. 3909–3943. DOI: 10.1109/ACCESS.2017.2685629.

[281] Stanley Shaw. *Report on Stack-Based Buffer Overflows*. Tech. rep. CyberStan, Feb. 2024. URL: https://cyberstan.co.uk/wp-content/uploads/2024/10/CSAO_coursework-2.pdf.

[282] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011.

[283] Yuxin Shi, Xinjin Lu, Kang An, Yusheng Li, and Gan Zheng. "Efficient index-modulation-based FHSS: A unified anti-jamming perspective". In: *IEEE Internet of Things Journal* 11.2 (2023), pp. 3458–3472.

[284] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press. Los Alamitos, CA, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[285] S Shylesh. "A study of software development life cycle process models". In: *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*. 2017, pp. 534–541.

[286] Muhammad Nasir Siddiqui, Kaleem Razzaq Malik, and Tauqeer Safdar Malik. "Performance Analysis of Blackhole and Wormhole Attack in MANET Based IoT". In: *2021 International Conference on Digital Futures and Transformative Technologies (ICoDT2)*. 2021, pp. 1–8. DOI: 10.1109/ICoDT252288.2021.9441515.

[287] William R Simpson. "Zero trust philosophy versus architecture". In: *World Congress on Engineering*. 2022.

[288] Raven Sims. *Implementing a Zero Trust Architecture For ICS/SCADA Systems*. Master's Thesis. Masters Theses & Doctoral Dissertations, 445. 2024.

[289] Dhananjay Singh, Gaurav Tripathi, and Antonio J. Jara. "A survey of Internet-of-Things: Future vision, architecture, challenges and services". In: (2014), pp. 287–292. DOI: 10.1109/WF-IoT.2014.6803174.

[290] Somnath Sinha and Kruthi. G. "Network layer DoS Attack on IoT System and location identification of the attacker". In: *2021 Third International Conference on Inventive Research in Computing Applications (ICIRCA)*. 2021, pp. 22–27. DOI: 10.1109/ICIRCA51532.2021.9545071.

[291] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. "Industrial Internet of Things: Challenges, Opportunities, and Directions". In: *IEEE Transactions on Industrial Informatics* 14.11 (2018), pp. 4724–4734. DOI: 10.1109/TII.2018.2852491.

[292] R Sivapriyan, K Manisha Rao, and M Harijyothi. "Literature Review of IoT based Home Automation System". In: (2020), pp. 101–105. DOI: 10.1109/ICISC47916.2020.9171149.

[293] Yuba Raj Siwakoti, Manish Bhurtel, Danda B Rawat, Adam Oest, and RC Johnson. "Advances in IoT security: Vulnerabilities, enabled criminal services, attacks, and countermeasures". In: *IEEE Internet of Things Journal* 10.13 (2023), pp. 11224–11239.

[294] Murugiah Souppaya and Karen Scarfone. *Technical Guide to Information Security Testing and Assessment*. NIST Special Publication 800-115. Gaithersburg, MD: National Institute of Standards and Technology, Sept. 2008. DOI: 10.6028/NIST.SP.800-115. URL: https://csrc.nist.gov/pubs/sp/800/115/final.

[295] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities*. Tech. rep. SP 800-218. Gaithersburg, MD: National Institute of Standards and Technology, 2022. DOI: 10.6028/NIST.SP.800-218.

[296] SPIFFE Project. *SPIFFE and SPIRE: Production-Ready Workload Identity*. https://spiffe.io/docs/latest/spiffe-about/overview/. Cloud Native Computing Foundation. Accessed: 2025. 2022.

[297] National Institute of Standards and Technology. *Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach*. Tech. rep. NIST Special Publication (SP) 800-37r2. Gaithersburg, MD: U.S. Department of Commerce, 2018. DOI: 10.6028/NIST.SP.800-37r2.

[298] Douglas Stebila and Michele Mosca. "Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project". In: *Selected Areas in Cryptography – SAC 2016*. Springer. 2017, pp. 14–37. DOI: 10.1007/978-3-319-69453-5_2.

[299] Michal Sterbak, Pavel Segec, and Jan Jurc. "Automation of risk management processes". In: *2021 19th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE. 2021, pp. 381–386.

[300] Nenad M Stojanović, Branislav M Todorović, Vladimir B Ristić, and Ivana V Stojanović. "Direct sequence spread spectrum: History, principles and modern applications". In: *Vojnotehnički glasnik/Military Technical Courier* 72.2 (2024), pp. 790–813.

[301] Michael Stoltz. "The Road to Compliance: Executive Federal Agencies and the NIST Risk Management Framework". In: *arXiv preprint arXiv:2405.07094* (2024).

[302] Blake E. Strom, Andy Applebaum, Doug P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. *MITRE ATT&CK: Design and Philosophy*. Tech. rep. MP180360R1. The MITRE Corporation, 2018. URL: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf.

[303] Panjun Sun, Shigen Shen, Yi Wan, Zongda Wu, Zhaoxi Fang, and Xiao-zhi Gao. "A survey of iot privacy security: Architecture, technology, challenges, and trends". In: *IEEE Internet of Things Journal* (2024).

[304] Nandhini Swaminathan and David Danks. "Application of the NIST AI Risk Management Framework to Surveillance Technology". In: *arXiv preprint* (2024). arXiv: 2403.15646.

[305] Synopsys, Inc. *Building Security In Maturity Model (BSIMM14)*. Tech. rep. Synopsys, 2023. URL: https://www.bsimm.com/download/.

[306] Hamed Taherdoost. "Understanding Cybersecurity Frameworks and Information Security Standards: A Review and Comprehensive Overview". In: *Electronics* 12.14 (2023), p. 3030. DOI: 10.3390/electronics12143030.

[307] Gregory Tassey. *The Economic Impacts of Inadequate Infrastructure for Software Testing*. Tech. rep. Planning Report 02-3. Gaithersburg, MD: National Institute of Standards and Technology, May 2002. URL: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf.

[308] Ekin Ecem Tatar and Murat Dener. "Wormhole Attacks in IoT Based Networks". In: *2021 6th International Conference on Computer Science and Engineering (UBMK)*. 2021, pp. 478–482. DOI: 10.1109/UBMK52708.2021.9558996.

[309] Tenable, Inc. *Nessus Professional Datasheet*. Accessed: December 2024. 2024. URL: https://www.tenable.com/products/nessus.

[310] The White House. *Executive Order 14028: Improving the Nation's Cybersecurity*. Executive Order issued May 12, 2021. May 2021. URL: https://www.govinfo.gov/app/details/DCPD-202100401.

[311] The White House. *FACT SHEET: Biden-Harris Administration Continues Work to Secure a Post-Quantum Cryptography Future*. https://bidenwhitehouse.archives.gov/ostp/news-updates/2024/08/13/fact-sheet-biden-harris-administration-continues-work-to-secure-a-post-quantum-cryptography-future/. 2024.

[312] The White House. *National Security Memorandum/NSM-8: Improving the Cybersecurity of National Security, Department of Defense, and Intelligence Community Systems*. National Security Memorandum NSM-8, issued January 19, 2022. Jan. 2022. URL: https://bidenwhitehouse.archives.gov/briefing-room/presidential-actions/2022/01/19/memorandum-on-improving-the-cybersecurity-of-national-security-department-of-defense-and-intelligence-community-systems/.

[313] Jacopo Tosi, Fabrizio Taffoni, Marco Santacatterina, Roberto Sannino, and Domenico Formica. “Performance evaluation of bluetooth low energy: A systematic review”. In: *Sensors* 17.12 (2017), p. 2898.

[314] Mengru Tsai, Shanhsin Lee, and Shiuhpyng Winston Shieh. “Strategy for implementing of zero trust architecture”. In: *IEEE Transactions on Reliability* 73.1 (2024), pp. 93–100.

[315] Bhagyashri Tushir, Yogesh Dalal, Behnam Dezfouli, and Yuhong Liu. “A Quantitative Study of DDoS and E-DDoS Attacks on WiFi Smart Home Devices”. In: *IEEE Internet of Things Journal* 8.8 (2021), pp. 6282–6292. DOI: 10.1109/JIOT.2020.3026023.

[316] Vicky Tyagi, Amar Saraswat, Ashwani Kumar, and Shalini Gambhir. “Securing IoT Devices Against MITM and DoS Attacks: An Analysis”. In: *Reshaping Intelligent Business and Industry: Convergence of AI and IoT at the Cutting Edge* (2024), pp. 237–249.

[317] Kyriaki A. Tychola and Konstantinos Rantos. “Cyberthreats and Security Measures in Drone-Assisted Agriculture”. In: *Electronics* 14.1 (2025). ISSN: 2079-9292. DOI: 10.3390/electronics14010149. URL: https://www.mdpi.com/2079-9292/14/1/149.

[318] Shahid Ul Haq, Yashwant Singh, Amit Sharma, Rahul Gupta, and Dipak Gupta. “A survey on IoT & embedded device firmware security: architecture, extraction techniques, and vulnerability analysis frameworks”. In: *Discover Internet of Things* 3.1 (2023), p. 17.

[319] Muhammad Usman, Usman Qamar, Irfan Hameed, and M Mudasir Kirmani. “Zero trust architecture: a comprehensive review”. In: *Journal of Computing and Social Informatics* 1 (2022), pp. 1–20.

[320] van Hauser and The Hacker's Choice. *THC-Hydra: A very fast network logon cracker which supports many different services*. `https://github.com/vanhauser-thc/thc-hydra`. AGPLv3 License; software originally released 2001. 2023.

[321] Mohit Kumar Verma and Rajendra Kumar Dwivedi. "A Survey on Wormhole Attack Detection and Prevention Techniques in Wireless Sensor Networks". In: *2020 International Conference on Electrical and Electronics Engineering (ICE3)*. 2020, pp. 326–331. DOI: 10.1109/ICE348803.2020.9122944.

[322] Juan de Vicente Mohino, Javier Bermejo Higuera, Juan Ramón Bermejo Higuera, and Juan Antonio Sicilia Montalvo. "The application of a new secure software development life cycle (S-SDLC) with agile methodologies". In: *Electronics* 8.11 (2019), p. 1218.

[323] Chip Walter. "Kryder's Law". In: *Scientific American* 293.2 (Aug. 2005), pp. 32–33. DOI: 10.1038/scientificamerican0805-32.

[324] Zhongxu Wan et al. "Artificial Intelligence for Cloud-Assisted Smart Factory". In: *IEEE Access* 6 (2018), pp. 55419–55430. DOI: 10.1109/ACCESS.2018.2871724.

[325] Ju Wang, Yiying Liang, Xuanyu Xu, Jinyi Wang, and Yi Zhong. "A High Dynamic Velocity Locked Loop for the Carrier Tracking of a Wide-Band Hybrid Direct Sequence/Frequency Hopping Spread-Spectrum Signal". In: *Electronics* 13.9 (2024), p. 1794.

[326] Abraham Itzhak Weinberg and Kelly Cohen. "Zero Trust Implementation in the Emerging Technologies Era: Survey". In: *arXiv preprint arXiv:2401.09575* (2024).

[327] Sean Whalen. "An introduction to arp spoofing". In: *Node99 [Online Document]* 563 (2001).

[328] William Whyte, Zhaohui Zhang, Scott Fluhrer, and Brice Knodel. "Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2)". In: *RFC Editor* (2023).

[329] Chathurika S. Wickramasinghe, Daniel L. Marino, Kasun Amarasinghe, and Milos Manic. "Generalization of Deep Learning for Cyber-Physical System Security: A Survey". In: *IEEE Access* 11 (2023), pp. 75174–75188. DOI: 10.1109/ACCESS.2023.3295882.

[330] Wireshark Foundation and Gerald Combs. *Wireshark: The world's most popular network protocol analyzer*. https://www.wireshark.org/. GNU General Public License version 2. 2024.

[331] Gang Wu, Junjun Jiang, Kui Jiang, Xianming Liu, and Liqiang Nie. "DSwinIR: Rethinking Window-Based Attention for Image Restoration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2025). DOI: 10.1109/TPAMI.2025.3646016.

[332] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, and Y. Jin. "Security analysis on consumer and industrial IoT devices". In: *Proc. 21st Asia South Pac. Design Autom. Conf. (ASP-DAC)*. 2016, pp. 519–524.

[333] Che Xu, Peng Zhu, Jiacun Wang, and Giancarlo Fortino. "Improving the local diagnostic explanations of diabetes mellitus with the ensemble of label noise filters". In: *Information Fusion* 117 (2025), p. 102928.

[334] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. "Modeling and Discovering Vulnerabilities with Code Property Graphs". In: *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 590–604. DOI: 10.1109/SP.2014.44.

[335] Qigui Yao, Qi Wang, Xiaojian Zhang, and Jiaxuan Fei. "Dynamic access control and authorization system based on zero-trust architecture". In: *Proceed-*

ings of the 2020 1st international conference on control, robotics and intelligent system. 2020, pp. 123–127.

[336] Asad Yaseen. "Reducing industrial risk with AI and automation". In: *International Journal of Intelligent Automation and Computing* 4.1 (2021), pp. 60–80.

[337] Claudio Zanasi, Silvio Russo, and Michele Colajanni. "Flexible zero trust architecture for the cybersecurity of industrial IoT infrastructures". In: *Ad Hoc Networks* 156 (2024), p. 103414.

[338] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. "Internet of Things for Smart Cities". In: *Internet of Things Journal, IEEE* 1 (Jan. 2012). DOI: 10.1109/JIOT.2014.2306328.

[339] Ramije Zeqiri, Florim Idrizi, and Halim Halimi. "Comparison of Algorithms and Technologies 2G, 3G, 4G and 5G". In: *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. IEEE. 2019, pp. 1–4.

[340] Jun Zhang, Lei Pan, Qing-Long Han, Chao Chen, Sheng Wen, and Yang Xiang. "Deep learning based attack detection for cyber-physical system cybersecurity: A survey". In: *IEEE/CAA Journal of Automatica Sinica* 9.3 (2021), pp. 377–391.

[341] Mingbo Zhang and Saman Zonouz. "Control Corruption without Firmware Infection: Stealthy Supply Chain Attacks via PLC Hardware Implants (MalTag)". In: *2024 ACM/IEEE 15th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. 2024, pp. 247–258.

[342] Rui Zhang, Zhiwei Hu, Jianjun Li, Feng Fan, and Feng Wen. "Network Time Protocol (NTP) implementation for laser inter-satellite networks". In: *Ad-*

vanced Fiber Laser Conference (AFL2023). Vol. 13104. SPIE. 2024, pp. 1707–1712.

[343] Ying Zhang and Ake Årvidsson. "Understanding the characteristics of cellular data traffic". In: *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*. 2012, pp. 13–18.

[344] Chang-Le Zhong, Zhen Zhu, and Ren-Gen Huang. "Study on the IOT Architecture and Gateway Technology". In: (2015), pp. 196–199. DOI: 10.1109/DCABES.2015.56.

[345] Weihua Zhuang, Xuemin Shen, and Qi Bi. "Ultra-wideband wireless communications". In: *Wireless communications and mobile computing* 3.6 (2003), pp. 663–685.

[346] Bahman Zohuri and Masoud Moghaddam. "Deep Learning Limitations and Flaws". In: *Modern Approaches on Material Science* 2.3 (2020), pp. 241–250.

[347] Marwah Al-Zubaidi and Azizol Abdullah. "A systematic literature review on security issues and challenges in DevSecOps". In: *IEEE Access* 11 (2023), pp. 83597–83615.