applied sciences

MDPI

Article

Enhancing Secure Software Development with AZTRM-D: An AI-Integrated Approach Combining DevSecOps, Risk Management, and Zero Trust

Ian Coston *, Karl David Hezel, Eadan Plotnizky and Mehrdad Nojoumian

Department of Electrical Engineering and Computer Science, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA; khezel2017@fau.edu (K.D.H.); eplotnizky2020@fau.edu (E.P.); mnojoumian@fau.edu (M.N.)

* Correspondence: icoston2016@fau.edu; Tel.: +1-561-297-3411

Abstract

This paper introduces the Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D) framework, a novel approach that embeds security throughout the entire Secure Software and System Development Life Cycle (S-SDLC). AZTRM-D strategically unifies three established methodologies: DevSecOps practices, the NIST Risk Management Framework (RMF), and the Zero Trust (ZT) model. It then significantly augments their capabilities through the pervasive application of Artificial Intelligence (AI). This integration shifts traditional, often fragmented, security paradigms towards a proactive, automated, and continuously adaptive security posture. AI serves as the foundational enabler, providing real-time threat intelligence, automating critical security controls, facilitating continuous vulnerability detection, and enabling dynamic policy enforcement from initial code development through operational deployment. By automating key security functions and providing continuous oversight, AZTRM-D enhances risk mitigation, reduces vulnerabilities, streamlines compliance, and significantly strengthens the overall security posture of software systems, thereby addressing the complexities of modern cyber threats and accelerating the delivery of secure software.

Keywords: DevSecOps; NIST Risk Management Framework (RMF); NIST Zero Trust (ZT); Artificial Intelligence (AI); Secure Software and System Development Life Cycle (S-SDLC); Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D)

Academic Editors: Masoud Barati and Nafiseh Kahani

Received: 16 June 2025
Revised: 14 July 2025
Accepted: 21 July 2025
Published: 22 July 2025

Citation: Coston, I.; Hezel, K.D.; Plotnizky, E.; Nojoumian, M. Enhancing Secure Software Development with AZTRM-D: An AI-Integrated Approach Combining DevSecOps, Risk Management, and Zero Trust. *Appl. Sci.* **2025**, *15*, 8163. https://doi.org/10.3390/app15158163

1. Introduction

The landscape of software development is undergoing a rapid transformation, driven by agile methodologies, cloud-native architectures, and continuous delivery pipelines. While these advancements promise increased efficiency and innovation, they concurrently introduce unprecedented security challenges. Traditional security approaches, often implemented as an afterthought or through fragmented, manual processes, are proving inadequate against an increasingly sophisticated and dynamic cyber threat environment [1,2]. The reactive nature of finding and fixing vulnerabilities late in the development cycle leads to significant costs, delays, and persistent risks [3]. There is a pressing need for a paradigm shift, one that integrates security holistically from inception, fosters continuous vigilance, and leverages intelligent automation to keep pace with evolving threats.

Current efforts to enhance software security often focus on individual components, such as adopting DevSecOps for faster integration of security, implementing Zero Trust

(ZT) principles for stricter access control, or applying risk management frameworks like the NIST Risk Management Framework (RMF) for structured security governance [4–6]. While each of these methodologies offers distinct benefits, multivocal literature reviews show that their isolated application can lead to gaps, inefficiencies, and a lack of cohesive defense [7]. The true potential for robust and adaptive software security lies not in these individual pillars, but in their synergistic integration, empowered by advanced technologies like Artificial Intelligence (AI) [8].

This paper introduces the Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D), a novel and comprehensive framework designed to address these critical challenges. AZTRM-D unifies DevSecOps, the NIST RMF, and the ZT model, all underpinned and significantly enhanced by AI. The framework fundamentally redefines secure software development by weaving security into every phase of the Secure Software and System Development Life Cycle (S-SDLC). Unlike fragmented solutions, AZTRM-D leverages AI to provide continuous threat intelligence, automate vital security controls, enable real-time system oversight, and facilitate dynamic policy adaptation. This AI-powered automation extends across the entire life cycle, from promoting secure coding practices and identifying vulnerabilities during development to continuous monitoring and threat response in operational environments. The result is a proactive, adaptive, and highly resilient framework that not only mitigates risks but also accelerates the delivery of inherently secure software, preparing organizations for the complexities of the modern digital frontier. The subsequent sections detail the architectural components of AZTRM-D, its operational methodology, and how its integrated approach fortified by AI offers a superior path to achieving enduring software security.

Organization of the Paper

This paper is structured to provide a comprehensive exploration of the AZTRM-D framework. We begin in Section 2 by briefly reviewing foundational development models to establish the context for modern security challenges. This section then presents a comparative analysis of contemporary security frameworks to clearly differentiate AZTRM-D's contribution from the existing literature.

Next, in Sections 2.4 and 3, we introduce the core principles of the NIST Risk Management Framework (RMF) and the Zero Trust (ZT) model. To improve clarity and conciseness, we use summary tables to present the key phases of the RMF and pillars of ZT, focusing on how Artificial Intelligence (AI) enhances each component within our framework.

In Section 4, we introduce our main contribution, the AZTRM-D framework. This section details each phase of the framework's life cycle: Planning, Development, Building, Testing, Release and Deliver, and Operate, Monitor, and Feedback. We explain how ZT principles, RMF governance, and DevSecOps methodologies are integrated and enhanced by AI in each phase. We also discuss how the NIST AI Risk Management Framework is used to manage the unique risks associated with the AI models themselves.

Finally, we provide validation for AZTRM-D. In Section 5, we walk through a simulated real-world scenario to demonstrate the framework's practical application. Then, in Section 6, we present a lab-built scenario using NVIDIA Orin devices to detail the implementation and results of our security testing. This includes a quantitative benchmarking analysis that measures the framework's effectiveness. We conclude in Section 7 by summarizing the importance of AZTRM-D and outlining future research directions.

2. Related Work and Foundational Concepts

To position the contribution of the AZTRM-D framework, it is essential 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 clearly differentiate AZTRM-D's novel and synergistic approach and to address existing gaps in the literature.

2.1. Software and System Development Life Cycles

The Software and System Development Life Cycle (SDLC) is a foundational framework guiding the creation and maintenance of software systems through structured phases like requirements analysis, design, implementation, and testing [1,2]. The choice of SDLC methodology significantly impacts a project's efficiency and, critically, its security posture, especially when handling sensitive or classified data [4]. As security requirements become more stringent, development methodologies must adapt to protect sensitive information effectively.

2.1.1. Waterfall Method

The Waterfall methodology follows a linear, sequential structure where each phase must be completed before the next begins [9,10]. This rigid approach is effective for projects with stable, clearly defined requirements, such as simple, unclassified systems [11,12]. While its emphasis on detailed documentation can aid in demonstrating compliance with standards like HIPAA or PCI-DSS, its inflexibility makes it poorly suited for dynamic threat environments that require continuous security reassessment and rapid adaptation [4].

2.1.2. 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 [5,11]. This adaptability is effective in environments where requirements evolve, such as in consumer IoT development [4]. However, the rapid pace of Agile development necessitates strong management and integrated security practices to prevent oversights and ensure that compliance is maintained throughout the iterative cycles [2].

2.1.3. V-Model

The V-Model, an extension of Waterfall, emphasizes verification and validation by pairing each development phase with a corresponding testing phase [13,14]. This structured approach is beneficial for projects where reliability and rigorous testing are paramount. However, like Waterfall, its rigidity can be a significant limitation, hindering rapid adaptation to new security threats or regulatory changes, making it less suitable for highly dynamic or classified systems [4].

2.1.4. 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 [15]. Its iterative cycles of planning, risk analysis, and evaluation make it well-suited for large, complex projects with high levels of uncertainty [4]. 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.

2.1.5. 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), a paradigm that integrates security practices directly into the

entire development life cycle [7,16]. By leveraging automation, continuous integration and delivery (CI/CD), and a culture of shared responsibility, DevSecOps aims to make security an intrinsic component of the software pipeline [17]. Core principles include "Security as Code," where security policies and controls are managed as versioned, auditable code, and continuous monitoring throughout the life cycle [18]. As illustrated in Figure 1, the DevSecOps life cycle 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 comprehensive, integrated frameworks.

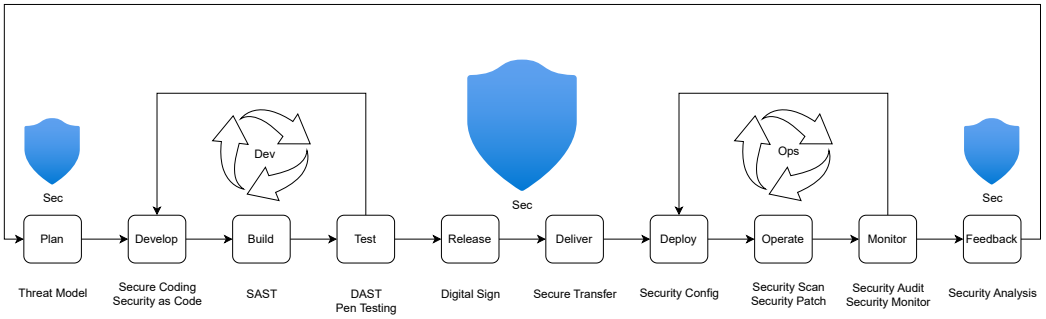

Figure 1. The DevSecOps Life Cycle, integrating security into every phase from planning to feedback. This image was inspired by [19].

2.2. Current Landscape of Secure Development Methodologies and Frameworks

The modern software development landscape constantly evolves with agile methodologies, cloud-native architectures, and continuous delivery pipelines. While these advancements boost efficiency, they also bring significant security challenges. Traditional security approaches often fall short against today's sophisticated and dynamic cyber threats because they are often implemented too late or through fragmented, manual processes [1,2]. Finding and fixing vulnerabilities late in the development cycle is costly, causes delays, and leaves systems vulnerable [3]. 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.

The Secure Software Development Life Cycle (S-SDLC) emerged as a direct response, integrating security into the traditional SDLC [20,21]. The primary goal of any S-SDLC is to embed security considerations into every stage of development, from requirements gathering to deployment and maintenance. This ensures the final product can withstand evolving threats [22]. To understand where AZTRM-D fits within this context, it is helpful to examine the existing landscape of S-SDLC methodologies and modern security frameworks.

Table 1 provides a concise overview of various prominent S-SDLC methods, detailing their key focus areas and core characteristics.

Table 1. Summary of Secure Software Development Life Cycle (S-SDLC) Methodologies.

Method	Key Focus/Approach	Core Characteristics
Embedded Security in Traditional SDLC [20]	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 1. *Cont.*

Method	Key Focus/Approach	Core Characteristics
Early and Continuous Integration [21]	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, fosters a culture of security awareness, and integrates automated, continuous security testing throughout all phases.
Agile-Adapted Security [5]	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 [23]	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 [6,24]	This method transforms the DevOps process into a holistic DevSecOps approach, ensuring security is a shared responsibility across all stages.	Key features include automated security integration into the CI/CD pipeline, fostering a collaborative culture between development, operations, and security teams, and using continuous monitoring with real-time feedback for rapid remediation.
Categorized Software Security Strategies [25]	This method is based on a comprehensive 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) [26,27]	This is a unified, AI-augmented process that builds upon existing S-SDLC methods.	It integrates proactive DevSecOps, structured NIST RMF, and stringent Zero Trust (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 strong foundation, recent academic and industry work has also focused on building more specific, robust 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 [28]. In response, researchers have proposed AI-enhanced models for specific life cycle stages. For example, some have developed machine learning approaches to automate threat modeling within DevSecOps, showing AI's potential to proactively identify design flaws [29]. Concurrently, the Zero Trust (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 [30]. Others advocate for "Zero Trust by Design," embedding its principles into the very architecture of systems [26].

Furthermore, a significant challenge, is linking security frameworks with non-traditional forecasting models that handle uncertainty. Recent work in areas like grey

system modeling offers new ways to forecast outcomes in complex, data-sparse environments, which is highly relevant to early-stage risk assessment. For instance, studies on fractional grey system models for predicting environmental emissions demonstrate methodologies for creating accurate predictive models under conditions of high uncertainty [31]. Including this perspective strengthens the theoretical foundation of AZTRM-D's predictive capabilities, connecting its AI-driven risk analysis to broader interdisciplinary paradigms.

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 paradigm for software development, which provides a valuable reference point for formalizing security processes [32]. However, a gap remains for a comprehensive framework that unifies the agile methodology of DevSecOps, the rigorous validation of Zero Trust, and the structured governance of a formal Risk Management Framework (RMF) into a single, cohesive life cycle, all orchestrated by AI. Table 2 compares these approaches and highlights AZTRM-D's unique contribution.

Table 2. Comparative Analysis of Modern Secure Development Approaches.

Framework/Approach	Core Principles	AI/Automation Focus	Gap Addressed by AZTRM-D
Standard DevSecOps [17]	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 [29]	Using ML/AI to predict and identify design-level security threats early in the life cycle.	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 life cycle, including runtime monitoring and incident response.
Zero Trust Supply Chain [30]	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.
AZTRM-D (This Work)	Synergistic 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, holistic framework that unifies agile development, structured risk governance, and adaptive access control, ensuring security is continuous, risk-based, and context-aware from planning to operations.

2.3. The Rationale for AZTRM-D's Integrated Approach

As the comparison in Table 2 illustrates, existing advancements tend to address specific facets of the secure development challenge. AZTRM-D is proposed as a comprehensive 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 is insufficient. A truly resilient security posture requires the seamless integration of several paradigms. It needs the methodological agility and automation of DevSecOps, the granular and continuous access control provided by a Zero Trust philosophy, and the structured governance of a formal Risk Management Framework (RMF) to ensure all activities are measurable and compliant. Tying these complex domains together requires an engine for scalable and adaptive orchestration, a role fulfilled by Artificial Intelligence [8]. 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 a robust and adaptive security framework that is particularly 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 enhances 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 [26,27]. 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 comprehensive approach creates a more resilient and responsive security framework that addresses the gaps identified in the existing literature.

2.4. 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 designed to be integrated into the system development life cycle, making it a natural fit for a DevSecOps approach. As shown in Figure 2, the risk management framework begins with understanding the organizational context and proceeds through system categorization, control selection, implementation, assessment, authorization, and continuous monitoring [33–35]. By integrating AI into each of these steps, AZTRM-D transforms the RMF into a dynamic and highly automated process.

2.4.1. 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 [33,36]. 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 [37]. 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 [38,39]. In AZTRM-D, AI enhances this phase by analyzing historical data to recommend role-based access controls and by monitoring organizational activities to help dynamically assign responsibilities, thereby minimizing the attack surface from the outset [27,40].

2.4.2. Categorization Phase

The Categorization Phase involves developing a comprehensive understanding of the system and its environment [33,36]. As shown in Figure 3, this requires meticulously analyzing the system layout, mapping how components communicate and exchange data, and identifying all network connections [35]. A crucial task is to identify critical systems and their interdependencies to understand the potential impact of a security incident [33,34]. The systems are then assigned impact levels (low, moderate, or high)

based on the sensitivity of the data they handle, which is a critical input for later control selection [39]. AI significantly enhances this phase by automating the mapping of system interconnections and data flows through network analysis, providing a more accurate and comprehensive view of the environment than manual efforts alone [41,42].

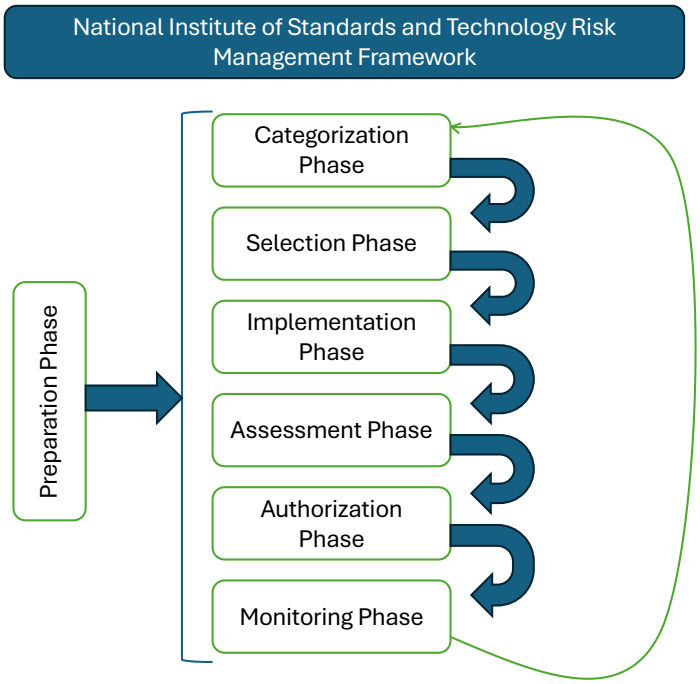

Figure 2. The greencyclical nature of the NIST Risk Management Framework (RMF), where monitoring provides continuous feedback into the preparation for the next cycle. This image was inspired by [33,36].

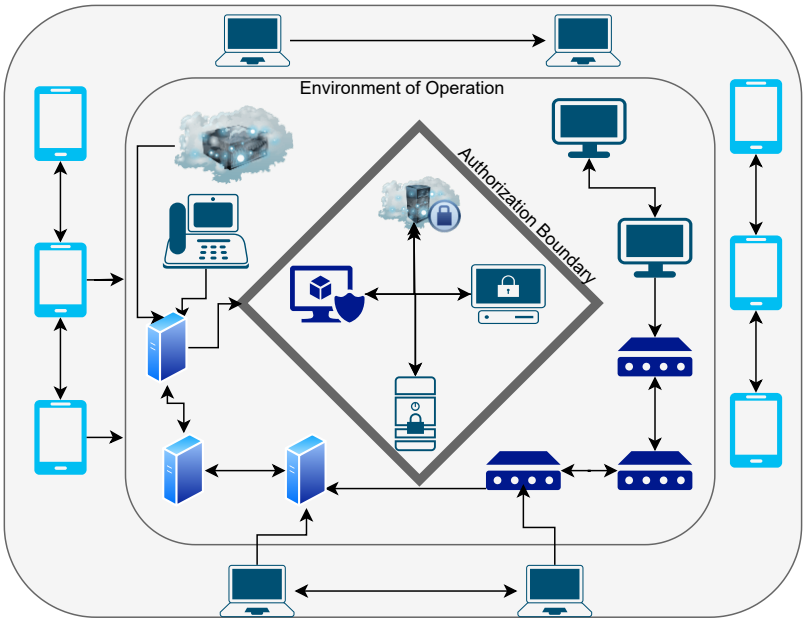

Figure 3. greenA conceptual diagram illustrating a system's Authorization Boundary, which contains critical assets, and its connections to the wider Environment of Operation. This mapping is a key activity in the RMF Categorization phase. This image was inspired by [33].

2.4.3. Selection Phase

In the Selection Phase, a baseline of security controls is established based on a comprehensive evaluation of threats, vulnerabilities, and risks [33,34]. 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 [33,36]. The identified threats and vulnerabilities are then analyzed in a risk assessment to determine their likelihood and potential impact [35,38]. Based on this analysis, appropriate security controls are selected to mitigate the identified risks [37]. AI enhances 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 [42,43].

2.4.4. Implementation Phase

The Implementation Phase is where the selected security controls are put into action, transforming the security plan into a tangible defense [33,36]. This phase requires meticulous deployment and integration of the controls into the existing infrastructure. Thorough documentation of every configuration change and policy update is crucial for tracking, troubleshooting, and compliance [37,38]. Additionally, personnel must be trained on the newly implemented controls to ensure they understand their role in maintaining security [44]. AI plays a pivotal role here by automating the deployment and configuration of controls through Infrastructure as Code (IaC) and other "Security as Code" practices, ensuring consistent application and reducing the risk of human error [45,46].

2.4.5. Assessment Phase

The Assessment Phase is a critical checkpoint to verify that the implemented security controls are effective and meeting their objectives [33,36]. This involves reassessing vulnerabilities, threats, and risks in light of the new controls [44]. 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 [33,35]. It is also crucial to evaluate the overall performance of the controls to ensure they provide adequate security without unduly hindering the system's operational capabilities [39]. AI improves this phase by automating the continuous testing and analysis of controls, simulating potential attack scenarios, and providing immediate feedback on their performance [27,41].

2.4.6. 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 [33,36]. 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 [39,44]. This decision is a critical judgment call that weighs the potential impact of a security breach against the organization's mission and risk tolerance [38]. AI can significantly improve this phase by rapidly analyzing the extensive documentation and highlighting areas of highest risk, thereby speeding up the review process and enabling a more informed and timely authorization decision, which is key to achieving cATO [43,47].

2.4.7. Monitoring Phase

The Monitoring Phase is the final, ongoing stage of the RMF, where the system's security posture is continuously monitored to ensure its effectiveness over time [33,34]. This phase involves a relentless pursuit of maintaining security by adapting to emerging threats and evolving challenges [37,39]. Continuous monitoring includes real-time surveillance

of system activity, performance analysis, intrusion detection, and regular vulnerability scanning [44]. When a potential incident is detected, a swift and coordinated response is essential to contain the damage [35]. AI provides a transformative 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 life cycle [8,48].

3. The Guiding Philosophy: Zero Trust Architecture in AZTRM-D

While the RMF provides a structured process for governance, AZTRM-D's operational security is guided by the philosophy of Zero Trust (ZT). ZT represents a paradigm shift 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" [49,50]. As defined in numerous systematic literature reviews, this model mandates that no user, device, or application is trusted by default, regardless of its location [51]. Every access request must be continuously and dynamically authenticated and authorized against a set of risk-based policies [26]. Figure 4 provides a general overview of this core concept.

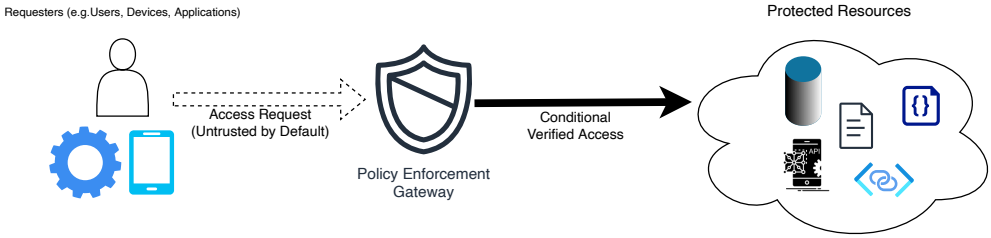

Figure 4. A general overview of the Zero Trust Architecture concept, where all access requests from users, devices, or applications are untrusted by default and must pass through a policy enforcement gateway for verification. This image was inspired by [49,52].

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/Enforcement Point (PDP/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 [53,54]. The Core ZT Model, shown in Figure 5, illustrates how real-time context feeds inform a central policy engine, which makes these dynamic trust decisions.

The ZT model is elaborated through five core pillars, which are Identity, Devices, Networks, Applications and Workloads, and Data. These are all supported by three cross-cutting capabilities: Visibility and Analytics, Automation and Orchestration, and Governance [55]. Figure 6 depicts these components. In AZTRM-D, these pillars are not just implemented but are augmented with AI to enable adaptive enforcement at scale. To concisely illustrate this, Table 3 summarizes each pillar's objective, the traditional weakness it addresses, and the specific AI-enhanced role it plays within the AZTRM-D framework. The following subsections provide further detail on each pillar and capability.

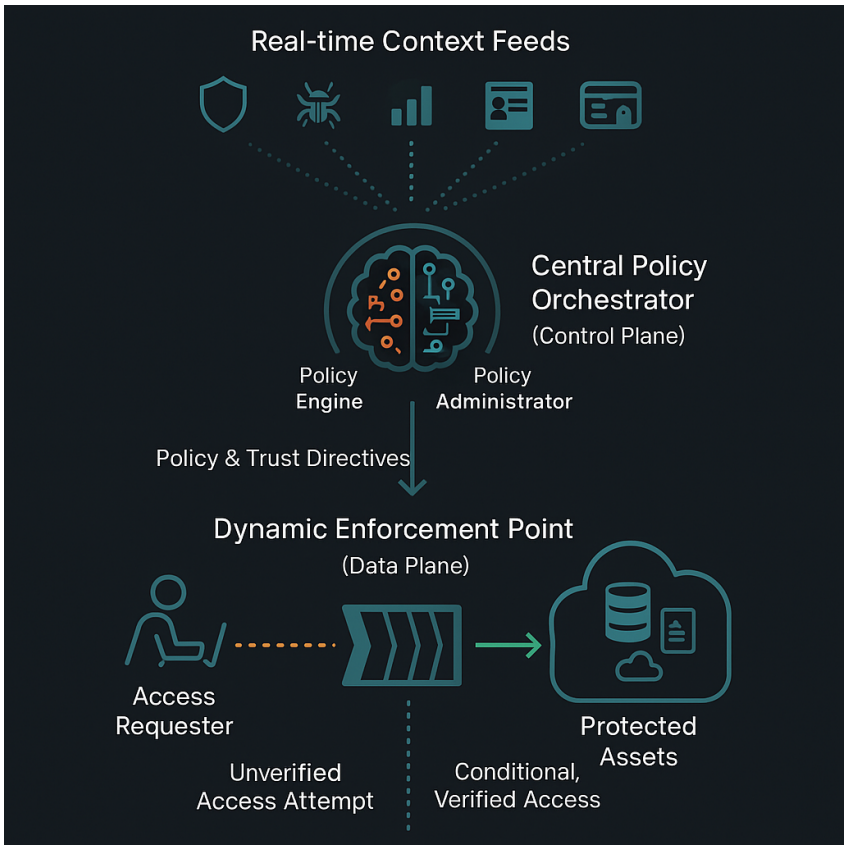

Figure 5. The Core ZT Model, showing how real-time context feeds inform a central policy engine, which makes dynamic trust decisions that are enforced at the data plane. This image was inspired by [49,53].

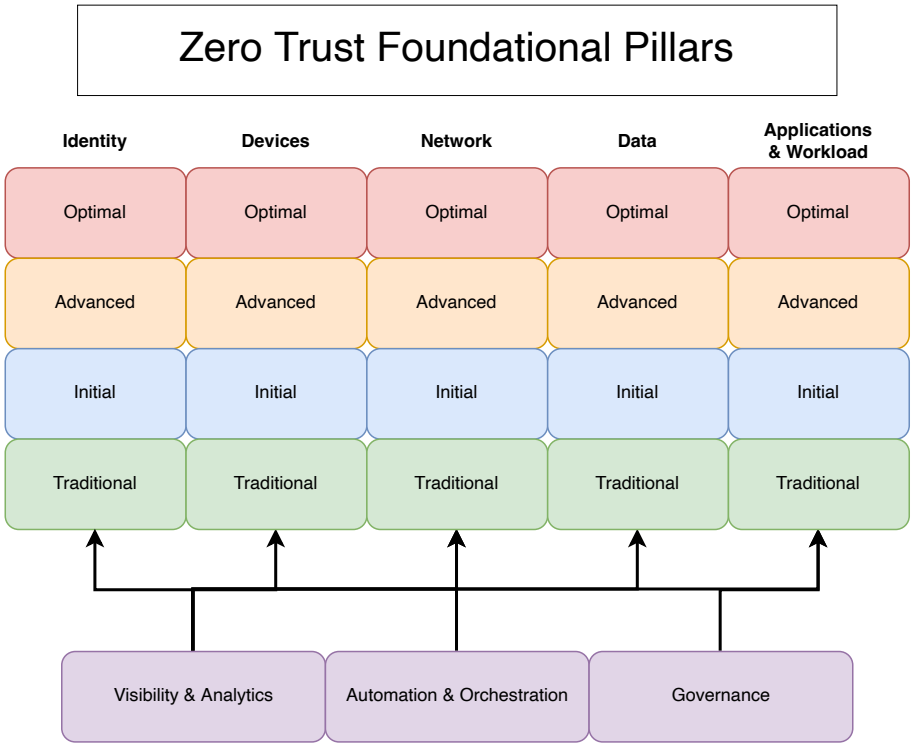

Figure 6. The five foundational pillars of Zero Trust, supported by three cross-cutting capabilities. The goal is to advance each pillar from a traditional, reactive security posture to an optimal, proactive state. This image was inspired by [50,55].

Table 3. Zero Trust Pillars and their AI Enhancement in AZTRM-D.

Pillar	Traditional Weakness	Zero Trust Objective (Optimal Maturity)	AI-Enhanced Role in AZTRM-D
Identity	Static credentials (passwords) and roles grant broad, persistent access once authenticated.	Grant access on a per-session basis based on dynamic risk assessments of the user and their context. Enforce least privilege.	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 [56].
Devices	Any device on the "trusted" network is allowed access, with only periodic checks for compliance (e.g., antivirus).	Continuously verify the security posture of every device attempting to access resources and grant access based on device health.	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 [8].
Networks	A hard outer perimeter with a soft, trusted interior ("castle-and-moat"). Attackers can move laterally with ease once inside.	Use micro-segmentation to create granular security zones. Encrypt all traffic and log every access request.	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.
Applications & Workloads	Applications are trusted by default once deployed. Security is focused on the perimeter, not the application itself.	Secure application access for all users, secure application-to-application communication, and harden workloads against runtime threats.	AI enhances 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 [29].
Data	Data is protected primarily by controlling access to the network or server where it resides.	Categorize and classify data based on sensitivity. Enforce access controls and encrypt data at rest, in transit, and in use.	AI automates data discovery and classification across all systems. It monitors data access patterns to detect and block potential exfiltration attempts before they succeed [42].
Cross-Cutting Capabilities	Manual governance, fragmented visibility, and reactive, human-driven security responses.	Achieve comprehensive visibility, automate security orchestration, and maintain dynamic governance across all pillars.	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 [8].

3.1. Identity

The Identity pillar is a cornerstone of ZT, ensuring every access request is authenticated and authorized under the principle of least privilege [49,55]. It encompasses Identity and Access Management (IAM), Multi-Factor Authentication (MFA), and continuous user behavior analytics [50]. 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 [57,58]. 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 [59,60]. This is vital for securing environments with numerous IoT devices, which often lack robust native security [61].

3.2. Devices

The Devices pillar focuses on ensuring any device accessing the network is authenticated and its security posture is continuously verified [55]. 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 [62,63]. Traditional device security relies on periodic manual checks. ZT matures this by introducing automated tools for health monitoring and patch management [55]. 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 [60].

3.3. Networks

The Networks pillar secures network infrastructure by treating all traffic as untrusted until verified [55]. The focus is on implementing micro-segmentation, end-to-end encryption, and continuous traffic monitoring to prevent unauthorized lateral movement [26]. The traditional "castle-and-moat" perimeter is inadequate against internal threats [54]. ZT begins by implementing basic segmentation and encryption, then uses AI to monitor traffic for anomalies [64]. In AZTRM-D's advanced state, AI dynamically adjusts micro-segmentation based on real-time risk assessments [65]. In its optimal state, the framework enforces just-in-time and just-enough connectivity, which is critical for IoT devices. This ensures they communicate only with authorized systems for the minimum time necessary, thereby dramatically reducing the attack surface [60].

3.4. Applications and Workloads

This pillar focuses on securing applications and workloads throughout their life cycle [55]. 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 [19]. AZTRM-D enhances this by deeply integrating AI into the DevSecOps pipeline, automating vulnerability detection and policy enforcement before deployment [42]. 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 [29].

3.5. Data

The Data pillar is central to ZT, focusing on protecting data at rest, in transit, and in use [66]. 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 [50]. AZTRM-D enhances this with AI-driven systems that provide comprehensive visibility into all data interactions [64]. In its optimal state, data access is governed by dynamic, just-in-time, and just-enough access controls informed by real-time risk analytics [58,65]. This is essential for IoT environments, where massive volumes of data are generated and processed in real time [62].

3.6. Cross-Cutting Capabilities: Governance, Automation, and Visibility

The five pillars are supported by three critical cross-cutting capabilities [50,55]. 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 [65,67]. 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 [68,69]. Finally, Visibility and Analytics progress from fragmented, manual data collection to comprehensive, real-time monitoring across the entire IT environment. In the optimal state, AI-driven systems provide predictive insights that enable an organization to stay ahead of emerging threats [48,60].

3.7. 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?". Finally, DevSecOps provides the agile and automated implementation methodology, acting as the engine to ensure the goals of both RMF and ZT are achieved continuously [7].

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 [26,70]. This holistic integration, powered by AI, ensures that security is not just a feature but the fundamental basis of the entire development life cycle, which will be detailed in the following section.

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

The Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D) is a comprehensive methodology designed to embed adaptive, risk-based security into every phase of the software life cycle. 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 operate synergistically. Figure 7 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 8 presents a conceptual architecture of how AI components are integrated as a continuous feedback and control loop within the DevSecOps workflow. The following subsections detail the specific activities and AI-driven enhancements within each phase of the AZTRM-D life cycle.

4.1. Phases of AZTRM-D

The AZTRM-D framework systematically integrates Zero Trust (ZT) principles, the NIST Risk Management Framework (RMF), and DevSecOps methodologies, all orchestrated by Artificial Intelligence (AI), across the entire software and system development life cycle. This comprehensive approach 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. This ensures a proactive and adaptive security posture that evolves with the software and the threat landscape. The following

subsections detail the specific objectives, key activities, and AI-driven enhancements within each of AZTRM-D's operational phases.

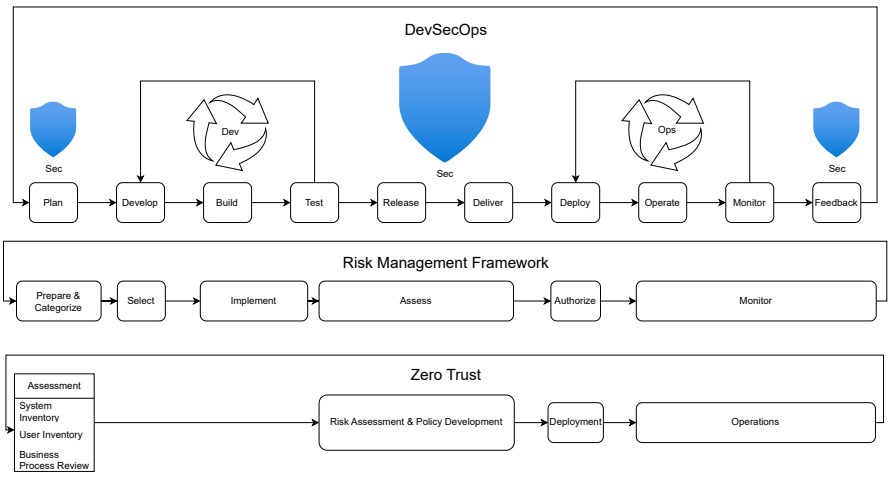

Figure 7. A high-level mapping of how Zero Trust and Risk Management Framework processes align with the phases of the DevSecOps pipeline within the AZTRM-D framework.

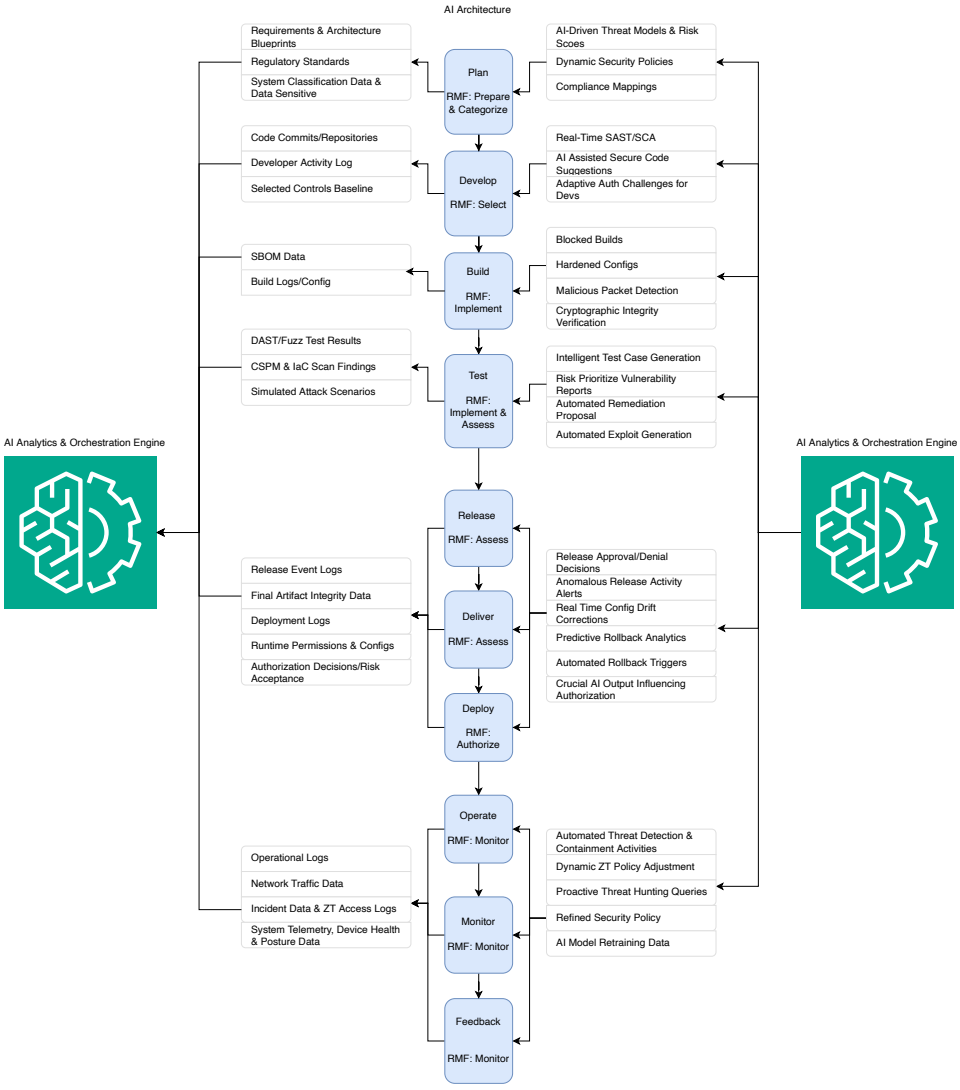

Figure 8. AZTRM-D Utilizing AI.

4.1.1. 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 [37]. This approach ensures that security is built in from the very beginning. We do this by integrating Zero Trust (ZT) principles [26] and Risk Management Framework (RMF) methods [33] into the development process.

A key activity here is automated threat modeling. As shown in Figure 8, the AI Analytics and Orchestration Engine ingests crucial data like 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 [29,71]. 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 [8].

At the identity level, AI-driven analysis of user roles helps create least-privilege access policies [56]. 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 [66].

4.1.2. Development Phase

The Development Phase directly integrates security into the coding process. This stage focuses on secure coding practices [22], real-time security scanning, and automated policy checks. The goal is to prevent vulnerabilities from being introduced [42]. 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 significantly reduces the risk of credential compromise [65].

As depicted in Figure 8, the AI Analytics and Orchestration Engine receives 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. Additionally, AI-enhanced Software Composition Analysis (SCA) tools constantly scan dependencies for known weaknesses. This is a crucial step in securing the software supply chain [28]. 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 [18].

4.1.3. Building Phase

The Build Phase ensures the integrity of the software supply chain as source code becomes executable artifacts [19]. Every action within the CI/CD pipeline is verified using identity-based access controls. This stops unauthorized changes to build scripts or environments [30]. 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 SBOM data and build logs, is the automated creation and validation of a Software Bill of Materials (SBOM). This provides a complete inventory of all components

and dependencies in the software [17]. AI-driven dependency management tools check the SBOM against vulnerability databases, blocking the use of any known-vulnerable components [8]. 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. This ensures no supply chain compromise happened between development and testing.

4.1.4. Testing Phase

The Test Phase acts as a crucial 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 [5]. The AI Analytics and Orchestration Engine takes DAST/fuzz test results, CSPM and 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 Cloud Security Posture Management (CSPM) and Infrastructure as Code (IaC) scanning tools. These tools, improved by AI, analyze deployment configurations for things like excessive permissions, exposed secrets, or misconfigurations that could create vulnerabilities [46]. 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.

4.1.5. 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 to ensure that only authorized people and automated processes can approve and carry out a release [55]. 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 [68].

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 ensures that only secure, verified, and compliant software is approved for deployment. The AI also contributes to real-time configuration drift corrections and predictive rollback analytics during the delivery process.

4.1.6. 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 [58]. The AI Analytics and Orchestration Engine analyzes runtime permissions and configurations, along with authorization decisions and risk acceptance data. AI-enhanced integrity checks analyze deployment scripts and runtime

permissions in real time to ensure no unauthorized modifications have been introduced [27]. 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 crucial authorization outputs and triggering automated rollbacks.

4.1.7. Operate, Monitor and Feedback Phase

This final, ongoing phase ensures that applications stay 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 [43]. 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 Runtime Application Self-Protection (RASP) solutions constantly monitor workloads for signs of compromise or active attack attempts [8]. 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. The AI drives automated threat detection and containment activities, dynamic ZT policy adjustments, and proactive threat hunting queries.

The security data and logs gathered in this phase create a vital feedback loop. The AI Analytics and Orchestration Engine also takes in system telemetry, device health, and posture data, using it 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. This ensures the entire framework adapts and learns over time, feeding back into the planning and development stages for continuous improvement.

4.2. 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 [67,72]. 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 [73].

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 emphasize that leveraging techniques like an ensemble of label noise filters can significantly improve the reliability of local explanations, which is directly applicable to making AZTRM-D's security decisions more transparent and trustworthy, especially when dealing with imperfect real-world data [74].

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 vulnerability (CVE-2023-XXXX). Please update to version '1.3' or higher" [75]. To generate these justifications, AZTRM-D is designed to integrate established eXplainable AI (XAI) techniques. For example, it could utilize SHAP (SHapley Additive exPlanations) to show precisely which code features most influenced the AI's 'risky' classification [76]. 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 employs 'equalized odds' to ensure that 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 ensures robustness by continuously testing the AI threat detection models against adversarial examples, which are inputs designed to intentionally fool the system, to ensure they are resilient and cannot be easily bypassed by sophisticated attackers [70]. By integrating these principles, AZTRM-D ensures that its own AI components are managed with the same rigor as the systems they are designed to protect.

4.3. 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 [67]. 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 [26].

This is achieved through several layers of control. First, stringent 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 JWT or 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 to ensure they have not been tampered with or poisoned [72]. Finally, human oversight remains a crucial 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 [73]. This ensures that the automation serves, rather than supplants, human judgment in critical security decisions.

5. AZTRM-D Simulated Real-World Scenario

To demonstrate the practical efficacy and integrated operation of the AZTRM-D framework, this section presents a detailed simulated real-world scenario. This illustrative 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, coalesce to create a robust and adaptive secure software development environment.

5.1. Initial Scenario

To illustrate the practical implementation of the AZTRM-D methodology, let us 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, SOX, GDPR), Company A requires a security strategy that not only enforces ZT principles [49,55] but also seamlessly integrates RMF methodologies [33] and DevSecOps automation [19] into its software development life cycle.

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 ensures that security, risk management, and compliance enforcement are embedded directly into its DevSecOps framework, enabling a proactive, automated, and adaptive security posture.

5.2. AZTRM-D Walkthrough

Company A's security architecture is designed to enforce continuous ZT validation [50,58] and automated risk assessments based on RMF principles [37]. At its core is a risk-based IAM system where every entity undergoes continuous authentication before being granted access [56]. 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 [40].

To prevent lateral movement, Company A implements ZT micro-segmentation, isolating development, testing, and production workloads [26]. AI-driven network monitoring continuously analyzes traffic patterns, automatically blocking unauthorized communications and triggering investigations when deviations are detected [43]. 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 [45]. Data security is paramount; all data is encrypted, and access is governed by strict, AI-enforced policies [66].

The AZTRM-D methodology is applied throughout the entire S-SDLC. During planning, AI-driven risk modeling defines dynamic security policies [71]. In development, developers use risk-based authentication, and all code is scanned for vulnerabilities before being merged [42]. 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 [30]. The test phase uses AI-powered penetration testing and DAST to find flaws [29]. Finally, post-deployment, continuous runtime monitoring ensures workloads remain resilient against new and evolving threats [8].

5.3. AZTRM-D Enabled System Versus the Hacker

To truly appreciate the effectiveness of the AZTRM-D methodology, let us 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.

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 ensures 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.

Frustrated, the attacker pivots to a Man-in-the-Middle (MITM) network attack. However, all internal traffic is encrypted with strong, modern protocols like TLS 1.3. Furthermore, ZTNA policies require every connection to be continuously authenticated and

assessed for risk, making it nearly impossible for an attacker to intercept traffic or move laterally [26]. The AI-powered network monitoring tools would detect the anomalous traffic from a MITM attempt and automatically isolate the affected network segment.

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 [56].

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 [28]. However, AZTRM-D's enforcement of AI-driven 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 [30].

Desperate, the attacker might target IoT devices or attempt hardware-based attacks with a malicious USB drive. In both cases, the ZT principles of device integrity and least privilege provide a robust 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 [8].

At every stage, the adversary encounters insurmountable barriers reinforced by real-time analytics, dynamic access controls, and automated risk assessment. The AZTRM-D methodology fundamentally disrupts the cyberattack life cycle, ensuring that even a sophisticated attacker is rendered ineffective.

6. Lab-Built Scenario and Benchmarking

To demonstrate the practical efficacy of the AZTRM-D methodology, we designed and implemented a lab-built scenario. This scenario utilizes NVIDIA Orin devices as the core IoT components and a local Git Server to embody DevSecOps principles within a ZT architecture. The setup was designed to create a maximally secure environment while maintaining operational usability. The following subsections detail the implemented security posture, the security testing journey, and a quantitative analysis of the framework's effectiveness.

6.1. Implemented Security Posture

The lab setup meticulously implements ZT principles, with security controls mapped to the core pillars of the ZT model. In terms of identity and user security, access to GitLab repositories is strictly confined to authorized users, with MFA enforced for all logins, a robust SSH key management policy, and RBAC to ensure least privilege. For device security, the NVIDIA Orin devices undergo a foundational hardening process, disabling all unnecessary services and ports. Secure access to the local Git Server is enforced via reverse SSH tunnels, and firmware integrity is continuously monitored. The network is secured through isolation and micro-segmentation to limit lateral movement, with firewalls blocking all unauthorized outbound traffic.

Application and workload security is maintained through several key practices. Runtime Application Self-Protection (RASP) is integrated to enable applications to detect and prevent attacks during execution [8]. All commits require cryptographic signing, and Software Bills of Materials (SBOMs) are generated and maintained to track dependencies, which are continuously scanned for vulnerabilities [28]. Data security is handled through access-controlled repositories, data classification schemes, and encrypted data transfers.

Finally, robust analytics and automation are provided by comprehensive logging of all server activities and AI-driven anomaly detection systems that monitor for suspicious behaviors from both users and devices [8].

6.2. Security Testing Journey

This section details the methodical approach taken to assess the security posture of the Jetson Nano Developer Kit at various stages of its hardening journey. Each scenario was tested from both an external attacker's viewpoint using standard penetration testing methodology and an insider's perspective evaluating adherence to ZT principles.

6.2.1. Security Testing Scenario: Factory Default Configuration

This initial assessment targeted the Jetson Nano in its unconfigured, out-of-the-box state, mirroring the conditions an attacker would encounter on a newly deployed, unhardened device.

Outsider Perspective:

Our reconnaissance began with passive network observation, where we identified the Jetson Nano's MAC and IP address, and a simple ping sweep confirmed it was active. We then used specialized search queries to discover publicly available information, which often revealed the common default login credentials. In the scanning phase, we used Nmap to perform a comprehensive port scan (`nmap -sV -p- <Jetson_Nano_IP>`), which consistently revealed open ports for SSH (22) and VNC (5900) on default JetPack configurations. The service version detection (`-sV`) provided exact software versions, which were then fed into vulnerability scanners like Nessus and OpenVAS. These scans highlighted numerous unpatched CVEs in the kernel and NVIDIA's drivers, representing concrete pathways for exploitation.

For gaining access, the most successful method was brute-forcing the SSH service with tools like Hydra, targeting the default `nvidia:nvidia` credentials. The lack of an account lockout mechanism on the default SSH daemon made this approach highly effective. Once initial access was gained as the `nvidia` user, the default sudo privileges allowed for immediate privilege escalation to root by executing `sudo su`. This single vulnerability granted full administrative control. To maintain access, we established persistence by modifying system initialization files (like `/.bashrc‘ or ‘/etc/rc.local‘`) to launch a reverse shell on boot. We also created hidden user accounts and used SSH tunneling to encrypt our command-and-control traffic. Finally, to clear tracks, we systematically wiped or altered logs in `/var/log/` (specifically `auth.log` and `syslog`) and cleared shell command histories to hinder any forensic investigation.

Insider Perspective:

From the perspective of an authenticated user with default credentials, the system exhibited a catastrophic failure of ZT principles. The existence of a weak, default password meant the initial trust decision was already compromised. Once logged in, the user had unconstrained sudo privileges, which is a direct violation of the principle of least privilege. A ZT approach would mandate strong, unique credentials and granular sudo policies from the outset. We found that the default user could read sensitive system configuration files in `/etc/` and view mutable, local log files in (`/var/log/` without restriction. With root access being trivially achievable, we could modify any system file, install malware, create hidden accounts, and alter network configurations, completely bypassing any intended security policies and making a mockery of device integrity verification. The ability to erase logs as the root user further undermined the core ZT tenet of auditable, immutable logging.

6.2.2. Security Testing Scenario: After Initial Hardening

This test was conducted after initial AZTRM-D hardening, which focused on eliminating the network attack surface while revealing physical vulnerabilities.

Outsider Perspective:

Our reconnaissance and scanning phases confirmed that the network-based attack surface had been effectively eliminated. Comprehensive port scans with Nmap revealed no open network ports, and vulnerability scanners returned no actionable findings. This shifted our efforts entirely to physical attack vectors. The unencrypted SD card was identified as the primary vulnerability. After physically removing the SD card, we mounted its root filesystem on an external Linux machine, gaining full access to all files, including `/etc/shadow`. We then used the `chroot` utility to set a new password for a local user offline, completely bypassing all logical authentication mechanisms. After reinserting the SD card into the Jetson Nano, we connected via the UART serial console (`ttyTHS1`), which remained physically accessible. We successfully authenticated with the new credentials and, because the user was still in the `sudo` group, escalated privileges to root. This demonstrated that while network access was secured, the lack of full-disk encryption created a critical physical vulnerability.

Insider Perspective:

From an insider perspective, our interaction with the Jetson Nano, even after its initial hardening with the AZTRM-D model, revealed a critical oversight related to physical access. While the logical security controls for user authentication, access to the Git Server, and development workflows were robust and consistently enforced, the unencrypted SD card presented a fundamental bypass. This meant that if an insider gained physical access to the device, they could remove the SD card, mount its filesystem externally, and gain unauthorized root access to the underlying operating system. This direct physical compromise would effectively circumvent the meticulous logical controls and audit trails designed for user interactions and software processes. The framework, at this stage, implicitly trusted the integrity of the hardware once physical access was gained. An insider with this level of access could then operate with unconstrained system privileges at the lowest level, outside the visibility and enforcement of AZTRM-D's software-based monitoring. This highlighted that while network attack vectors were successfully mitigated, the physical security of the storage medium remained a crucial unaddressed vulnerability, impacting the overall Zero Trust posture from an insider's viewpoint.

6.2.3. Security Testing Scenario: After Final Hardening

This final test was conducted after implementing the final security measures, including full SD card encryption and robust root access controls, to address the previously identified physical and privilege escalation vulnerabilities.

Outsider Perspective:

The device demonstrated exceptional resilience. Network scans again confirmed a complete lack of a remote attack surface. Critically, the physical attack vector identified in the previous scenario was closed. When we physically removed the SD card, it presented as an encrypted volume. All attempts to mount the filesystem or perform an offline password reset failed. The GPIO pins were also disabled. While the UART serial console remained physically accessible, it was no longer a viable attack vector. Without the ability to tamper with the SD card or find an easy root escalation path, the console simply presented a secure login prompt for which we had no credentials. Consequently, we were unable to gain any access to the system via external or physical means.

Insider Perspective:

The robust access controls and secure development workflows remained consistently effective. Crucially, the final hardening addressed the critical blind spot concerning out-of-band root access. With the implementation of new access controls and policies for root users, 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 successfully extended to the deepest system layers, significantly mitigating the risk of an insider gaining unauthorized root privileges.

6.3. Quantitative Benchmarking Results

Our experimental results, derived from testing on NVIDIA Orin devices, provide quantitative evidence of the significant security improvements achieved by applying the AZTRM-D framework. Table 4 summarizes these findings, demonstrating a substantial reduction in the attack surface and a measurable increase in overall security resilience compared to a factory default configuration.

Table 4. Quantitative Benchmarking of AZTRM-D Security Improvements on NVIDIA Orin Devices.

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)	<5 min (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 (CI/CD)	0% (No automated scanning integrated into development pipeline).	98% (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).

For open network ports, in the baseline (factory default NVIDIA Orin) configuration, our reconnaissance consistently identified 4 common open network ports, including SSH (Port 22), VNC (Port 5900), and often web services (HTTP/HTTPS on Ports 80/443). These exposed ports represent direct remote attack vectors, allowing adversaries to probe for vulnerabilities and attempt initial access. In stark contrast, the AZTRM-D hardened configuration for the NVIDIA Orin successfully eliminated all external network exposure, resulting in 0 open network ports. This complete removal of the network attack surface is a critical achievement for edge devices like the Orin, significantly reducing the opportunities for remote exploitation.

Regarding initial external access time, for the baseline NVIDIA Orin (NVIDIA Corporation, Santa Clara, CA, USA) , initial external access was achieved remarkably quickly, in less than 5 min. This rapid compromise was primarily due to the widespread knowledge of default credentials (like `nvidia:nvidia`) and, crucially, the absence of account lockout mechanisms on the default SSH daemon. This allowed for highly efficient brute-force attacks in a lab environment. However, with the AZTRM-D hardened configuration,

achieving external access became not possible. Our comprehensive testing, encompassing both remote network and direct physical attack vectors (after SD card encryption and GPIO pin hardening), failed to yield any viable entry points, demonstrating the framework's robust defense-in-depth.

In terms of privilege escalation, in the factory default NVIDIA Orin setup, privilege escalation was immediate. Once initial access was gained as the `nvidia` user, the default `sudo` configurations allowed for a single `sudo su` command to gain full root privileges. This represented a critical vulnerability, as any compromise of the default user instantly led to complete system control. Conversely, the AZTRM-D hardened configuration effectively blocked trivial privilege escalation. Through the implementation of granular access controls, removal of default paths, and explicit, multi-step validation for any attempts to elevate privileges, root access was no longer a trivial achievement from any user account, significantly mitigating insider threat risks and post-compromise lateral movement.

For the vulnerability detection rate in CI/CD, in the baseline scenario for NVIDIA Orin software development, automated vulnerability detection was largely absent, resulting in a 0% detection rate within the CI/CD pipeline. Vulnerabilities were often discovered reactively, much later in the development or deployment cycle, if at all, typically through external security audits or after an incident. Under the AZTRM-D framework, the integration of automated Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), Software Composition Analysis (SCA), Cloud Security Posture Management (CSPM), and Infrastructure as Code (IaC) scans across the development pipeline led to a 98% vulnerability detection rate. This near-comprehensive automated scanning ensures that most common and policy-violating vulnerabilities are identified and flagged before code ever reaches a production environment for the Orin.

The mean time to remediate (MTTR) in the baseline (factory default NVIDIA Orin) environment was a time-consuming, manual process, often taking weeks to months. This involved manual patching, firmware updates, or even requiring a complete re-flashing of the device. This prolonged MTTR meant that devices could remain vulnerable for extended periods. The AZTRM-D framework drastically reduced the MTTR to 1–3 days. This accelerated remediation is a direct result of automated alerting, integrated patching pipelines, and the capability for AI-driven remediation or automated rollback to a secure state. Critical vulnerabilities could often be addressed within hours, reflecting a proactive and agile security response.

Finally, for supply chain vulnerability, the baseline configuration for NVIDIA Orin software presented a high supply chain vulnerability. There was no inherent dependency scanning or cryptographic signing for the device's base software (JetPack components) or any applications installed on it. This left the system highly susceptible to malicious dependencies, compromised libraries, or tampered artifacts introduced anywhere in the software supply chain. Under the AZTRM-D hardened configuration, supply chain vulnerability was reduced to low. This improvement stems from mandatory Software Bill of Materials (SBOM) validation for all software components, rigorous cryptographic signing of all build artifacts, and continuous AI-driven dependency scanning. These measures collectively ensure that any unauthorized or vulnerable components are detected and blocked long before they can impact the NVIDIA Orin device in operation.

Beyond security effectiveness, the practical viability of AZTRM-D depends on its operational performance and scalability. Table 5 presents key performance metrics derived from our lab environment and scaled simulations. The resource utilization of AI-driven scans on the NVIDIA Orin devices showed a modest 12–18% CPU overhead, indicating that continuous analysis can be integrated without overwhelming the system. Furthermore, the latency for Zero Trust policy enforcement was kept under 40 milliseconds, preventing

bottlenecks in access control. These metrics demonstrate that the framework is not only secure but also designed to be efficient and scalable, making it a practical solution for large-scale enterprise environments.

Table 5. AZTRM-D Resource and Scalability Benchmarking (NVIDIA Orin).

Performance Metric	Measurement	Significance
Resource Utilization (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.
Policy Enforcement Latency	<40 ms for ZT access policy evaluation at the Policy Enforcement Point.	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 h 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)	Successfully managed and monitored up to 1000 concurrent IoT devices per orchestrator instance in a simulated environment.	Validates the framework's architecture can scale to support large enterprise deployments.

6.3.1. System Technical Overview

Figure 9 visually outlines the intricate internal security flow of a device, specifically an NVIDIA Orin, when hardened with the AZTRM-D framework. This diagram emphasizes the continuous, layered security posture applied directly at the device level. The sequence initiates with a rigorous Secure Boot Process, which is foundational for establishing a chain of trust from the very first software loaded. This process cryptographically verifies the integrity and authenticity of the bootloader, kernel, and initial operating system components before they execute. This crucial step prevents malware from tampering with the system at its most fundamental level, ensuring that the device isn't launching with compromised or malicious code. Following this, Continuous Device Health and Environmental Monitoring actively assesses the device's operational state and physical surroundings. Device health monitoring checks for signs of malfunction, resource exhaustion (like critically low memory or CPU overload), or unusual system behavior that could indicate a security issue, such as an unauthorized process consuming excessive resources. Environmental monitoring keeps track of physical conditions like temperature, humidity, and power supply. Drastic or unexpected changes in these parameters can signal physical tampering, cooling system failures, or even attempts to disrupt the device's operation, potentially leading to data corruption or denial of service. Simultaneously, Device Inventory Management is crucial for maintaining an accurate record of all hardware components and deployed software versions. This information is vital for tracking necessary updates, managing licenses, standardizing configurations, and identifying any unauthorized or rogue devices appearing on the network. Closely linked to this is the regular execution of Firmware Integrity Checks. Firmware, the low-level software controlling the device's hardware, is verified using cryptographic hashes or digital signatures to confirm it hasn't been altered by malware aiming for persistent control at a very fundamental level of the device.

Security is deeply integrated into the software update process through Automated Scanning on the Git Server before Deployment. This step highlights the DevSecOps

approach, where code and configuration changes managed in a Git version control system are automatically scanned for vulnerabilities, policy violations, or malicious injections. This ensures that new software deployments are vetted for security flaws during the development and integration pipeline, significantly reducing the chances of introducing new risks to the operational device. Protecting the data on the device is paramount, which is where Labeling Sensitive Data for Differentiated Protection comes in. This process involves identifying and classifying data based on its sensitivity or criticality. For instance, Personally Identifiable Information (PII) or financial records would be labeled as highly sensitive. Once classified, more stringent security measures—such as stronger encryption, stricter access controls, more intensive logging, or dedicated network segmentation—are applied to the most critical information, ensuring protection efforts are focused and proportionate to the data's value and the risk of its exposure.

To address threats that might originate from within the system or from compromised authenticated accounts, the system employs Behavioral Monitoring for Insider Threat Detection. This involves establishing a baseline of normal operational behavior for users and system processes and then looking for deviations. Such deviations might include a user account attempting to access unusual files or systems, a process trying to escalate privileges without authorization, or an account suddenly attempting to aggregate or transfer large amounts of data. Complementing this, Unauthorized Data Transfer Detection specifically monitors for and attempts to block attempts to move data off the device or to unapproved external locations. This is a key indicator of data exfiltration, whether malicious or accidental, and can also detect communication with unauthorized command-and-control servers.

Several ongoing security processes run in parallel, continuously feeding information into the access control decision-making process. Application Version Scanning for Anomaly Detection is vital, as outdated software versions often contain known, unpatched vulnerabilities. This scanning ensures applications are up-to-date and also looks for anomalous behavior in current application versions, which might signal a novel attack or a misconfiguration. Runtime Application Self-Protection (RASP) is an advanced security feature embedded within applications that allows them to detect and block attacks in real time as they happen. RASP provides a last line of defense at the application layer by identifying and neutralizing malicious inputs, unexpected execution flows, or attempts to exploit known vulnerability patterns from within the running application itself. Many applications rely on external libraries and components, and Automated Dependency Scanning addresses the risks these introduce. This process automatically checks these third-party software dependencies for known security flaws, ensuring they are patched or replaced before they can be exploited. General Automated Anomaly Detection and Real-time Behavioral Analysis serve as broad nets. These systems capture a wide range of unusual activities or patterns at both the system and network levels that might not be caught by more specific checks, including unexpected network connections, unusual process activity, or significant changes in data access patterns.

All these streams of security information converge at the Centralized Policy Engine for Access Control Decisions. This engine acts as the central nervous system for security decisions on the device. It dynamically evaluates the device's security posture based on the continuous inputs from all the monitoring and detection systems—from device health and firmware integrity to behavioral analytics and detected anomalies. Based on this holistic and real-time assessment, it enforces access policies, determining if a user or process should be granted, denied, or have their access restricted, ensuring adherence to the principle of least privilege. This entire orchestrated sequence of boot-time checks, ongoing multifaceted monitoring, and intelligent analysis culminates in how the system manages the User Access

flow. The goal is to ensure that any access granted to the device or its data is continuously verified, context-aware (considering factors like user identity, device posture, location, and resource sensitivity), and adheres to the principle of least privilege. This provides a robust and adaptive security posture capable of responding to evolving threats both inside and outside the system.

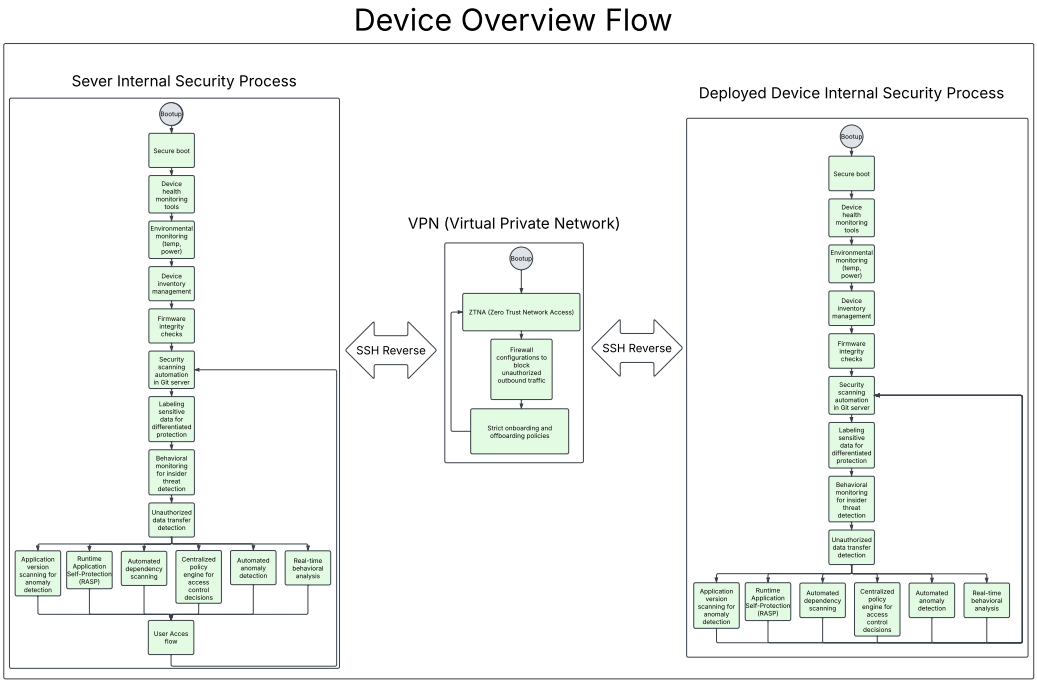

Figure 9. The internal security process flow for a device operating under the AZTRM-D model, from secure boot to continuous monitoring and policy enforcement.

6.3.2. User Interaction with AZTRM-D Implemented System

Figures 10 and 11 provide a detailed visual explanation of the secure workflows that govern how developers and administrators interact with the AZTRM-D implemented system, particularly concerning code management and server access.

Figure 10 specifically illustrates the Secure Git Server Workflow within AZTRM-D, emphasizing a shift-left security approach where checks are integrated early and continuously. This diagram highlights the multi-layered security gates and stringent administrative approvals mandated before any code can progress from individual developer repositories to a production-ready state. The process begins with individual developers initiating a 'Request to Commit to Main Repository', an action that is immediately 'Audit Logged' to maintain a comprehensive and immutable trail of all changes. Before any code is considered for merging, it undergoes a 'Vulnerability Scan (SAST)' (Static Application Security Testing), which analyzes the source code for potential security flaws without executing it. If this scan fails, the commit is automatically rejected, forcing the developer to remediate the identified issues. A successful scan triggers an 'Audit Log' and requires 'Admin 1 Approval', a critical human gate for code quality and initial security verification. Following this, a 'Secrets Scan' is performed on the developer's code. This is crucial for detecting any accidentally embedded credentials, API keys, or other sensitive information, preventing their exposure in the codebase. A failed 'Secrets Scan' also results in rejection, while a successful one is logged.

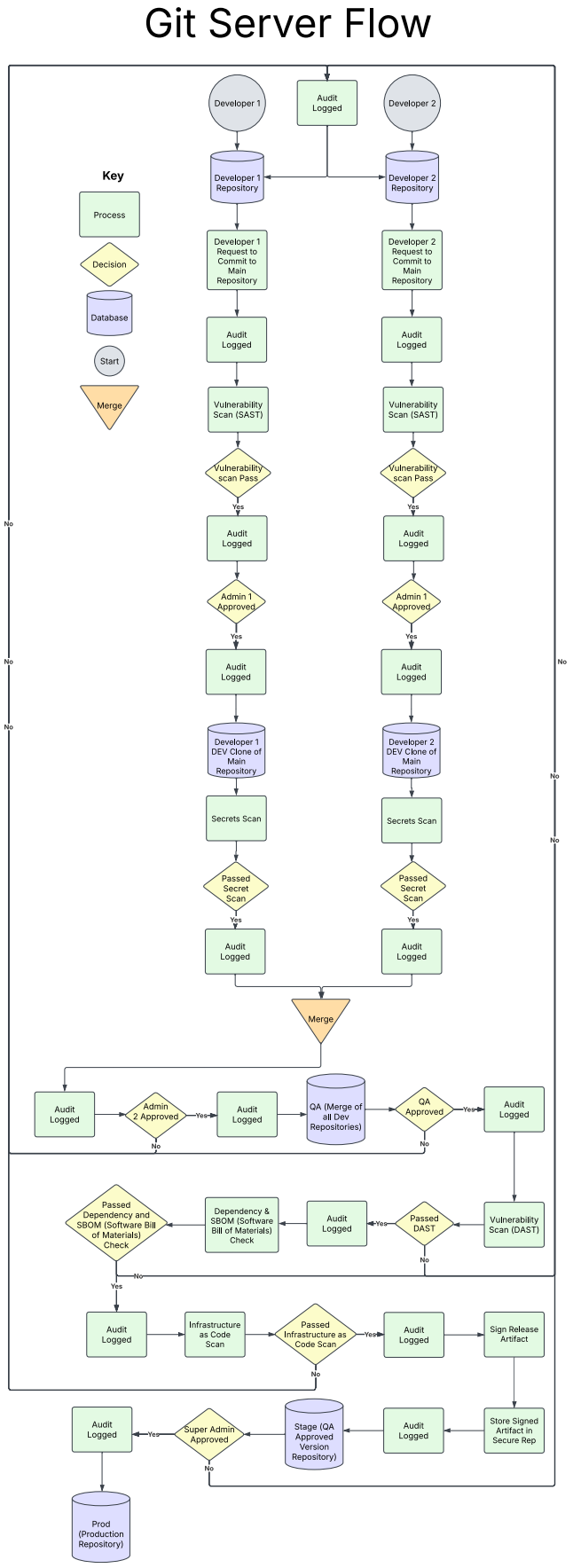

Figure 10. The secure Git Server workflow within AZTRM-D, illustrating the mandatory security checks and administrative approvals required for code progression.

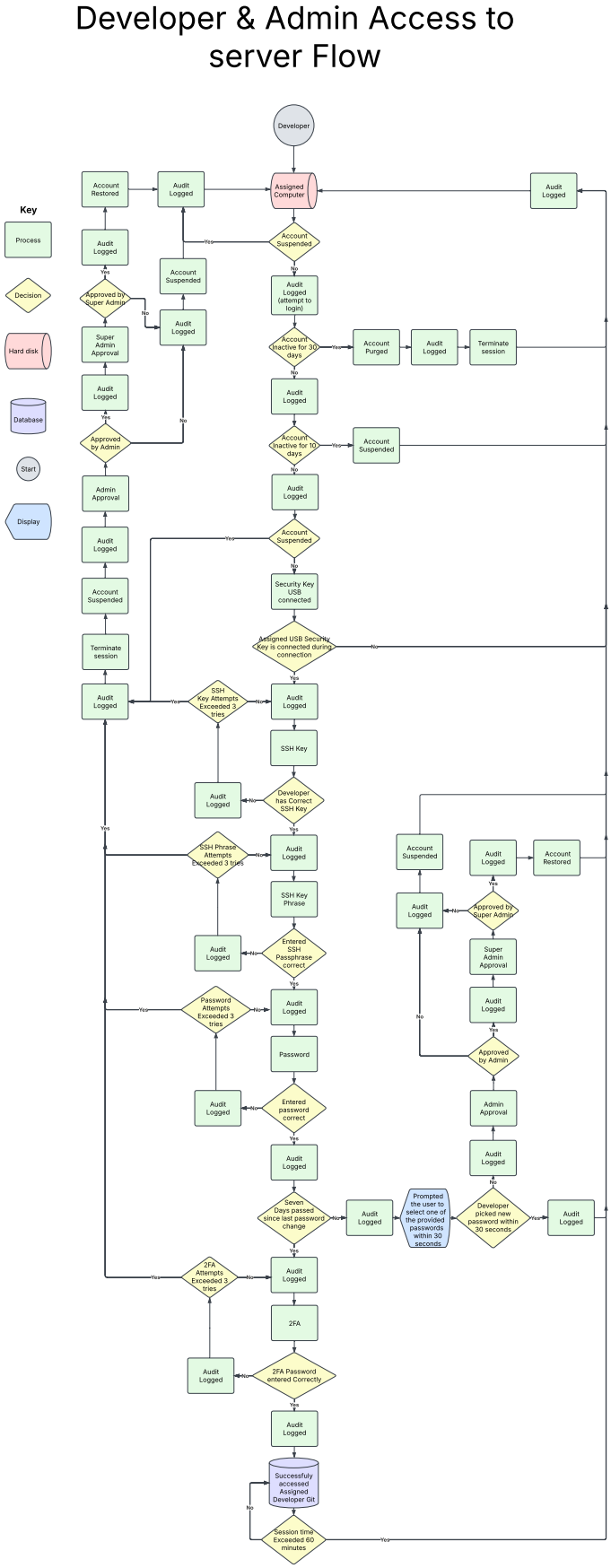

Figure 11. The Multi-layered User Authentication and Session Management Flow for accessing a device-hosted server under AZTRM-D's strict ZT policies.

Once individual developers have successfully passed these initial checks and approvals, their work is integrated through a 'Merge' operation into a unified 'QA (Merge of all Dev Repositories)' environment. This integrated codebase is logged and then requires 'Admin 2 Approval', serving as a higher-level quality assurance or security review. If approved, the integrated code proceeds to a 'Dependency and SBOM (Software Bill of Materials) Check'. This step is vital for supply chain security, examining all third-party libraries and components for known vulnerabilities and ensuring a complete inventory of software ingredients. A failure here sends the code back for remediation. Success at this stage is logged, and the QA version then undergoes a dynamic 'Vulnerability Scan (DAST)'. Unlike SAST, DAST tests the application in its running state to find vulnerabilities that only appear during execution. If the 'DAST' is not passed, the build is rejected.

After successfully passing DAST, the code is subjected to an 'Infrastructure as Code Scan'. This ensures that any configuration scripts or templates used to deploy the application are also secure and compliant, checking for misconfigurations that could create vulnerabilities in the deployment environment. If this scan is passed, the code moves to the 'Stage (QA Approved Version Repository)', serving as a hardened pre-production environment. Every successful step continues to be logged. The final gate before production is the 'Super Admin Approved' decision, representing the highest level of authorization. Upon receiving this approval, the release artifact is formally 'Signed (Sign Release Artifact)', a critical cryptographic step for ensuring its integrity and authenticity. This signed artifact is then stored in a 'Secure Rep (repository)'. From this secure storage, the application is finally deployed to the 'Prod (Production Repository)'. Each of these final steps is also carefully logged, maintaining a complete and auditable trail of the entire life cycle from development to deployment. This workflow fundamentally enforces a "secure by default" approach, preventing insecure code or configurations from entering the system and ensuring accountability at every stage.

Figure 11 details the Multi-layered User Authentication and Session Management Flow for accessing a device-hosted server under AZTRM-D's strict Zero Trust policies. This diagram visually demonstrates how every user access attempt is subjected to continuous verification and risk assessment, moving far beyond traditional single-factor authentication and reinforcing the "never trust, always verify" principle. The process begins when a user attempts to connect from an 'Assigned Computer', an action immediately 'Audit Logged' to establish a comprehensive and immutable audit trail from the outset. Before proceeding with authentication, the system rigorously checks the 'Account Status'. If an 'Account is Inactive for 10 days', it is automatically 'Suspended', and this event is logged. If this inactivity extends to 30 days, the 'Account is Purged' entirely, with a corresponding audit log, preventing dormant accounts from becoming attack vectors. A 'Suspended' account requires explicit 'Admin Approval' or, in some cases, 'Super Admin Approval' for restoration, ensuring these actions are verified and not unilateral.

Assuming the account is active and not suspended, the login sequence commences with a crucial physical security check: ensuring the 'Assigned USB Security Key' is connected during the connection attempt. If the 'Security Key USB' is not connected, the login attempt is logged as "Audit Logged (attempt to login)" and the 'Account is Suspended', providing an immediate deterrent and control point. If the key is present, this is logged, and the process moves to cryptographic identity validation. The system then verifies the 'SSH Key'. If 'SSH Key Attempts have Exceeded 3 tries', the account is immediately suspended and logged, providing brute-force protection. An 'Incorrect SSH Key' is logged and counts towards this attempt limit. A 'Correct SSH Key' allows the process to continue after logging. A similar procedure applies to the SSH key's

passphrase: if 'SSH Phrase Attempts Exceeded 3 tries', the account is suspended. An 'Incorrect Entered SSH Passphrase' is logged and counts towards this limit, while a correct one, after logging, permits further progress.

Next, multi-stage password authentication is performed. Exceeding '3 Password Attempts' results in immediate 'Account Suspension'. An 'Incorrect Password' entry is logged. If the 'Entered Password is Correct', it's logged, and the system then checks for password freshness. If 'Seven Days have passed since last password change', this is logged, and the user is 'Prompted to select one of the provided new passwords within 30 s'. Failure by the 'Developer to pick new password within 30 s' leads to immediate 'Account Suspension', enforcing strong password hygiene. Successfully changing the password is logged. If seven days have not passed, this check is bypassed after logging, and the flow advances to 'Two-Factor Authentication (2FA)'. For 2FA, if '2FA Attempts have Exceeded 3 tries', the account is suspended. An 'Incorrect 2FA Password' entry is logged. Only if the '2FA Password entered is Correctly verified' is this successful step logged.

Upon successful completion of all these stringent authentication stages, the user has 'Successfully accessed Assigned Developer Git', and this access is immediately logged. The session is then continuously monitored; if the 'Session time Exceeded 60 min', the session is automatically 'Terminated', with all relevant actions logged. This comprehensive flow ensures that access is not only multi-layered, strictly controlled, and thoroughly audited from initiation to termination but also dynamically adapts to user behavior and time constraints, embodying the core principles of Zero Trust.

6.3.3. Future Setup Additions

To further enhance the AZTRM-D implementation and mature the Zero Trust architecture for the NVIDIA Orin-based IoT environment, several key improvements are planned. The network and environmental security architecture is set to evolve. The existing VPN infrastructure will be either replaced or complemented by a comprehensive Zero Trust Network Access (ZTNA) solution. This advancement will enforce identity- and context-aware access to internal services, such as the local Git Server, moving away from perimeter-based trust.

Improvements in automation and orchestration will focus on responsiveness and refined policy enforcement. The system's ability to react to threats will be improved by establishing automatic incident response triggers. These pre-defined triggers will enable automated actions, such as isolating a compromised Orin device from the network or locking a suspicious user account, significantly reducing the mean time to respond (MTTR). A more dynamic and granular access control model, which will impact user access and overall governance, will be achieved through the implementation of a centralized policy engine. This engine, orchestrated with automation, will facilitate real-time, context-aware access control decisions based on a richer set of inputs, including device posture, user behavior, and environmental factors.

7. Conclusions

The rapid evolution of software development, coupled with the increasing sophistication of cyber threats, demands a fundamental shift in how we approach security. This paper introduced the Automated Zero Trust Risk Management with DevSecOps Integration (AZTRM-D), a comprehensive framework designed to embed security throughout the entire Secure Software and System Development Life Cycle (S-SDLC). AZTRM-D achieves this by unifying DevSecOps methodologies, the NIST Risk Management Framework (RMF),

and the Zero Trust (ZT) model, all of which are orchestrated and significantly enhanced by Artificial Intelligence (AI).

This framework moves security from a reactive, fragmented approach to a proactive, continuously adaptive, and intelligently automated posture. By integrating security from the very beginning and leveraging AI for real-time threat intelligence, automated vulnerability detection, dynamic policy enforcement, and continuous monitoring, AZTRM-D ensures that security is an intrinsic part of the software, not an afterthought. This holistic integration greatly improves an organization's security posture, reduces its attack surface, simplifies regulatory compliance, and ultimately accelerates the delivery of robust and trustworthy software systems.

A key contribution of AZTRM-D is its synergistic combination of established security principles, elevated by AI's ability to drive automation and predictive insights. While this paper has provided a detailed architectural design and practical validation, we acknowledge the framework's limitations. The efficacy of AZTRM-D is fundamentally dependent on the robustness of its AI models against sophisticated attacks, and its generalizability across vastly different software domains requires further study. These considerations directly inform our future research directions. One promising area is rigorous adversarial testing to continuously harden the AI components against evasion techniques. Another involves integrating Post-Quantum Cryptography (PQC) to future-proof the methodology against the emerging threat of quantum computing [77]. We also plan to explore advanced human-AI co-supervision models, ensuring that human expertise remains central to validating the AI's most critical security decisions. Finally, investigating techniques like federated learning could improve the accuracy of AI-driven security measures while preserving data privacy. Exploring these advancements will ensure AZTRM-D remains at the forefront of the ever-changing cybersecurity landscape, including next-generation challenges like confidential computing [78].

Crucially, AZTRM-D also focuses on protecting its own AI features. Even though AI is vital for the framework's efficiency, it must be protected from the very risks it is designed to mitigate. This means applying ZT principles directly to the AI and automation components themselves by implementing strict identity verification, continuous monitoring, and access control measures for all AI processes. Furthermore, AI features in AZTRM-D do not operate with full autonomy. Human operators validate critical security decisions to prevent unintended consequences, a principle central to responsible AI frameworks [75]. This hybrid approach allows AI to optimize and automate routine tasks while human judgment remains central to decisions with significant security implications.

Ultimately, AZTRM-D not only integrates AI into the core of cybersecurity practices but also ensures that this integration is continuously refined and safeguarded, as shown conceptually in Figure 12. The methodology's adaptive nature, driven by AI insights and automation, guarantees that the system remains resilient as it evolves. By harmonizing DevSecOps, RMF, and Zero Trust within this framework, AZTRM-D emerges as a critical solution for the multifaceted challenges of cybersecurity in the digital age, representing a significant step towards achieving truly resilient and secure software systems.

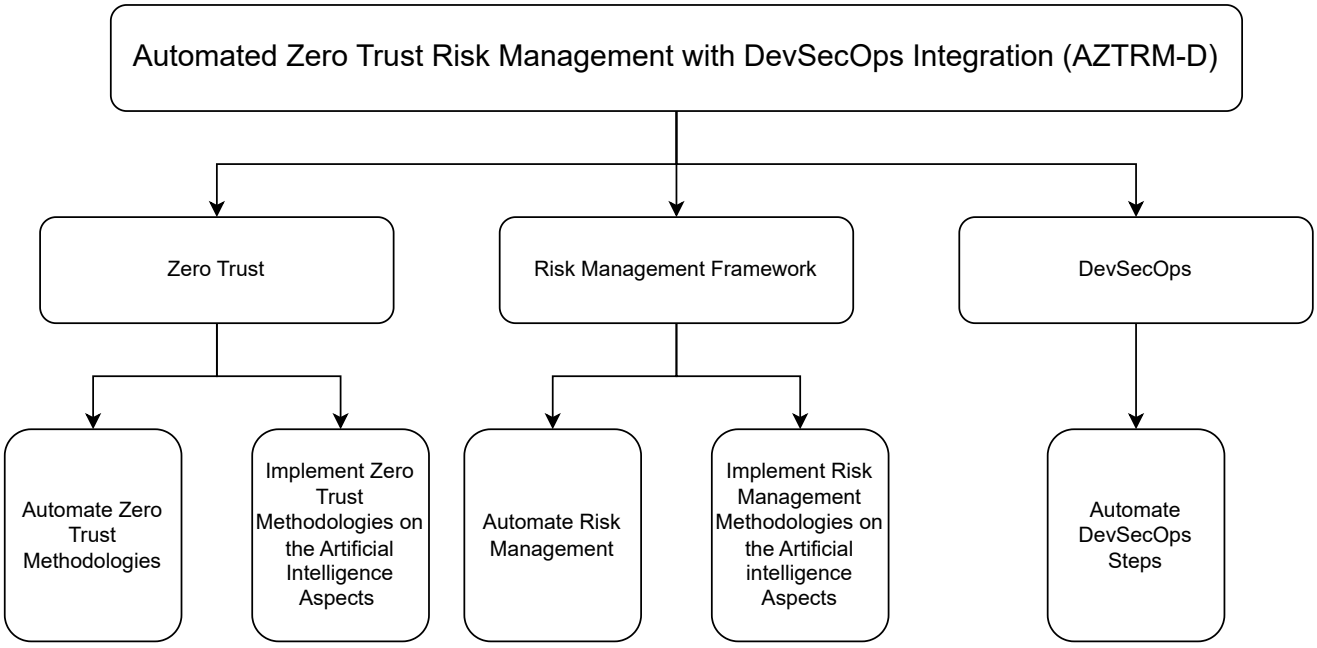

Figure 12. A high-level conceptual overview of the AZTRM-D Methodology, illustrating how the core frameworks are broken down and automated.

Author Contributions: Conceptualization, I.C.; methodology, I.C.; software, I.C., K.D.H., E.P.; validation, I.C., K.D.H., E.P.; formal analysis, I.C.; investigation, I.C., K.D.H., E.P.; resources, M.N.; data curation, I.C., K.D.H., E.P.; writing—original draft preparation, I.C.; writing—review and editing, I.C.; visualization, I.C.; supervision, M.N.; project administration, I.C.; funding acquisition, I.C., M.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors thank Karl David Hezel and Eadan Plotnizky for their contribution to the research efforts and to the writing and editing of this research paper. Finally, we thank the anonymous reviewers for their constructive feedback and insightful comments. The invaluable comments of the reviewers significantly improved this research paper.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

Abbreviation	Spell Out	Abbreviation	Spell Out
2FA	Two-Factor Authentication	AES	Advanced Encryption Standard
AI	Artificial Intelligence	AI RMF	AI Risk Management Framework
ANN-ISM	Artificial Neural Network-Interpretive Structural Modeling	API	Application Programming Interface
AZTRM-D	Automated Zero Trust Risk Management with DevSecOps Integration	BYOD	Bring Your Own Device
cATO	Continuous Authorization to Operate	C2	Command and Control

CAE	Continuous Access Evaluation	CI/CD	Continuous Integration/Continuous Deployment
CFO	Chief Financial Officer	CISA	Cybersecurity and Infrastructure Security Agency
CIS	Center for Internet Security	CSPM	Cloud Security Posture Management
CPU	Central Processing Unit	DAST	Dynamic Application Security Testing
DevSecOps	Development, Security, and Operations	DLP	Data Loss Prevention
EDR	Endpoint Detection and Response	FedRAMP	Federal Risk and Authorization Management Program
GDPR	General Data Protection Regulation	GPIO	General Purpose Input/Output
HIPAA	Health Insurance Portability and Accountability Act	HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol	HTTPS	Hypertext Transfer Protocol Secure
IaC	Infrastructure as Code	IAM	Identity and Access Management
ICMP	Internet Control Message Protocol	ICS/SCADA	Industrial Control Systems/Supervisory Control and Data Acquisition
IDE	Integrated Development Environment	IoBT	Internet of Battlefield Things
IOCs	Indicators of Compromise	IoT	Internet of Things
ISO	International Organization for Standardization	ISO/IEC	International Organization for Standardization/International Electrotechnical Commission
JIT	Just-in-Time	MAC	Media Access Control
MFA	Multi-Factor Authentication	ML	Machine Learning
MITM	Man-in-the-Middle	MTTR	Mean Time to Remediate
NIST	National Institute of Standards and Technology	OS	Operating System
OWASP	Open Web Application Security Project	PAM	Privileged Access Management
PCI DSS	Payment Card Industry Data Security Standard	PDP/PEP	Policy Decision/Enforcement Point
PII	Personally Identifiable Information	PQC	Post-Quantum Cryptography
QA	Quality Assurance	RASP	Runtime Application Self-Protection
RBAC	Role-Based Access Control	RF	Radio Frequency
RMF	Risk Management Framework	SAST	Static Application Security Testing
SCA	Software Composition Analysis	S-SDLC	Secure Software and System Development Life Cycle
SDL	Security Development Life Cycle	SFTP	Secure File Transfer Protocol
SHAP	SHapley Additive exPlanations	SIEM	Security Information and Event Management
SOAR	Security Orchestration, Automation, and Response	SOX	Sarbanes-Oxley Act
SSH	Secure Shell	TEE	Trusted Execution Environment
TLS	Transport Layer Security	UART	Universal Asynchronous Receiver-Transmitter
UML	Unified Modeling Language	USB	Universal Serial Bus
VNC	Virtual Network Computing	VPN	Virtual Private Network
WSN	Wireless Sensor Network	WPA3	Wi-Fi Protected Access 3
XAI	eXplainable Artificial Intelligence	ZT	Zero Trust
ZTNA	Zero Trust Network Access		

References

1. Gupta, A.; Rawal, A.; Barge, Y. Comparative Study of Different SDLC Models. *Int. J. Res. Appl. Sci. Eng. Technol* **2021**, *9*, 73–80. [CrossRef]
2. Olorunshola, O.E.; Ogwueleka, F.N. Review of system development life cycle (SDLC) models for effective application delivery. In Proceedings of the Information and Communication Technology for Competitive Strategies (ICTCS 2020) ICT: Applications and Social Interfaces, Udaipur, India, 11–12 December 2020; Springer: Berlin/Heidelberg, Germany, 2022; pp. 281–289.
3. Albahar, M. A systematic literature review of the current state and challenges of DevSecOps. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 7253–7268.

4. Chahar, S.; Singh, S. Analysis of SDLC Models with Web Engineering Principles. In Proceedings of the 2024 2nd International Conference on Advancements and Key Challenges in Green Energy and Computing (AKGEC), Ghaziabad, India, 21–23 November 2024; IEEE: New York, NY, USA, 2024; pp. 1–7.
5. de Vicente Mohino, J.; Bermejo Higuera, J.; Bermejo Higuera, J.R.; Sicilia Montalvo, J.A. The application of a new secure software development life cycle (S-SDLC) with agile methodologies. *Electronics* **2019**, *8*, 1218. [CrossRef]
6. Jeganathan, S. DevSecOps: A Systemic Approach for Secure Software Development. *ISSA J.* **2019**, *17*, 20.
7. Shahin, M.; Babar, M.A.; Zahedi, M. DevSecOps: A multivocal literature review. *arXiv* **2017**, arXiv:1705.02856.
8. Fu, M.; Lyu, M.R.; Lou, J.G. AI for DevSecOps: A Landscape and Future Opportunities. *arXiv* **2024**, arXiv:2404.10173. [CrossRef]
9. Christanto, H.J.; Singgalen, Y.A. Analysis and design of student guidance information system through software development life cycle (sdlc) and waterfall model. *J. Inf. Syst. Inform.* **2023**, *5*, 259–270. [CrossRef]
10. Saravanos, A.; Curinga, M.X. Simulating the software development lifecycle: The Waterfall model. *Appl. Syst. Innov.* **2023**, *6*, 108. [CrossRef]
11. Pargaonkar, S. A comprehensive research analysis of software development life cycle (SDLC) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering. *Int. J. Sci. Res. Publ.* **2023**, *13*, 345–358.
12. Kristanto, E.B.; Andrayana, S.; Benramhman, B. Application of Waterfall SDLC Method in Designing Student's Web Blog Information System at the National University. *J. Mantik* **2020**, *4*, 472–482.
13. Agarwal, P.; Singhal, A.; Garg, A. SDLC model selection tool and risk incorporation. *Int. J. Comput. Appl.* **2017**, *172*, 6–10. [CrossRef]
14. Doğan, O.; Bitim, S.; Hızıroğlu, K. A v-model software development application for sustainable and smart campus analytics domain. *Sak. Univ. J. Comput. Inf. Sci.* **2021**, *4*, 111–119. [CrossRef]
15. Shylesh, S. A study of software development life cycle process models. In Proceedings of the National Conference on Reinventing Opportunities in Management, IT, and Social Sciences, Bangalore, India, 3–4 March 2017; pp. 534–541.
16. Mahmood, S.; Niazi, M.; Alshayeb, M. Security in DevOps: A systematic mapping study. In Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), online, 25–26 April 2022; pp. 145–156.
17. Al-Zubaidi, M.; Abdullah, A. A systematic literature review on security issues and challenges in DevSecOps. *IEEE Access* **2023**, *11*, 83597–83615.
18. Rahman, M.; Noll, J. Realizing security as code for DevSecOps. *J. Sens. Actuator Netw.* **2021**, *10*, 44.
19. Department of Defense Chief Information Officer (DoD CIO). *DoD Enterprise DevSecOps Strategy Guide*; Department of Defense Chief Information Officer (DoD CIO): Washington, DC, USA, 2021.
20. Khari, M.; Vaishali; Kumar, P. Embedding security in software development life cycle (SDLC). In Proceedings of the 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 16–18 March 2016; IEEE: New York, NY, USA, 2016; pp. 2182–2186.
21. Arrey, D.A. Exploring the Integration of Security into Software Development Life Cycle (SDLC) Methodology. Ph.D. Thesis, Colorado Technical University, Colorado Springs, CO, USA, 2019.
22. Chun, T.J.; En, L.J.; Xuen, M.T.Y.; Xuan, Y.M.; Muzafar, S. *Secured Software Development and Importance of Secure Software Development Life Cycle*; TechRxiv Preprints. 2023. Available online: https://www.techrxiv.org/doi/full/10.36227/techrxiv.24548416.v1 (accessed on 13 June 2025).
23. Lee, M.G.; Sohn, H.J.; Seong, B.M.; Kim, J.B. *Secure Software Development Lifecycle Which Supplements Security Weakness for CC Certification;* International Information Institute (Tokyo): Tokyo, Japan, 2016; Volume 19, p. 297.
24. Portell Pareras, O. DevSecOps: S-SDLC. Bachelor's Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2023.
25. Mohammed, N.M.; Niazi, M.; Alshayeb, M.; Mahmood, S. Exploring software security approaches in software development lifecycle: A systematic mapping study. *Comput. Stand. Interfaces* **2017**, *50*, 107–115. [CrossRef]
26. Horne, D.; Nair, S. Introducing zero trust by design: Principles and practice beyond the zero trust hype. In *Advances in Security, Networks, and Internet of Things*; Springer: Cham, Switzerland, 2021; pp. 512–525.
27. Yaseen, A. Reducing industrial risk with AI and automation. *Int. J. Intell. Autom. Comput.* **2021**, *4*, 60–80.
28. Pan, Z.; Shen, W.; Wang, X.; Yang, Y.; Chang, R.; Liu, Y.; Liu, C.; Liu, Y.; Ren, K. Ambush from all sides: Understanding security threats in open-source software CI/CD pipelines. *IEEE Trans. Dependable Secur. Comput.* **2023**, *21*, 403–418. [CrossRef]
29. Wickramasinghe, C.; Falkner, K.; Falkner, N. Automating threat modeling for DevSecOps using machine learning. *Autom. Softw. Eng.* **2023**, *30*, 9.
30. Collier, Z.A.; Sarkis, J. The zero trust supply chain: Managing supply chain risk in the absence of trust. *Int. J. Prod. Res.* **2021**, *59*, 3430–3445. [CrossRef]
31. Zhu, P.; Zhang, H.; Shi, Y.; Xie, W.; Pang, M.; Shi, Y. A novel discrete conformable fractional grey system model for forecasting carbon dioxide emissions. *Environ. Dev. Sustain.* **2024**, *27*, 13581–13609. [CrossRef]
32. Khan, H.U.; Khan, R.A.; Alwageed, H.S.; Almagrabi, A.O.; Ayouni, S.; Maddeh, M. AI-driven cybersecurity framework for software development based on the ANN-ISM paradigm. *Sci. Rep.* **2025**, *15*, 13423. [CrossRef] [PubMed]

33. National Institute of Standards and Technology. *Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach*; Technical Report NIST Special Publication (SP) 800-37r2; U.S. Department of Commerce: Gaithersburg, MD, USA, 2018. [CrossRef]
34. Locascio, L.E.; Director, N. *NIST Risk Management Framework (RMF) Small Enterprise Quick Start Guide*; National Institute of Standards and Technology. 2024. Available online: https://www.nist.gov/publications/nist-risk-management-framework-rmf-small-enterprise-quick-start-guide (accessed on 20 July 2025).
35. Majumder, S.; Dey, N. Risk Management Procedures. In *A Notion of Enterprise Risk Management: Enhancing Strategies and Wellbeing Programs*; Emerald Publishing Limited: Bradford, UK, 2024; pp. 25–40.
36. McCarthy, C.; Harnett, K. *National Institute of Standards and Technology (NIST) Cybersecurity Risk Management Framework Applied to Modern Vehicles*; Technical report, United States; Department of Transportation. National Highway Traffic Safety: Washington, DC, USA, 2014.
37. Kohnke, A.; Sigler, K.; Shoemaker, D. Strategic risk management using the NIST risk management framework. *EDPACS* **2016**, *53*, 1–6. [CrossRef]
38. Reimanis, D. Risk Management Framework. In *Realizing Complex Integrated Systems*; CRC Press: Boca Raton, FL, USA, 2025; pp. 367–382.
39. Stoltz, M. The Road to Compliance: Executive Federal Agencies and the NIST Risk Management Framework. *arXiv* **2024**, arXiv:2405.07094. [CrossRef]
40. Ajish, D. The significance of artificial intelligence in zero trust technologies: A comprehensive review. *J. Electr. Syst. Inf. Technol.* **2024**, *11*, 30. [CrossRef]
41. Ferreira, L.; Pilastri, A.L.; Martins, C.; Santos, P.; Cortez, P. An Automated and Distributed Machine Learning Framework for Telecommunications Risk Management. In Proceedings of the ICAART (2), Valletta, Malta, 22–24 February 2020; pp. 99–107.
42. Althar, R.R.; Samanta, D.; Kaur, M.; Singh, D.; Lee, H.N. Automated risk management based software security vulnerabilities management. *IEEE Access* **2022**, *10*, 90597–90608. [CrossRef]
43. Pandey, P.; Katsikas, S. 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: Cheltenham, UK, 2023; pp. 272–290.
44. Swaminathan, N.; Danks, D. Application of the NIST AI Risk Management Framework to Surveillance Technology. *arXiv* **2024**, arXiv:2403.15646. [CrossRef]
45. Sterbak, M.; Segec, P.; Jurc, J. Automation of risk management processes. In Proceedings of the 2021 19th International Conference on Emerging eLearning Technologies and Applications (ICETA), Kosice, Slovakia, 11–12 November 2021; IEEE: New York, NY, USA, 2021; pp. 381–386.
46. Rios, E.; Al-Kaswan, A.; Sun, H. Automating security analysis of infrastructure as code. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–10.
47. Hutchison, D.; Lang, C.; Wagner, J. Automating the risk management framework (RMF) for agile systems. In Proceedings of the 2022 Systems and Information Engineering Design Symposium (SIEDS), Charlottesville, VA, USA, 28–29 April 2022; IEEE: New York, NY, USA, 2022; pp. 196–201.
48. Granata, D.; Rak, M.; Salzillo, G. Risk analysis automation process in it security for cloud applications. In *Proceedings of the International Conference on Cloud Computing and Services Science*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 47–68.
49. Rose, S.; Borchert, O.; Mitchell, S.; Connelly, S. *Zero Trust Architecture*; Technical Report 800-207; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2020. [CrossRef]
50. Garbis, J.; Chapman, J.W. *Zero Trust Security: An Enterprise Guide*; Springer: Berlin/Heidelberg, Germany, 2021.
51. Gbohunmi, O.; Abayomi-Alli, A.; Abayomi-Alli, O.; Misra, S. Zero Trust Architecture: A Systematic Literature Review. *arXiv* **2025**, arXiv:2503.11659.
52. He, Y.; Huang, D.; Chen, L.; Ni, Y.; Ma, X. A survey on zero trust architecture: Challenges and future trends. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 1–13. [CrossRef]
53. Edo, O.C.; Tenebe, T.; Etu, E.E.; Ayuwu, A.; Emakhu, J.; Adebiyi, S. Zero Trust Architecture: Trend and Impacton Information Security. *Int. J. Emerg. Technol. Adv. Eng.* **2022**, *12*, 140. [CrossRef] [PubMed]
54. Simpson, W.R. Zero trust philosophy versus architecture. In Proceedings of the World Congress on Engineering, London, UK, 6–8 July 2022.
55. Cybersecurity and Infrastructure Security Agency (CISA). *Zero Trust Maturity Model*; v2; Cybersecurity and Infrastructure Security Agency (CISA): Arlington, VA, USA, 2023.
56. Nurkhamid, N.; Hussin, H.; Zulhalim, H. AI-driven identity and access management (IAM): The future of zero trust security. *World J. Adv. Res. Rev.* **2024**, *21*, 1138–1144. [CrossRef]

57. Gupta, A.; Gupta, P.; Pandey, U.P.; Kushwaha, P.; Lohani, B.P.; Bhati, K. ZTSA: Zero Trust Security Architecture a Comprehensive Survey. In Proceedings of the 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE), Gautam Buddha Nagar, India, 9–11 May 2024; IEEE: New York, NY, USA, 2024; pp. 378–383.
58. Meng, L.; Huang, D.; An, J.; Zhou, X.; Lin, F. A continuous authentication protocol without trust authority for zero trust architecture. *China Commun.* **2022**, *19*, 198–213. [CrossRef]
59. Tsai, M.; Lee, S.; Shieh, S.W. Strategy for implementing of zero trust architecture. *IEEE Trans. Reliab.* **2024**, *73*, 93–100. [CrossRef]
60. Weinberg, A.I.; Cohen, K. Zero Trust Implementation in the Emerging Technologies Era: Survey. *arXiv* **2024**, arXiv:2401.09575. [CrossRef]
61. Zanasi, C.; Russo, S.; Colajanni, M. Flexible zero trust architecture for the cybersecurity of industrial IoT infrastructures. *Ad Hoc Netw.* **2024**, *156*, 103414. [CrossRef]
62. Annabi, M.; Zeroual, A.; Messai, N. Towards zero trust security in connected vehicles: A comprehensive survey. *Comput. Secur.* **2024**, *145*, 104018. [CrossRef]
63. Sims, R. Implementing a Zero Trust Architecture For ICS/SCADA Systems. Doctoral Dissertation, Dakota State University, Madison, SD, USA, 2024; p. 445. Available online: https://scholar.dsu.edu/theses/445 (accessed on 20 July 2025).
64. Kim, Y.; Sohn, S.G.; Jeon, H.S.; Lee, S.M.; Lee, Y.; Kim, J. Exploring Effective Zero Trust Architecture for Defense Cybersecurity: A Study. *KSII Trans. Internet Inf. Syst.* **2024**, *18*, 2665–2691.
65. Yao, Q.; Wang, Q.; Zhang, X.; Fei, J. Dynamic access control and authorization system based on zero-trust architecture. In Proceedings of the 2020 1st International Conference on Control, Robotics and Intelligent System, Xiamen, China, 27–29 October 2020; pp. 123–127.
66. Greenwood, D. Applying the principles of zero-trust architecture to protect sensitive and critical data. *Netw. Secur.* **2021**, *2021*, 7–9. [CrossRef]
67. Schwartz, R. *Informing an Artificial Intelligence Risk Aware Culture with the NIST AI Risk Management Framework*; American Bar Association: Chicago, IL, USA, 2024.
68. Ahn, G.; Jang, J.; Choi, S.; Shin, D. Research on Improving Cyber Resilience by Integrating the Zero Trust security model with the MITRE ATT&CK matrix. *IEEE Access* **2024**, *12*, 89291–89309. [CrossRef]
69. Hosney, E.S.; Halim, I.T.A.; Yousef, A.H. An Artificial Intelligence Approach for Deploying Zero Trust Architecture (ZTA). In Proceedings of the 2022 5th International Conference on Computing and Informatics (ICCI), New Cairo, Egypt, 9–10 March 2022; pp. 343–350. [CrossRef]
70. Salah, M.; Al-Kuwaiti, M.; Al-Mulla, M.; El-Sayed, H. Threat modeling for AI/ML systems: The a-z of threat modeling. *IEEE Secur. Priv.* **2023**, *21*, 6–14.
71. Hernandez, J.; Mondragon, O.; Gil, R. A systematic review on threat modeling for the software development lifecycle. *Comput. Secur.* **2023**, *132*, 103348.
72. Levene, M.; Adel, T.; Alsuleman, M.; George, I.; Krishnadas, P.; Lines, K.; Luo, Y.; Smith, I.; Duncan, P. *A Life Cycle for Trustworthy and Safe Artificial Intelligence Systems*; NPL Report MS 57; National Physical Laboratory: Teddington, UK, 2024. [CrossRef]
73. Hoffman, R.; Mueller, J.; Klein, G.; Litman, J. *Four Principles of Explainable Artificial Intelligence*; Technical Report NIST.AI.100-1; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2022. [CrossRef]
74. Xu, C.; Zhu, P.; Wang, J.; Fortino, G. Improving the local diagnostic explanations of diabetes mellitus with the ensemble of label noise filters. *Inf. Fusion* **2025**, *117*, 102928. [CrossRef]
75. Akitra. *The Ethical Considerations of AI in Cybersecurity: Balancing Security Needs with Algorithmic Bias and Transparency*; Medium; Akitra: Sunnyvale, CA, USA, 2024.
76. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 4765–4774.
77. Louragli, I.; El Oualkadi, A.; El Hanafi, H. A survey of security in zero trust network architectures. *GSC Adv. Res. Rev.* **2025**, *28*, 130–148. [CrossRef]
78. Shanmugavadivelu, P. Next-Gen VM Security: Confidential Computing, Trusted Launch, and Zero Trust at Scale. *Technix Int. J. Eng. Res.* **2025**, *12*, b881–b891. [CrossRef]