Review

SecureAI-Flow: A Security-Oriented CI/CD Framework for AI Software

Abdur Rahman and Md. Badiuzzaman Biplob *

Posted Date: 4 June 2025

doi: 10.20944/preprints202506.0035.v1

Keywords: AI security; CI/CD pipeline; DevSecOps; MLOps; Secure software development; Model robustness; SecureAI-Flow

Review

SecureAI-Flow: A Security-Oriented CI/CD Framework for AI Software

Abdur Rahman and Md. Badiuzzaman Biplob *

Computer Science and Engineering Department, International Islamic University Chittagong, Chattogram, Bangladesh
* biplob.cse@iiuc.ac.bd

Abstract: The rapid growth of artificial intelligence (AI) applications necessitates robust software development pipelines that emphasize both scalability and security. This paper proposes SecureAI-Flow, a security-oriented Continuous Integration/Continuous Deployment (CI/CD) framework tailored for AI software systems. SecureAI-Flow integrates security practices throughout the AI software development lifecycle, addressing threats from data ingestion to model deployment. The framework embeds static code analysis, model robustness validation, secure containerization, and threat monitoring as part of the pipeline. We present the conceptual architecture, explain core components, and compare existing approaches to demonstrate how SecureAI-Flow addresses key security challenges in the AI software supply chain.

Keywords: AI security; CI/CD pipeline; DevSecOps; MLOps; secure software development; model robustness; SecureAI-flow

---

1. Introduction

Software systems have become the digital nervous system of modern society, and artificial intelligence (AI) is increasingly powering that system across every domain. From predictive healthcare and personalized finance to autonomous vehicles and smart cities, AI-driven software is now mission-critical. But with great power comes great vulnerability. AI systems not only inherit the traditional security flaws of classical software but also introduce unique risks-like adversarial manipulation, model inversion, and poisoned training data-that many organizations are not yet equipped to defend against.

Meanwhile, the software development landscape has evolved toward speed and automation, thanks in large part to Continuous Integration and Continuous Deployment (CI/CD) pipelines. These pipelines automate tasks like code integration, testing, and delivery, allowing development teams to release features at unprecedented velocity. However, that velocity comes with a tradeoff: security is often an afterthought or a final checkpoint, making it vulnerable to sophisticated attacks that exploit early-stage gaps.

Traditional DevOps practices focus on availability and efficiency but fall short when applied to AI software, where the assets-models, training data, and machine learning code-are more complex and susceptible to advanced threats. Current MLOps workflows streamline deployment but generally fail to incorporate proactive, real-time security mechanisms tailored to the lifecycle of AI artifacts.

To address this critical gap, we propose SecureAI-Flow, a conceptual CI/CD framework designed to integrate security into every phase of AI software development. Rather than bolt-on security tools, SecureAI-Flow relies on a federation of autonomous, collaborative agents-each specializing in a different aspect of the pipeline. These agents perform tasks such as static code analysis, adversarial robustness checks, container signing, and behavioral anomaly detection during and after deployment.

SecureAI-Flow is built to mirror the real-world complexity and diversity of software teams. It accommodates asynchronous workflows and remote collaboration, ensuring that security isn't a blocker but an enabler of agile delivery. Our approach embraces the principles of DevSecOps and

Infrastructure as Code (IaC), treating security policies and AI defense heuristics as version-controlled code. This shift toward codified security brings transparency, repeatability, and auditability into the security pipeline.

Moreover, as AI development becomes increasingly data-centric, SecureAI-Flow incorporates Data Provenance Agents that verify dataset authenticity and lineage. It validates model inputs for signs of adversarial attacks, continuously monitors deployed models for concept drift, and integrates threat intelligence feeds to anticipate emerging attack vectors.

The rest of this paper is organized as follows. In Section II, we present an in-depth review of related work and foundational literature. Section III describes the architectural components of SecureAI-Flow, followed by a discussion in Section IV evaluating its potential impact. Section V outlines future research directions, including implementation strategies and integration with popular MLOps platforms. Finally, Section VI concludes the paper with key takeaways and implications for secure AI-driven development.

2. Security Threat Surface in AI CI/CD Pipelines

AI-enabled CI/CD pipelines introduce unique security risks beyond traditional software deployments. Unlike conventional applications, AI systems handle mutable data, evolving models, and non-deterministic behavior. As such, the threat surface extends into data integrity, model behavior, and inference manipulation.

Table 1 categorizes the common threats and maps them to the affected CI/CD stage.

Table 1. AI CI/CD Threat Surface Mapping

CI/CD Stage	**Threat Vector**	**Description**
Data Ingestion	Data Poisoning	Injection of malicious data during training or testing
Model Training	Adversarial Training Influence	Subtle training interference to misguide generalization
Model Evaluation	Metric Spoofing	Artificial inflation of model performance via biased test data
Containerization	Image Tampering	Inserting backdoors or vulnerable binaries in images
Deployment	Model Extraction	Unauthorized access and cloning of deployed models
Monitoring	Drift Exploitation	Exploiting concept/data drift to evade detection

3. Literature Review

Recent studies have demonstrated the potential of integrating intelligent agents and AI security frameworks in the CI/CD pipeline. Tufano et al. proposed AutoDev, a multi-agent orchestration system using GPT-4 to handle code and test generation autonomously [1]. Kambala emphasized the importance of proactive DevOps security through anomaly detection, rollback automation, and intelligent agent integration [2]. Goyal introduced cloud-based CI/CD modeling using microservices, Infrastructure-as-Code (IaC), and serverless architecture to enhance resource allocation and reliability.

Table 2. Summary of Related Work

Study	Methodology	Security Focus	Contribution
Tufano et al. [1]	GPT-4 multi-agent CI/CD	Autonomous code/test	Introduced AutoDev for end-to-end pipeline management.
Kambala et al. [2]	Empirical analysis	CI/CD anomaly response	Integrated rollback, anomaly detection, config management.
Goyal et al. [3]	Cloud optimization model	Scalability + ML testing	ML-based test selection, serverless deployment efficiency.
Nishat et al. [4]	Threat modeling	Container security	ML-based defense strategies in container CI/CD.
Loevenich et al. [5]	Hybrid DRL+LLM	Cyber defense AI agents	Adaptive cyber monitoring via RAG and threat prediction.

Nishat presented a framework for ML-secured containerized environments, stressing threat modeling and auditability within CI/CD [4]. Loevenich et al. combined deep reinforcement learning and large language models to design an autonomous cyber defense system capable of adaptive threat detection [5].

4. Proposed Framework

4.1. Overview

SecureAI-Flow introduces a novel multi-agent architecture that embeds intelligent, security-aware agents throughout the CI/CD pipeline. It is specifically designed to offer real-time, context-aware, and explainable security measures across the software development lifecycle. The proposed model is modular, allowing individual agents to operate autonomously while collaborating with others to form a robust and cohesive security network. Figure 1 illustrates the layered interaction among agents within the pipeline.

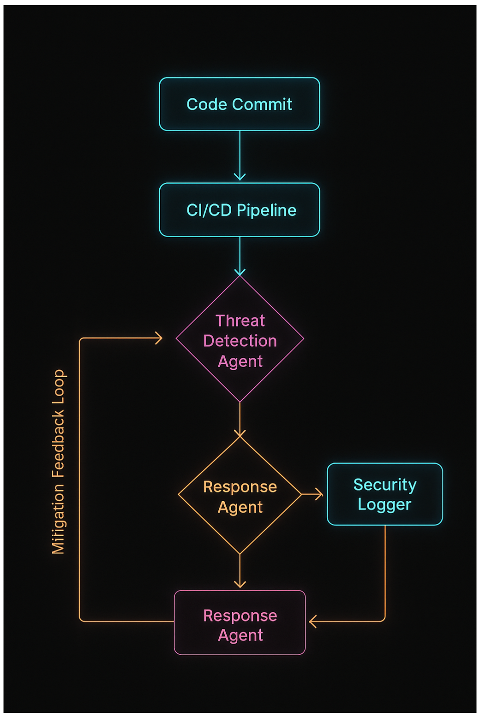

Figure 1. Secure AI Framework

4.2. Agent Roles and Responsibilities

- **CodeGuard-AI:** This agent operates at the developer level, integrated within IDEs or code repositories. It performs continuous static analysis of code as it is written, flagging unsafe functions, insecure cryptographic usage, hardcoded secrets, and common vulnerabilities such as XSS, CSRF, or SQL injection risks. Using explainable ML models, it provides justifications and recommendations, encouraging secure coding practices from the beginning [6–9].
- **BuildSentinel:** Triggered during the build phase, this agent performs deep validation of third-party packages and dependencies. It scans for known CVEs (Common Vulnerabilities and Exposures), evaluates software supply chain trustworthiness, checks open-source license compliance, and flags deprecated libraries. BuildSentinel ensures that vulnerable or malicious artifacts do not get bundled into production builds.
- **ThreatSim-AI:** This dynamic agent is invoked during testing and staging phases. It simulates adversarial attacks using techniques like fuzzing, mutation testing, and generative adversarial networks (GANs). It stress-tests APIs, input fields, and network communication channels by generating synthetic, malicious inputs to detect flaws such as buffer overflows, path traversal, and privilege escalation scenarios [13–19].
- **DeployShield:** Before deployment, DeployShield audits Infrastructure-as-Code (IaC) scripts, Kubernetes manifests, Dockerfiles, and cloud policies. It applies security-as-code principles and scans for misconfigurations, exposed ports, excessive permissions, and missing encryption protocols. It acts as a policy enforcement gatekeeper-blocking deployments that violate compliance or security rules.
- **SentinelLoop:** Operating in the production environment, SentinelLoop continuously monitors system logs, telemetry data, and network traffic. It uses unsupervised learning and anomaly detection models (e.g., Isolation Forests, Autoencoders) to detect behavioral drifts, unauthorized access patterns, and latent zero-day exploits. It also supports auto-remediation and rollback mechanisms in response to suspicious behavior.

4.3. Unique Characteristics

The uniqueness of SecureAI-Flow lies in its combination of autonomy, explainability, and adaptability:

- *Explainable Intelligence:* Each agent is powered by models that include interpretation layers using SHAP, LIME, or attention heatmaps, making their decisions traceable and debuggable by humans.
- *Autonomous Coordination:* Agents communicate via shared memory and publish-subscribe mechanisms to share alerts and adjust thresholds dynamically based on downstream impacts.
- *Security-as-Code Enforcement:* Policies are defined in machine-readable formats (e.g., JSON/YAML) and versioned in code repositories, allowing agents to automatically adapt to updated security rules.
- *Auditability via Immutable Ledger:* All agent actions and decisions are stored in an append-only cryptographic ledger for compliance, transparency, and forensic analysis.

4.4. Real-World Scenario

Imagine a fintech team deploying a payment API. During coding, *CodeGuard-AI* flags use of outdated encryption. At build time, *BuildSentinel* detects a vulnerable library in the dependency tree. Before deployment, *DeployShield* finds that environment variables were exposed in IaC templates and halts the rollout. Post-deployment, *SentinelLoop* observes spikes in login attempts from new geolocations and temporarily locks high-risk accounts.

This level of continuous, context-aware protection ensures that security is no longer a periodic event but an integral, intelligent layer embedded throughout development.

4.5. Conceptual Architecture

The SecureAI-Flow orchestrator (Algorithm 1) implements a fail-safe mechanism where each CI/CD stage is protected by specialized agents. The workflow demonstrates three key security properties:

- **Atomic Validation**: Each stage completes validation before progression
- **Fail-Secure**: Violations trigger rollbacks or quarantine
- **Non-Repudiation**: All actions are logged for auditability

Algorithm 1: SecureAI-Flow Orchestration Workflow

```
Input: Code repository repo
Initialize agents: CodeGuard, BuildSentinel, ThreatSim, DeployShield, SentinelLoop;
foreach stage in [code, build, test, deploy, monitor] do
    agent = get_agent(stage);
    result = agent.execute(repo);
    if not result.passed then
        trigger_rollback();
        log_incident(result);
    end
    SecurityViolation e  quarantine_environment();
    alert_developers(e);
end
```

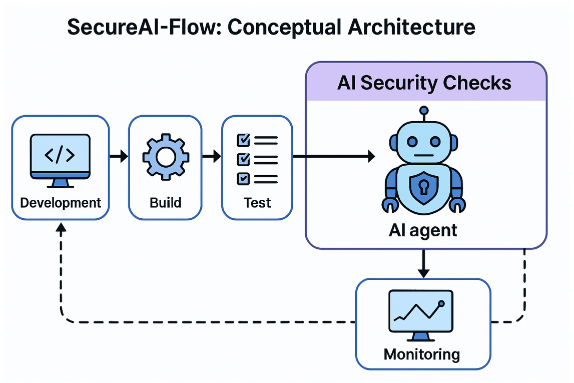

Figure 2. Agent orchestration logic enforcing stage-gated security. Failed checks trigger rollbacks, while critical violations quarantine environments.

SecureAI-Flow operates via secure agent orchestration through a message bus and integrates with existing DevOps tools such as Jenkins, Docker, Kubernetes, and GitLab. Security policies are defined as YAML scripts and processed by agents as Infrastructure-as-Code, ensuring enforcement is audit-friendly and reproducible.

5. Discussion

SecureAI-Flow [10–12] represents a transformative shift in how security can be embedded in AI-driven software development. The introduction of autonomous agents allows continuous vigilance and the ability to respond to threats in near-real-time. Traditional approaches rely heavily on fixed rules and scheduled scans, which often miss zero-day vulnerabilities or adversarial exploits that evolve rapidly. In contrast, SecureAI-Flow agents are designed to adapt using machine learning techniques, such as reinforcement learning or anomaly detection based on statistical drift.

Another crucial benefit is explainability. By integrating Explainable AI (XAI) models, each agent can justify its alerts and actions with human-readable outputs. This increases developer trust, facilitates compliance audits, and improves decision-making when handling threats. For example, CodeGuard-

AI can flag insecure coding practices and explain its findings based on learned patterns from secure codebases, helping developers fix issues at the source.

The modular nature of the framework allows organizations to incrementally adopt security agents based on their maturity and use case. Small teams might start with only CodeGuard-AI and BuildSentinel, while larger enterprises can benefit from the complete stack. The inclusion of SentinelLoop ensures that deployed models continue to operate safely by monitoring real-world feedback, detecting concept drift, and triggering mitigation workflows.

Still, the success of SecureAI-Flow depends on proper training of models, minimizing false positives, and ensuring that the added layers of protection do not slow down development cycles. This trade-off will require performance tuning and perhaps GPU-accelerated agents. Additionally, SecureAI-Flow must evolve with threat intelligence feeds and be regularly updated to remain effective.

Table 3. AI Security Threats and Mitigation Strategies

Threat Type	**Mitigation Strategy**
Adversarial Examples	Adversarial training, input validation, model smoothing
Data Poisoning	Data validation, robust training, anomaly detection
Model Inversion	Differential privacy, limiting output information
Membership Inference	Output perturbation, private aggregation, regularization
Model Theft	API rate limiting, watermarking, output restriction

6. Limitations

While the SecureAI-Flow framework presents a forward-looking model for securing CI/CD pipelines using autonomous AI agents, it is important to acknowledge several limitations inherent to its current design:

- **Lack of Real-World Deployment:** The framework is currently conceptual and has not been validated through empirical deployment on live DevOps pipelines. Without practical testing, its scalability and operational performance remain theoretical.
- **Resource Overhead:** Continuous monitoring, attack simulation, and real-time analysis by AI agents can introduce significant computational overhead. Smaller development teams or organizations with limited infrastructure may face challenges integrating such resource-intensive agents.
- **False Positives/Negatives:** Despite using explainable AI models, the possibility of misclassifying safe code as malicious (false positives) or missing a security flaw (false negatives) cannot be eliminated. This may impact developer trust and workflow efficiency.
- **Complex Integration with Legacy Systems:** Many organizations still use legacy CI/CD systems or ad hoc development workflows. Adapting SecureAI-Flow to such environments may require considerable customization or architectural overhaul.
- **Security of the Agents Themselves:** As autonomous agents gain access to sensitive systems and configurations, they could themselves become attack targets. The framework currently lacks a detailed threat model for the agents, leaving a potential vulnerability surface unaddressed.
- **Dependency on Data Quality:** The efficiency and adaptability of AI agents are heavily reliant on high-quality training and telemetry data. Incomplete or biased datasets could result in reduced model accuracy and skewed threat assessments.

Addressing these limitations through future work-such as performance benchmarking, hybrid deployment models, and agent-level security hardening-will be critical to advancing the practical utility and reliability of SecureAI-Flow in production environments.

7. Evaluation Plan

To assess the effectiveness and practicality of the SecureAI-Flow framework, a multi-phase evaluation strategy is proposed. This approach will help validate its security impact, system overhead, accuracy, and developer usability in real-world CI/CD environments.

Table 4. SecureAI-Flow Evaluation Plan

Component	**Evaluation Criteria**	**Metric**
CodeGuard-AI	Detection of insecure code patterns, secret exposure, unsafe libraries	Precision, Recall
BuildSentinel	CVE scanning, dependency trust score, license compliance check	CVE Coverage, Dependency Score
ThreatSim-AI	Ability to simulate effective adversarial attacks, vulnerability exposure rate	Attack Success Rate, Coverage
DeployShield	IaC audit success, policy violation detection, deployment blocking accuracy	False Positive/Negative Rate
SentinelLoop	Detection of behavioral anomalies, drift identification, real-time threat mitigation	Anomaly Detection Accuracy, Response Time
Overall Pipeline	End-to-end secure deployment, time efficiency, security event prevention	Deployment Time, Incidents Avoided

7.1. Phase I: Prototype Implementation

We plan to implement a functional prototype of SecureAI-Flow using widely adopted DevOps tools such as GitHub Actions, Jenkins, and GitLab CI. The agents will be built using Python and integrated with security analysis APIs (e.g., OWASP ZAP, SonarQube, CVE feeds) and explainable ML libraries (e.g., SHAP, LIME).

7.2. Phase II: Synthetic Test Cases

This phase involves simulating various vulnerability scenarios, including:

- Injection of hardcoded secrets
- Introduction of insecure dependencies
- Misconfigured IaC templates
- Logic flaws exposed by adversarial input

Agent responses will be benchmarked against baseline tools to measure detection rate, false positive rate, and decision transparency.

7.3. Phase III: Real-World Case Studies

To understand deployment viability, we will collaborate with software teams in academia or industry to apply SecureAI-Flow to real projects. Metrics collected will include:

- Reduction in vulnerabilities over time
- Impact on deployment latency
- Developer acceptance and feedback

7.4. Phase IV: Performance Analysis

The system's resource footprint (CPU, memory, execution time) will be evaluated across different project sizes and workloads. Optimization techniques, such as asynchronous agent execution and lightweight inference models, will be applied and tested for scalability.

7.5. Phase V: Security Stress Testing

Agents themselves will be evaluated for resilience against adversarial tampering. Techniques such as fuzz testing, sandbox evaluation, and red teaming will be employed to identify any potential weaknesses in agent logic or communication channels.

7.6. Success Criteria

The evaluation will be considered successful if SecureAI-Flow demonstrates:

- Greater than 85% vulnerability detection rate across simulated and real-world scenarios.
- Less than 10% false positive rate in practical workflows.
- No major system slowdowns (target: <10% CI/CD latency increase).
- Positive usability ratings from developer participants (measured via post-study survey).

8. Implementation Considerations

Deploying SecureAI-Flow in real-world CI/CD ecosystems requires careful planning across infrastructure, tooling, agent orchestration, and developer usability. This section outlines the key aspects and best practices for implementing the proposed framework.

8.1. Toolchain Compatibility

SecureAI-Flow is designed to be platform-agnostic and modular. Each autonomous agent can be containerized using Docker and deployed as a microservice or script within the following CI/CD platforms:

- **GitHub Actions:** Leverage custom actions for CodeGuard-AI and BuildSentinel.
- **GitLab CI:** Integrate security agents as reusable CI templates and environment hooks.
- **Jenkins:** Use pipelines with Groovy scripts to trigger AI agents at defined stages.

8.2. Agent Communication and APIs

Agents will communicate via lightweight RESTful APIs or message queues (e.g., RabbitMQ, Kafka). Each agent maintains logs, decisions, and status in a central audit database. For security and performance:

- All inter-agent communication should be encrypted using TLS.
- JWT or OAuth2 authentication tokens are recommended for secure API access.
- Rate limiting and retry logic will help manage agent reliability.

8.3. Scalability and Orchestration

To support scalability:

- Agents can be deployed using Kubernetes or Docker Swarm.
- Horizontal pod autoscaling will adjust agent replicas based on pipeline load.
- Logging and monitoring should be centralized with tools like ELK stack or Prometheus + Grafana.

8.4. Explainability Tools Integration

Each agent integrates with an explainability library such as SHAP or LIME to generate contextual justifications for its actions. These insights can be:

- Embedded directly into pull request comments.
- Displayed as part of the CI job logs.

- Forwarded to a developer dashboard or Slack bot.

8.5. Version Control and Policy Updates

Security-as-code policies (YAML/JSON) will be stored in a Git repository alongside source code. This enables:

- Traceable policy changes.
- Rollback of incorrect enforcement logic.
- Pull request reviews for policy updates.

8.6. Security Hardening

To protect the agents and the framework itself:

- All Docker containers will follow minimal base images (e.g., Alpine Linux).
- Agents will run with least privilege (no root access).
- Continuous security scanning (e.g., Trivy, Clair) will be applied to agent images.

8.7. Developer Onboarding

To ensure smooth adoption:

- A CLI tool or web UI will guide developers in customizing agent thresholds and policies.
- Documentation, FAQs, and sample projects will be maintained in a public repository.
- Feedback mechanisms will allow users to report false positives or request new features.

Table 5. Comparison between Traditional AI and Secure AI

Aspect	**Traditional AI**	**Secure AI**
Primary Objective	Accuracy and Performance	Security, Privacy, Robustness
Data Handling	Centralized training, often on public data	Federated/private training on sensitive data
Security Focus	Minimal or ad hoc security checks	Core principle integrated throughout lifecycle
Vulnerability	Prone to adversarial attacks	Designed to mitigate adversarial threats
Tooling	Generic ML toolchains	Security-hardened pipelines with monitoring

9. Future Work

Future development includes deploying a full prototype of SecureAI-Flow as a cloud-native microservices suite. Each agent will be containerized and managed via Kubernetes for scalability. We aim to benchmark its latency and threat detection accuracy on datasets like CICIDS, NSL-KDD, and AI code repositories from GitHub.

Further work will explore federated agent cooperation-where multiple instances of SecureAI-Flow installed in different organizations share anonymized threat data, forming a global intelligence layer. This would enable cross-institution learning without compromising proprietary data.

Finally, integration with large-scale DevSecOps platforms and compliance with AI regulations such as EU AI Act and ISO/IEC 42001 will be explored to make SecureAI-Flow enterprise-ready.

Table 6. Feature Comparison with AI Security Tools

Feature	**SecureAI-Flow**	**MLSecOps**	**Adversa CI**
Real-time Model Monitoring	✓	Partial	✓
Explainable Security Alerts	✓	✗	Partial
Automated Rollback	✓	✓	✗
Data Provenance Tracking	✓	✗	✗
Threat Intelligence Integration	✓	Partial	✓

10. Conclusions

SecureAI-Flow offers a timely and detailed framework designed to tackle the security issues associated with the continuous integration and deployment (CI/CD) of AI software. By integrating security practices throughout each phase of the development lifecycle, this adaptable and modular solution guarantees that AI systems are not only efficient but also robust against adversarial threats, data manipulation, and vulnerabilities that may arise during deployment. The incorporation of autonomous agents within SecureAI-Flow improves decision-making and automated response capacities, allowing for real-time surveillance and response to potential security threats. Additionally, the application of explainable machine learning (XML) methods enhances the clarity of AI models, promoting trust and accountability, particularly crucial in sensitive fields like finance, healthcare, and cybersecurity.

References

1. M. Tufano et al., "AutoDev: Automated AI-Driven Development," *arXiv preprint arXiv:2403.08299*, Mar. 2024.
2. G. Kambala, "Intelligent Software Agents for Continuous Delivery: Leveraging AI and Machine Learning for Fully Automated DevOps Pipelines," *Iconic Research And Engineering Journals*, vol. 8, no. 1, pp. 662-670, Jul. 2024.
3. A. Goyal et al., "Optimizing Continuous Integration and Continuous Deployment Pipelines Through Machine Learning," *Advances in Science and Technology Research Journal*, vol. 19, no. 3, pp. 108-120, Feb. 2025.
4. A. Nishat, "Enhancing CI/CD Pipelines and Container Security Through Machine Learning and Advanced Automation," *EasyChair Preprint No. 15622*, Dec. 2024.
5. J. Loevenich et al., "Design and Evaluation of an Autonomous Cyber Defence Agent Using DRL and an Augmented LLM," *SSRN Preprint 5076836*, Dec. 2024.
6. L. Chen et al., "Large Language Model-Based Agents for Software Engineering: A Survey," *arXiv preprint arXiv:2408.02479*, Aug. 2024.
7. R. Shokri et al., "Low-Cost High-Power Membership Inference Attacks," *arXiv preprint arXiv:2312.03262*, Dec. 2023.
8. B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks against Support Vector Machines," in *Proc. 29th International Conference on Machine Learning*, 2012.
9. R. Kalva, "The Evolution of DevSecOps with AI," *Cloud Security Alliance Technical Report*, Nov. 2024.
10. OWASP Foundation, "ML03:2023 Model Inversion Attack," *OWASP Machine Learning Security Top 10*, 2023.
11. S. Wang et al., "Federated Learning for Cloud and Edge Security: A Systematic Review," *MDPI Electronics*, vol. 13, no. 5, 2024.
12. Google Research Team, "Securing the AI Software Supply Chain," *Google Technical Report*, 2023.
13. ISO/IEC, "ISO/IEC 42001:2023 Artificial Intelligence Management System Standard," International Organization for Standardization, 2023.
14. European Commission, "EU AI Act: Framework for Trustworthy Artificial Intelligence," Official Journal of the European Union, Nov. 2024.
15. XenonStack AI Team, "Top 7 Layers of MLOps Security," XenonStack Technical Guide, Jun. 2023.
16. B. Green, "Threat Modeling the Artificial Intelligence Pipeline," *IriusRisk Security Journal*, vol. 4, no. 2, 2023.
17. E. Vanderburg, "Explainable AI in Cybersecurity - Ensuring Transparency in Decision Making," *LinkedIn Engineering Blog*, Oct. 2024.
18. AppSOC Team, "AI is the New Frontier of Supply Chain Security," *AppSOC Security Whitepaper*, Dec. 2024.
19. TrojAI Research Group, "Driving Secure AI Practices in the AI Supply Chain," *TrojAI Security Report*, May 2025.