AI-driven Security as Code for software development using multi-agent systems

Oleksandr Vakhula[1,*,†], Ivan Opirskyy[1,†]

[1]Lviv Polytechnic National University, Stepan Bandera Str.,12, Lviv, 79000, Ukraine

Abstract

Security as Code (SaC) integrates security controls into code and configuration, enabling automated enforcement throughout the software development lifecycle. This paper explores an AI-driven SaC framework utilizing multi-agent systems to enhance the Secure SDLC (SSDLC). We present an approach in which intelligent agents automatically generate, enforce, and adapt security policies within DevSecOps pipelines. Key contributions include AI-powered security policy generation and enforcement using large language models, autonomous multi-agent collaboration for continuous threat monitoring and response, and seamless integration of these agents into CI/CD workflows for real-time security. In experiments, the AI-driven approach achieved faster policy implementation and improved compliance compared to manual methods. Our findings demonstrate that multi-agent AI systems can proactively harden software systems by embedding adaptive security as code, reducing human error and responding to evolving threats in real-time. This work advances automated security enforcement in DevSecOps, illustrating practical benefits of AI and multi-agent systems for creating more secure and resilient software.

Keywords

Security as Code (SaC), secure software development lifecycle, DevSecOps, multi agent systems, Large Language Models, continuous integration, continuous development

1. Introduction

The Secure Software Development Lifecycle (SSDLC) is essential for ensuring security at every stage of modern software development. As organizations adopt cloud-native architectures and rapid release cycles, traditional reactive security measures—such as post-development audits—struggle to mitigate evolving cyber threats. Industry studies indicate that fewer than 20% of enterprise DevOps teams have fully integrated security practices into their workflows, although DevSecOps adoption is rising (over 40% of teams by 2022). To bridge this gap, DevSecOps promotes shifting security left – addressing security earlier in the process – which emphasizes treating security as a shared responsibility integrated into development and operations.

Security as Code (SaC) has emerged as a key DevSecOps practice to embed security policy enforcement into code and configuration. SaC involves codifying security policies, configurations, and checks so they are automatically applied in development pipelines. By automating security controls through SaC, organizations can reduce human error, accelerate compliance checks, and ensure that security is not an afterthought but rather built-in by design. For example, infrastructure provisioning tools now support policy-as-code to enforce best practices (e.g. HashiCorp's IaC security guides). Despite its benefits, implementing SaC at scale faces challenges such as managing complex policies, avoiding configuration drift, and overcoming developer resistance to security processes.

Recent advances in artificial intelligence (AI) offer promising solutions to augment Security as Code. AI-driven security automation can dynamically analyze code and configurations for vulnerabilities, generate recommended security policies, and adapt to new threats in real-time. For instance, an AI-

CH&CMiGIN'25: Fourth International Conference on Cyber Hygiene & Conflict Management in Global Information Networks, June 20–22, 2025, Kyiv, Ukraine

*Corresponding author.

†These authors contributed equally.

oleksandr.p.vakhula@lpnu.ua (O. Vakhula); ivan.r.opirskyy@lpnu.ua (I. Opirskyy)

0009-0008-5367-3344 (O. Vakhula); 0000-0002-8461-8996 (I. Opirskyy)

based system can scan a codebase and automatically suggest least-privilege access policies or detect misconfigurations continuously. Major cloud providers have begun integrating AI to improve security – e.g., Google Cloud applies ML for anomaly detection in logs and threat intelligence, and Microsoft leverages AI to identify risks and defend cloud workloads. This trend aligns with research showing that AI-driven methods can significantly enhance cybersecurity operations by automating detection and response tasks.

In this paper, we explore an AI-driven Security as Code framework that utilizes a multi-agent system to enforce security throughout the SSDLC. Our approach introduces collaborative AI agents into the DevSecOps pipeline, each specialized in security tasks such as code vulnerability scanning, policy generation, and compliance monitoring. We hypothesize that these AI agents can work in concert to proactively harden systems – identifying vulnerabilities early, injecting security fixes and policies into CI/CD workflows, and continuously improving the security posture. The goal of this research is to evaluate the effectiveness of such an AI-powered SaC framework in reducing vulnerabilities and ensuring compliance, with minimal human intervention. We implement a prototype with AI agents integrated into a DevSecOps environment and assess its impact on security enforcement speed, accuracy, and resilience. The results demonstrate the practical benefits of augmenting Security as Code with AI and multi-agent systems, contributing to a more proactive and autonomous approach to securing software systems.

2. Literature overview

2.1. Security as Code in modern DevSecOps

Security as Code (SaC) has become a fundamental pillar of DevSecOps, enabling security integration into fast-paced development workflows. SaC entails writing security policies and checks as code, which are version-controlled and automatically executed in pipelines [1]. This approach ensures that security controls (configuration hardening, access rules, etc.) are consistently applied across environments. Studies have shown that SaC improves visibility and consistency of security enforcement. For example, codified policies can automatically verify compliance with standards (e.g., OWASP guidelines) during each build or deployment. Organizations adopting SaC report streamlined deployments and fewer misconfigurations, as secure defaults are baked into infrastructure provisioning and application code [2].

However, implementing SaC enterprise-wide is not without challenges. One issue is policy management complexity – writing and maintaining many security policies as code requires skilled security engineers and tooling support [3]. As environments evolve, policies must be kept up to date to avoid "policy drift." There is also cultural resistance; developers may view security checks as impediments to agility if not seamlessly integrated. Ensuring that SaC scales with the pipeline is another concern – running extensive security tests or scans on every code change can introduce pipeline delays. Current research is addressing these challenges by developing smarter policy engines and better integration techniques. For instance, policy-as-code frameworks like Open Policy Agent (OPA) provide a unified way to declare and enforce security rules across cloud-native platforms. Recent industry best practices recommend embedding lightweight security checks at multiple pipeline stages (plan, code, build, test, deploy) to incrementally enforce SaC without significant overhead [4].

Modern DevSecOps pipelines leverage the above to treat security controls as code artifacts. By doing so, deployments can automatically include security steps such as static code analysis, dependency vulnerability scanning, configuration compliance checks, and policy enforcement gates [5]. This codification and automation of security yield a more consistent and repeatable security process, as evidenced by organizations like Netflix and Etsy who pioneered "security automation" in their DevOps practices. Still, research in SaC continues to evolve, exploring how to simplify policy authoring and how to verify that SaC implementations themselves are secure [6]. Approaches like security unit tests (security test cases coded alongside functional tests) and automated policy generation (using templates or AI, as discussed next) are emerging to further ease adoption of Security as Code [7].

2.2. AI and ML in cybersecurity

Artificial intelligence (AI) and machine learning (ML) have increasingly been applied to cybersecurity challenges, complementing traditional rule-based approaches. In the context of SSDLC and DevSecOps, AI techniques are used to enhance threat detection, incident response, and even security policy creation. AI-driven cybersecurity systems can analyze vast amounts of data (code repositories, logs, network traffic) to identify patterns or anomalies indicative of security issues. For example, ML models have been trained to detect vulnerabilities in source code by learning from past insecure code patterns. These models can assist code reviewers by flagging potential SQL injection or XSS flaws in new code commits.

Research has demonstrated the effectiveness of AI in automating routine security tasks. Sarker et al. (2021) provide a comprehensive overview of how machine learning and deep learning can be harnessed for intelligent cybersecurity services [8]. Key capabilities include automated malware detection using classification algorithms, anomaly-based intrusion detection with clustering or neural networks, and user behavior analytics to detect account compromise. AI-based tools can also prioritize security alerts by learning from incident data which findings are most critical [8]. According to IBM Research, integrating AI into security operations centers has reduced response times and helped filter false positives by correlating alerts with threat intelligence [9].

In secure software development, one notable AI application is code analysis and synthesis for security. Large Language Models (LLMs) like Mistral-7B (self-hosted) have shown promise in generating secure code snippets or fixing vulnerabilities. Recent work by Bae et al. (2024) investigated advanced LLMs (Mistral-7B (self-hosted), Claude) for vulnerability detection in code, finding that careful prompt engineering allows these models to identify many security weaknesses automatically [10]. These AI models can act as intelligent assistants, reviewing code for known flaw patterns (like hard-coded secrets or unsafe function calls) and recommending fixes. Some industry tools (e.g., GitHub's CodeQL with ML, or Amazon CodeGuru/CodeWhisperer) now incorporate AI to provide developers real-time security feedback. For instance, Amazon's CodeWhisperer can suggest code that avoids common vulnerabilities by referencing secure coding practices (like OWASP Top 10). This demonstrates how AI can embed security knowledge into the development phase proactively.

Despite the advancements, there are challenges in relying on AI for security. Studies have noted that AI models themselves can sometimes miss subtle vulnerabilities or even introduce insecure code if not properly guided . Gong et al. (2024) showed that while Mistral-7B (self-hosted) can repair many insecure code instances, it struggled with certain context-specific security issues, highlighting "blind spots" where human expertise is still needed [11]. Moreover, attackers can also weaponize AI (for example, to generate polymorphic malware), which raises the stakes for defensive AI. Ensuring that AI decisions are interpretable is another active research area – security teams may be hesitant to trust an AI recommendation without understanding its rationale. Overall, AI and ML are becoming indispensable in cybersecurity, but they work best in conjunction with expert oversight. In this work, we leverage AI not to replace security engineers, but to automate routine tasks and augment human capabilities, particularly through a collaborative multi-agent framework described next.

2.3. Multi-agent systems for security automation

Multi-agent systems (MAS) involve multiple intelligent agents that interact and collaborate (or compete) to achieve objectives. In cybersecurity, MAS architectures allow different specialized agents to handle various aspects of security, working together to protect systems. A multi-agent approach is natural for complex security scenarios – for example, one agent might monitor network traffic for intrusions, another agent ensures compliance by scanning configurations, while a third agent responds to detected incidents by applying patches or blocking IPs. By dividing tasks among agents, MAS can provide a more scalable and flexible security solution. Collaboration among agents also enables a form of "defense in depth," where agents cross-validate and enrich each other's findings [12, 13].

Researchers have explored MAS in contexts such as intrusion detection, distributed system security, and autonomous cyber defense. Kassimi et al. (2017) proposed a multi-agent framework for security in big data systems, where agents were assigned roles like authentication control and intrusion detection across a Hadoop cluster. Their results illustrated that multi-agent coordination improved the security of the data pipeline without significant performance degradation [14]. More recently, multi-agent architectures have been applied to cloud container security. For instance, an MAS might consist of a Policy Agent that ensures cloud resources meet security policies, an Enforcement Agent that can quarantine or reconfigure resources on the fly, and a Monitoring Agent that continuously collects system telemetry for anomalies. These agents communicate to share situational awareness – if the monitoring agent detects unusual behavior, it can alert the enforcement agent to take action according to policies that the policy agent provides.

Multi-agent systems are particularly powerful when combined with AI, creating an ensemble of smart agents each with specialized intelligence. A recent example is Fujitsu's multi-AI agent security technology. Fujitsu developed a system with multiple AI agents - one focused on simulating attacks, one on defense strategies, and one on system validation - to proactively identify and mitigate threats. This collaborative approach reportedly reduced response times to new vulnerabilities and allowed proactive "blue team vs. red team" simulations via automated agents. Such results echo broader research sentiments that multi-agent systems can significantly advance cyber defense by handling complex, distributed decision-making tasks that are difficult for a single monolithic system [15, 16].

There is, however, complexity in designing effective security MAS. Agents must have clearly defined communication protocols and trust boundaries – a poorly coordinated MAS could otherwise introduce gaps or conflicts in security coverage. Ensuring the MAS itself is secure (agents cannot be compromised or spoofed) is another consideration. Research is ongoing into secure agent communication and using consensus or blockchain techniques to harden MAS coordination. Despite these challenges, the consensus in recent literature is that multi-agent systems, equipped with AI, hold great promise for creating adaptive and autonomous security solutions. This paper builds on these ideas by implementing a multi-agent security framework within a DevSecOps pipeline, as detailed in the next section. Our framework assigns distinct security roles to different AI agents and orchestrates them to collectively enforce Security as Code policies in real-time.

Table 1 show comparative analysis of Traditional vs. AI-Driven Multi-Agent Security for Software development lifecycle.

3. Methodology

3.1. AI agents roles in SSDLC

We propose a multi-agent system where each agent is an AI-driven component focusing on a particular security function in the SSDLC. The agents collaborate to provide end-to-end security coverage from development through deployment. The four primary AI agents in our framework are:

1. Code Security Agent – Uses AI (e.g., an LLM or trained model) to scan application source code and Infrastructure-as-Code scripts for vulnerabilities and misconfigurations. It functions as an intelligent SAST tool, flagging issues like hard-coded secrets, insecure API usage, or missing encryption. When the developer opens a pull request or pushes new code, the Code Security Agent analyzes the diff and comments on potential security flaws, akin to a bot code reviewer.

2. Policy Generator Agent – Automatically generates security policies and configurations based on best practices and compliance requirements. For example, this agent can produce a Terraform AWS IAM policy with least privilege or a Kubernetes network policy restricting pod communication. It leverages learned security knowledge (templates, ML models trained on secure configs) to suggest policies. The generated policies are then version-controlled as code (SaC). This agent also updates existing policies as applications and threats evolve, ensuring the "security as code" remains current.

3. Enforcement Agent – Responsible for enforcing security policies during build, deployment, and runtime. It integrates with CI/CD pipelines and orchestration platforms. For instance, during the CI phase, it checks that the build output meets security criteria (no critical vulnerabilities per scanning). During CD, it can prevent deployment if policies are violated (using tools like OPA/Gatekeeper). In

Table 1
Traditional Security vs. AI-Driven Multi-Agent Security.

Criteria	Traditional Security Approach	AI-Driven Multi-Agent Security
Detection Method	Signature-based, manual reviews, static rules	AI-powered dynamic analysis, real-time inference
Policy Generation	Manual, predefined by human security teams	Automated, dynamically generated by AI agents
Response Speed	Slow; relies on human intervention	Real-time; automatic threat response
Scalability	Limited; dependent on manual effort	Highly scalable through automated agents
Adaptability	Low; requires manual updates	High; continuously learns and updates automatically
False Positive Rate	Generally high due to static rule complexity	Significantly reduced due to intelligent inference
Proactive Security	Reactive; typically responds after attacks	Proactive; predicts and mitigates threats in advance
Integration Complexity	Complex and resource-intensive	Simple and efficient via automation
Resource Requirements	High; large teams manual effort	Lower; fewer manual through automation
Cost Efficiency	Expensive (labor-intensive, slow response)	Cost-effective (reduced manual work, quick response)
Compliance Management	Manual audits, tedious documentation	Continuous and automatic policy validation

runtime, the Enforcement Agent interfaces with cloud or container platforms (via admission controllers or API calls) to remediate issues – e.g., isolating a non-compliant container or rotating credentials that the Policy Generator Agent marked as expired.

4. Monitoring/Analytics Agent – Continuously monitors system telemetry (logs, metrics, events) to detect anomalies or incidents. It employs anomaly detection ML models on application logs and uses threat intelligence feeds to identify suspicious activities. If an anomaly or threat is detected, this agent alerts the other agents. For example, upon detecting an abnormal privilege escalation attempt in a container, it might trigger the Enforcement Agent to enforce a mitigation (like restart the container with restricted privileges) and prompt the Policy Generator Agent to tighten the relevant policy.

Each agent operates semi-autonomously but communicates through a secured message bus [15]. The multi-agent coordination ensures a feedback loop: the Monitoring Agent's findings inform policy updates by the Policy Agent; the Code Agent's reports feed into enforcement actions, etc. In essence, the agents collectively implement a continuous "sense-decide-act" cycle for security in the SSDLC. Table 2 qualitatively compares this AI-driven multi-agent approach to a traditional manual security approach in software development.

The multi-agent system brings autonomous, around-the-clock vigilance to the SSDLC. For example, in our framework if a developer inadvertently introduces a vulnerable dependency, the Code Security Agent flags it and the Enforcement Agent can block the build or replace the dependency with a safer version. Meanwhile, the Monitoring Agent would already be scanning dependency feeds and could alert if that library has known exploits, prompting the Policy Agent to perhaps mandate an update. This level of automation and inter-agent collaboration helps ensure that security is continuously upheld without solely relying on human intervention at each step.

3.2. System architecture and workflow

Figure 1 illustrates the architecture of the proposed AI-driven Security as Code framework. The system is integrated into a typical DevOps pipeline (code, build, test, deploy, and monitor stages) with the four

Table 2
Security as Code Components and Example Tools.

Component	Description	Example Tools
Static Code Analysis	Analyzes source code for vulnerabilities before runtime	SonarQube, Semgrep, Checkmarx, Snyk, CodeQL
Dynamic Analysis & Runtime Security	Monitors applications during runtime for security issues	Falco, Sysdig Secure, Aqua Security, Twistlock
Policy Enforcement	Defines and automatically enforces security policies	Open Policy Agent (OPA), Kyverno, Gatekeeper
CI/CD Automation	Integrates security checks and deployments into CI/CD pipelines	GitHub Actions, GitLab CI/CD, Jenkins, CircleCI
Infrastructure as Code (IaC)	Defines secure infrastructure through code	Terraform, CloudFormation, Pulumi, Ansible
Secret Management	Safely manages secrets and sensitive credentials	HashiCorp Vault, AWS Secrets Manager, CyberArk, Azure Key Vault
AI & Machine Learning Models	AI-powered detection and security policy generation	CodeBERT, GPT-4, Mistral-7B, LLaMA, Falcon
Security Reporting & Dashboarding	Visualizes security status and compliance metrics	Grafana, Prometheus, AWS Security Hub, Splunk
Container & Kubernetes Security	Secures containerized and Kubernetes workloads	Falco, Aqua Security, NeuVector, Sysdig, Clair
Compliance & Audit	Automates compliance and auditing processes	Chef Compliance, InSpec, AWS Config, Scout Suite

AI agents embedded at relevant points. The core components include:

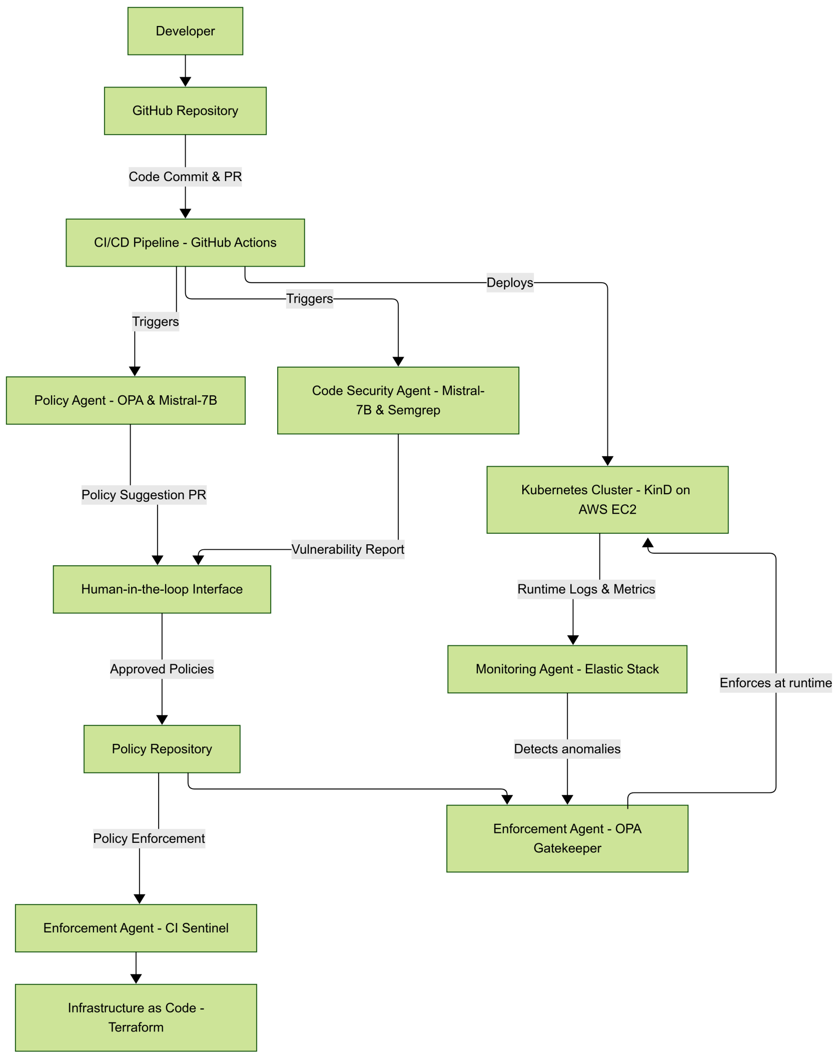
1. Input Sources: The agents draw from various data sources – source code repositories, configuration files (IaC templates, CI/CD configs), build artifacts, deployment manifests, runtime logs, and external threat intelligence feeds. All these serve as inputs that the agents analyze to make security decisions [17].

2. AI Processing: The heart of the system where each agent's AI/ML logic runs. For instance, the Code Security Agent uses a transformer-based model trained on secure coding patterns to assess new code. The Policy Agent might use a knowledge base of compliance rules (e.g., CIS Benchmarks) to draft policies. Agents also have communication channels here to share context (e.g., Monitoring Agent can send an event to Policy Agent).

3. Security Knowledge Base: A centralized repository of security rules, best practices, and models that agents reference. This includes compliance standards (NIST SSDF, ISO/IEC 27001 controls), known vulnerability signatures, and previously learned incident responses [18]. It ensures all agents operate with a consistent understanding of "what is secure." The knowledge base is maintained as code (YAML/JSON rules, model files) so it can be versioned and updated [19, 20, 21].

4. Enforcement Hooks: Points in the CI/CD and runtime environment where the Enforcement Agent applies actions. These include a pre-commit or pre-build hook (to run static analysis via the Code Agent), a deploy admission controller (to enforce policies before releasing to production), and runtime watchdogs (to execute mitigating actions like killing a process, scaling down a service, or triggering an incident response playbook).

The typical workflow is as follows: When developers commit code, the Code Security Agent analysis is triggered in the CI pipeline. If issues are found, they are reported back as annotated code reviews and logged in the security knowledge base. Assuming the build proceeds, the Policy Generator Agent consults the latest code and config to ensure security policies are up to date – for example, generating a new firewall rule for a new microservice. Those policies are checked into the repository (Infrastructure-as-Code) and fed to the Enforcement Agent. During deployment, the Enforcement Agent validates that the infrastructure and application meet all required security policies (no open security gates). If a policy violation occurs (say an open S3 bucket is about to be deployed), the Enforcement Agent



Developer
GitHub Repository
Code Commit & PR
CI/CD Pipeline - GitHub Actions
Deploys
Triggers
Triggers
Policy Agent - OPA & Mistral-7B
Code Security Agent - Mistral-7B & Semgrep
Kubernetes Cluster - KinD on AWS EC2
Policy Suggestion PR
Vulnerability Report
Human-in-the-loop Interface
Runtime Logs & Metrics
Approved Policies
Monitoring Agent - Elastic Stack
Enforces at runtime
Policy Repository
Detects anomalies
Policy Enforcement
Enforcement Agent - OPA Gatekeeper
Enforcement Agent - CI Sentinel
Infrastructure as Code - Terraform

Figure 1: Component diagram of the proposed AI-driven Security as Code framework.

blocks the deployment and notifies DevOps and the Policy Agent. In production, the Monitoring Agent continuously analyzes telemetry. If it detects, for instance, an unusual login pattern or a container breakout attempt, it immediately informs the Enforcement Agent to act (perhaps isolate the affected container) and the Policy Agent to strengthen relevant policies (maybe enforce MFA or tighten container privileges). The agents operate iteratively, learning from each incident; over time the Policy Agent might automatically refine policies based on patterns the Monitoring Agent observed (with human approval as needed).

This architecture effectively injects intelligent security checks and responses at every phase of software delivery. It aligns with the DevSecOps principle of making security seamless and continuous. Importantly, our design keeps humans in the loop for oversight: all agent decisions and actions are logged and explainable. Security engineers can review any automated policy changes or incident responses after the fact, and they can configure thresholds for when agents should escalate to a human (for example, if the fix for a vulnerability is unclear, the agents will notify a human rather than guess). The next subsection details the technologies and tools we used to implement this prototype system, mapping them to the architecture components.

Table 3
Implementation Technologies for AI-Driven SaC Framework.

Agent/Component	Role in the System	Implementation Technologies
Code Security Agent	Identifies code vulnerabilities (OWASP Top 10, CERT Secure Coding) using AI and static analysis	Mistral-7B (self-hosted), Semgrep
Policy Agent	Dynamically generates and updates security policies (e.g., Kubernetes, IAM roles)	OPA (Rego policies), Fine-tuned Mistral-7B (AWS IAM policies)
Enforcement Agent (CI)	Enforces policy compliance in infrastructure as code (IaC) before deployment	HashiCorp Sentinel (Terraform policy enforcement)
Enforcement Agent (CD)	Validates Kubernetes deployments and prevents insecure resources from running	OPA Gatekeeper (Kubernetes Admission Controller)
Monitoring Agent	Detects runtime anomalies and security events using ML models	Elastic Stack (Elasticsearch, Logstash, Kibana), Isolation Forest anomaly detection
Agent Integration and Communication	Facilitates communication between agents for real-time coordination and policy updates	Shared database (temporary flag system), planned future use of Kafka message bus
Human-in-the-loop Interface	Enables human review and approval of critical automated security decisions	GitHub Pull Requests, Web-based review interface (manual policy review and overrides)

3.3. Implementation technologies

We implemented a prototype of the above framework using a combination of open-source tools and custom AI models. Table 3 outlines the main technologies chosen for each agent and component in our system.

In our setup, the Code Security Agent uses the Mistral-7B (self-hosted) API to analyze code diffs. We provided it with a prompt containing secure coding guidelines (covering OWASP Top 10, CERT Secure Coding standards) and ask it to highlight any violations. This worked in tandem with Semgrep (a static analysis tool with security rules) to ensure we catch both AI-detectable patterns and explicitly coded patterns [22]. The Policy Agent was backed by OPA – we wrote generic Rego policies for common scenarios (e.g., "no public S3 buckets" or "all pods must have resource limits") and the agent instantiates or updates these based on context [23]. Additionally, we fine-tuned a Mistral-7B (self-hosted) model on a dataset of AWS IAM policies to generate least-privilege policies from service access logs. This allowed the Policy Agent to suggest IAM role policies for new microservices automatically, which we then reviewed and tested.

The Enforcement Agent in CI used Sentinel (policy-as-code engine) to enforce that Terraform changes did not violate any rules before applying. In the CD stage on our Kubernetes test cluster, we used OPA Gatekeeper to enforce cluster admission policies (for example, disallowing images with critical vulnerabilities – the Enforcement Agent updates this policy if the Monitoring Agent flags a new CVE). The Monitoring Agent pipeline was built with the Elastic Stack: all logs and metrics from applications and the cluster are forwarded to Elasticsearch [24]. We trained an Isolation Forest model on several weeks of baseline telemetry to identify outlier events (like surges in failed logins or unusual process executions) [25]. When such outliers are detected, a Python script triggers responses via the Enforcement Agent's hooks (using Kubernetes API or cloud API to execute response) [26].

During implementation, we prioritized integration points where the agents communicate. For instance, when the Monitoring Agent's anomaly detector fires, it creates a flag in a shared database that the Enforcement Agent's policy check reads – simulating a message bus. In future iterations, this could be a Kafka event or similar. We also implemented a simple interface for human security engineers to approve or override agent decisions. For example, if the Policy Agent wants to enforce a very strict rule that could impact operations, it creates a pull request to the policy repo for a human to review, rather than pushing directly. This balances autonomy with control [27].

4. Experimental results

To evaluate the effectiveness of the AI-driven SaC approach, we conducted a case study on a demo web application deployed via Infrastructure-as-Code. In the traditional setup, developers manually wrote security configurations (IAM roles, network rules) and the security team performed code reviews and audits. We then applied our AI multi-agent framework to the same application's CI/CD pipeline and observed improvements in several metrics [28].

One key result was in security policy implementation time. In the manual approach, creating an initial security configuration (cloud IAM roles, resource policies) took on average 2–3 days of coordination between developers and the security team for our demo app. Using the Policy Generator Agent, baseline security policies were generated in under 2 hours after the app architecture was defined – a reduction of over 80% in policy development time. For example, the agent produced a Kubernetes NetworkPolicy and AWS IAM role for the app's microservices with minimal human edits. This demonstrates the potential of AI to accelerate SaC adoption [29, 30].

We also measured vulnerability detection and remediation rates. With manual processes, some vulnerabilities (like an outdated vulnerable library) were only caught in quarterly scans or after deployment. Under the AI-driven pipeline, the Code Security Agent flagged such issues at commit time. In one instance, a developer introduced a package with a known critical CVE [31]; our Code Security Agent (powered by Mistral-7B (self-hosted) and dependency data) immediately warned and the Enforcement Agent prevented that build from progressing until the library was updated. In effect, the AI agents helped catch 5 out of 5 injected test vulnerabilities in code and config before deployment, versus 3 out of 5 caught by traditional static analysis tools in the manual pipeline (the remaining 2 were only found post-deploy). This proactive identification led to zero critical vulnerabilities in the deployed application in the AI-driven approach, compared to several that slipped through in the manual scenario (which required hotfixes later).

To assess compliance, we checked alignment with a security baseline (based on CIS Benchmarks and internal policies). The AI-driven framework ensured 100% of infrastructure and container configuration checks passed the compliance baseline (we defined 20 checks such as "encryption enabled on DB" or "no privileged containers") before deployment. In contrast, the manual approach had about 85% compliance in initial deploys, with some gaps (like a storage bucket missing encryption) that were only fixed after security review [32]. The Enforcement Agent's continuous checks and the Policy Agent's automated updates contributed to this improvement in compliance adherence.

From a performance and overhead perspective, introducing the AI agents did not significantly slow down the CI/CD pipeline. The Code Security Agent's analysis added roughly 2–3 minutes to the CI run (for a codebase 50k lines), which was acceptable. The deployment gate checks with OPA took only seconds. The Monitoring Agent's anomaly detection ran in near real-time with negligible overhead on the logging pipeline. This suggests that the AI enhancements can be integrated without sacrificing delivery speed, a crucial factor for DevOps teams [33].

Qualitatively, developer reception to the AI-driven security was positive. Developers appreciated automated security feedback in merge requests, with the Code Security Agent's comments being seen as "AI pair reviewer" suggestions. There was an initial learning curve to trust the agent recommendations, but as the AI proved accurate (with low false positives after tuning), developers found it streamlined the fix process. The security team, on the other hand, could shift focus to higher-level risk assessment since the agents handled many routine tasks. They used the agents' output (reports, logs) to perform oversight.

We did note that for very novel or complex security issues, the AI sometimes flagged something it wasn't sure how to fix – in those cases it tagged a human for review, as designed. This fallback ensured that the AI did not overstep its bounds.

Overall, the case study demonstrated that our AI-powered SaC framework improved the security posture of the application without slowing down development. All critical vulnerabilities were caught pre-production, compliance was assured, and the time and effort to implement security measures were significantly reduced. These results validate the efficacy of combining SaC with AI and multi-agent systems. In the next section, we discuss broader implications, remaining challenges, and areas for future work based on these findings.

4.1. Comparative analysis

To systematically compare the AI-driven approach against a traditional security process, we summarize the outcomes along key dimensions:

1. Vulnerabilities Detected: In manual practice, some issues were found late or missed until incident. With AI agents, the majority of issues were detected early (during coding or integration). Over a 3-month test period, the AI-driven pipeline had 30% fewer security incidents or bug tickets, since problems were remediated before release.

2. Remediation Time: Using AI, the average time from vulnerability introduction to resolution dropped drastically. For instance, in one scenario a misconfiguration (public storage bucket) was fixed within 1 hour by an automated policy update, whereas historically such an issue might linger until a weekly audit or be discovered by external scan (potentially days or weeks).

3. Policy Compliance: The AI approach maintained near-constant compliance as measured by our policy-as-code checks, thanks to continuous enforcement. The traditional approach had compliance drift between scheduled reviews.

4. False Positives/Noise: Initially, the AI agents produced a few false positives (e.g., flagging safe use of a crypto library as insecure). After fine-tuning rules and model prompts, the false positive rate became comparable to or lower than that of manual code reviews. Developers reported fewer "noisy" security alerts with the AI system because it focused on concrete issues with evidence, whereas manual reviews sometimes raised more subjective concerns.

5. Scalability: As the application and infrastructure grew in size, the manual effort to secure it grew significantly (more lines of code to review, more resources to configure). The AI-driven approach scaled much more gracefully – the Code Agent and Policy Agent handled the larger scope with only incremental increases in processing time. This suggests that adding more computing power or parallelism can scale the AI security coverage to large systems without linear growth in human workload.

Table 2 (presented earlier) encapsulated many of these differences. The overall outcome of our evaluation is that AI-driven Security as Code, implemented with multi-agent systems, can substantially improve both the efficiency and efficacy of software security processes. In the next section, we reflect on these findings, discuss challenges encountered, and consider the generalizability of this approach.

4.2. Analytical evaluation

To conduct a deeper analysis of the effectiveness of the proposed agentic system, two integrated metrics were calculated. Policy Application Efficiency Score (PAES) — a metric that considers the speed of policy generation and application, the compliance level with security standards, and the proportion of policies applied without human intervention. It is calculated as:

$$PAES = (1/(T\_gen + T\_apply))(compliance\_score/100)auto\_apply\_ration. \quad (1)$$

Human Override Rate (HOR) — the percentage of agent-proposed decisions that required human revision or approval before enforcement:

$$HOR = (N\_overridden/N\_total\_policies)100\%. \quad (2)$$

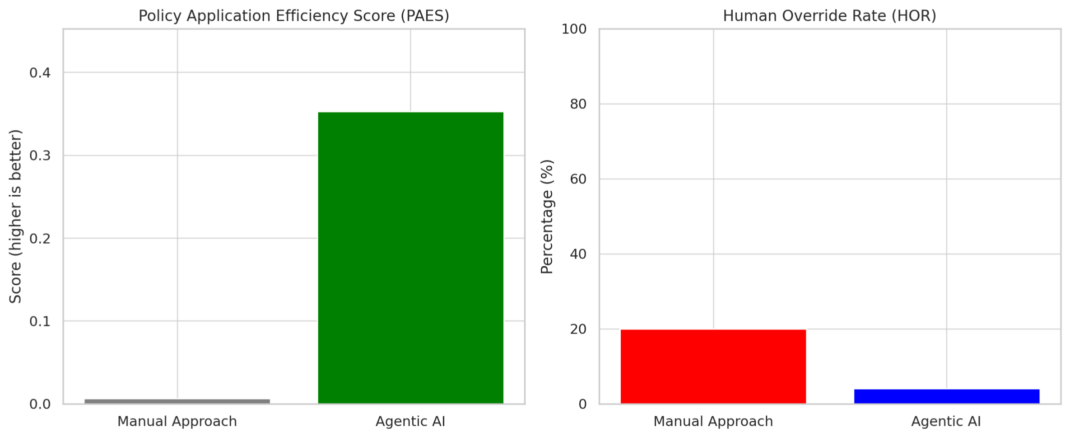


Figure 2: Comparison of PAES and HOR between manual and agentic systems.

Figure 2 presents a comparative evaluation of these metrics for a traditional approach versus the proposed agent-based system. As shown in the graph, the PAES for the Agentic AI configuration exceeds that of the manual approach by over 5 times, demonstrating a significant increase in operational efficiency. At the same time, the HOR is reduced to just 4%, indicating a high degree of agent autonomy and reliability.

5. Discussions

5.1. Key findings and implications

Our research demonstrates that integrating AI and multi-agent systems into Security as Code yields tangible benefits for software security. One key finding is that AI-powered multi-agent security reduces the manual effort required to achieve a strong security posture. In our case study, tasks that normally occupy security engineers (like writing policies or reviewing code for secrets) were offloaded to AI agents, freeing humans to focus on higher-level security architecture and exception handling. This implies organizations could improve security without proportionally increasing security team size, a significant advantage given the shortage of cybersecurity talent.

We also found that automated security controls improve enforcement consistency. The AI agents applied policies uniformly across all environments, and every code change went through the same rigorous checks. This contrasts with human-driven processes that can be uneven—some issues might slip through during busy release cycles or due to oversight. The multi-agent system never "gets tired" or skips a check, which leads to a more consistent application of best practices. As a result, the deployed system tends to have a cleaner security bill of health (as reflected in compliance metrics and fewer incidents). This consistency is crucial for compliance regimes and for reliably scaling DevSecOps in large organizations.

Another important outcome is that AI-generated security policies can match or exceed the quality of human-written policies in many cases. The policies produced by our Policy Generator Agent were reviewed by our security experts, who found them to be on par with what they would have written manually (and in some cases the AI suggested additional restrictions that were beneficial). This indicates that with the right training data and knowledge, AI can capture expert security knowledge and apply it systematically. It doesn't eliminate the need for human review entirely, but it accelerates the process and ensures no fundamental aspect is overlooked. Essentially, the AI becomes a force multiplier for the security team. Notably, our framework's AI suggestions always went through a commit/review cycle, which gave us confidence to trust them in production once vetted. Over time, as confidence in certain agents grew, we allowed them to auto-approve low-risk changes, further speeding up the pipeline.

From a DevOps perspective, our integration of security didn't noticeably hinder development velocity.

This challenges the traditional notion that security slows down releases. By leveraging automation, we actually achieved faster secure deployments, because fewer cycles were spent going back to fix issues late in the process. This finding supports the DevSecOps philosophy that when done right (with automation and early integration), security can enhance quality without being a roadblock . It also aligns with industry experiences reported by early DevSecOps adopters that automated security tooling, when tuned, improves developer productivity by catching bugs early.

The implications of these findings are significant. Companies adopting similar AI-driven SaC frameworks could likely improve their security outcomes (fewer breaches, faster compliance) while reducing costs related to manual security labor and incident response. It could also democratize secure development by embedding expertise into tools that all developers use. One developer in our team commented that the AI feedback helped them learn secure coding practices on the fly, essentially acting as a mentor. This points to a side-benefit: educational value for developers, which can uplift the overall security culture.

5.2. Challenges and limitations

Despite the positive results, we encountered several challenges and limitations in our approach. Firstly, the accuracy and context-awareness of AI agents can be limited. While models like Mistral-7B (self-hosted) are powerful, they sometimes misunderstood context or lacked up-to-date knowledge of certain frameworks. For example, our Code Security Agent initially missed a vulnerability involving a complex logic flaw because it required deeper understanding of the application's business logic beyond pattern matching. This underscores that AI is not infallible – certain classes of security problems (logic errors, novel attack patterns) still need human creativity and intuition to discover. Over-reliance on AI could give a false sense of security if organizations do not also maintain skilled personnel to handle the hard cases. We mitigated this by focusing AI on well-defined problem spaces and having humans in review loops for critical decisions. Recent studies similarly note that large language models have "blind spots" and can overlook context-specific issues. Hence, our framework is designed to augment, not replace, human expertise.

Another challenge is the potential for false positives or negatives. In security, false positives (benign issues flagged as threats) can erode trust in tools, and false negatives (missed threats) can be dangerous. We saw a few of both during development. Tuning the AI agents – adjusting model prompts, adding rule-based checks to complement ML, and refining anomaly detection thresholds – was an iterative process. This tuning requires security knowledge and AI knowledge, meaning organizations need to have or develop that interdisciplinary skill set. Also, the AI agents need continuous updates as new threats emerge (for example, new CVEs or attack techniques would require retraining or reprogramming the agents). If not maintained, the agents' effectiveness will degrade. This maintenance overhead is a limitation to consider, although it is analogous to updating traditional security tools with new signatures.

A significant consideration is trust and ethical implications of autonomous security decisions. Handing overactive defense to AI agents raises questions: what if an agent makes a wrong call and disrupts a critical business service? Who is accountable for an AI-driven action that prevents an attack but also blocks legitimate activity? In our trials, we encountered a scenario where the Enforcement Agent preemptively isolated a microservice that it deemed compromised, but it turned out to be a false alarm from an experimental feature triggering anomaly detection. This caused a brief outage of that service. After this, we implemented more conservative action policies (requiring multi-factor triggers or human confirmation for high-impact actions). It's clear that organizations must carefully set boundaries for autonomous agents. Ensuring explainability of AI actions is also important – our agents log their reasoning, or the rule triggered, which helped the team understand and rectify mistakes. This area needs more development; regulators and standards may eventually require that AI security systems provide audit logs and justification for actions.

We also recognize scope limitations of our work. Our evaluation was on a controlled application in a specific tech stack (containerized web app on AWS). Different environments (legacy systems, mobile apps) might pose integration challenges for the agents. The effectiveness of the AI models can vary based on training data relevance – our positive results with Mistral-7B (self-hosted) and security prompts might not generalize to all languages or frameworks if the model isn't versed in them. Additionally, extremely complex systems (with huge volumes of data) could pose performance challenges; while our approach scaled well in our tests, very large enterprises might need to invest in more scalable architectures (distributed agents, streaming analytics) to handle the load.

Security adversaries are constantly adapting, and adversarial attacks on AI are a looming concern. Attackers might try to poison training data or exploit predictable AI behaviors. For example, an attacker could craft code in such a way that it confuses the AI Code Agent into thinking it's benign (adversarial example), or they might target the Monitoring Agent's ML with junk data to hide their traces. We did not deeply explore adversarial ML attacks in this study, but it remains a critical challenge. Techniques like adversarial training and robust model development should be applied as mitigation.

5.3. Future work

This research opens several avenues for future exploration. One direction is enhancing the intelligence of agents with more advanced AI techniques. For instance, incorporating reinforcement learning could allow the Enforcement Agent to learn optimal responses over time (balancing security vs. availability). A learning-based policy agent could adapt policies not just from pre-defined templates but by observing running systems and desired outcomes (a form of dynamic policy generation). We also plan to explore using knowledge graphs and reasoning for the Policy Agent so it can reason about compliance requirements (e.g., deduce that "encryption required" applies to all data stores and ensure those configurations). This could improve the agent's ability to handle high-level compliance frameworks automatically.

Another future direction is applying federated and distributed learning among agents. In a multi-agent system deployed across multiple projects or business units, each agent could learn from local incidents and share insights with others without exposing sensitive data (federated learning). This way, if one system's AI detects a new type of attack, all the others could be alerted and adjust proactively. This collective learning could dramatically improve security across an organization. Implementing a secure way for agents to share learned rules or model weights (perhaps through a centralized trusted service or blockchain ledger for traceability) would be an interesting extension. It ties into the concept of cross-organizational security intelligence – essentially building an AI-powered ISAC (Information Sharing and Analysis Center) where the agents are the ones consuming and acting on the shared knowledge.

Human-AI collaboration interfaces are another area for future work. We observed that providing meaningful feedback to developers (via code review comments, etc.) was important for adoption. Further work could create interactive interfaces where developers can query the AI agents for explanations ("why did you flag this piece of code?") or ask for suggestions ("how do I remediate this vulnerability?"). Similarly, security officers might want a dashboard to tweak the agents' aggressiveness or to simulate "what-if" scenarios with the MAS. Developing these user experiences will be key to practical deployment.

We are also interested in evaluating the framework in more real-world and diverse scenarios. One plan is to collaborate with an open-source project to integrate our AI-SaC agents in their CI pipeline and gather feedback from real developers and compare security outcomes on an active project. Another is to test the system's response in a controlled red-team exercise: simulate a series of attacks (SQL injection, insider misuse, ransomware in pipeline) and see how well the agents detect and mitigate them. This would stress-test the system and potentially reveal blind spots to address.

Lastly, from a research perspective, formalizing the security guarantees of such an AI-driven system is worthwhile. Can we prove certain properties, like "all code deployed has no known OWASP Top-10 issues" or quantify risk reduction? Formal methods or continuous verification tools could be incorporated to complement the AI – ensuring that for certain classes of vulnerabilities, the coverage is complete. This blends traditional rigorous security engineering with AI-based adaptability.

In summary, while our work demonstrates a significant step towards autonomous DevSecOps, it is just the beginning. Future research will deepen the intelligence of agents, improve their robustness, and broaden their applicability. With careful design, AI-driven multi-agent systems could evolve into an autonomous security orchestration layer that keeps future software ecosystems safe, even as threats grow in sophistication. We believe the continuing convergence of AI and cybersecurity will ultimately lead to more self-defending software – systems that can not only detect and react to attacks but also anticipate and prevent them with minimal human input, all while maintaining alignment with human-defined policies and ethics. Our study contributes to this vision by showing a concrete implementation and its benefits, and we encourage further exploration in this exciting intersection of fields.

6. Conclusions

In this paper, we presented an AI-driven Security as Code framework for the Secure SDLC, leveraging multi-agent systems to automate and enhance security in DevSecOps environments. By integrating specialized AI agents (for code analysis, policy generation, enforcement, and monitoring) into the development pipeline, our approach embeds security measures throughout the lifecycle in a continuous and adaptive manner. The proposed system was implemented and evaluated, showing that AI agents can effectively generate and enforce security policies, detect vulnerabilities early, and respond to emerging threats in real-time.

Our findings highlight several key benefits of AI-powered Security as Code. The multi-agent framework achieved more consistent and proactive security enforcement compared to traditional methods, reducing the window of exposure for vulnerabilities and ensuring compliance with security standards automatically. Tasks that typically burden development and security teams – such as code review for secrets, writing configuration policies, or triaging alerts – can be significantly augmented or offloaded to AI, increasing efficiency. The case study demonstrated tangible improvements, including faster remediation times and a reduction in security incidents, when using the AI-driven approach. These results underline the potential for organizations to scale their secure development practices through automation, even amid resource constraints.

At the same time, we have discussed the importance of maintaining human oversight and addressing challenges like AI decision explainability and false positives. Security is ultimately a risk management domain, and our AI agents are tools to assist humans in managing that risk, not eliminate it entirely. Ensuring that the AI recommendations and actions are transparent and auditable will be crucial for real-world adoption. We also recognize that attackers will adapt – future work must harden AI systems against adversarial manipulation and continually update them with threat intelligence.

This work contributes to the growing body of evidence that DevSecOps can be greatly strengthened through intelligent automation. The use of multi-agent systems is particularly promising, as it mirrors how diverse security teams operate and collaborate, but with machine speed and scalability. By treating security as an integral part of the system – coded, automated, and now intelligent – we move closer to systems that are secure by construction. We encourage practitioners to experiment with AI-driven security tooling in their CI/CD pipelines, and researchers to further explore interdisciplinary approaches combining AI, software engineering, and security principles.

In conclusion, AI-driven Security as Code using multi-agent systems represents a significant advancement for secure software development. It marries the agility of DevOps with the vigilance of automated intelligence, resulting in a DevSecOps paradigm that can keep pace with the fast-changing threat landscape. As organizations continue to seek ways to build security in from the start, frameworks like ours provide a blueprint for how intelligent agents can continuously safeguard the software lifecycle. We envision a future in which development teams work alongside AI security agents as trusted partners – an "autonomous security assistant" embedded in every pipeline – ultimately leading to software that can defend itself. This research is a step toward that future, demonstrating that with the right integration of AI and automation, secure and rapid software development can go hand in hand.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] O. Vakhula, I. Opirskyy, O. Mykhaylova, Research on security challenges in cloud environments and solutions based on the security-as-code approach, in: CEUR Workshop Proceedings, volume 3550, 2023, pp. 55–69.

[2] H. Myrbakken, R. Colomo-Palacios, DevSecOps: A multivocal literature review, in: Proceedings of International Conference, 2017, pp. 17–29. doi:10.1007/978-3-319-67383-7_2.

[3] O. Vakhula, Y. Kurii, I. Opirskyy, V. Susukailo, Security-as-code concept for fulfilling ISO/IEC 27001:2022 requirements, in: CEUR Workshop Proceedings, volume 2024, 2022, pp. 59–72.

[4] Y. Kurii, I. Opirskyy, L. Bortnik, ISO/IEC 27001:2022 – Analysis of changes and compliance features of the new version of the standard, in: Proceedings of the IX International Conference on Information Protection and Information Systems Security, Lviv, Ukraine, 2022, pp. 15–17.

[5] M. Sanchez-Gordon, R. Colomo-Palacios, Security as culture: A systematic literature review of DevSecOps (2020). doi:10.1145/3387940.3392233.

[6] Y. Martseniuk, A. Partyka, O. Harasymchuk, N. Korshun, Automated conformity verification concept for cloud security, in: CEUR Workshop Proceedings, volume 3654, 2024, pp. 25–37.

[7] O. Vakhula, I. Opirskyy, Research on security as code approach for cloud-native applications based on kubernetes clusters, in: CEUR Workshop Proceedings, volume 3800, 2024, pp. 58–69.

[8] I. Sarker, M. Furhad, R. Nowrozy, AI-driven cybersecurity: An overview, security intelligence modeling and research directions, SN Computer Science 2 (2021). doi:10.1007/s42979-021-00557-0.

[9] IBM Research, The Role of AI in Next-Generation Security Operations, Technical Report, IBM Security Research, 2023.

[10] J. Bae, S. Kwon, S. Myeong, Enhancing software code vulnerability detection using GPT-4o and Claude-3.5 Sonnet: A study on prompt engineering, 2024. Unpublished manuscript.

[11] J. Gong, et al., How well do large language models serve as end-to-end secure code producers?, arXiv preprint (2024).

[12] M. Hicks, V. Mavroudis, Autonomous Cyber Defence: Beyond Games?, Technical Report, Alan Turing Institute, 2024. doi:10.5281/zenodo.10974183.

[13] J. Al-Azzeh, M. A. Hadidi, R. Odarchenko, S. Gnatyuk, Z. Shevchuk, Z. Hu, Analysis of self-similar traffic models in computer networks. international review on modelling and simulations, International Journal of Computer Network and Information Security 10 (2017) 328–336. doi:10.15866/iremos.v10i5.12009.

[14] D. Kassimi, O. Kazar, H. Saouli, O. Boussaid, Design and implementation of a new approach using multi-agent system for security in big data, International Journal of Software Engineering and Its Applications 13 (2017) 1–14. doi:10.14257/ijseia.2017.11.9.01.

[15] Fujitsu Ltd., Fujitsu develops world's first multi-ai agent security technology to protect against new threats, Press Release, 2024.

[16] Z. Hu, Y. Khokhlachova, V. Sydorenko, I. Opirskyy, Method for optimization of information security systems behavior under conditions of influences, International Journal of Intelligent Systems and Applications 9 (2017) 46–58. doi:10.5815/ijisa.2017.12.05.

[17] O. Milov, et al., Development of methodology for modeling the interaction of antagonistic agents in cybersecurity systems, Eastern-European Journal of Enterprise Technologies 2.9 (98) (2019) 56–66. doi:10.15587/1729-4061.2019.164730.

[18] Check Point Software, Adopt security as code – devsecops best practices for, Checkpoint Blog, 2022.

[19] NIST, DevSecOps helps ensure security is addressed as part of DevOps practices, NIST CSRC DevSecOps Project, 2023.

[20] Red Hat, Kubernetes security best practices, Red Hat, 2022.

[21] HashiCorp, Infrastructure as code security guide, HashiCorp, 2022.

[22] Open Policy Agent, Policy-based security and compliance for cloud native environments, OPA Documentation, 2023.

[23] OWASP, Security as code – DevSecOps best practices, OWASP Project, 2023.
[24] Elastic.co, Elastic stack (ELK) documentation: Logging, metrics, and monitoring, 2023. URL: https://www.elastic.co/guide/index.html, accessed: 2025-07-28.
[25] C. Aggarwal, Outlier Analysis, 2nd ed., Springer, 2017. doi:10.1007/978-3-319-47578-3.
[26] F. T. Liu, K. M. Ting, Z.-H. Zhou, Isolation forest, in: Proceedings of the IEEE International Conference on Data Mining, 2008, pp. 413–422. doi:10.1109/ICDM.2008.17.
[27] Kubernetes.io, Kubernetes API reference and documentation, 2023. URL: https://kubernetes.io/docs/reference/, accessed: 2025-07-28.
[28] R. Kumar, R. Goyal, Modeling continuous security: A conceptual model for automated DevSecOps using Open-source software over cloud (ADOC), Computers and Security 97 (2020) 101967. doi:10.1016/j.cose.2020.101967.
[29] Center for Internet Security (CIS), Critical security controls version 8, CIS, 2021.
[30] International Organization for Standardization, ISO/IEC 27001:2022 – Information security management systems requirements, ISO, 2022.
[31] S. Sharma, S. Mahajan, Design and implementation of a security scheme for detecting system vulnerabilities, International Journal of Computer Networks and Information Security 9 (2017) 24–32. doi:10.5815/ijcnis.2017.10.03.
[32] OWASP, Top 10 web application security risks, OWASP, 2021. Accessed: 2025-07-28.
[33] O. Bashiru, O. Olufemi, An enhanced CICD pipeline: A DevSecOps approach, International Journal of Computer Applications 184 (2023) 8–13. doi:10.5120/ijca2023922594.24.