SafeOps: An integrated platform for automated security analysis and compliance in DevSecOps pipelines

PREPARED BY

Hind Soulaimani

Maryam Chentouf

Houssam Joudar

Oussama Ben Zian

Supervised by Professor Saiida Lazaar

Departeent of Cybersecurity and cyber intelligence, ENSA Tanger

Abstract

The integration of security protocols into the software development lifecycle is often hindered by the complexity of managing disparate analysis tools. This paper presents *SafeOps*, an open-source platform designed to automate security analysis and compliance auditing within CI/CD pipelines. Built on a modular microservices architecture, the system combines rule-based vulnerability detection with unsupervised machine learning for anomaly detection. *SafeOps* addresses the usability gap in security tooling by providing an intuitive web interface that enables users to ingest pipeline logs, visualize security findings, and generate compliance reports. By enhancing the traceability and reproducibility of security assessments, the software facilitates the effective adoption of DevSecOps practices for development teams, researchers, and educators.

Keywords: DevSecOps; Security Automation; Static Analysis; Anomaly Detection; Compliance; Machine Learning

1 Motivation and significance

The widespread adoption of Continuous Integration and Continuous Deployment (CI/CD) pipelines has revolutionized software delivery, enabling teams to deploy updates rapidly and frequently. However, this velocity often comes at the cost of security. CI/CD environments have become prime targets for attackers, as they are prone to critical vulnerabilities categorized by industry standards such as OWASP, SLSA, and CIS benchmarks. Common security issues include:

- **Secret and credential exposure:** API keys, tokens, and passwords accidentally committed to logs or source code.
- **Supply chain vulnerabilities:** Unpinned dependencies and GitHub Actions using mutable tags instead of cryptographic hashes.
- **Excessive permissions:** Overly permissive workflow configurations granting write access to repository contents.
- **Configuration weaknesses:** Missing timeouts, insecure network calls, and debug logging in production.

Traditional security assessments, often performed manually at the end of the development cycle, are insufficient to address these dynamic threats without creating bottlenecks. There is a pressing need for automated tools that integrate seamlessly into existing workflows. *SafeOps* addresses this challenge by enabling DevOps teams to transition effectively to **DevSecOps**, embedding a dedicated layer of security directly into the pipeline. The platform automates the detection of misconfigurations and vulnerabilities early in the lifecycle ("shift-left"), ensuring that speed does not compromise the security posture of modern software supply chains.

Nr.	**Code metadata description**	**Metadata**
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used for this code version	`https://github.com/oussama-benzian/SafeOps-ENSAT`
C3	Permanent link to Reproducible Capsule	`https://github.com/oussama-benzian/SafeOps-ENSAT/wiki/Docs`
C4	Legal Code License	MIT License
C5	Code versioning system used	Git
C6	Software code languages, tools, and services used	Python, JavaScript
C7	Compilation requirements, operating environments & dependencies	Container runtime, Python 3.9+, Node.js 18+
C8	If available Link to developer documentation/manual	`https://github.com/oussama-benzian/SafeOps-ENSAT/wiki/Docs`
C9	Support email for questions	N/A

Table 1: Code metadata

2 Software Description

SafeOps is engineered as a distributed, microservices-based platform designed to automate security analysis within CI/CD pipelines. The system adopts a decoupled architecture to ensure scalability and fault tolerance, utilizing asynchronous messaging for data processing and synchronous REST APIs for user interaction.

2.1 Software Architecture

The platform follows an event-driven microservices architecture with seven specialized services orchestrated through a message broker and unified API gateway. Figure 1 depicts the system topology.

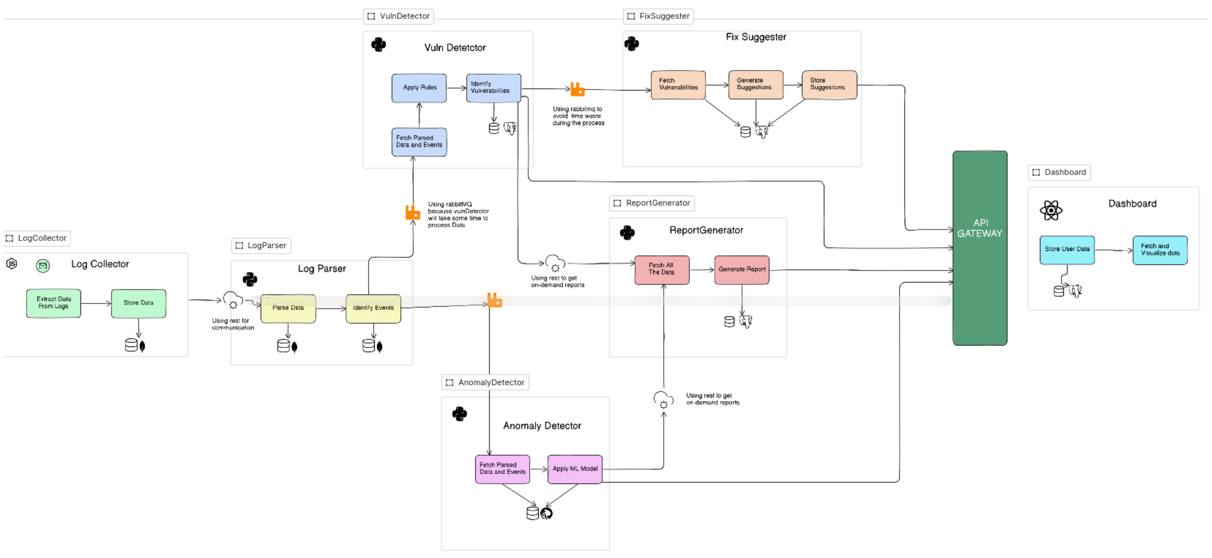


Figure 1: The SafeOps Microservices Architecture

2.1.1 Service Components

The architecture comprises the following microservices:

1. **Log Collector:** A lightweight service responsible for ingesting CI/CD logs from external sources such as GitHub Actions workflow runs. It normalizes incoming data and persists raw logs to a document database for downstream processing.
2. **Log Parser:** Consumes raw logs and extracts structured events including job metadata, step durations, output sizes, and workflow configurations. Parsed events are published to a message queue for parallel processing by detection engines.
3. **Vulnerability Detector:** A rule-based static analysis engine that scans parsed workflow content against a configurable ruleset. Rules are defined using regular expression patterns categorized by security standards (OWASP, SLSA, CIS). Detected vulnerabilities are persisted to a relational database with full provenance.

4. **Anomaly Detector:** An unsupervised machine learning component that analyzes pipeline behavioral metrics to identify deviations from established baselines. Uses the Isolation Forest algorithm to detect unusual patterns that rule-based scanning cannot identify.

5. **Fix Suggester:** Consumes detected vulnerabilities from the message queue and generates remediation recommendations. Uses template-based suggestions mapped from rule definitions to provide actionable fixes.

6. **Report Generator:** Aggregates security findings, anomaly detections, and pipeline statistics into comprehensive compliance reports. Supports PDF generation for audit documentation.

7. **Dashboard:** A single-page application providing real-time visualization of security posture, vulnerability distribution, anomaly trends, and pipeline health metrics.

Nr.	**(Executable) software metadata description**	**Metadata**
S1	Current software version	v1.0.0
S2	Permanent link to executables of this version	
S3	Permanent link to Reproducible Capsule	`https://github.com/oussama-benzian/SafeOps-ENSAT/wiki/Docs`
S4	Legal Software License	MIT License
S5	Computing platforms/Operating Systems	Linux, Unix-like, Distributed/Web-based
S6	Installation requirements & dependencies	Docker, Docker Compose
S7	Link to user manual	`https://github.com/oussama-benzian/SafeOps-ENSAT/wiki/Docs`
S8	Support email for questions	N/A

Table 2: Software metadata

2.1.2 Data Storage Layer

The platform employs a polyglot persistence strategy optimized for each data type:

- **Document database:** Stores raw logs and parsed events with flexible schema for heterogeneous CI/CD data.

- **Relational database:** Manages structured entities including vulnerabilities, security rules, and fix suggestions with referential integrity.

- **Time-series database:** Persists pipeline metrics and detected anomalies with native time-based aggregation and retention policies.

2.1.3 Communication Patterns

Services communicate through two patterns:

- **Synchronous REST API:** The API gateway routes user requests to appropriate services, enabling real-time queries for dashboard visualization.
- **Asynchronous messaging:** A message broker decouples the log parser from detection engines, enabling horizontal scaling of compute-intensive analysis without blocking the ingestion pipeline.

2.2 End-to-End Workflow

The platform follows a predictable processing pipeline that keeps ingestion, analysis, and visualization decoupled:

- **Ingest:** CI/CD logs or workflow files are submitted through the Log Collector and stored in MongoDB.
- **Parse:** The Log Parser extracts structured events and publishes them to the message broker.
- **Detect:** Vulnerability and anomaly services consume events in parallel to generate findings.
- **Remediate:** Fix Suggester produces template-based recommendations tied to detected issues.
- **Report:** Report Generator compiles findings into PDF/JSON artifacts for audit use.
- **Visualize:** The Dashboard surfaces real-time status, trends, and findings for operators.

2.3 Security Rule Engine

The Vulnerability Detector implements a comprehensive ruleset aligned with industry security standards. Rules are defined declaratively with the following attributes: unique identifier, descriptive name, security category, severity level (CRITICAL, HIGH, MEDIUM, LOW), regex pattern, and remediation guidance.

Table 3 summarizes the security categories and example detections:

Table 3: Security Rule Categories in SafeOps

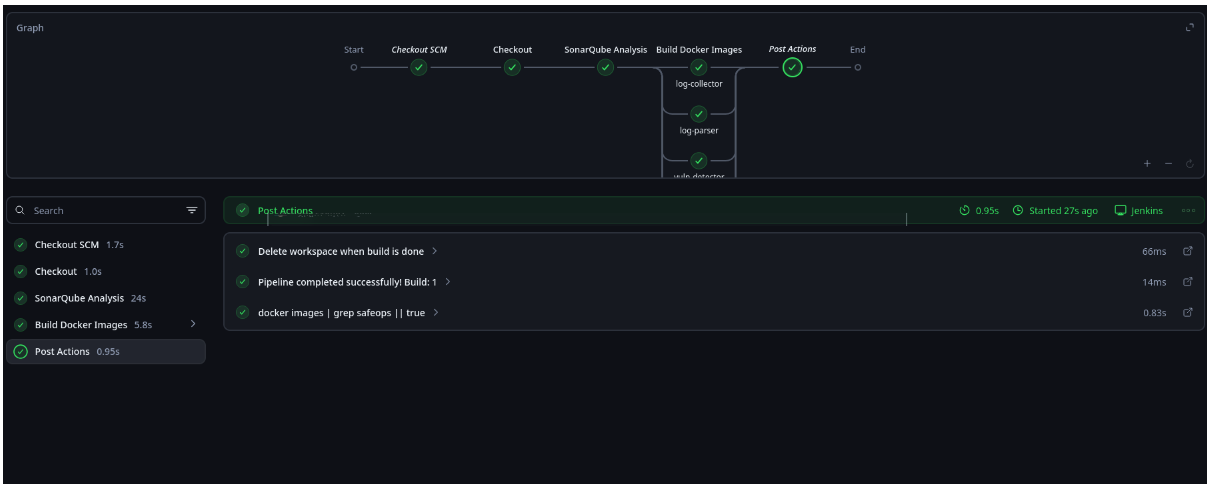
Standard	**Example Rule**	**Severity**
OWASP	Exposed API Key	CRITICAL
OWASP	Hardcoded Password	CRITICAL
OWASP	Private Key Exposure	CRITICAL
SLSA	Unpinned GitHub Action	HIGH
SLSA	Missing Artifact Integrity	MEDIUM
CIS	Write-All Permissions	HIGH
CIS	Missing Job Timeout	LOW

2.4 Jenkins and SonarQube-Based Development Pipeline

During the development of SafeOps, a Jenkins-based CI/CD pipeline was employed to automate building, testing, and quality assurance tasks. SonarQube was integrated into this pipeline to perform continuous static code analysis, enabling early detection of code smells, security hotspots, and maintainability issues throughout the implementation phase.

Each commit to the SafeOps repository triggered a Jenkins pipeline that executed unit tests, performed dependency checks, and launched SonarQube scans. The analysis results were used by the development team to iteratively improve code quality and security compliance prior to deployment. This setup ensured that the SafeOps platform itself adhered to the same DevSecOps principles it is designed to enforce.

Pipeline execution logs generated by Jenkins were also used as input samples during the development and validation of the SafeOps log ingestion and analysis components. SonarQube reports were retrieved via its REST API and served as reference artifacts to validate the correctness of the Vulnerability Detector's findings. This dual usage allowed SafeOps to be tested against realistic CI/CD outputs produced by an industry-standard toolchain.

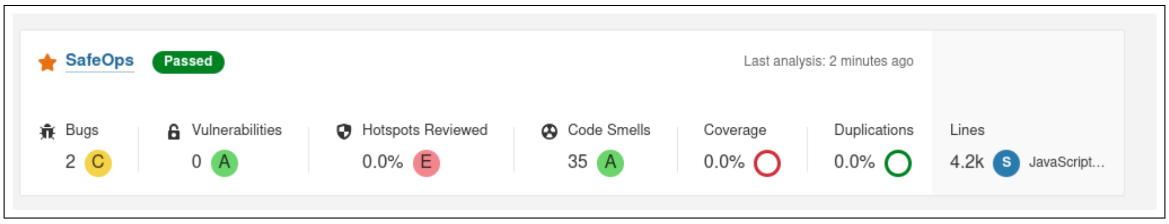


Checkout SCM 1.7s
Checkout 1.0s
SonarQube Analysis 24s
Build Docker Images 5.8s
Post Actions 0.95s

Post Actions — 0.95s — Started 27s ago — Jenkins

```
Delete workspace when build is done          66ms
Pipeline completed successfully! Build: 1    14ms
docker images | grep safeops || true         0.83s
```

Figure 2: Jenkins CI pipeline used during SafeOps development, including build, test, and SonarQube analysis stages



SafeOps Passed — Last analysis: 2 minutes ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
2 C	0 A	0.0% E	35 A	0.0%	0.0%	4.2k S JavaScript...

Figure 3: SonarQube code quality and security analysis results generated during SafeOps development

3 Data and Model Selection

The Anomaly Detector component employs unsupervised machine learning to identify behavioral deviations that static rule-based analysis cannot capture.

3.1 Algorithm Comparison

To select the optimal model for production, we compared Isolation Forest (IF) against One-Class SVM (OCSVM). The evaluation was conducted on a dataset of 25,000 samples containing simulated attack scenarios (e.g., cryptomining, data exfiltration) and realistic pipeline patterns.

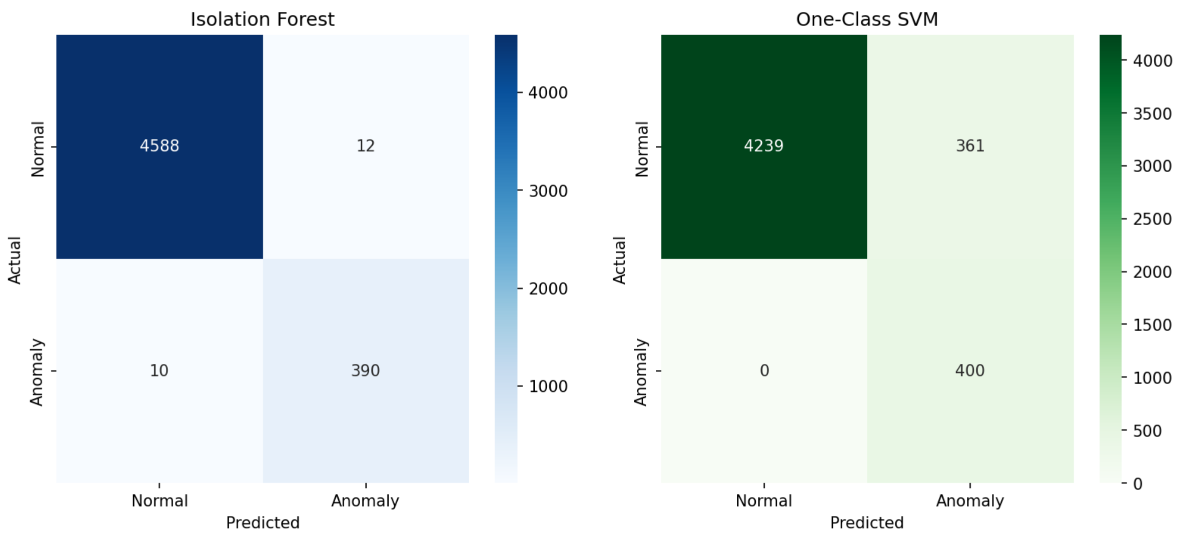


Figure 4: Model Comparison: Isolation Forest vs One-Class SVM.

As shown in Table 4 and Figure 4, Isolation Forest demonstrated superior performance for this use case. While OCSVM achieved perfect recall, it suffered from low precision (0.5256), leading to a high rate of false positives. Isolation Forest provided a balanced performance with significantly better precision and computational efficiency ($O(n \log n)$ vs $O(n^2)$), making it the preferred choice for real-time monitoring.

Table 4: Algorithm Performance Comparison

Metric	**Isolation Forest**	**One-Class SVM**
Precision	**0.9701**	0.5256
Recall	0.9750	**1.0000**
F1-Score	**0.9726**	0.6891
Accuracy	**0.9956**	0.9278

3.2 Production Model Implementation

The platform implements the Isolation Forest algorithm [Liu et al.(2008)]. The model is trained on 20 distinct features, including temporal metrics (build duration), structural features (step count), and resource usage (memory, CPU, network I/O).

The production model performance, detailed in Table 5 and Figure 5, shows robust detection capabilities for attacks such as Resource Abuse ($\approx$ 99% detection rate) and DoS attacks.

Table 5: Production Metrics (Isolation Forest)

Metric	Score
Precision	98.30%
Recall	98.30%
Accuracy	99.73%
ROC-AUC	99.99%

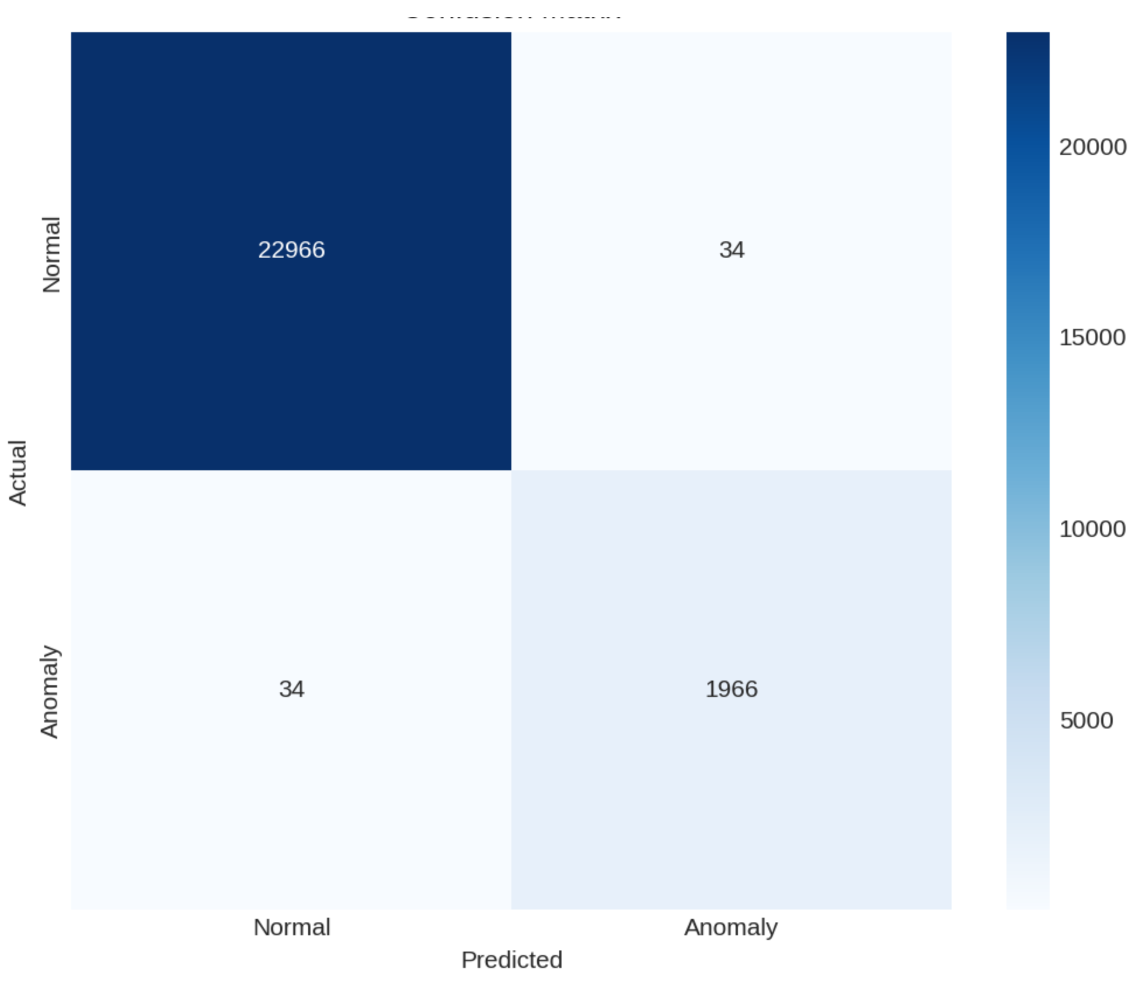
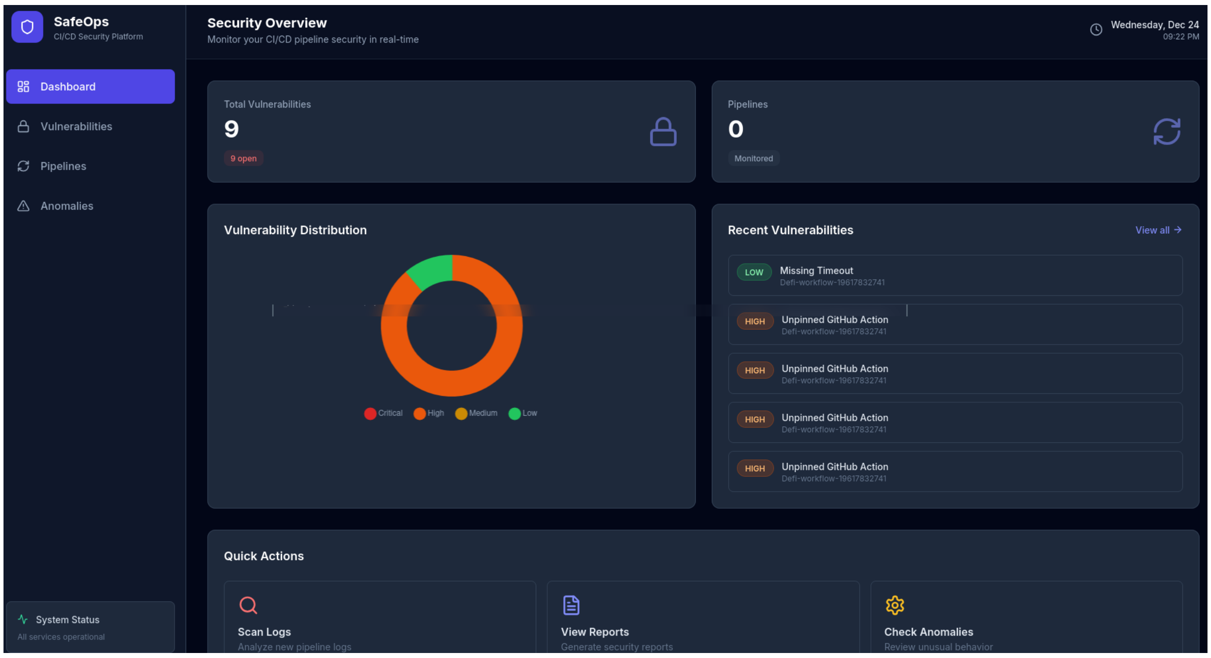


Figure 5: Production Model Analysis (Confusion Matrix).

4 Illustrative Examples

This section demonstrates the practical application of the platform in a real-world DevSecOps workflow, illustrating the progression from data ingestion to detailed security analysis.



SafeOps
CI/CD Security Platform

Dashboard
Vulnerabilities
Pipelines
Anomalies

System Status
All services operational

Security Overview
Monitor your CI/CD pipeline security in real-time

Wednesday, Dec 24
09:22 PM

Total Vulnerabilities
9
9 open

Pipelines
0
Monitored

Vulnerability Distribution

● Critical ● High ● Medium ● Low

Recent Vulnerabilities — View all →

LOW	Missing Timeout	Defi-workflow-19617832741
HIGH	Unpinned GitHub Action	Defi-workflow-19617832741
HIGH	Unpinned GitHub Action	Defi-workflow-19617832741
HIGH	Unpinned GitHub Action	Defi-workflow-19617832741
HIGH	Unpinned GitHub Action	Defi-workflow-19617832741

Quick Actions

Scan Logs
Analyze new pipeline logs

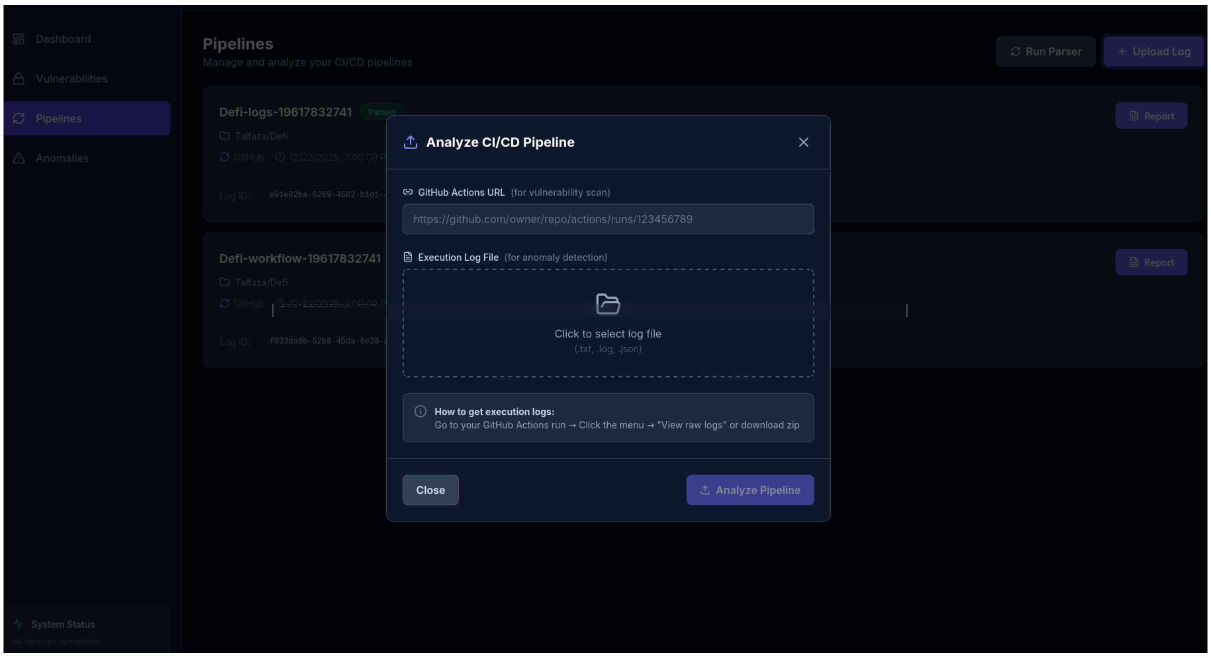
View Reports
Generate security reports

Check Anomalies
Review unusual behavior

Figure 6: The Main Dashboard. The interface features a donut chart for vulnerability distribution, a real-time feed of recent findings, and quick-action cards for initiating new scans.

4.1 Pipeline Analysis and Ingestion

To begin a security assessment, users interact with the analysis modal depicted in Figure 7. The platform supports a dual-input method: users can provide a repository URL for static vulnerability scanning or upload raw execution logs for anomaly detection. This decoupling ensures that both source code and runtime behavior can be analyzed in parallel.

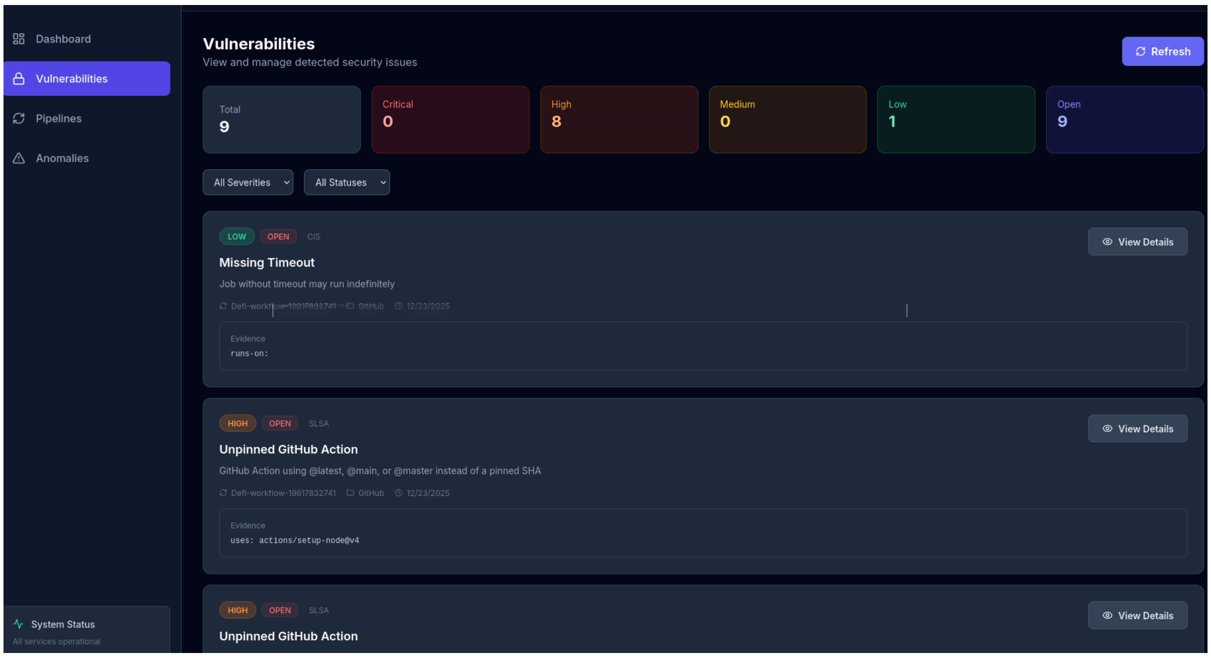


How to get execution logs:
Go to your GitHub Actions run → Click the menu → "View raw logs" or download zip

Figure 7: Pipeline Analysis Interface. Users can input a repository URL for static analysis or upload execution logs for behavioral anomaly detection.

4.2 Vulnerability Management

Once analysis is complete, findings are presented in the Vulnerabilities view (Figure 8). This interface categorizes issues by severity (Critical, High, Medium, Low) and allows for filtering by status. The figure illustrates the detection of specific compliance failures, such as ”Unpinned GitHub Action” (a supply chain risk) and ”Missing Timeout” configurations. Each entry provides evidence context, enabling developers to pinpoint exactly where the misconfiguration resides in the CI/CD definition.



Vulnerabilities

View and manage detected security issues

Total	Critical	High	Medium	Low	Open
9	0	8	0	1	9

LOW OPEN CIS

Missing Timeout

Job without timeout may run indefinitely

Defi-workflow-19617832741 GitHub 12/23/2025

Evidence

```
runs-on:
```

HIGH OPEN SLSA

Unpinned GitHub Action

GitHub Action using @latest, @main, or @master instead of a pinned SHA

Defi-workflow-19617832741 GitHub 12/23/2025

Evidence

```
uses: actions/setup-node@v4
```

HIGH OPEN SLSA

Unpinned GitHub Action

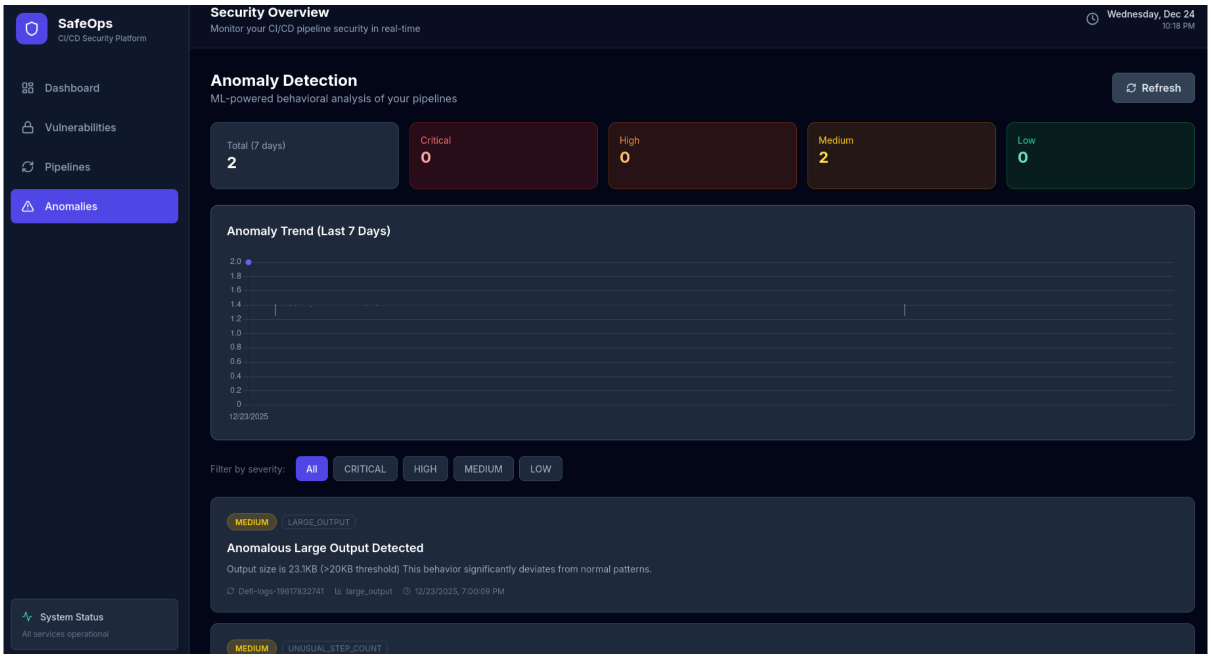
System Status

All services operational

Figure 8: Vulnerability Management View. A filtered list showing detected "High" severity issues, including unpinned dependencies and missing timeout configurations, with associated evidence.

4.3 Behavioral Anomaly Detection

Beyond static rules, the platform visualizes behavioral deviations using its machine learning engine. Figure 9 displays the Anomaly Detection dashboard, which tracks deviation trends over a 7-day period. The interface highlights specific outliers, such as ”Anomalous Large Output Detected,” assigning a numeric ”Deviation Score” (e.g., 0.04) to quantify the abnormality. This visualization aids in identifying subtle threats, such as data exfiltration attempts or resource abuse, that traditional static scanners often miss.



Anomaly Detection

ML-powered behavioral analysis of your pipelines

Total (7 days)	Critical	High	Medium	Low
2	0	0	2	0

Anomaly Trend (Last 7 Days)

MEDIUM LARGE_OUTPUT

Anomalous Large Output Detected

Output size is 23.1KB (>20KB threshold) This behavior significantly deviates from normal patterns.

Defi-logs-19617832741 large_output 12/23/2025, 7:00:09 PM

MEDIUM UNUSUAL_STEP_COUNT

Figure 9: Anomaly Detection Dashboard. The trend graph tracks behavioral deviations over time, while the details pane highlights specific anomalies like ”Large Output” with calculated deviation scores.

5 Impact and Conclusions

5.1 Impact

The development of *SafeOps* addresses critical gaps in CI/CD security:

- **Operational efficiency:** Asynchronous processing enables continuous security analysis without blocking deployment pipelines. The decoupled architecture allows detection engines to scale independently based on workload.
- **Detection of novel threats:** While rule-based detection identifies known patterns, the Isolation Forest component detects statistical anomalies that may indicate zero-day exploits or insider threats that evade signature-based detection.
- **Standards compliance:** The ruleset codifies OWASP, SLSA, and CIS benchmarks into automated checks, enabling organizations to demonstrate compliance through generated audit reports.
- **Accessibility:** The unified dashboard democratizes security analysis, enabling developers without specialized security expertise to identify and remediate issues within their workflow.

5.2 Conclusion

This paper presented *SafeOps*, an open-source microservices platform for automated CI/CD security analysis. The system architecture combines rule-based vulnerability detection aligned with industry standards (OWASP, SLSA, CIS) with unsupervised anomaly detection using the Isolation Forest algorithm.

Future work will focus on:

- Enhancing the Fix Suggester with context-aware remediation using large language models
- Expanding the ruleset to cover additional CI/CD platforms beyond GitHub Actions
- Implementing real-time alerting and integration with incident management systems

References

[Liu et al.(2008)] Liu, F. T., Ting, K. M. and Zhou, Z. H., 2008. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining* (pp. 413–422). IEEE. doi: 10.1109/ICDM.2008.17.

[OWASP(2023)] OWASP Foundation, 2023. OWASP Top 10 CI/CD Security Risks. Available at: `https://owasp.org/www-project-top-10-ci-cd-security-risks/`.

[SLSA(2023)] Supply-chain Levels for Software Artifacts, 2023. SLSA Specification. Available at: `https://slsa.dev/spec/v1.0/`.