European Journal of
Technology
(EJT)

Security Policy Enforcement and Behavioral Threat Detection in DevSecOps Pipelines

Khaja Kamaluddin

European Journal of Technology
ISSN 2520-0712 (online)
Vol.6, Issue 4, pp1 10-30, 2022

www.ajpojournals.org

Security Policy Enforcement and Behavioral Threat Detection in DevSecOps Pipelines

Khaja Kamaluddin
Masters in Sciences, Fairleigh Dickinson University, Teaneck, NJ, USA, Aonsoft International Inc,1600 Golf Rd, Suite 1270, Rolling Meadows, Illinois, 60008 USA

Article history
Submitted 21.04.2022 Revised Version Received 22.05.2022 Accepted 24.06.2022

Abstract

Purpose: The evolution of DevSecOps reflects a critical shift from traditional DevOps by embedding security seamlessly throughout the software development lifecycle. This research explores the convergence of security policy enforcement with behavioral threat detection within CI/CD pipelines, focusing on practices and tools. We discuss the limitations of legacy DevOps security approaches, including late-stage vulnerability identification and insufficient runtime protection, and highlight the rising need for behavior-based detection to counter advanced threats and insider breaches.

Materials and Methods: While static analysis and Infrastructure-as-Code scanning are useful strategies for evaluating security policies, a more comprehensive approach examines both compliance-focused tools and behavioral monitoring techniques.

Findings: Compliance as-code frameworks define policies that are automatically checked, yet anomaly detection within system calls, container events, and source code changes offers a dynamic perspective on threats. Previously, integration of these checks into CI/CD platforms like Jenkins and GitLab relied on manual security reviews of alerts and build checkpoints to demonstrate how security checkpoints and alerts were managed before the adoption of AI-driven automation. Through case studies such as the Solar Winds breach and practical pipeline examples, we illustrate how combined policy and behavior-based controls can enhance threat prevention. However, we also identify the significant challenges to solutions, including high false positive rates and limited cross-layer correlation capabilities.

Unique Contribution to Theory, Practice and Policy: Finally, the article looks ahead to the anticipated future of DevSecOps, emphasizing machine learning-driven behavior modelling, unified enforcement engines, and a zero-trust approach centered on identity and behavior analytics.

Keywords*: DevSecOps (JEL: O33, O32), Behavioral Detection (JEL: D83, L86), CI/CD Security (JEL: O33, L86), Jenkins Pipeline (JEL: L86, O32), Runtime Threat Monitoring (JEL: D83, L86), Infrastructure as Code (IaC) (JEL: O33, L86), Falco (JEL: L86, K24)*

INTRODUCTION

Modern DevOps pipelines have enabled rapid software delivery by automating development, testing, and deployment processes. However, this speed has introduced critical security vulnerabilities, as the same automation that accelerates workflows can propagate insecure code, misconfigurations, and unvetted dependencies into production. In many early DevOps implementations, security was often addressed only after deployment, making remediation reactive, time-constrained, and potentially ineffective [1].

To counter this, the DevSecOps movement promotes embedding security practices throughout the development lifecycle, a strategy commonly referred to as "shifting left" [2]. This approach integrates automated security checks such as code analysis, configuration validation, and access control enforcement directly into CI/CD workflows. It ensures applications are secure by design rather than relying on post-deployment patching. Tools like SonarQube [6], Checkov, and Open Policy Agent (OPA) [7] allow teams to enforce static security policies during early development and build phases, minimizing the risk of insecure releases.

However, policy enforcement alone is no longer sufficient to protect against modern threats. Traditional static analysis tools are limited to known patterns and predefined rules, which makes them ineffective against zero-day exploits, insider threats, and behaviorally dynamic attacks. This has led to the incorporation of behavioral threat detection within DevSecOps pipelines. By analyzing the real-time behavior of applications and infrastructure, tools such as Falco and OSSEC provide continuous runtime monitoring to detect anomalies like privilege misuse, unauthorized access, or suspicious system calls [5].

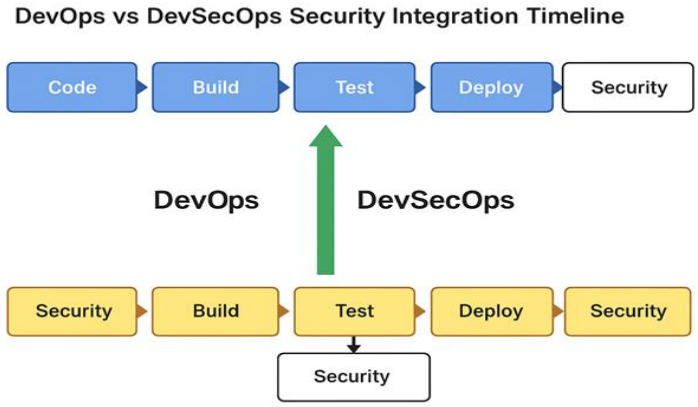

Figure 1: DevOps vs DevSecOps Security Integration Timeline

Figure 1 illustrates this evolution from DevOps to DevSecOps, showing how security has gradually shifted from the end of the pipeline to earlier phases such as coding, building, and testing. Initially, DevOps workflows prioritized speed and functionality, with security interventions occurring after deployment. As threats evolved, security began moving leftward, integrated at each stage to form a layered defense model.

This article examines how security policy enforcement and behavioral threat detection two complementary strategies were implemented in traditional DevSecOps workflows using tools available up to that time. By focusing on pre-AI and pre-cloud-native orchestration technologies, the study offers a clear picture of the security landscape before the emergence of modern intelligent detection systems [3][4]. The findings reinforce the importance of proactive

and continuous security integration to safeguard today's fast-paced software development environments.

This article focuses on the following key research objectives related to DevSecOps security:

i. To evaluate tools and techniques for enforcing security policies in DevSecOps pipelines using solutions such as SonarQube, OPA, and InSpec.

ii. To evaluate the effectiveness of existing behavior-based threat detection methods, including runtime tools like OSSEC, Auditd, and Falco.

iii. To demonstrate how security enforcement and behavioural detection were integrated into CI/CD workflows employing traditional DevSecOps and pipeline ideas before any innovations appeared.

Background and Motivation

The demand for faster, more reliable software delivery has led to the widespread adoption of DevOps, a methodology that promotes collaboration between development and operations teams while automating repetitive tasks [8]. DevOps has revolutionized deployment speed, enabling organizations to push features and updates in hours rather than weeks. However, this increased velocity has also intensified security challenges. In early DevOps implementations, speed and agility were prioritized, often at the expense of robust security protocols. As software systems grew more interconnected and dependent on micro services, APIs, and containerized infrastructure, the attack surface expanded, making traditional post-deployment security practices insufficient [9].

This section reflects on the initial shortcomings of security in DevOps workflows and explains the growing need for integrated controls. Specifically, it motivates the dual inclusion of behavioral threat detection and policy enforcement within DevSecOps pipelines, ensuring both proactive rule-based compliance and adaptive monitoring of dynamic runtime threats.

Limitations of Traditional DevOps Security

In its early phases, DevOps focused primarily on automation, scalability, and continuous delivery. Security, when included at all, was often confined to the final stages of the software delivery pipeline just before production deployment [10]. This reactive model introduced several technical limitations that left systems vulnerable:

i. **Lack of runtime visibility:** Traditional pipelines offered limited insight into how applications and infrastructure behaved post-deployment. Without monitoring system calls, process activity, or file access, it was difficult to detect privilege abuse or stealthy intrusions.

ii. **Manual audits were inefficient and error-prone:** Security assessments often relied on human review of configurations, logs, and documentation. These processes were vulnerable to oversight due to large log volumes, inconsistent formatting, time constraints, and analyst fatigue, which introduced subjectivity and missed anomalies.

iii. **Delayed vulnerability detection:** Security issues identified late in the pipeline required rework or emergency patches, often after the application had already entered production making the process disruptive and high-risk.

iv. **Lack of policy enforcement at key stages:** In many pipelines, security policies were not automatically applied at the commit, build, or deploy stages. This inconsistency meant that insecure code or configurations could proceed unchecked through the pipeline.

v. **Isolated security functions:** Security teams often operated in silos, disconnected from development and operations. This lack of integration hindered timely feedback, delayed remediation, and reduced overall pipeline transparency.

These limitations underscored the need for DevSecOps, a model where security is embedded throughout the software lifecycle and enhanced with both preventive (policy enforcement) and detective (behavioural analysis) controls.

Emerging Threat Landscape

Organisations witnessed a sharp increase in cyberattacks targeting the software delivery process. Key developments included:

i. **Software Supply Chain Attacks:** The SolarWinds attack illustrated that cyber thieves could exploit code and automated pipelines used by other organisations [11].
ii. **Insider Threats:** Developers or administrators who have special access may slip past security controls either accidentally or by purpose.
iii. **Zero-Day exploit:** The use of open-source elements in projects became a major risk for encountering unusual vulnerabilities.
iv. **Container Exploit and Misconfiguration:** They introduced additional risks when a large number of Docker and Kubernetes were used.

These evolving threats required more than just traditional vulnerability scanning; they demanded active monitoring of system behaviour to detect subtle indicators of compromise.

The Need for Behaviour-Based Detection

As attacks became more evasive and context-aware, security teams turned to behavior-based detection to spot subtle indicators of compromise that static scanners might miss [12]. Instead of simply verifying configurations or dependencies, these tools actively monitor how systems behave in real-time to flag unusual activity such as:

i. Execution of unauthorized processes or binaries
ii. Unexpected privilege escalation attempts
iii. Modifications to sensitive files during CI/CD runs
iv. Unscheduled outbound network access, lateral movement, or port scanning during or after deployment

Thanks to these technologies, both well-known and unexpected attacks could be found even during CI/CD pipeline execution. If used along with Open Policy Agent (OPA) and InSpec, these tools helped achieve both full compliance and current threat awareness. Comparison of security focus of DevSecOps and DevOps is given in Table 1 below.

Table 1: Comparison of Traditional DevOps vs DevSecOps Security Focus

Feature	Traditional DevOps	DevSecOps
Security Integration Point	Post-deployment	Throughout CI/CD stages
Security Tools Used	Manual audits, scanners	Automated SAST, DAST, OPA, InSpec
Threat Detection Approach	Signature-based	Behavioral and anomaly-based
Focus	Speed and delivery	Security, compliance, and delivery balance
Response Time	Delayed (after release)	Real-time or near real-time
Human Involvement	High (manual checks)	Low to moderate (automated enforcement)
Risk Mitigation Strategy	Reactive	Proactive and layered
Policy Enforcement	Ad hoc or manual	Continuous and codified

Security Policy Enforcement in Devsecops

By using security policy enforcement, DevSecOps pipelines ensure security practices are always applied to every step in software development [13]. The main idea of DevSecOps was to create security guidelines and apply them in CI/CD and IaC tools. Following this approach reduced mistakes made by staff, improved compliance and found possible threats at the beginning of development.

Defining Security Policies

Security policies show the restrictions a system must have in place for proper confidentiality, integrity and availability. Policies in this area are normally put into the following categories:

i. **Authentication & Authorization:** Verifying who users are and giving them rightful access to code, workspace and deployment processes.
ii. **Compliance:** It means guaranteeing that applications and systems follow industry regulations including HIPAA, GDPR or FedRAMP.
iii. **Secure Coding:** Setting specified coding rules, checking code with static analysis and confirming the safety of outside libraries used by the code.

These policies were grounded in widely recognised frameworks and benchmarks:

i. **OWASP Top 10 (2017):** Provided guidelines on the most critical security risks for web applications.
ii. **NIST SP 800-53:** Offered comprehensive security and privacy controls for federal systems.
iii. **CIS Benchmarks:** Defined secure configuration practices for operating systems, applications, and cloud platforms.

Table 2: Common Security Policies and Mapping to Compliance Frameworks

Security Policy	OWASP Top 10 (2017)	NIST SP 800-53	CIS Benchmarks
Input validation and sanitization	A1: Injection	SI-10, SC-28	Web server input validation
Secure authentication (e.g., MFA)	A2: Broken Authentication	IA-2, IA-5	OS & Identity service configs
Secure password storage	A2, A5	IA-5 (1), SC-12	Password policy enforcement
Least privilege access control	A5: Broken Access Control	AC-6, AC-17	User access control benchmarks
Secure configuration of services	A6: Security Misconfigurations	CM-2, CM-6	Docker/K8s/OS hardening
Dependency and library scanning	A9: Using Components	SI-2, SA-11	Package manager checks
Logging and monitoring	A10: Insufficient Logging	AU-2 to AU-6	Log service configuration

Policy Enforcement Techniques

Policy enforcement relies on automated tools and techniques integrated into the software delivery pipeline. These techniques can be grouped into three major categories:

i. **Static Application Security Testing (SAST):** Tools like SonarQube, Checkmarx, and Fortify analyse source code and binaries without executing them [14]. They detect vulnerabilities such as SQL injection, buffer overflows, and improper error handling. SAST tools were typically embedded into build stages to prevent insecure code from being merged.
ii. **Infrastructure-as-Code (IaC) Scanning:** With the rise of declarative infrastructure tools like Terraform and Kubernetes [15], security enforcement shifted left into infrastructure definitions:

Checkov and Conftest scanned Terraform and Kubernetes manifests.

Terraform Sentinel and OPA (Open Policy Agent) enforced custom policies at provisioning time.

Enforcement Points in CI/CD: Security Policies were Enforced Via

i. **Git Hooks:** Preventing insecure code from being committed to repositories.
ii. **Jenkins or GitLab CI/CD Gates:** Conditional pipeline stages that fail if security tests do not pass.
iii. **Kubernetes Admission Controllers:** Rejecting unsafe deployments at runtime using OPA or Kyverno.

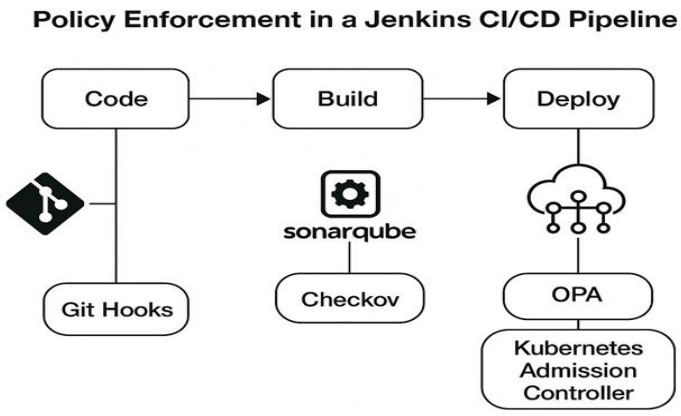

Figure 2: Typical Policy Enforcement in a Jenkins CI/CD Pipeline

Compliance-as-Code

Compliance-as-Code (CaC) allows organisations to automate regulatory and internal policy compliance using code, enabling version control, peer review, and continuous testing [16]. This approach ensures repeatable and auditable enforcement across development environments.

Key tools for CaC included are:

i. **InSpec:** Developed by Chef, InSpec allows writing human-readable security and compliance tests.
ii. **Sentinel**: HashiCorp's policy-as-code framework for Terraform, Nomad, and Vault.
iii. **OPA:** A general-purpose policy engine used with Kubernetes, Envoy, and Terraform.

These tools were embedded into CI/CD pipelines to validate infrastructure compliance before deployment.

Table 3: Tools for Policy-as-Code Enforcement with DevSecOps Integration

Tool	Functionality	CI/CD Integration Point
InSpec	Compliance testing for OS, containers, cloud infrastructure	Test/Deploy stage
Terraform Sentinel	IaC policy enforcement for provisioning	Terraform apply/plans
OPA	Generic policy engine for Kubernetes, CI/CD, APIs	Admission control, build/test
Conftest	Policy checks for Kubernetes, Terraform, and Dockerfiles	Git hooks, build stage

Before the emergence of AI-driven tools, DevSecOps teams enforced security by embedding policy checks at every CI/CD phase. Using open-source tools to set up these rules made DevSecOps pipelines effective, repeatable and secure.

Afterward, we will discuss how adding behavioural threat detection to the pipeline helped it deal with shift in threats and enforced protection at run time, instead of only identifying static threats.

Behavioral Threat Detection in Devsecops

Concept and Importance

DevSecOps behavioral threat detection expands upon traditional security approaches by identifying suspicious patterns in system, application, or user activity, rather than relying solely on known attack signatures. Traditional signature-based detection works by comparing incoming data (e.g., files or network packets) against a predefined list of known malicious code or behavior patterns [17]. While effective for identifying previously catalogued malware, this method struggles to detect new, evolving, or obfuscated threats. Its dependency on frequently updated signature databases also introduces delays and blind spots.

In contrast, behavioral detection analyzes runtime activity to spot anomalies that deviate from established norms. Instead of requiring a known malware signature, it flags suspicious events such as unusual system calls, unexpected file modifications, strange outbound connections, or privilege escalations even if the underlying code is previously unseen. This makes it particularly effective against zero-day exploits, file less malware, polymorphic threats, and complex multi-stage attacks like Advanced Persistent Threats (APTs).

As modern attack techniques increasingly bypass static defenses, behavioral detection serves as an adaptive, context-aware layer that complements policy enforcement. It focuses on dynamic system behavior, enabling the detection of subtle intrusions that evade traditional rules-based methods.

Advantages of Behavioral Detection in Modern Threat Landscapes

i. **Zero-day and Unknown Threat Detection:** Can detect previously unseen threats by monitoring anomalous behaviour.
ii. **Reduced Dependence on Updates:** Less reliant on signature databases that require constant updating.
iii. **Contextual Analysis:** Behavioural systems consider the context of activities, reducing false positives caused by benign but rare activities.
iv. **Early Detection:** Can identify malicious activity early in its lifecycle before damage or exfiltration occurs.
v. **Adaptability:** Effective against polymorphic and fileless malware that mutate or hide to evade signature-based detection.

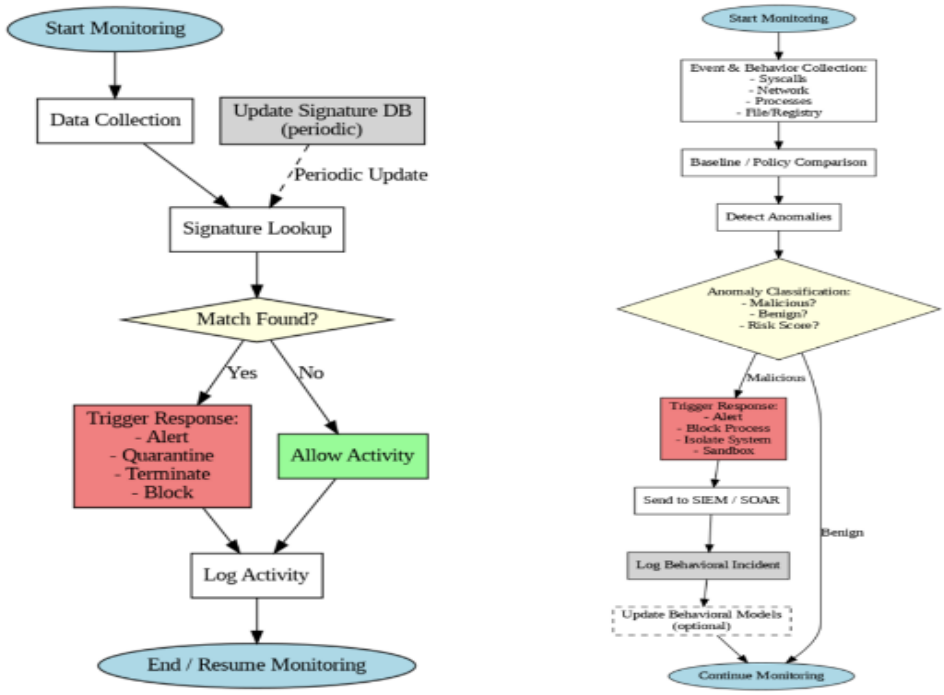

Figure 3: Signature-Based vs Behavioral Detection Flowchart

Figure 3 presents a side-by-side comparison of detection methods. On the left, signature-based detection matches incoming data against known threat patterns. On the right, behavioral detection collects live system events and inspects them for deviations from expected behavior, allowing the system to identify threats without needing prior knowledge of specific malware.

Host and System-Level Tools

Several open-source and commercial tools have been widely adopted for behavioral threat detection at the host and system level. These tools primarily monitor system calls, audit logs, and runtime behavior to detect anomalies and suspicious activities in real time [18]. While they originated as stand-alone detection systems, modern DevSecOps practices increasingly incorporate these tools into CI/CD workflows and infrastructure-as-code pipelines to provide security as part of the development lifecycle.

Auditd

Auditd is a native Linux auditing system that records low-level events such as file accesses, permission changes, and user authentication attempts. Administrators configure audit rules to monitor sensitive system activity such as changes to /etc/, execution of privileged commands, or login attempts. In a DevSecOps pipeline, Auditd is commonly deployed on build servers, staging environments, or ephemeral test machines via automation tools like Ansible or Terraform, ensuring consistent audit coverage across dynamic infrastructure. Its logs can be forwarded to centralized SIEM tools for real-time analysis [19].

OSSEC and Wazuh

OSSEC is a Host-based Intrusion Detection System (HIDS) that analyzes logs, checks file integrity, and detects rootkits in real time. Wazuh, its modern fork, extends OSSEC with cloud-native features such as agent management, Kubernetes monitoring, and scalable dashboards. Both tools use rule-based and anomaly-based detection techniques. In DevSecOps environments, OSSEC/Wazuh agents are frequently deployed as part of infrastructure-as-code

templates or baked into base machine images used in pipelines. Their alerts can be integrated with webhook-based alerting platforms (e.g., Slack, Jira, PagerDuty) to trigger automated responses within CI/CD stages.

Falco

Falco is a CNCF-hosted, real-time behavioral monitoring tool designed for containerized and Kubernetes environments, though it also works on traditional hosts. It uses eBPF or syscall tracing to detect abnormal behavior such as container escapes, privilege escalation, or unauthorized network activity. Falco is commonly deployed as a Kubernetes DaemonSet in production and test clusters, providing runtime visibility for every node. DevSecOps teams can integrate Falco alerts into CI/CD pipelines by using Falco Sidekick, forwarding findings to tools like Prometheus, Grafana, or incident management systems, enabling automated enforcement or rollback.

Custom Syslog-Based Alerts and Anomaly Scripts

Many organizations also rely on custom scripts to parse syslog entries, detect anomalous login attempts, suspicious process executions, or unusual network traffic. These scripts are often integrated into CI/CD validation stages to flag builds or infrastructure configurations that deviate from expected behavior. They may also be run as part of post-deployment checks or scheduled jobs triggered by pipeline orchestration tools like Jenkins, GitLab CI, or Argo CD.

The table 4 typically compares tools like Auditd, OSSEC, Wazuh, Falco, and custom scripts, mapping their capabilities against monitored elements: system calls, file integrity, process anomalies, network behavior, and container runtime monitoring.

Developer & Source Code Behavior Monitoring

Although it is important to analyze the actions of hosts, examining what developers do and their source code is a quickly growing area in DevSecOps behavioral threat detection. Mistakes from developers using malicious or infected tools can create security risks, so it is important to detect them fast [20].

Git Commit Anomaly Detection (Manual Rule-Based)

Anomalies in Git commit transitions were detected by workforce members writing their own rules and heuristics prior to the use of advanced tools. If people are committing code in unusual ways, the system may ring the alarm. CI techniques often merged these rules into their pipelines.

Review of Git Audit Trails and Manual Scripting in CI Pipelines

Scouts and DevOps engineers made scripts to check Git logs and analyze commits, referring them to people's roles, branch policies and code metrics. Checking manually needs to go hand in hand with automated checks to find signs of insider threats or breaches in accounts.

While it takes more effort and isn't very automated, using this approach was an important way to add threat detection into what developers do every day. It revealed that guaranteeing honest code and saying who wrote it is key in securing software delivery.

Runtime Behavior Analysis in Containers

Because containers exist only as long as the application runs, they bring new security challenges. Finding threats related to container breakouts or lateral action within containers depended on behavioral detection.

Falco Rule Sets

Sysdig open source and free of charge. Common principles of Falco were:

i. Detecting shell access in containers (reverse shells or interactive shells).
ii. Alerting on attempts to write to sensitive files or directories outside expected paths.
iii. Monitoring for process spawning patterns indicative of privilege escalation or breakout attempts.
iv. Detecting network connections initiated from containers to unusual IP addresses or ports.

These rules were customizable and often integrated with alerting systems to feed security incident and event management (SIEM) tools.

Container Breakout Detection and Reverse Shell Activity

Containers can be compromised when an attack gets from the container to the host or other containers; this threat was addressed with behavioral detection. By checking for sudden use of mounting, modifying namespaces or using privileged binaries, defenders were able to alert the security team.

Process creations and network behavior were also used to detect when a reverse shell was set up by an attacker to an external server in real time.

Role of Tools like Sysdig Secure, Twistlock (Legacy), and Aqua Security

Before consolidation in the cloud-native security space, tools like Twistlock (acquired by Palo Alto Networks) and Aqua Security were prominent commercial solutions offering runtime behavioral threat detection for containers and Kubernetes. They extended capabilities to vulnerability scanning, compliance checks, and runtime protection.

Sysdig Secure, building upon the open-source Sysdig and Falco foundation, provided enhanced monitoring and response capabilities, including integration with Kubernetes audit logs and cloud-native SIEM solutions.

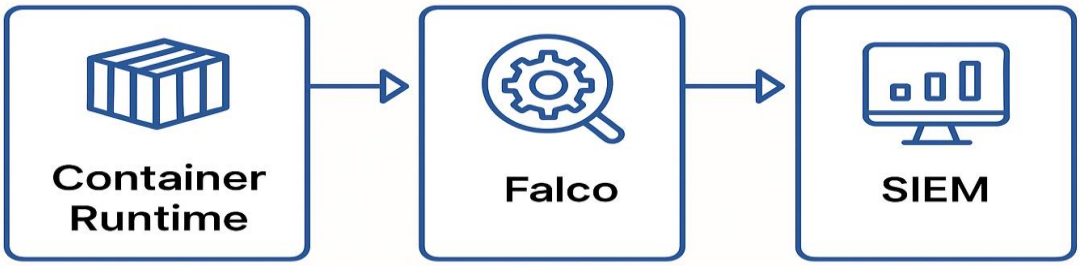

Figure 4: Sample Falco Detection Flow from Container Runtime to SIEM Alert

Behavioural threat detection has become a foundational element of DevSecOps practices, complementing traditional signature-based methods by focusing on system and application behaviours. Tools like Auditd, OSSEC, Wazuh, and Falco established effective frameworks for host and container-level monitoring. Moreover, integrating developer and source code behaviour analysis into CI/CD pipelines enhanced early detection of insider or supply chain threats. Container runtime behaviour analysis, powered by flexible rule sets and tools like Falco and commercial counterparts, addressed the evolving security challenges posed by ephemeral, cloud-native environments. As threat actors continue to innovate, behavioural detection's adaptive, context-aware nature remains critical to robust security posture in modern DevSecOps.

Table 4: Behavioral Threat Detection Tools and Their Monitoring Capabilities

Tool / Capability	System Call Monitoring	File Integrity Monitoring	Process Anomaly Detection	Network Behavior Monitoring	Container Runtime Monitoring	Notes
Auditd	Yes	Limited (via rules)	Limited	Limited	No	Native Linux tool; strong syscalls audit; config intensive
OSSEC	Indirect (via logs)	Yes	Yes	Yes	No	HIDS with log analysis and file integrity
Wazuh	Indirect (via logs)	Yes	Yes	Yes	Limited	Enhanced OSSEC with cloud support and scalability
Falco	Yes	No	Yes	Yes	Yes	Real-time syscall analysis; container-focused
Custom Syslog Scripts	Indirect (via logs)	Varies	Varies	Varies	Varies	Highly customized, depends on implementation

Integrating Security and Detection in DevSecOps Pipelines

Common CI/CD Tools & Integrations

Using CI/CD tools is now a standard process in many DevSecOps pipelines for automating the release of software. Security improvements were possible thanks to native plug-ins and custom scripting used by Jenkins, GitLab CI/CD and CircleCI [21].

i. Jenkins enabled security checks through plugins such as OWASP Dependency-Check, Checkmarx, and SonarQube, typically embedded into pipeline stages via Groovy or pipeline DSL.

ii. GitLab CI/CD offered YAML-based configuration (.gitlab-ci.yml) that allowed tight integration of SAST, container scanning, and license compliance jobs.

iii. CircleCI provided orb-based extensions for integrating tools like Snyk and Aqua Security.

DevSecOps pipelines are now commonly built with the help of lightweight GitHub Actions. YAML scripts were used to configure the security workflows which responded to actions including push, pull request or a deployment. Possible actions are using linters, static analysis programs or image vulnerability checks during the early stages of the pipeline.

Stages for Security Checkpoints

To effectively detect and mitigate risks, security controls were inserted at various points in the pipeline. The following illustrates typical checkpoints based on DevSecOps implementations:

1. **Pre-Commit Stage:**
   i. **Static Application Security Testing (SAST):** Code was scanned for vulnerabilities by SonarQube, Bandit (for Python) and ESLint (JavaScript) before it could be pushed.
   ii. **Infrastructure as Code (IaC) Scanning:** Infrastructure build scripts were checked with Checkov or tfsec: the selected tools examined Terraform, Kubernetes YAML and Ansible playbooks.
2. **Build/Test Stage:**
   i. **Dependency Scanning:** CVE data was obtained for my code's dependencies and packages by using OWASP Dependency-Check and Snyk, among others.
   ii. **Policy Enforcement:** Organizational policies were followed and implemented thanks to OPA (Open Policy Agent) plus internal scripts.
3. **Deployment Stage:**
   i. **Runtime Hardening:** Before releasing the image, executables were signed, container privileges were reduced and AppArmor/SELinux enforcement was used.
   ii. **Gate Checks:** Deployments were held back using either policies or manually approved based on what the scans or reports found.

Table 5: Security Checkpoints in CI/CD Pipeline

Stage	Checkpoint Type	Example Tools
Pre-Commit	SAST, IaC Scanning	SonarQube, Checkov, Bandit
Build/Test	Dependency Scanning, Policy Checks	Snyk, OWASP DC, OPA
Deploy	Runtime Hardening, Gate Enforcement	Falco, Notary, Open Policy Agent

Log Management and Alerting

Security logging and monitoring was an essential component of DevSecOps detection workflows. Practices emphasized self-managed solutions with manual rule correlation:

i. The ELK Stack (Elasticsearch, Logstash, Kibana) was widely adopted for centralized logging. Applications and pipeline tools forwarded logs to Logstash, where custom parsing and filtering prepared data for visualization in Kibana.

ii. Graylog provided an alternative log aggregation platform with a simpler UI and plugin architecture. It supported pipeline logs, container events, and security audit logs.

iii. Manual correlation rules, such as matching repeated failed builds with suspicious Git activity, were common due to the lack of mature machine learning-based detection.

Working together, teams used SIEM systems such as Splunk or Wazuh to connect to their logs through specially created connectors. Yet, analysts could spot unusual actions, guidance breaches and unauthorized insider risks, as the automation offered by neighbors was still not as powerful.

Devsecops Breach Prevention and Behavioral Detection
Devsecops Breach Prevention: Lessons from the Solarwinds Attack

To understand why traditional DevOps pipelines could not detect unusual activities in the SolarWinds breach in late 2020, we look at that case as a prime example. If behavioural detection had been used, early signs might have been seen by the system [22].

i. **Alterations in the code and hidden logic included in the build**
The injected backdoor code did not align with standard coding or version control behavior. Behavioral baselines for commit frequency, contributor activity, and file access patterns could have flagged these anomalies.
ii. **Transmission to external parties**
The malware's outbound communications to attacker-controlled servers were abnormal for build or deployment environments. Behavioural monitoring of expected network activity would have identified these unauthorized connections.
iii. **Sudden shifts in build frequency or number of deliverables**
A spike in the number of builds or unexpected changes in deliverable outputs may signal compromised automation. Deviations from the usual build cadence could have been detected through pipeline behavior baselining.
iv. **Unexpected execution of administrative tools**
Tools such as PowerShell or credential scripts executed during builds are atypical in normal CI/CD workflows. Behavioural profiling of runtime activity on build agents could have flagged such unauthorized tool usage.

These deviations are difficult to detect with signature-based tools alone. However, if behavioral logging and monitoring were applied consistently during both code commits and runtime execution, such activity could have been flagged early. Logging sensitive script changes and tracking anomalous actions across build environments allows organizations to detect sophisticated supply chain threats before full impact occurs.

Jenkins Pipeline Security Integration: Behavior & Policy Triggers

Jenkins-based DevSecOps pipelines now incorporate both static and behavioral security controls to protect applications throughout the CI/CD lifecycle. In the early stage, tools like Checkov and Conftest scan Infrastructure as Code (IaC) files such as Terraform or Kubernetes manifests to detect misconfigurations and enforce policy compliance. These scans occur before code is merged, helping teams prevent insecure configurations from reaching later stages. The results can be automatically converted into structured reports for security teams to review.

During the build and deployment phase, Docker is used to containerize applications, simulating production conditions. At this stage, runtime security becomes essential.

To detect suspicious behavior during container execution, Falco is integrated into the final pipeline stage. Running alongside live containers, Falco monitors system calls in real time, identifying threats like unauthorized shell access, unexpected file changes, or abnormal network activity. After a short observation period, containers are shut down, and Falco logs are stored for review.

Together, Checkov and Conftest enforce static security early, while Falco delivers runtime visibility. This layered approach ensures vulnerabilities are addressed at multiple stages—making the Jenkins pipeline both secure by design and resilient to runtime threats.

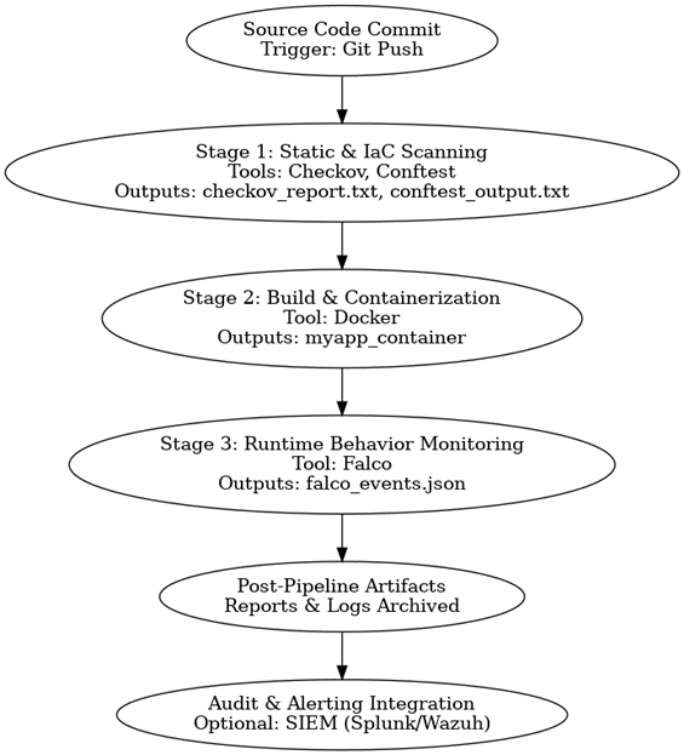

Figure 5: Jenkins DevSecOps Pipeline with Security and Behavioral Triggers

Detecting Behavioral Anomalies in CI Environments

One noticeable behavioural anomaly seen in CI/CD pipelines is when an unauthorised shell belonging to a CI agent like Jenkins or GitLab Runner obtains access to a container. The access can happen when someone steals the builders' credentials or if malicious code is put into the build process.

Typically, secure pipelines handle workflow steps differently from the usual order. This behaviour can be detected by Falco, which sets off an alert when a shell process is run inside a container it doesn't expect one. The tools set up baseline behaviour and identify events that move beyond this standard.

Early on, leading DevSecOps teams monitoring app and system behaviour using Falco or AppArmor were able to track activities and find anomalies in close to real time, helping to improve security throughout DevSecOps practices.

Limitations of Methods

Even though DevSecOps pipelines offered good security options such as static analysis, rule-based detection and log aggregation, they still faced certain issues. There was greater risk to how threats were detected, the ability to react and the total stability of the company's security operations.

Absence of ML/AI for Behavior Profiling

Tools from this era did not employ machine learning (ML) or artificial intelligence (AI) to model and understand normal system behavior over time. Without such behavioral baselining, distinguishing between legitimate and malicious activity required manual investigation. This limited the ability of detection systems to identify novel or subtle threats (e.g., insider threats, slow-moving exfiltration), which were not covered by predefined rules.

Fragmented Toolchain with Poor Cross-Layer Correlation

DevSecOps pipelines often spanned multiple tools for code, build, and runtime stages. However, most tools operated in isolation and lacked integration standards. This made it difficult to correlate a security issue in runtime with its origin in source code or infrastructure configuration. For example, a vulnerability discovered during runtime might not be easily traceable back to a particular Git commit, developer, or IaC misconfiguration limiting root cause analysis.

Weak Integration between Behavior Logs and Policy Engines

Although behavior detection tools like Falco generated useful logs, these were rarely integrated with policy engines like OPA (Open Policy Agent) or Conftest in real time. As a result, enforcement of security policies remained largely static and reactive. There was no mechanism to dynamically adjust policies based on detected behavioral trends or emerging anomalies.

Table 6: Gaps in DevSecOps Security Toolchains

Limitation	Description
High False Positives	Static rules triggered alerts on rare but legitimate actions
No Behavioral Modeling	Lack of ML/AI led to blind spots in anomaly detection
Toolchain Fragmentation	Poor visibility across CI, build, deploy, and runtime phases
Disconnected Logs & Policies	Runtime logs not linked to enforcement mechanisms or access controls
Lack of Real-Time Correlation	Security events across layers weren’t contextualized or correlated

Future Outlook

Security experts and DevSecOps practitioners were envisioning a future pipeline model that would overcome the limitations of rule-based, fragmented systems. The emphasis shifted toward intelligent automation, unified observability, and identity-driven security. Several trends were anticipated to transform DevSecOps practices into proactive, adaptive frameworks.

ML-Based Behavior Modeling for Developer and System Patterns

The future of threat detection was expected to be driven by machine learning (ML), enabling pipelines to learn from normal developer behavior and system operations. By analyzing coding habits, commit frequency, build dependencies, and runtime behavior, ML algorithms could establish baselines and detect subtle anomalies. This evolution aimed to reduce false positives and enable early detection of insider threats, compromised credentials, or supply chain attacks.

It includes:

i. Learning typical Docker image changes or IaC edits by a given developer
ii. Alerting on deviation from established deployment sequences
iii. Detecting unauthorized CI agent access based on historical command patterns

Unified Policy + Behavior Engines

A key limitation of early toolchains was the disconnect between policy definition (e.g., OPA rules) and behaviour detection (e.g., Falco logs). The future vision included unified engines that could evaluate both real-time behaviour and declarative policy within the same logic layer. These systems would respond dynamically updating policies or enforcement criteria as behavioural anomalies emerged, effectively closing the feedback loop between detection and prevention.

Such engines could:

i. Trigger policy enforcement based on live behavioral context
ii. Modify access privileges or enforce rate limits on suspicious activity
iii. Integrate runtime intelligence into CI gatekeeping mechanisms

Real-Time Zero Trust Enforcement

ZTA is expected to become the core principle of future security architecture. In a modern DevSecOps pipeline, this means that every person or service must be continuously verified and authorized in real time, based on identity and contextual attributes. Access decisions no longer depend on the network perimeter, but instead on factors like user behavior, device state, and live threat intelligence.

Expected capabilities:

i. Real-time policy enforcement at commit, build, and deploy phases
ii. Automated revocation of tokens or privileges upon behavioral anomaly
iii. Integration with cloud-native security frameworks (e.g., Istio, SPIFFE)
    - SPIFFE assigns dynamic, short-lived identities to workloads, allowing context-aware authentication between services.
    - Istio enforces runtime policies and secures service-to-service communication using mutual TLS, traffic policies, and access control layers.

While these predictions point toward a dynamic, self-defending pipeline, real-world implementation remains challenging. AI-based behavioral analysis depends on high-quality, labeled training data which may be scarce or biased in live DevSecOps environments. Likewise, deploying identity-aware controls like SPIFFE and Istio at scale can incur significant infrastructure overhead, configuration complexity, and require organizational maturity in DevOps practices. Bridging the gap between ideal and practical ZTA adoption demands thoughtful integration planning, strong data governance, and robust resource allocation.

Shift from Perimeter to Identity & Behavior-Based Controls

Traditional perimeter tools were becoming less useful with the rise of microservices, temporary containers and work done remotely. The future approach aims to guide control models based on identity and behavior. Security would mirror how much work the code handles, protecting it equally whether on-premises, in the cloud or in a mixture of both.

Strategic shifts included:

i. Fine-grained access control based on developer roles and behavior
ii. Continuous verification of workloads and agents
iii. Decentralized policy distribution aligned with GitOps principles

This vision marked a shift from reactive security to a proactive, adaptive DevSecOps pipeline that learns, enforces, and evolves with real-time intelligence.

CONCLUSION AND RECOMMENDATIONS

Conclusion

The article has highlighted the importance of adding policy enforcement and threat detection to CI/CD pipelines as the security practices in DevSecOps grow. It was the increasing difficulty in software supply chains and new kinds of serious threats, like attacks from within a company and runtime attacks that encouraged the move from traditional DevOps to DevSecOps. With static application security testing (SAST), infrastructure code scans, compliance-as-code and Falco and OSSEC behaviour monitoring, we were able to detect and mitigate risks at the earliest stage.

Nevertheless, these earlier designs depended on fixed rules, which meant they were not good at low false positives and missed changes in attack patterns. The use of different security tools at various points in development, build and runtime also made it difficult to see everything clearly or relate it all together. These holes made it obvious that we needed more coordinated and smart security approaches.

Despite these limitations, the combination of security policy enforcement with behavior-based detection marked a crucial advancement in shifting security left and embedding it deeply within the software lifecycle. The case studies and practical pipeline integrations discussed illustrate the tangible benefits and feasibility of this approach using technologies.

Moving forward, DevSecOps is set to improve through more automation, smarter features and machine learning that creates changing behaviour models. The use of one security system and fast zero trust checks will be necessary to defend today's complex and dispersed application environments. Organisations must make security part of the code and keep an eye on behaviours to remain resilient to severe threats as they speed up software production.

Besides, the results from DevSecOps adoption give useful knowledge for shaping future security approaches. People in organisations must understand that security requires continual effort and teamwork from developers, security experts and operations staff. If teams are aware of security and use advanced detecting and enforcing tools, they will be ready to react to emerging threats. Eventually, as DevSecOps practices improve, both fast development and security will work together seamlessly in software.

Recommendation

Based on the findings, it can be stated that a range of strategic suggestions can be made to improve the safety verification of the security policy and increase the identification of behavioral threats in DevSecOps pipelines. First, a layered security is necessary since it involves the usage of both static and dynamic analysis tools (sonarqube + OWASPZAP and infrastructure-as-code (IaC) scanning and compliance-as-code enforcement such as open policy agent (OPA) and inspect. In order to enhance visibility into behaviors, run-time threat detection mechanisms (e.g., Falco or Sysdig Secure) should be deployed to complement the use of anomaly detection using machine learning (e.g., Elastic ML, Splunk UBA) to provide baselines and mitigate false positives. The integration of the tool chain is also to be enhanced

by correlating alerts throughout the code, build, and runtime layers via the SIEM platforms, such as Wazuh or Splunk, as well as automating the response, such as the termination of containers, in the case of the detection of malicious activity. In addition to tooling, DevSecOps culture should be developed, which can be done by introducing shift-left security training opportunities, threat modeling, and having Security Champions, to fill the gaps in the organization. Lastly, to support the changing threats, the organizations are also advised to embrace Zero Trust principles (e.g. SPIFFE/ SPIRE towards workload-identity) and adaptive security models steered by the Artificial Intelligence to predict and prevent future attacks using the previous attack behaviour as training data.

REFERENCES

[1] R. Manchana, "The DevOps Automation Imperative: Enhancing Software Lifecycle Efficiency and Collaboration," Eur. J. Adv. Eng. Technol., vol. 8, no. 7, pp. 100–112, 2021.

[2] R. Kumar and R. Goyal, "When security meets velocity: Modeling continuous security for cloud applications using DevSecOps," in Innovative Data Communication Technologies and Application: Proc. ICIDCA 2020, Singapore: Springer, 2021.

[3] F. Yashu, M. Saqib, S. Malhotra, D. Mehta, J. Jangid, and S. Dixit, "Thread mitigation in cloud native application development," Webology, vol. 18, no. 6, pp. 10160–10161, 2021. [Online]. Available: https://www.webology.org/abstract.php?id=5338s

[4] W. Tounsi and H. Rais, "A survey on technical threat intelligence in the age of sophisticated cyber-attacks," Comput. Secur. vol. 72, pp. 212–233, 2018.

[5] Y. Smeets, "Improving the adoption of dynamic web security vulnerability scanners," M.S. thesis, Radboud Univ., Nijmegen, Netherlands, 2015.

[6] V. Lenarduzzi et al., "Are sonarqube rules inducing bugs?," in Proc. 27th IEEE Int. Conf. Softw. Anal., Evol. Reeng. (SANER), 2020, pp. 217–227.

[7] F. Hoces de la Guardia, S. Grant, and E. Miguel, "A framework for open policy analysis," Sci. Public Policy, vol. 48, no. 2, pp. 154–163, 2021.

[8] C. A. Cois, J. Yankel, and A. Connell, "Modern DevOps: Optimizing software development through effective system interactions," in Proc. IEEE Int. Prof. Commun. Conf. (IPCC), 2014, pp. 1–5.

[9] D. H. Ryu, H. Kim, and K. Um, "Reducing security vulnerabilities for critical infrastructure," J. Loss Prev. Process Ind., vol. 22, no. 6, pp. 1020–1024, 2009.

[10] J. Hamunen, "Challenges in adopting a Devops approach to software development and operations," M.S. thesis, 2016.

[11] W. J. Heinbockel, E. R. Laderman, and G. J. Serrao, "Supply chain attacks and resiliency mitigations," The MITRE Corporation, 2017, pp. 1–30.

[12] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, "Behavior-based features model for malware detection," J. Comput. Virol. Hacking Tech., vol. 12, pp. 59–67, 2016.

[13] P. Bitra and C. S. Achanta, "Development and Evaluation of an Artefact Model to Support Security Compliance for DevSecOps," 2021.

[14] J. Yang et al., "Towards better utilizing static application security testing," in Proc. 2019 IEEE/ACM 41st Int. Conf. Softw. Eng.: Softw. Eng. Pract. (ICSE-SEIP), 2019, pp. 525–534.

[15] S. Chinamanagonda, "Automating Infrastructure with Infrastructure as Code (IaC)," SSRN, 2019. [Online]. Available: https://ssrn.com/abstract=4986767

[16] S. R. Gopireddy, "Automated Compliance as Code for Multi-Jurisdictional Cloud Deployments," Eur. J. Adv. Eng. Technol., vol. 7, no. 11, pp. 104–108, 2020.

[17] A. Bahaa et al., "Monitoring real time security attacks for IoT systems using DevSecOps: a systematic literature review," Information, vol. 12, no. 4, p. 154, 2021.

[18] P. Cui, DevSecOps of Containerization, Ph.D. dissertation, Auburn Univ., 2020.

[19] B. A. Kuperman and E. H. Spafford, "Audlib: a configurable, high-fidelity application audit mechanism, Softw" Pract. Exp., vol. 40, no. 11, pp. 989–1005, 2010.

[20] J. Diaz et al., "Self-service cybersecurity monitoring as enabler for DevSecOps," IEEE Access, vol. 7, pp. 100283–100295, 2019.

[21] B. Jammeh, "DevSecOps: Security expertise a key to automated testing in CI/CD pipeline," M.S. thesis, Bournemouth Univ., 2020.

[22] J. Martínez and J. M. Durán, "Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study," Int. J. Safety Secur. Eng., vol. 11, no. 5, pp. 537–545, 2021.

License