

1 Abstract

Mit DevGuard stellen wir ein Open-Source-Projekt vor, das Sicherheit für Entwickler:innen zugänglich, pragmatisch und automatisierbar macht. Angesichts zunehmender Angriffe auf Software-Supply-Chains und wachsender regulatorischer Anforderungen (z.B. CRA) adressiert DevGuard zentrale Herausforderungen der sicheren Softwareentwicklung: komplexe Tool-Integration, unübersichtliche Ergebnisse, hohe False-Positive-Raten und fehlende Compliance-Nachweise.

DevGuard bündelt bewährte Open-Source-Scanner in einem entwicklerfreundlichen CLI-Wrapper, integriert nahtlos in CI/CD-Pipelines und unterstützt u.a. SBOMs, SAST, SCA, Secret Scanning und Compliance as Code. Durch automatische Risikoanalysen, VEX-Support und Community-gestützte Bewertungen reduziert es Aufwand und Frustration. Tools wie in-toto sorgen für nachvollziehbare Supply-Chain-Sicherheit, Attestations und Rego-Policies ermöglichen eine durchsetzbare Compliance-Automatisierung.

In unserem Talk zeigen wir Funktionen, Architektur und wie DevGuard durch einfache Nutzung ein starkes Sicherheitsniveau fördert – von der Community, für die Community.

2 Froscon CFP – DevGuard

Relevante Links:

DevGuard Website
DevGuard Dokumentation
DevGuard GitHub Repository
DevGuard Live

2.1 Mission

DevGuard ist ein Open-Source-Projekt, entwickelt von Entwickler:innen für Entwickler:innen, mit dem Ziel, Sicherheit nahtlos in den Softwareentwicklungsprozess zu integrieren. Sicherheit soll für alle Entwickler:innen zugänglich und effizient umsetzbar sein – unabhängig vom vorhandenen Security-Know-how. DevGuard soll durch die Erhöhung der Usability von Softwaresicherheits-Tools das allgemeine Security-Level der Softwareentwicklung erhöhen und Angriffe auf die Software-Supply-Chain erschweren.

2.2 Problemstellung

Mit dem Anstieg von Angriffen auf Software und deren Lieferketten (z. B. SolarWinds, XZutils) und zunehmenden regulatorischen Anforderungen, wie dem Cyber Resilience Act (CRA), steigen die Anforderungen an sichere Softwareentwicklung massiv. Gleichzeitig fehlt es vielen Entwickler:innen an Ausbildung in der Entwicklung sicherer Software und die Kenntnis über passende Tools, um diesen Anforderungen gerecht zu werden. Zwar existieren viele leistungsfähige Open-Source-Security-Scanner, doch deren Integration ist komplex, die Ergebnisse oft schwer bewertbar und bis zu 80% der Funde können False Positives sein, was einen hohen Zeitverlust und zu einer gleichzeitigen hohen Frustration der Entwickler:innen führt. Auch nehmen die Anforderungen an die Dokumentation der Schwachstellen und deren Behandlung, die durch Regularien gefordert werden, zu, was zu einer weiteren Belastung der Entwickler:innen führt.

2.3 Zielsetzung

DevGuard bietet eine Entwickler:innen freundliche Lösung für das zentrale Schwachstellenmanagement, die Absicherung der Software-Supply-Chain sowie die Umsetzung von Compliance as Code. Es kombiniert bewährte Open-Source-Werkzeuge in einem einfach integrierbaren CLI-Wrapper, automatisiert Risikoanalysen und hilft dabei, relevante Schwachstellen zu priorisieren und korrekt zu beheben.

2.4 Funktionen

- **Developer-zentrierte Integration:** Nahtlose Einbindung in bestehende CI/CD-Pipelines, unterstützt die OWASP DevSecOps Pipeline mit Tools für SCA, Container Scanning, IaC Scanning, SAST,

DAST und Secret Scanning (teilweise in Vorbereitung).

- **Automatisiertes Security-Monitoring:** Erstellung und kontinuierliche Auswertung von SBOMs zur Überwachung von Abhängigkeiten und des Softwarezustands.
- **Risikobewertung:** Priorisierung durch pragmatische Automatisierung basierend auf CVSS, ExploitDB, Tiefe der Abhängigkeit und EPSS.
- **Compliance-Unterstützung:** Unterstützung bei der Einhaltung gängiger Standards wie ISO/IEC 27001 oder PCI-DSS.
- **Compliance as Code:** Automatisierte Sicherheitsprüfungen über signierte Attestations, die z. B. SBOMs oder Schwachstellenreports enthalten. Diese werden mit `cosign` erzeugt und signiert, mit Rego-Policies (OPA) geprüft und durch den Sigstore Policy Controller in Kubernetes erzwungen.
- **Supply Chain Security:** Integration von Projekten wie `in-toto` zur Nachvollziehbarkeit und Absicherung des Software-Build-Prozesses. Perspektivisch wird auch die Unterstützung für die Umsetzung des SLSA-Framework angestrebt.

3 Kernkonzepte

Im Folgenden werden die 3 Kernkonzepte und deren Umsetzung in DevGuard beschrieben:

3.1 Zentrales Schwachstellenmanagement

DevGuard verfolgt das Ziel, ein zentralisiertes und Entwickler:innen zentriertes Schwachstellenmanagement bereitzustellen, das sich nahtlos in moderne CI/CD-Workflows integrieren lässt. Die Grundlage bildet eine vollständige Umsetzung der OWASP DevSecOps Pipeline (siehe Abbildung 4.1) – angereichert durch automatisierte Nachweisführung in Form von Attestations und kontinuierlicher Schwachstellenüberwachung.

3.1.1 Integration in die CI/CD-Pipeline

Die DevGuard CI/CD-Komponenten lassen sich direkt in Repositorys auf GitHub oder GitLab integrieren. Nach der Integration übernimmt die Pipeline automatisiert verschiedene Sicherheitsscans (z. B. SCA, Container Scanning, IaC Scanning) und generiert Prüfberichte im SBOM- oder SARIF-Format. Diese werden analysiert und die gefundenen Schwachstellen werden zentral in DevGuard erfasst und bewertet. Zusätzlich werden für durchgeführte Prüfungen kryptografisch signierte Attestations erzeugt, die zur Compliance-Dokumentation dienen.

3.1.2 Zentrale Datenbasis

Die Grundlage für das Schwachstellenmanagement bildet eine aggregierte Schwachstellendatenbank, die alle sechs Stunden aktualisiert wird. Sie kombiniert Informationen aus mehreren etablierten und vertrauenswürdigen Quellen, darunter:

- CVE / NVD
- Exploit DB
- GitHub Exploits
- OSV (inkl. PyPI, Go, Rust, etc.)
- EPSS (Exploit Prediction Scoring System)
- OpenSSF, Ubuntu, Bitnami, OSS-Fuzz u. v. m.

Diese breite Datenbasis ermöglicht eine fundierte Risikobewertung und Priorisierung der Schwachstellen auf Basis von Ausnutzbarkeit, CVSS-Schweregrad und realer Bedrohungslage.

3.1.3 Risikobasierte Priorisierung

Nicht jede gefundene Schwachstelle ist unmittelbar kritisch. DevGuard ersetzt die klassische Bewertung nach CVSS (z. B. `--severity CRITICAL`) durch eine praxisnahe Risikobetrachtung (z. B. `--risk CRITICAL`). Dies geschieht durch Kombination von Metriken wie EPSS, ExploitDB-Präsenz und **Projektkontext**. So werden Ressourcen gezielt auf wirklich relevante Schwachstellen gelenkt.

3.1.4 Vulnerability Management und Synchronisation

DevGuard bietet eine zentrale Übersicht aller Schwachstellen eines Projekts. Entwickler:innen können diese eigenhändig nach-priorisieren, kommentieren und direkt aus DevGuard heraus manuell oder automatisiert Tickets in GitLab, GitHub und in Zukunft Jira erzeugen. Die Bearbeitung erfolgt anschließend in der bekannten GitHub oder GitLab Umgebung bequem per Slash-Commands aus dem Issue.

3.1.5 Gemeinsames Schwachstellenmanagement mit VEX und Crowd-Input

Zur Reduzierung von False Positives und doppelter Analysearbeit nutzt DevGuard den **Vulnerability Exploitability eXchange (VEX)**-Standard. Damit können Projekte maschinenlesbar dokumentieren, dass eine bestimmte Schwachstelle zwar formal vorliegt, aber im eigenen Kontext nicht ausnutzbar ist (False-Positive). Diese Einschätzung wird über VEX verfügbar gemacht und schützt auch andere Nutzer vor unnötigem Aufwand. Jeder DevGuard-Nutzer kann zu jederzeit den eigenen VeX seiner Software einsehen und mit Nutzern teilen.

Ergänzend wird Wissen der „Crowd“ genutzt: Wenn eine ausreichend große Zahl an Nutzern eine Schwachstelle als irrelevant einstuft, kann diese Information als verifizierter Community-Eintrag übernommen werden. So entsteht ein intelligentes, kollaboratives Schwachstellenmanagement.

3.2 Supply-Chain-Security

Die Absicherung der Software Supply Chain ist ein zentraler Bestandteil moderner Softwareentwicklung. DevGuard integriert Sicherheitstools wie `in-toto` und soll etablierte Frameworks wie SLSA (Supply Chain Levels for Software Artifacts) umsetzen, um die Integrität der gesamten Build- und Deployment-Pipeline sicherzustellen – automatisiert, nachvollziehbar und mit minimalem Konfigurationsaufwand für Entwickler:innen.

3.2.1 Integration von `in-toto`

`In-toto` ist ein Framework zur Integritätssicherung von Softwarelieferketten. Es erstellt kryptografisch signierte Link-Dateien für jeden Schritt der Pipeline, die genau dokumentieren:

- Welche Dateien (Materials) in einen Schritt eingeflossen sind.
- Welche Artefakte (Products) daraus erzeugt wurden.
- Welche Kommandos ausgeführt wurden.
- Wer den Schritt ausgeführt hat (Signatur).

Diese Link-Dateien werden später gegen eine `root.layout` geprüft, die die gesamte erwartete Supply Chain, autorisierte Schlüssel und erlaubte Materialien/Produkte definiert.

3.2.2 Automatisierte Umsetzung mit DevGuard

DevGuard vereinfacht die sonst komplexe Nutzung von `in-toto` durch:

- **Automatische Layout-Generierung:** DevGuard erstellt und verwaltet die `root.layout`-Datei dynamisch basierend auf den verwendeten CI/CD-Workflows.
- **Transparentes Key-Management:** Entwickler:innen verwenden Zugangstoken, DevGuard verwaltet die Public Keys. Private Keys verbleiben lokal.
- **CI/CD-Integration:** Link-Dateien werden automatisch pro Pipeline-Schritt erstellt, signiert und in DevGuard gespeichert.

- **Kontinuierliche Verifikation:** Bei Abschluss der Pipeline wird die gesamte Kette automatisch gegen die `root.layout` verifiziert. Ein API-Endpoint stellt das Ergebnis zur Verfügung.

3.2.3 SBOM-Integration zur Komponentenübersicht

Neben der Nachvollziehbarkeit der Build-Schritte, bietet DevGuard eine vollständige Übersicht über alle verwendeten Softwarekomponenten. Mittels SBOMs (Software Bill of Materials), die automatisch erzeugt und ausgewertet werden, erkennt DevGuard bekannte Sicherheitslücken in eingesetzten Abhängigkeiten. Diese Informationen fließen in das zentrale Schwachstellenmanagement und ermöglichen ein sicheres Lifecycle-Management aller Komponenten.

3.2.4 SLSA-Kompatibilität und attestations

Zur weiteren Härtung der Lieferkette erstellt DevGuard Attestations (signierte JSON-Dateien), die Metadaten zur Build-Herkunft, durchgeführten Scans, genutzten Tools, Signaturen oder Lizenzinformationen enthalten. Diese Attestations werden im CI/CD-Prozess generiert und in der Container-Registry sowie bei DevGuard gespeichert.

Für erhöhte Vertrauenswürdigkeit unterstützt DevGuard auch die Erzeugung von SLSA-konformer Provenance. Durch Nutzung von GitLab Multi-Project Pipelines kann DevGuard Attestations sogar mit einem eigenen, vertraulichen Schlüssel signieren, der dem Maintainer nicht bekannt ist. So entsteht eine fälschungssichere Beweiskette, gemäß SLSA Level 3.

3.3 Compliance as Code

Compliance as Code verfolgt das Ziel, Sicherheits- und Compliance-Prüfungen automatisiert und reproduzierbar in moderne CI/CD-Pipelines zu integrieren. Anstelle manueller Prüfungen werden sogenannte *Attestations* verwendet (kryptografisch signierte JSON-Dokumente), die als Nachweis dienen, dass bestimmte Sicherheitsprüfungen erfolgreich durchgeführt wurden.

Typische Inhalte solcher Attestations sind:

- Generierung einer Software Bill of Materials (SBOM),
- durchgeführte Schwachstellen-Scans,
- Prüfung auf kritische Schwachstellen,
- Verwendung vertrauenswürdiger Build-Tools,
- kryptografische Signatur durch eine autorisierte Instanz.

Die Attestations lassen sich mit `Open Policy Agent` (OPA) und der Policy-Sprache `Rego` verifizieren. `Rego`-Policies prüfen gezielt Inhalte der Attestations – etwa, ob bestimmte Schlüssel existieren oder Werte gültig sind.

Für die automatische Durchsetzung der Compliance-Anforderungen kommt der `Sigstore Policy Controller` in Kubernetes-Clustern zum Einsatz. Dieser verwendet `ClusterImagePolicies`, um Regeln zu definieren, welche Images unter welchen Bedingungen ausgeführt werden dürfen. Wird eine Attestation nicht erfüllt oder nicht vertrauenswürdig signiert, wird das entsprechende Deployment blockiert.

DevGuard automatisiert diesen kompletten Prozess. Es erstellt Attestations entlang der DevSecOps-Pipeline, lädt sie automatisch hoch und unterstützt Organisationen dabei, individuelle oder standardisierte Sicherheitsrichtlinien (z. B. nach ISO/IEC 27001 oder PCI-DSS) effizient durchzusetzen. So wird Compliance Teil des automatisierten Entwicklungsprozesses – nachvollziehbar, überprüfbar und sicher.

4 Technische Details

4.1 Verwendeter Tech-Stack

- Golang
- TypeScript

- Container
- Kubernetes (Helm-Chart)

4.2 DevSecOps Pipeline

DevGuard unterstützt die Entwickler:innen dabei, die OWASP DevSecOps Pipeline (Abbildung 1: 4.1) mit einer kuratierten Open Source Security Scanner - Liste umzusetzen. **Badge-Programm UI:**

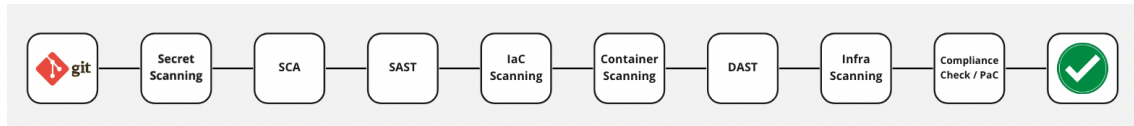


Abbildung 4.1: OWASP - DevSecOps Pipeline

Aktuell verwendete Scanner

- **secret scanning:** gitleaks
- **sast:** semgrep
- **iac:** checkov
- **sca:** trivy
- **container-scanning:** trivy
- **signing:** cosign

DevGuard verfügt über eine SARIF-Schnittstelle. Dieses Dateiformat wird von den meisten gängigen Analysetools unterstützt. Über die SARIF-Schnittstelle können somit weitere, eigene Tools an DevGuard angebunden werden. So können Entwickler:innen auch selbst gewählte Security-Scanner nutzen und die Ergebnisse über DevGuard verwalten.

5 Inhalte des Froscon Vortrags

Auf der diesjährigen FrOSCon möchten wir das DevGuard-Projekt erstmals einem breiten Fachpublikum vorstellen. Unser Ziel ist es, konstruktives Feedback zur bisherigen Umsetzung einzuholen, neue Anwendungsfälle zu diskutieren und gemeinsam mit der Community Potenziale für Weiterentwicklungen zu identifizieren.

Darüber hinaus möchten wir Entwicklerinnen und Entwickler:innen motivieren, DevGuard zur Erhöhung der Sicherheit in ihren eigenen Softwareprojekten einzusetzen – unabhängig davon, ob es sich um persönliche Projekte, Open-Source-Initiativen oder unternehmensinterne Anwendungen handelt. DevGuard soll ein Projekt *von der Community, für die Community* sein. Deshalb freuen wir uns über jede Form der Beteiligung: sei es durch das aktive Mitwirken an der Entwicklung, das Erstellen von Issues oder Feature Requests, das Teilen von Erfahrungsberichten oder durch Beiträge zur Dokumentation.

Gemeinsam wollen wir ein Werkzeug schaffen, das moderne Sicherheitsanforderungen pragmatisch adressiert und gleichzeitig einfach nutzbar bleibt.

Agenda:

- Vorstellung des Projekts
- Einführung in die Funktionen
- Vorstellung der technischen Details
- Vorstellung der Road-Map
- Call to Action