SignPath

Software Integrity in the Age of Rising Supply Chain Attacks

Protecting Code and Pipelines from Emerging Threats, and enforcing Compliance

Table of content

Introduction: A False Sense of Security 3

The Rise of Software Supply Chain Attacks 4

What you need to know about Software integrity? 5

- What Exactly is Software integrity? 5
- What makes software supply chains an attractive target for attackers? 5
- Why is ensuring software integrity so difficult? 6
- Industry response and current challenges 6

Rethinking Code Signing for Modern DevSecOps 7

SignPath DevSec360: The New Standard for Zero Trust Software Security 8

- DeepSign 9
- Pipeline Integrity 11
- Integration into the development process 12
- Problems Solved & Key Benefits 13

Four-Stage Maturity Model for Secure Pipelines 15

Conclusion: From Technical Detail to Strategic Imperative 16

About SignPath 17

Introduction:
A False Sense of Security

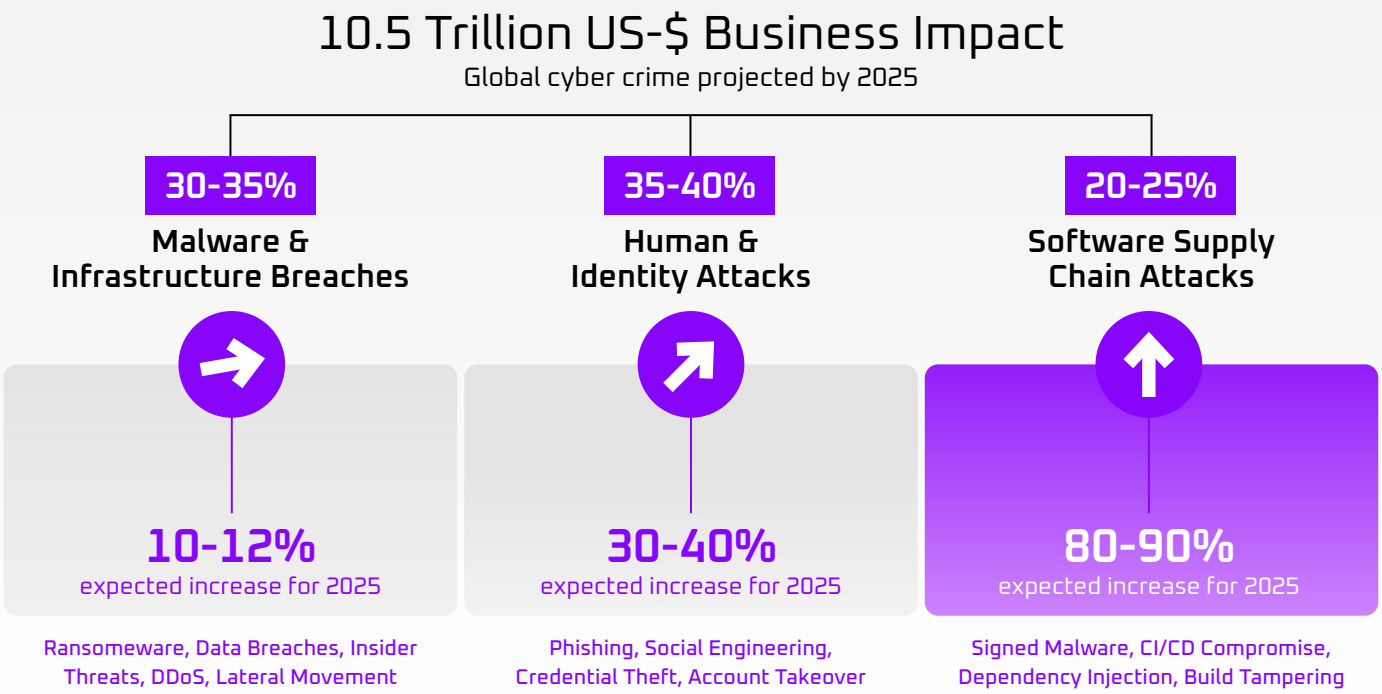
Software supply chain attacks have become one of the most dangerous threats to IT systems. They allow attackers to infiltrate entire networks by compromising a single weak link in the supply chain, including Open-Source projects, software vendors, contractors, or internal development teams. Their impact is amplified by the fact that modern software ecosystems are complex and interconnected, including CI/CD pipelines and automated deployment tools. Software powers nearly every critical business function today, and this extensive reliance creates a broad attack surface. When trust in the integrity of a single component or process is broken, it can cascade through the entire system and lead to widespread compromise and operational disruption.

And it's not just about malicious actors. Violations of software integrity can also occur unintentionally, through human error, flawed processes, or automated workflows with no real oversight. Software is now developed in complex and fast-moving environments that often prioritize speed and agility over security. This makes it incredibly difficult to ensure, let alone verify, that code remains trustworthy, tamper-free, and secure throughout its lifecycle.

Traditional perimeter-focused security controls such as firewalls, endpoint protection, and access management are no longer sufficient to address the risks inherent in modern software supply chains. While these measures protect runtime environments, they do little to safeguard the integrity of development pipelines, build processes, or third-party dependencies. As a result, organizations often overestimate their security posture, unaware that attackers can exploit upstream components long before software reaches production.

Guaranteeing software integrity isn't just a technical challenge anymore; it has become a strategic pillar of cybersecurity. In this paper, we'll explore why maintaining integrity is so hard, why traditional approaches have failed and how modern organizations can finally take control of their software supply chains.

The Rise of Software Supply Chain Attacks



Cybercrime causes massive business impact; software supply chain attacks are increasing disproportionately.

Figure 1: Cyberattacks are rising sharply across almost all vectors [1]

Cyberattacks are increasing across all fronts, but one trend is particularly alarming: the surge in software supply chain attacks. Long overshadowed by more visible threat vectors, this area has rapidly become one of the most critical challenges in cybersecurity. Already responsible for 20 to 25 percent of all security incidents, software supply chain attacks are growing at an unprecedented pace. Between 2021 and 2023 alone, the number of such attacks increased by over 430 percent. No other category is growing faster. This steep rise underscores the urgency to secure the entire development and delivery process, not just endpoints and infrastructure.

During 2025, 45% of organizations are expected to be affected by software supply chain attacks, with an 80–90% increase projected.

These attacks exploit build infrastructure, dependencies, and developer workflows to compromise signed software releases, making them hard to detect and expensive to contain. Critical components of modern software delivery such as CI/CD pipelines and open-source packages are increasingly targeted.

This shift signals a dangerous evolution: threat actors are moving upstream, embedding themselves deeper into the software development lifecycle. The result is a growing class of vulnerabilities that cannot be mitigated by endpoint or perimeter security alone.

To respond effectively, organizations need to rethink their security approach by placing greater emphasis on software integrity: securing development and release pipelines and fostering end-to-end trust across the software supply chain.

Without urgent action, the compounded effect of these attack vectors will erode digital trust, disrupt operations, and expose businesses to long-term financial and reputational damage.

[1] https://www.bdemerson.com/article/complete-cybercrime-statistics

What you need to know about Software integrity?

What Exactly is Software integrity?

Software integrity is a broad term that covers several critical properties of software, and of the systems running it.

Software integrity ensures that all software components remain authentic, verifiable, and free from interference across the entire software lifecycle, from development to execution. The following key objectives can be grouped into two focus areas: Software Integrity and Code Integrity.

Software Integrity	Process Integrity	Software production process	No unauthorized changes are made to the source code, third-party components, build environments, configurations, or any other part of the production. Policy conformance and alignment with a secure software development lifecycle (SSDLC).
		Release process	Only tested and approved releases enter distribution and update channels.
	Code Integrity	Deliverables	Software files originate from the stated publisher and have not been modified by third parties.
		Installation and update process	Only verified and authorized software files are installed and updated on target systems.
		System	Only approved and verified software is running on a given system. Nothing more, nothing less.

Table 1 - Integrity guarantees

Together, these principles form the foundation for maintaining trust in software, not just during development, but across every stage of its delivery and use.

What makes software supply chains an attractive target for attackers?

Ensuring the integrity of the entire software lifecycle is far more than just a recommended best practice. It's a critical line of defense against modern cyberattacks.

From an attacker's perspective, compromised software is a highly attractive entry point: if malicious code is successfully introduced into a target system, it often already has access to sensitive data. If not, it creates an initial foothold from which attackers can move laterally within the IT environment, escalate privileges, and ultimately compromise sensitive systems. Sometimes this happens through an open "front door," but more often through poorly secured side entrances, such as those commonly found in development environments or within the software supply chain. Once established, these attacks are often hard or impossible to detect.

Development and CI/CD environments are often much more flexible and less restrictive than production systems. Development teams frequently have broad privileges, use a wide range of tools, and have direct control over build processes and deployment artifacts. While this freedom accelerates software delivery, it also significantly expands the attack surface. The risk becomes especially high when security policies are bypassed, incomplete patches are applied, or unvetted fixes are deployed. These vulnerabilities often go unnoticed until they are actively exploited.

Many software vendors and subcontractors operate with smaller cybersecurity budgets and less stringent security requirements than their customers such as large enterprises or government agencies. This makes them attractive targets for attackers seeking to gain access to downstream environments via trusted components with potentially devastating impact: a successful attack on a widely used component can affect thousands or even millions of systems worldwide.

Therefore, protecting source code alone is not enough. What matters is end-to-end integrity. From the very first line of code through every review, handoff, build process, and testing environment, all the way to final deployment. Only through verifiable workflows, cryptographically secured artifacts, and strict policy enforcement can a consistently trustworthy software development process be established, forming the foundation for genuine supply chain security.

Why is ensuring software integrity so difficult?

The challenges are numerous. Modern attackers deploy increasingly sophisticated methods that directly target every stage of the software lifecycle: from production through distribution, deployment, and updates. Without adopting a zero-trust mindset for these processes, many opportunities exist for software integrity to be compromised.

Any breach of network boundaries, or unauthorized access to user accounts or credentials, can be the starting point for an attack. Insider threats, whether disgruntled employees or external contractors with broad permissions, pose a major risk. Upstream components, social engineering, and phishing are common methods for attackers to gain the access they need to manipulate software releases.

Moreover, security that depends on human carefulness or adherence to policies is fragile. Simple negligence can have catastrophic consequences. Paper policies that are difficult to enforce, or that obstruct agility and productivity, often fail in practice.

Attackers only need to find a single vulnerability or oversight, while defenders must be perfect. This imbalance is especially stark in software production, where the attack surface is disproportionately large due to the complex and intentionally agile interactions of people, tools, and processes.

Industry response and current challenges

The industry's response has been mixed. On one hand, application security has received increased attention, and there are promising initiatives to improve software supply chain transparency, such as the standardization of Software Bills of Material (SBOMs). These efforts aim to help organizations understand and track the components used in their software.

On the other hand, the fundamental challenge remains: protecting software integrity from advanced and persistent attacks is still in its early stages of attention and investment. Many organizations have yet to implement comprehensive solutions or adopt the zero-trust principles needed to defend against these threats effectively.

Real-world wake-up call: the SolarWinds "Sunburst" attack

This trend has been visible since the 2010 Stuxnet[2] attack, but it was the 2020 "Sunburst" attack on U.S. software vendor SolarWinds that served as a massive wake-up call for the IT industry.

A highly skilled Russian Advanced Persistent Threat (APT) group infiltrated SolarWinds' network and employed sophisticated techniques to insert backdoors into its flagship product.

These backdoors were then distributed via regular software updates to approximately 18,000 organizations worldwide, including roughly 80% of S&P 500 companies and numerous government agencies across several countries. The attackers used this foothold to conduct espionage and data theft on a massive scale.

Despite the severity of this attack, supply chain compromises have continued to occur, underlining how difficult it is to fully secure software supply chains and maintain software integrity.

SOLARWINDS

[2] Stuxnet, discovered in 2010, was the first widely published and discussed instance of a cyberweapon specifically designed to target industrial control systems (ICS). The worm exploited multiple zero-day vulnerabilities and made use of stolen digital code-signing certificates to appear trustworthy and bypass security controls. This attack highlighted the critical importance of software integrity, demonstrating how manipulated software components and weak verification practices in the software supply chain can be leveraged to execute highly targeted and destructive operations.
See also: https://simple.wikipedia.org/wiki/Stuxnet

Rethinking Code Signing for Modern DevSecOps

Code signing is widely regarded as a cornerstone of software integrity. It enables users and systems to detect tampered artifacts, identify impersonation attempts, and verify that a given software component has not been modified after publication. With most platforms offering build-in or easily configurable enforcement policies for verifying code signing signatures, code signing has become both a standard and convenient method for establishing trust.

But this sense of trust can be misleading. While code signing is critical, it often creates a false sense of security. Many organizations assume that a signed artifact is automatically safe. In reality, signed malware, misused credentials, and compromised signing workflows have been central to some of the most damaging supply chain attacks in recent years.

Historically, attackers exploited weak key protection. Once inside a publisher's network, they could locate private code signing keys, often stored in plain files and protected only by hardcoded passwords. Although these practices have long been discouraged, they remained widespread until industry regulations in 2023 began requiring hardware-protected key storage for many platforms.

That change has reduced the risk of key theft. But it has not eliminated the threat, it has merely shifted it.

Modern software development is fast-paced and automated. Manual signing processes involving physical tokens or user intervention do not scale well. They are too slow, too error-prone, and their perceived security benefit is often undermined by repetition and operational fatigue. To keep up, many organizations now rely on fully automated code signing systems that will sign anything presented with valid credentials.

The consequence is that while private keys may be secure from direct theft, attackers are now focusing on compromising build pipelines or directly signing their code by stealing code signing credentials. If they succeed, they can still sign malicious code. The mechanism is protected, but the outcome is the same.

Code signing remains essential, but it is not enough on its own. It must be part of a broader integrity strategy that not only controls who is signing, but also enforces what is being signed, how it is built, and that it meets defined security policies.

Definition Code Integrity

Code Integrity is the assurance that software code remains authentic, unaltered, and originates from a trusted and/or well-known source throughout its lifecycle. It is enforced through digital signatures using public key cryptography. A digital signature allows systems to cryptographically verify that the code has not been modified since it was published and signed by a legitimate developer or organization. Code integrity helps prevent tampering, unauthorized modifications, and the execution of untrusted or malicious code within the software supply chain. It does not provide any guarantees about the software process or security practices, the quality, or even of the intentions of the verified publisher.

But what does it take to truly deliver on code signing's promise of software integrity?

The simplified answer is no longer sufficient: it's not enough to ensure the integrity of artifacts after release - we must evolve from code integrity to holistic software integrity. That means integrating artifact-level protection with robust measures that safeguard the entire software development and delivery process. This includes code reviews, build pipelines, signing and release workflows, key management, and policy enforcement.

Delivering on this promise requires more than a cryptographic signature. It calls for a platform that ensures integrity across the entire software supply chain – secure by design, automated by default, and governed by enforced policies.

SignPath DevSec360: The New Standard for Zero Trust Software Security

DevSec360 represents a holistic 360-degree approach to securing the entire software development lifecycle. The core principle is Zero Trust. No step in the development or release process is trusted blindly; every action must be verifiably legitimate.

DevSec360 is built on two essential components: DeepSign and Pipeline Integrity. DeepSign applies cryptographic signing based on the full structural depth of software artifacts and a broad set of verifiable integrity criteria, ensuring robust end-to-end trust. Pipeline Integrity protects the entire CI/CD pipeline by automatically enforcing policies for source code management, build processes, and signing, ensuring the trustworthiness of the entire software supply chain.

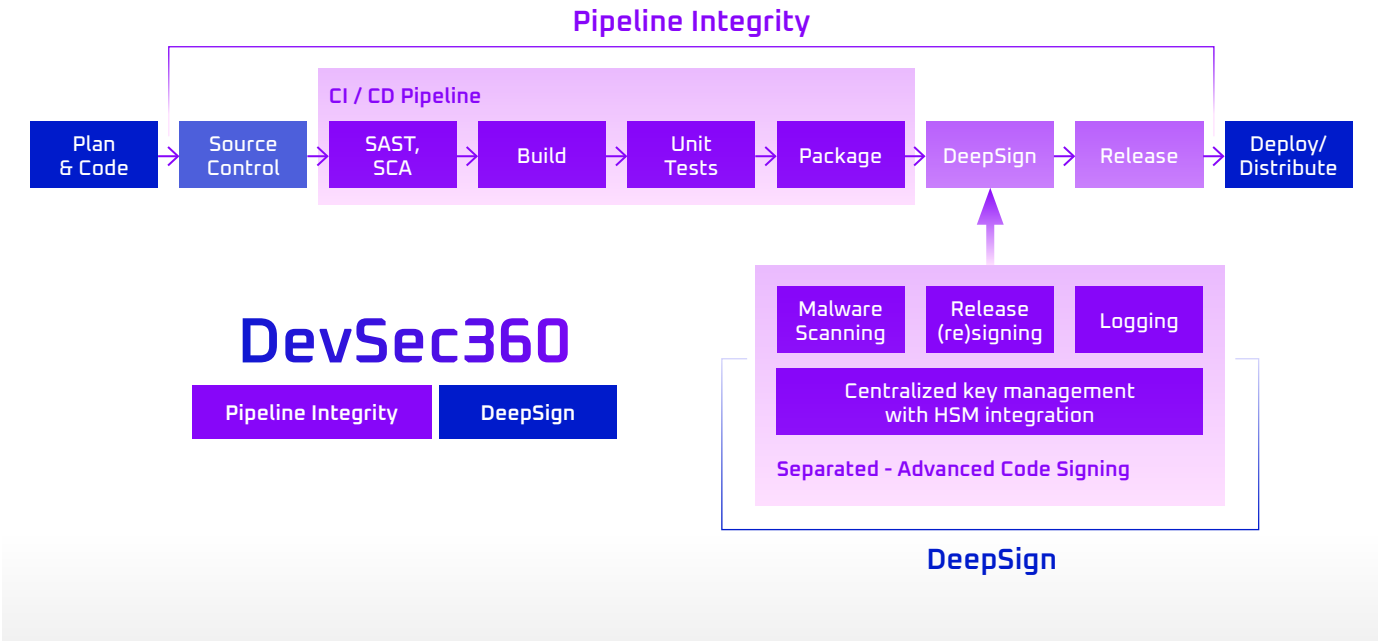


Figure 2: Development workflow with DevSec360

DevSec360 is the first platform to fully implement zero-trust principles across the entire software workflow. The solution integrates seamlessly with existing CI/CD pipelines and provides centralized policy and key management as well as automatic policy enforcement and auditing.

Development teams can continue to work in agile environments while security teams define and enforce policies. DevSec360 bridges both worlds without slowing down developers. In the face of increasing software supply chain attacks, DevSec360 ensures that only verified artifacts from secure processes are signed using the organization's trusted certificates.

This combination of automation and intentional distrust makes the approach unique and highly effective.

DeepSign

Artifact-based signing with deep inspection and zero blind spots

DeepSign is the advanced signing mechanism of the DevSec360 platform. Unlike traditional code signing solutions, which typically transmit only a **hash digest** of the file to the signing service, DeepSign is fully artifact-based: the complete file is submitted to the platform for signing. This gives DevSec360 full visibility into the artifact and enables comprehensive validation before any signature is applied.

Scanning & Verification of Incoming Artifacts: Every file is scanned using up-to-date antivirus engines to detect malware. In addition, the platform validates file structure and metadata against project-specific profiles. For example, checking whether the declared publisher matches the expected identity, or whether an installer includes only approved components. DeepSign can also inspect and validate existing signatures of embedded third-party or upstream components.

Nested Signing Automation: Another key strength of DeepSign is its ability to process nested artifacts for files with signable internal components, such as installers, packages, ZIP archives, or modular applications like Java archives. The platform can automatically unpack these components, verify and sign them individually, and reassemble the full package. This is all done in a single signing process. Developers no longer need to create complex, multi-step signing workflows in their builds.

Definition of a hash digest

A hash digest (or hash code) is a mathematically derived numerical representation of a larger piece of data. Using cryptographic hash algorithms, we can guarantee that a hash digest cannot represent any other data, therefore signing the hash digest is equivalent to signing the data. This leads to efficient processing by dedicated key-protection hardware, but it leaves code signing services oblivious to the nature of the signed artifact.

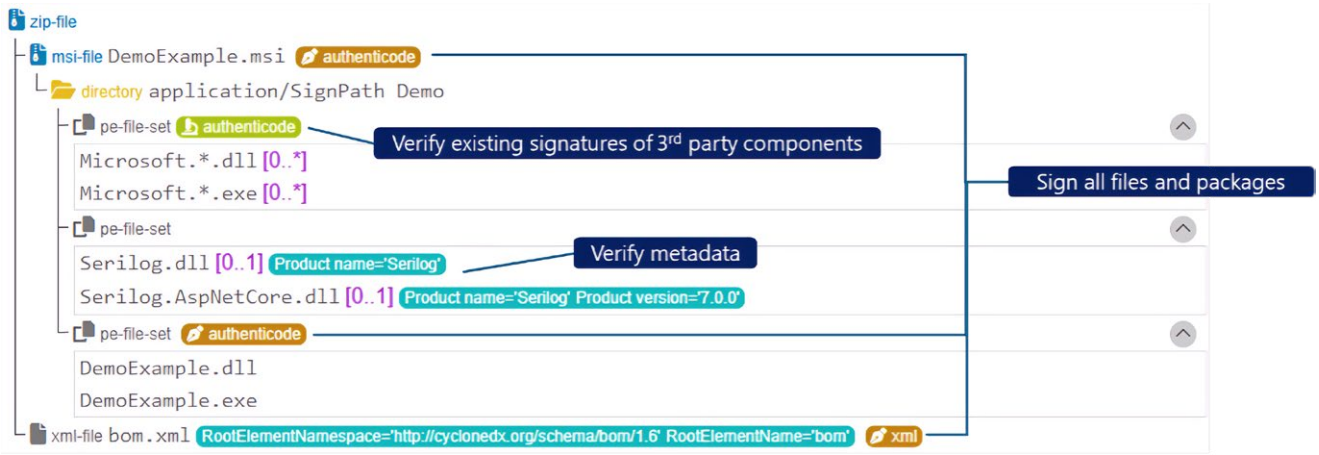


Figure 3 - DeepSign artifact configuration for an installer and its payload

Content-Aware Signing: DeepSign eliminates blind spots in the signing process. The platform knows exactly what it is signing and will reject any signing request if the file doesn't match expected formats or contains unexpected content. Unlike hash-only approaches, this means attackers cannot slip in manipulated files because any deviation is detected and blocked. For development, operations, and end users, this translates into the highest levels of signing security without requiring deep technical effort.

Re-Signing Without Rebuild: Another major advantage: previously signed artifacts can be re-signed at a later stage. This enables promoting release candidates after functional testing and additional security reviews, or handling certificate expiration, algorithm changes, or key compromise. All earlier verifications remain intact, eliminating the need for a full rebuild and saving significant time and drift risk.

Flexible PKI Integration: Cryptographic keys are stored in Hardware Security Modules (HSMs). The DevSec360 platform supports both standard and Extended Validation (EV) certificates, issued by either public Certificate Authorities or private PKIs.

HSM Options	**DevSec360 SaaS** (recommended)	**DevSec 360 Self-Managed**
Shared HSM	**Thales Luna cluster (HA) operated by SignPath**	N/A
Cloud HSM	Thales Luna DPoD Cloud HSM	
Customer-operated HSM	Thales Luna and other vendors	
	On-prem connector	Direct integration

Table 2 - DevSec360 HSM options

Pipeline Integrity

Enforcing Security Policies Throughout the Development Process

Pipeline Integrity monitors and secures all steps leading up to the signing process. While DeepSign focuses on verifying the resulting files, Pipeline Integrity ensures that the process used to create that file is trustworthy.

For every signing request, the platform automatically evaluates the CI/CD pipeline (i.e., the build process), including:

> **Source Code Provenance:** It verifies that the build originates from the correct source code repository and the intended branch, that code reviews were performed, and other repository-level criteria. This prevents any artifact from being signed if it comes from an unauthorized or insufficiently protected repository or branch.
>
> **Build Process Integrity:** Pipeline Integrity ensures the build or its artifacts have not been tampered with. It verifies that every release actually originates from the specific build execution and was not swapped or altered during or afterward. Additional policies include build agent conditions and caching restrictions. Even if signing credentials are misconfigured, an unauthorized build will not be signed.
>
> **Compliance with Development Policies:** Depending on the configured security policy, Pipeline Integrity also ensures all required development procedures were followed. In addition to enforcing a continuous chain of code reviews, it can enforce execution of security scanners and thresholds for security findings. Pipeline Integrity can be configured to only allow signing of artifacts that are the result of reviewed, scanned, and approved source code.

Through these layered checks, Pipeline Integrity in interaction with DeepSign acts as the gatekeeper of the pipeline: only when all conditions are met is the signing approved.

This zero-trust approach avoids problems that originate at the project or team level, such as insecure build configurations, bypassing reviews, skipping tests or releasing unauthorized builds.

By enforcing these measures, Pipeline Integrity guarantees adherence to security policies within the development lifecycle and effectively prevents human error or targeted attacks from compromising the signing and release process.

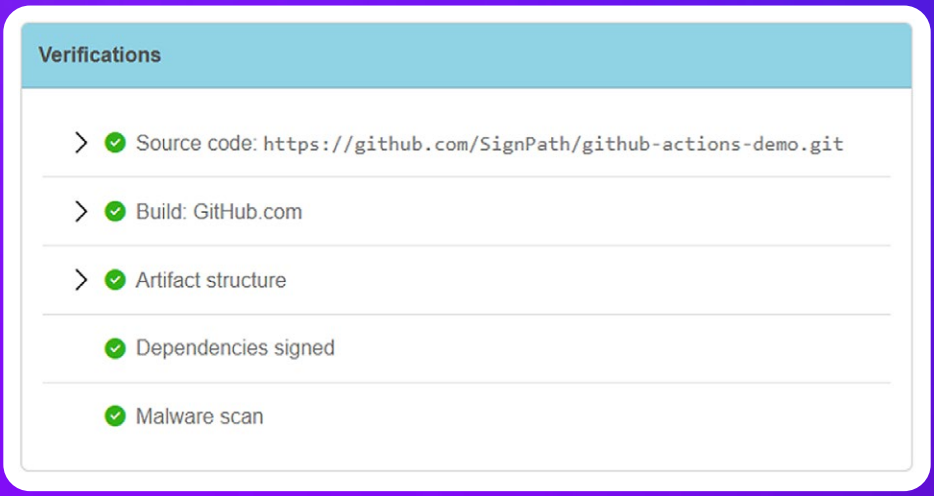


Figure 4 - Overview of verifications in DevSec360

Integration into the development process

How DeepSign and Pipeline Integrity Work Together in Practice

Integrating DeepSign and Pipeline Integrity into existing CI/CD pipelines is straightforward and requires minimal effort. Teams use the DevSec360 plugin for their CI/CD platform to automatically submit release artifacts for signing. Supported platforms include GitHub, GitLab, Jenkins, Azure DevOps, TeamCity and more to come.

Once the plugin is installed and project-specific parameters are configured, the process executes automatically. Each time a build completes, the plugin transfers the release to DevSec360. The platform then verifies artifact and pipeline integrity, enforces all configured policies and only then signs the release's artifacts.

From the development team's perspective, the integration is seamless. Developers continue to work with their usual tools and workflows, while DevSec360 ensures that only verified and policy-compliant artifacts are signed. The signing keys remain securely stored in hardware modules, and the entire process is logged and auditable.

DevSec360 acts as a transparent control layer at the release boundary, providing assurance that every signed artifact meets the defined standards for security and integrity.

```yaml
name: build-and-sign
run-name: Building and signing the software
on:
  push:

jobs:
  build_and_sign:
    runs-on: ubuntu-latest
    steps:

    # ... building the software

    - name: sign
      uses: signpath/github-action-submit-signing-request@v1
      with:
        api-token: '${{ secrets.SIGNPATH_API_TOKEN }}'
        organization-id: '${{ vars.SIGNPATH_ORGANIZATION_ID }}'
        project-slug: 'MyApplication'
        signing-policy-slug:  '${{ env.SIGNPATH_SIGNING_POLICY_SLUG }}'
        github-artifact-id:  "${{steps.upload-unsigned-artifact.outputs.artifact-id}}"
        wait-for-completion: true
        output-artifact-directory: './myapplication-signed'

    # .. deploy steps
```

Figure 5 - Using DevSec360 in a GitHub Actions workflow

Problems Solved & Key Benefits

In summary, the DevSec360 Platform, featuring DeepSign, and Pipeline Integrity, delivers comprehensive protection for software development and delivery.

Development teams benefit from a seamless, automated workflow, while security teams gain assurance that every released software package is thoroughly validated, compliant, and securely signed.

This combination of integrity, efficiency, tracebility and transparancy lays the foundation for trustworthy software releases, enabling marketing and design teams to translate the abstract topic of software security into clear, compelling visual concepts.

	SignPath DevSec360		
Capabilities	**Zero Trust Platform Summary**	**DeepSign**	**Pipeline Integrity**
Functional scope	Zero-trust software integrity	Integrated code signing	Policy verification and enforcement
Provisioning of keys / certificates	✓	✓	✓
Signing functionality	Deep support	Executables • packages • installers • containers • scripts • manifests • SBOMs • config files	Attestations
Application security scope	✓	Secure code siging	Dev process and pipeline security
Integrity guarantee	End-to end software integrity	Artifact integrity	Process and configuration integrity
Artifact content integrity	✓	✓	
Malware scan	✓	✓	
Control-plane integration	✓		✓
Artifact origin integrity	✓		✓
Process and configuration	✓		✓
Signing operations	Per release		
File format support level	Deep support	Existing signatures • structure • metadata • signing	
Custom file formats	✓		
Signing of nested files	✓		
Supported platforms	File-formats	Windows • Linux • Kubernetes • Docker • Java • Android • Embedded	SLSA • in-toto
Integration and configuration	Declarative	Configuration of artifact verification and signing	Process policy restrictions
Integration level	CI/CD (declarative)	Explicit CI/CD step	Control-plane CI/CD integration
Single point of CI/CD integration with isolated credentials	✓	✓	✓
Re-signing capability	✓	✓	✓

	SignPath DevSec360		
Capabilities	**Zero Trust Platform Summary**	**DeepSign**	**Pipeline Integrity**
Authorization scope	High-level authorization	Projects • signing policies	Repositories • branches • build configurations
Permissions per project	✓	✓ (based on content)	✓ (based on verified origin)
Authentication of build systems	✓	✓	Attestations
Define & enforce process policies	✓		Dev process and pipeline security
Approval of releases	✓	✓	✓
Automated / Manual approval	✓	✓	
Quorum-based approval	✓	✓	
Verified information	Artifacts • processes	Release artifacts	Process definition and execution
Artifact information	✓	✓	
Source code origin	✓		✓
Build system & configuration	✓		✓
Security scans performed	✓		✓
Auditing and traceability	Integrated • complete	Signing process • artifacts	Source and build process
Structured audit trail	Deep audit information	All input and output files	Policies • approvals • attestations
Release-level auditability	✓	Signing policy • metadata	Full release context • build workflow • build execution • scanning results

Table 3 - DevSec360 HSM options

Four-Stage Maturity Model for Secure Pipelines

Creating a Secure Software Development Lifecycle (SSDLC) is a continuous journey. This four-stage maturity model helps organizations assess their current posture and provides clear guidance to enhance trust, automation, and policy enforcement across the pipeline.

Stage	Description
1 **Ad-hoc:** Inconsistent Signing, No Traceability	▪ SSDLC policies are only provided as paper policies. ▪ Code signing is performed manually, often by individual developers using unmanaged keys. ▪ There are no enforced processes, no central oversight, and no audit trail. ▪ The pipeline is highly vulnerable to human error, credential misuse, and undetected tampering.
2 **Structured:** Centralized Keys, Basic CI/CD Integration	▪ Key management becomes centralized and access-controlled. ▪ Keys are secured via HSMs or cloud-based hardware protection. ▪ Signing is integrated into CI/CD pipelines, often through scripts or plugins. ▪ Initial policies and approval steps are introduced, but enforcement remains partial. ▪ Visibility improves, but auditability and consistency are still limited.
3 **Trusted Traceability:** End-to-End Transparency and Control	▪ The pipeline establishes a verifiable chain of custody for every artifact. ▪ Each component can be traced back to its original source, including who made which changes and how artifacts were built and signed. ▪ Automated security and compliance scans are applied. ▪ Signing actions are logged, and artifacts become provably trustworthy. ▪ This transparency builds a solid foundation for regulatory compliance and defense against supply chain attacks.
4 **Zero Trust Integrity:** Fully Automated, Policy-Driven, Hardware-Backed	▪ At the highest level, the pipeline enforces strict, automated security policies for every code change and artifact. ▪ Nothing is signed or deployed without verification. ▪ All processes are fully auditable and continuously monitored. ▪ The result is a resilient, trustworthy software supply chain aligned with modern zero-trust security principles.

Table 4 - Pipeline Security Maturity Stages

Conclusion:
From Technical Detail to Strategic Imperative

Software supply chain security has evolved from a specialized concern to a central responsibility. Modern attacks no longer focus solely on code, but increasingly exploit weaknesses in development pipelines, target trusted signing systems, and manipulate components deep within the supply chain. The consequences reach far beyond technical impact, affecting business continuity, legal exposure, and long-term customer trust.

This development is mirrored by a tightening regulatory landscape. Frameworks such as the European Union's Cyber Resilience Act and NIS2 directive and others, as well as sector-specific compliance frameworks demand that organizations ensure and demonstrate integrity, traceability, and security throughout the entire software lifecycle. These are no longer optional controls. They are rapidly becoming legal obligations.

Organizations that establish secure development and signing processes today are better prepared for the future. Not only do they reduce operational risk, but they also position themselves for regulatory compliance and greater resilience under pressure. DeepSign and Pipeline Integrity are not marginal optimizations. They are essential for implementing security by design where it matters most: at the boundary between development and operations (or distribution).

Delivering trustworthy software on scale requires more than policy. It requires automation, enforcement, continuous validation, and complete visibility across the delivery process. The DevSec360 platform enables organizations and teams to achieve this without slowing down development or adding unnecessary complexity.

Security is no longer a support function. It is an integral part of the product, a differentiator in the market, and a prerequisite for long-term competitiveness. Now is the time to make it part of the process.

About SignPath

SignPath – Zero Trust Software Integrity for Modern Development Teams

SignPath is a European provider of software supply chain security based in Vienna. Its platform, DevSec360, establishes a Zero Trust approach to software development that goes beyond traditional code signing. By fully verifying all security-relevant build information, the signing process becomes a technically enforced release gate—enabling consistent policy enforcement directly within the CI/CD pipeline.

DevSec360 combines centralized, artifact-based signing with automated verification of source code origin, review status, build context, and policy compliance. Only verified, authorized, and fully traceable releases are signed. The integration is CI/CD-native and does not disrupt development processes.

Customers include organizations such as SolarWinds, Bosch, Dräger, Hitachi Energy, or Airbus.

AIRBUS jamf FESTO Dräger

SOLARWINDS HITACHI

SignPath

SignPath.com | Vienna, Austria