digicert®

Top 5 strategies to secure your software supply chain

Focus on automation of secure code signing, threat and malware detection, and SBOM generation

Software supply chain incidents have increased in frequency and severity for several years. A report by TechTarget's Enterprise Strategy Group found that 91% of organizations experienced one in 2023.

Companies large and small are experiencing these attacks. By now, almost everyone has heard of supply chain attacks at SolarWinds, CircleCI, and 3CX. In March 2024, the maintainer of a very popular Linux compression utility was tricked into introducing a backdoor into the code. The problem was detected before most popular Linux distributions released it in production versions, but the implications could have been horrific.

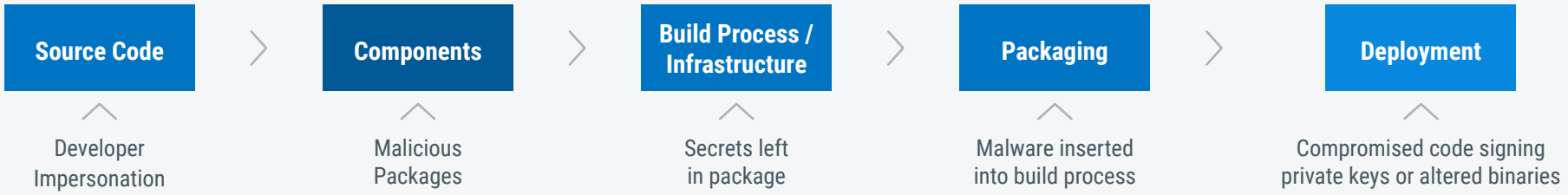
While basic security measures, such as Zero Trust principles, provide companies with a first line of defense, an in-depth defense approach is needed to secure a company's complex software supply chains and software development lifecycles.

Challenges

Malicious actors use a variety of tactics when attacking a software supply chain (see Figure 1). They might target a compromised software developer's password to break into a software build system, steal, or misuse unprotected code signing private keys, insert malware in common open-source software packages, or use a combination of all these tactics. As the Sunburst attack on SolarWinds demonstrated in 2019, attack tactics can be quite sophisticated and complex, but effective defenses need not be as cutting-edge. Adherence to best practices, defense-in-depth principles, and taking a consistent approach to security is within everyone's reach.

Other challenges are organizational and cultural: Software development teams may resist security tools for many reasons: they can be cumbersome, break automated CI/CD build pipelines, or slow down the software release

Figure 1: Supply chain attacks can happen at any stage



Attacks can occur during any stage of the software development lifecycle. Stand-alone existing security tooling might detect specific types of attacks but will miss others. Therefore, organizations should adopt both "shift left" and "shift right" strategies to secure the entire software development lifecycle.

process. False positives do happen, and even warnings can make security products seem annoying and unhelpful.

Security teams have a different set of challenges. They must provide security guidance and enforce compliance with many software teams distributed both organizationally and geographically across the organization. In addition, these teams often use different programming languages and environments, deploy software to different platforms, and use different software to build infrastructure and applications. This heterogeneity can make defining, enforcing, and monitoring software security policy across an enterprise extremely difficult.

Finally, most organizations, especially in some regions like the EU, must comply with a growing body of government regulations and private standards. Several regulations and industry recommendations specify not only best practices but also requirements that companies must follow to ensure the integrity of the software supply chain. For example, the US Executive Order on Improving the Nation's Cybersecurity and the EU's Cyber Resilience Act require that Software Bills of Materials (SBOMs) be generated to show software transparency. NIST's Security Considerations for Code Signing outlines steps that should be followed for secure code signing. The Payment Card Industry Data Security Standard requirement 6 includes several mandates for developing and maintaining secure software and systems.

This solution brief outlines five strategies, or best practices companies can leverage to address these challenges and how DigiCert can help. The key aspect of these best practices is that they occur throughout the software development lifecycle instead of focusing on a single SDLC stage (see Figure 3, page 3).

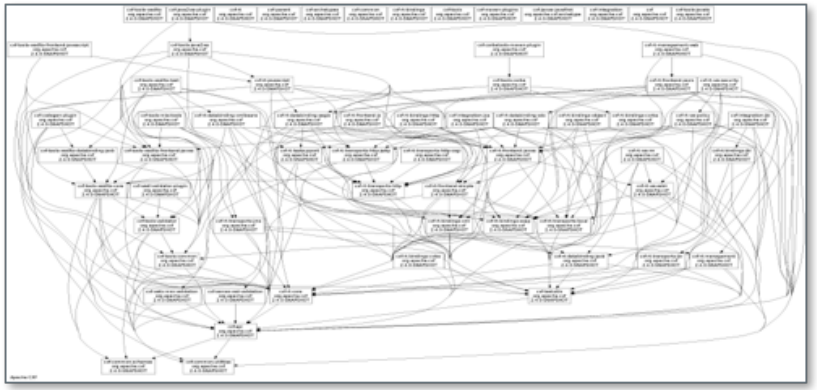
#1: Do not underestimate the complexity of your software development environments

Why are so many software supply chain attacks successful? There are many reasons: IT groups are often understaffed, siloed, and isolated.

A large organization probably has a large and complex attack surface, open to diverse attacks from around the world. But one of the biggest reasons is undoubtedly the complexity of modern software systems.

Figure 2 shows a structural representation of the Apache HTTP software project in 2024. Begun in 1995 by 8 developers, it now has about 2 million lines of code with over 630,000 worldwide contributors.

Figure 2: Apache HTTP Software Project

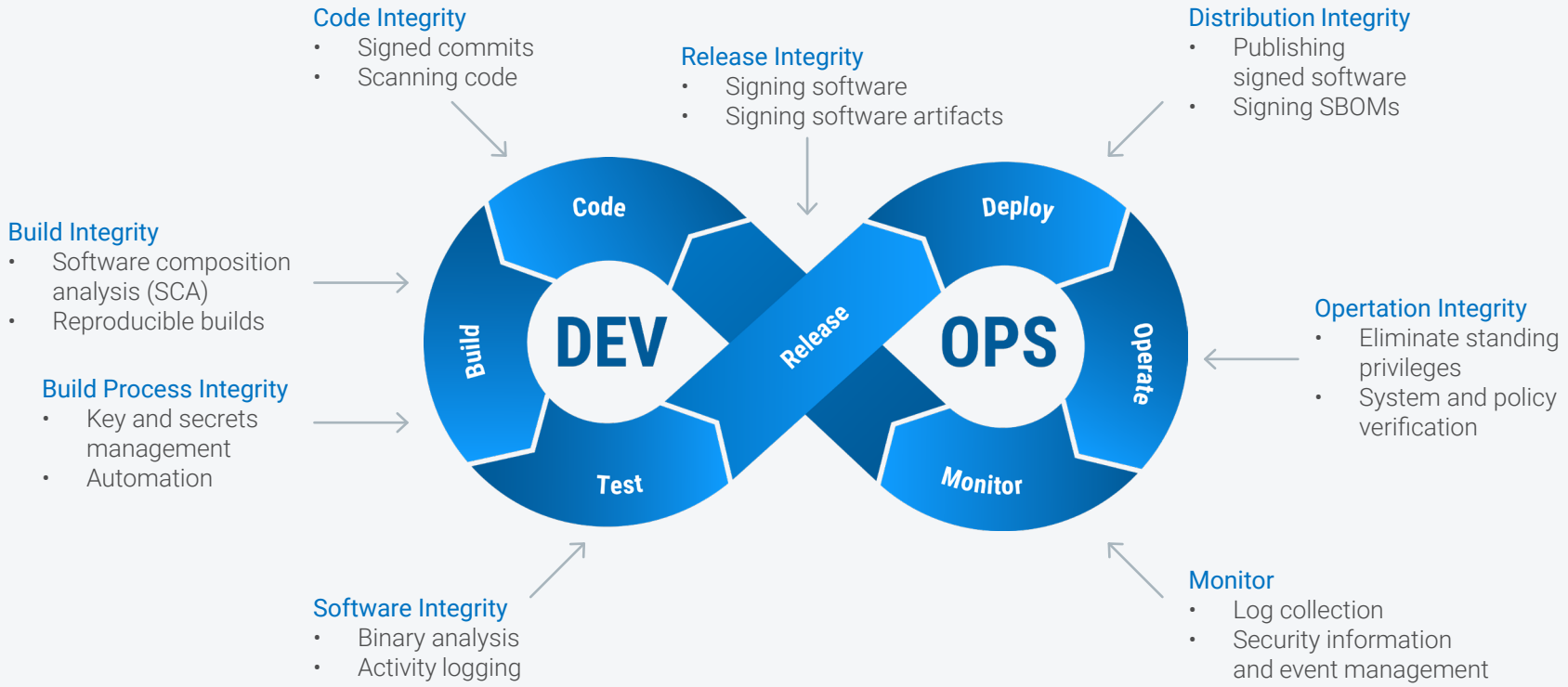


It is a project at the core of many important applications worldwide, and probably nobody can say they are fully conversant with all its aspects.

If you are operating on the Internet, you are almost certainly using Apache HTTP now and then and probably using it all the time. It has responsible maintainers, and its issues are manageable, yet frequent, as there must be with any project this complex.

Even the software you buy to provide security is imperfect and needs to be bolstered by other measures, a practice we call defense-in-depth. In late 2023, authentication service Okta determined that attackers had compromised their systems, stolen the identity information of a large number of Okta users, and were able, as KrebsOnSecurity put it, "…to steal authentication tokens from some Okta customers, which the attackers could then use to make changes to customer accounts, such as adding or modifying authorized users."

Figure 3: Enterprise policy driven approach and visibility



As part of their post-mortem on the incident, Okta announced improved support for several features which are good practice to follow wherever they are available:

- Multifactor authentication on all accounts.
- IP binding – when possible, only allow connections from or to a particular service using specific IP addresses known to be legitimate. This is one way to defeat reuse of hijacked HTTP tokens, another capability the attackers had.
- As a general rule, eliminate standing privileges. These are default, but often unnecessary privileges for an account. Pare down account privileges to those necessary to perform tasks appropriate for the account.

#2: Ensure the authenticity and tamper resistance of your software

Code signing is a basic security measure that has been in use for over 30 years. It uses cryptography to create a digital signature of a software package (executable, library, app, etc.). Users receiving the package can use cryptographic software and the software distributor's public key to show that the software is authentic, i.e., that the purported publisher of the software did, in fact, create it and that the code has not been tampered with by a third party. Many operating systems, like iOS, macOS, and Windows, will check the digital signature of software before installing and running the package.

Software signing is so effective that malicious actors have targeted code

signing systems to steal, compromise, or misuse a company's code signing certificates and private keys. Stolen code signing keys were an essential element of the Stuxnet worm, a nation-state attack that sabotaged Iranian uranium enrichment facilities. Stuxnet was a sophisticated attack using four zero-day Windows vulnerabilities, publicly revealed in 2010.

As with all PKI applications, it is essential that the signer protects private code signing keys. For this reason, the CA/Browser forum requires the storage of private keys associated with publicly trusted code signing certificates in a FIPS 140 Level 2, Common Criteria EAL 4+ device such as a hardware security module (HSM).

While secure key storage is a necessary first step, securing code signing across an enterprise requires additional diligence, if only because a compromised username/password could allow a malicious actor to access the secure storage.

Instead, we recommend the use of an enterprise-strength code signing system that offers many additional security measures, including:

- Automation of code signing certificate and key management, including rotation and revocation.
- An approval process that specifies not only who can use a code signing key but who needs to approve the access and under what circumstances the key can be used (such as time of day, build machine, etc.).
- A role-based system specifying which users can perform signing operations, which are authorized to approve signing operations, and which are allowed to monitor.
- Where approvals are needed, the system should allow for multiple approvals to be required and for m of n approval, meaning approval by some number m out of a set of n users authorized to approve.
- Maintenance of an immutable record of all code signing activities to refer to in case of a compromised certificate.
- A single code signing system that supports a large volume of distributed code signing operations and many different software deployment and development platforms and tools.
- A policy-based approach of control measures, generally implementing best practices we describe in this paper, that must be satisfied before software can be released.
- Sign intermediate software artifacts throughout the SDLC, including source code before it is committed, build scripts and recipes, and container and other software infrastructure files.
- Enterprise-wide visibility, policy definition, and enforcement using a centralized facility, regardless of where various software teams are located.

If you need a written code signing policy, we've got you covered. Start with our downloadable template to easily create a code signing policy tailored for your organization.

#3: Scan for threats, vulnerabilities, and malware

While code signing can prevent tampering after the software is released, what happens if the software is tampered with before it is signed and released? For example, a malicious actor with a breached build system could insert malware into a company's source code repository. Or a software developer could download an open-source package that has hidden malware in it. Or a previously undetected hidden vulnerability could be discovered.

Malicious code embedded in an open-source library has been a problem for a long time and the cause of many software supply chain vulnerabilities. In their report on The State of Software Supply Chain Security, ReversingLabs describes a campaign called "Operation Brainleeches" that began in May 2023. Brainleeches used npm packages to host files used in widespread phishing campaigns targeting Microsoft 365 customers in an effort to obtain their login credentials. There are countless examples of such land mines in public, respected open-source repositories.

Various security tools on the market address some of these issues, such as static application security testing (SAST) tools. However, these tools often

only focus on a single aspect (such as vulnerabilities with open-source software) or a particular stage of the SDLC.

Instead, a comprehensive method is needed to scan final software source code and binaries for threats, vulnerabilities, presence of embedded secrets and malware. This scan should occur regardless of the original source of the components of the software: open-source code, third-party commercial libraries, or proprietary code written by the company. Many companies are innovating in the task of avoiding malware, for instance, by providing curated open-source repositories.

#4: Ensure software transparency (SBOM)

As mentioned earlier, government and industry regulations now more frequently require software publishers to reveal what is in their software, much like a nutrition label indicates the ingredients in food. Known as software bills of materials (SBOMs), these documents catalog all the components in a software package no matter the component's source (e.g., open source, commercial library, etc.). SBOMs can be used not only to catalog the "software ingredients" but to assess the quality of those ingredients, such as versions used, known vulnerabilities, etc.

Even if not required for compliance, organizations should generate SBOMs so they can:

- **Understand changes from one build to the next:** Technology professionals know the frustration of troubleshooting problems between two allegedly identical software releases. A software bill of materials, like its physical manufacturing counterpart, allows organizations to understand which ingredients are necessary to reproduce their software and how those ingredients change over time.
- **Evaluate and prioritize the attack surface of their software:** Look for vulnerable or targeted versions of libraries or components, look at software dependencies, and check for missing mandated security patches.
- **Look for outdated components** like missing mitigations and insecure code signing practices.
- **Take proactive defensive measures** by using software structure to support detection and threat incident response teams and for threat modeling.

Do not just generate SBOMs, tell your software suppliers to provide them and implement software to check them as part of your evaluation. Some SBOMs can be digitally signed as a whole or by component.

When you deploy SBOMs, you actively foil the efforts of cybercriminals trying to use obscurity to slip harmful code into your software. A Software Bill of Materials is one of the most effective methods for protecting your business.

#5: Automate with Enterprise Visibility and Control

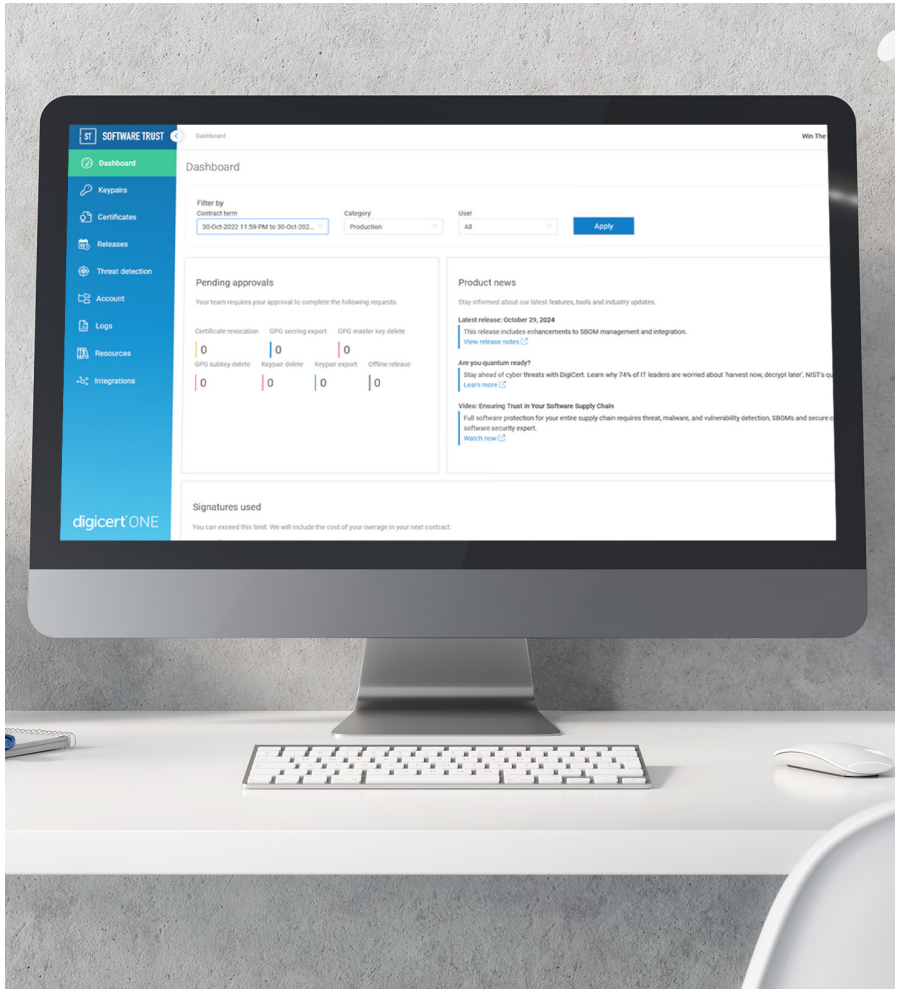
You may have noticed that many security operations need to happen almost constantly. Security experts agree that the only way they will get done is if their operation is automated.

In the SDLC, security operations need to be built into the process, these days generally through a CI/CD pipeline. These systems are designed to include security operations and to design workflow around their results.

The right approach is to have this process driven by enterprise policy and integrated with other enterprise security systems. With centralized visibility and control, organizations can strengthen compliance and reporting while enhancing overall security. Moreover, it's essential to implement these processes in ways that support and streamline developers' workflows, rather than taxing them, fostering a more efficient and secure development environment.

Get agile or fail

Security in a software supply chain is a moving target. There are many opportunities for attack, and you must depend on the security of other parties. Best practices dictate that security be built into the CI/CD process at every stage, using a wide variety of security techniques. The only way to do that is to automate the process. A high level of automation will not just prevent incidents by ensuring the regular application of security measures, but it will help you to respond to incidents, to change process if necessary, and to generate the information needed for compliance reporting.



DigiCert Software Trust Manager

DigiCert® Software Trust Manager is a digital trust solution that protects the integrity of software across the software supply chain, reducing risk of code compromise, enforcing corporate and regulatory policy, and delivering fine-grained key usage and access controls for code signing.

DigiCert Software Trust Manager minimizes the risks of security breach and malware propagation in the development, build and release of software, enabling companies to protect against software supply chain attacks and comply with government regulations.

DigiCert Software Trust Manager helps large enterprises easily implement these five best practices.

Discover how to protect your software with Software Trust Manager