ENDOR LABS

WHITEPAPER

Implementing Software Supply Chain Security

What to Consider When Designing a Program

By Chris Hughes, Darren Meyer, and Jenn Gile

Introduction

Software supply chains are top of mind for security professionals, but there is little industry agreement on what it includes or how to secure it. Meanwhile, vendors and talking heads are quick to provide definitions (and tools) that may seem self-serving. Practitioners and CISOs alike are left with fundamental questions, such as:

- Why do I need to worry about it?
- How is it defined?
- How do I analyze my current state?
- How do I protect it?
- What tools do I need?

In this guide, we set out to establish a shared understanding so you can make informed decisions on your security strategy.

READ TIME: 5 MINUTES

The Big Picture

Why organizations should prioritize software supply chain security.

READ TIME: 5 MINUTES

Defining the Software Supply Chain

Components that contribute to development and delivery of your software.

READ TIME: 20 MINUTES

Securing the Software Supply Chain

Recommendations, resources, and methodologies.

About the Authors

Chris Hughes

Chief Security Advisor

Chris Hughes is the Chief Security Advisor at Endor Labs and President & Co-Founder at Aquia. Chris also serves as a Cyber Innovation Fellow at CISA and is the author of *Software Transparency and Effective Vulnerability Management* from Wiley. Chris hosts the Resilient Cyber podcast and the Substack by the same name and is passionate about all things software supply chain security and helping organizations achieve a secure digital transformation.

Darren Meyer

Staff Research Engineer

Darren Meyer is an Application Security practitioner and leader with 25 years of experience, 19 of which have been focused on AppSec products and programs. Currently the Staff Research Engineer for Endor Labs, he's passionate about practical and affordable security programs, socio-technical systems, and getting great results from his home espresso machine.

Jenn Gile

Director, Product Marketing

Jenn Gile is a seasoned Application Security and DevOps marketer and community builder. She writes prolifically on tech topics including open source, Zero Trust, Layer 7 traffic management, compliance, and WebAssembly. She is also the author of NGINX's ebook *Taking Kubernetes from Test to Production*. With a background in learning and development, Jenn prioritizes education-based marketing programs that provide intrinsic value to the community.

1 The Big Picture

Malicious actors have discovered a more effective way.

Software supply chain security continues to be one of the most widely discussed topics in the cybersecurity industry, and rightfully so. Reports document a continuing rapid growth in supply chain attacks, with twice as many observed attacks in 2023 as there were in all previous years combined, as well as seeing a three-fold increase in malicious packages over the same period. These threats impact thousands of organizations and millions of users worldwide. These include attacks against both proprietary software vendors and their products, as well as widely used open source software (OSS) projects and components.

Rather than investing effort into compromising a single target, malicious actors can compromise a widely used product or open source component, causing a massive downstream impact across the entire ecosystem. They have also realized it is much easier to attack the brittle supply chain components than it is to target mature enterprise environments that have (and continue to make) investments in efforts such as Zero Trust and moving away from perimeter-based security models.

Rather than traditional "push" style attacks that force their way into an enterprise target, software supply chain attacks often rely on a "pull" style model of attack, where victims pull in malicious components potentially compromising themselves.

The software supply chain has become the "soft underbelly" of our digital ecosystem, making it an appealing attack vector to malicious actors around the world.

Four Trends in Software Development

Software development is now software assembly.

Before we get into the components of the software supply chain, it's important to understand how software development has fundamentally changed in the early 21st century as organizations adopt modern application development techniques. To meet the demands of customers, organizations have shifted away from traditional waterfall software development to more agile and iterative release cycles which may be shorter with new feature velocity accelerating. But without changing the way software is architected, teams have no chance at meeting their SLAs.

TREND 1 Containerization

Over 50% of applications are containerized.

Increasingly, modern applications are run in containers, which are defined as "standard units of software that packages up code and all of its dependencies". This allows for applications to be lightweight, portable and standalone, enabling scalability, replication, and re-use.

TREND 2 Open Source Software

OSS represents 70-90% of modern code bases.

OSS is a tremendous asset because it reduces the amount of code your developers write by leveraging what already exists. It's estimated that developing this code from scratch would cost firms $8.8 trillion. When applications can be created faster by leveraging existing components, organizations experience expedited development timelines and cost savings. That said, as we will discuss below, OSS isn't without its own risks and considerations.

TREND 3 Continuous Integration & Delivery

84% of developers are involved in CI/CD activities.

Increasing the development and shipment velocity while meeting functional and non-functional requirements required the automation of various development activities along the SDL, e.g. compilation, unit and integration tests or packaging. Those activities are increasingly performed by CI/CD pipelines that can go as far as deploying releases in production environments.

TREND 4 Artificial Intelligence

70% of packages using OpenAI's APIs were brand new.

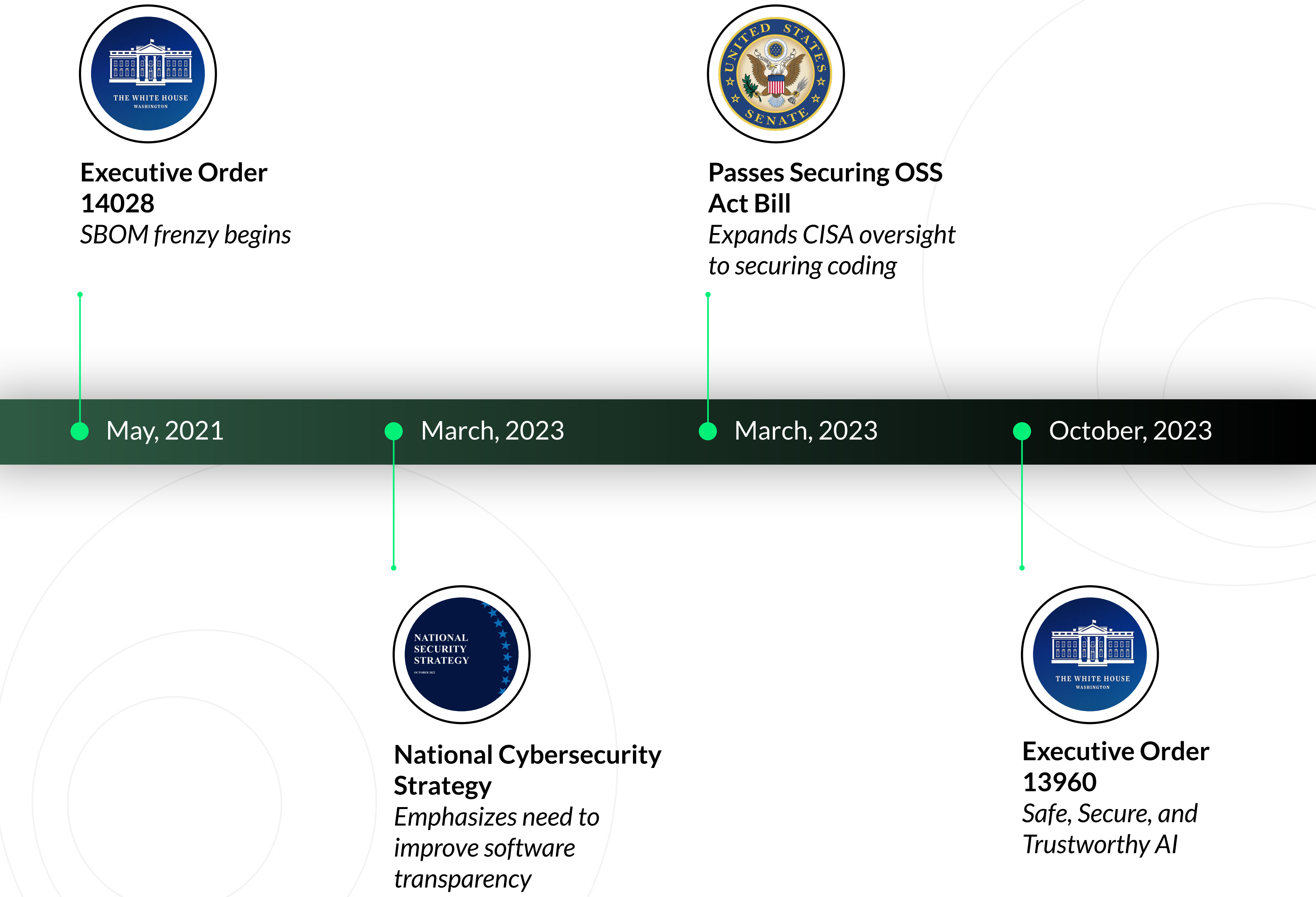
AI models are another tool to accelerate development and beat competition. Just like your applications, models can contain OSS code and components. AI/ML resources and software may share principles much like the OSS world, in terms of sharing of training data, models, weights and a wide variety of tools.

Citations: Dell Container Adoption Trends, Synopsis 2024 Open Source Security and Risk Analysis Report, CD Foundation State of Continuous Delivery Report 2023, Endor Labs State of Dependency Management 2023.

Compliance Standards

A burden or a driver?

Software supply chain attacks have spurred widespread concern both from industry and government. This concern is highlighted in U.S. Executive Order (EO) 14028 on *Improving the Nation's Cybersecurity*, which led to a slew of efforts from Federal agencies (such as NIST and CISA) and industry organizations (such as OWASP and The Linux Foundation), as well as by commercial organizations including Google and Microsoft. And from PCI DSS to DORA (Digital Operational Resilience Act) to FedRAMP, there are numerous global standards seeking to protect consumers and national security from malicious actors. While some compliance activities may seem like checkboxes that add more overhead without adding value, the fact remains that compliance is a useful tool to drive change that otherwise might not get prioritized.



May, 2021

Executive Order 14028
SBOM frenzy begins

March, 2023

National Cybersecurity Strategy
Emphasizes need to improve software transparency

March, 2023

Passes Securing OSS Act Bill
Expands CISA oversight to securing coding

October, 2023

Executive Order 13960
Safe, Secure, and Trustworthy AI

2 Defining the "Software Supply Chain"

Ask five people, get five different answers

As with many industry terms, there isn't total agreement on what is included in a software supply chain, and therefore what it means to secure it. Definitions range from "everything" to "just the tools I bring in from outside my organization." At Endor Labs, we've adopted the broader definition popularized by Google, because it integrates better into the software development lifecycle and is more reflective of the reality that many organizations produce products that end up in someone else's supply chain (a "consumer point of view"). The benefit of taking a broad view is it encourages security professionals to address product security holistically rather than as a siloed, binary "did I build it vs buy it" strategy. That definition is:

> A software supply chain consists of all the code, people, systems, and processes that contribute to development and delivery of your software, both inside and outside of your organization. It includes:
>
> - Code you create, its dependencies, and the internal and external software you use to develop, build, package, install, and run your software.
> - Processes and policies for system access, testing, review, monitoring and feedback, communication, and approval.
> - Systems you trust to develop, build, store, and run your software and its dependencies.
> - Software you consume from external suppliers and service providers.

As defined here, your software supply chain contains at least six discrete components: Your developers, code, dependencies, AI models, containers, and CI/CD pipelines.

Your Developers

Your developers represent an absolutely critical part of your software supply chain. They're the individuals responsible for writing first-party code and pulling in third-party dependencies, and they're a crucial attack vector (with regards to their development accounts, endpoints, and more).

Developers are also your first line of defense because they're in the position to ensure that trusted and secure components are integrated into your organization's applications. Additionally, technology stacks and industry-leading security tools are looking to integrate with developer workflows and empower developers to make more secure choices around software creation, consumption, and use.

Your Code

Although most modern codebases are primarily made up of open source software (OSS), organizations still write their own code ("first-party" code). Your first-party code isn't necessarily part of your broader software supply chain, but it still has implications for the security of your systems and products. It also has downstream ramifications for your customers and consumers, which can be impacted by vulnerabilities in your first-party code.

Your Dependencies

If you're following current software development practices, your first-party code relies on additional OSS components to function properly. These components are referred to as "direct dependencies." While the number of dependencies varies (based on programming language, complexity of the software/application, etc.), the average application has several hundreds of direct dependencies. Additionally, each of those direct dependencies have OSS dependencies of their own, referred to as "transitive dependencies."

Like your first party code, vulnerabilities in OSS packages and libraries can present risks to your organization, your customers and consumers. Because each OSS package you directly depend on tends to have multiple dependencies of its own, these transitive dependencies can be significant sources of risk. In fact, research by Endor Labs found that 95% of vulnerabilities that affect customer applications exist within their transitive, rather than direct, dependencies.

Organizations leveraging OSS dependencies must understand the context around vulnerabilities and vulnerable dependencies. Moving beyond traditional approaches such as CVSS scoring, this includes known exploitations (KEV), likelihood of exploit (EPSS), and reachability.

Your AI Models

AI models are increasingly a core consideration in the modern software supply chain. While your AI models are likely a combination of first-party, third-party, and OSS code (found in outlets like HuggingFace), we think it's worth pulling them out separately as they have unique attributes and concerns.

Organizations need to have safeguards in place to protect their consumption and use of AI, whether delivered as a SaaS or self-hosted, as well as ensure any OSS AI resources they're using are from trusted sources and have security measures in place, much like broader OSS usage. Industry-leading sources, such as OWASP's AI Exchange, list supply chain threats as a part of the AI threat landscape. They point out that, much like traditional source code and software components, data and models may involve multiple suppliers and require supply threat risk management to mitigate threats. Security recommendations include standard supply chain risk management practices including provenance, pedigree, signature verification, trusted package repositories, and dependency verification tools.

Your Containers

Like the open source in your applications, OSS is also used for containerization. The challenge, as discussed in the section regarding dependencies, is your containers may also have vulnerabilities. As a result, you need to introduce the practice of container hardening — reducing a container's attack surface and making it less vulnerable to exploits. Otherwise, developers end up building on top of vulnerable containers full of unnecessary dependencies, typically referred to as "bloated containers". Those vulnerabilities in container images now place any applications running on top of them at risk as malicious actors can exploit vulnerabilities in the underlying containers. They also attempt to compromise widely used publicly available container images from popular repositories such as Docker Hub in efforts to impact downstream consumers and their associated applications.

Your CI/CD Pipelines

There may be a tendency to focus on the application layer when discussing software supply chain security. The reality is that underlying infrastructure and systems, which may also be defined as Infrastructure-as-Code (IaC), also represent part of your software supply chain and must be secured to mitigate risk to the products and software you produce. A lack of governance around repositories and their hygiene as well as the pipelines and associated infrastructure all can allow for nefarious exploitation. If malicious actors can compromise pipelines and the infrastructure used to develop application layer code, they can still have their desired impact to facilitate software supply chain attacks, and these techniques have been used in various notable software supply chain incidents.

3 Securing the Software Supply Chain

One vendor to rule them all? Not quite.

It’s tempting to look for a single vendor or product that “solves'' software supply chain security concerns. But like other broad security practices, such as Zero Trust, the reality is that SSCS is complex. It's impractical to expect one vendor to cover every area of the software supply chain attack surface. Due to the wide attack surface of the modern software supply chain (your developers, code, dependencies, CI/CD pipelines, and more), as well as a wide array of programming languages and frameworks, no single vendor comprehensively covers every aspect of the software supply chain.

In this section we discuss some of the specific components of modern SSCS as well as key recommendations, resources, and methodologies to secure these components or implement the associated activities.

Open Source Governance

Code Security

AI Security

Software Bill of Materials

Secret Management

CI/CD Pipelines

Attestation & Code Signing

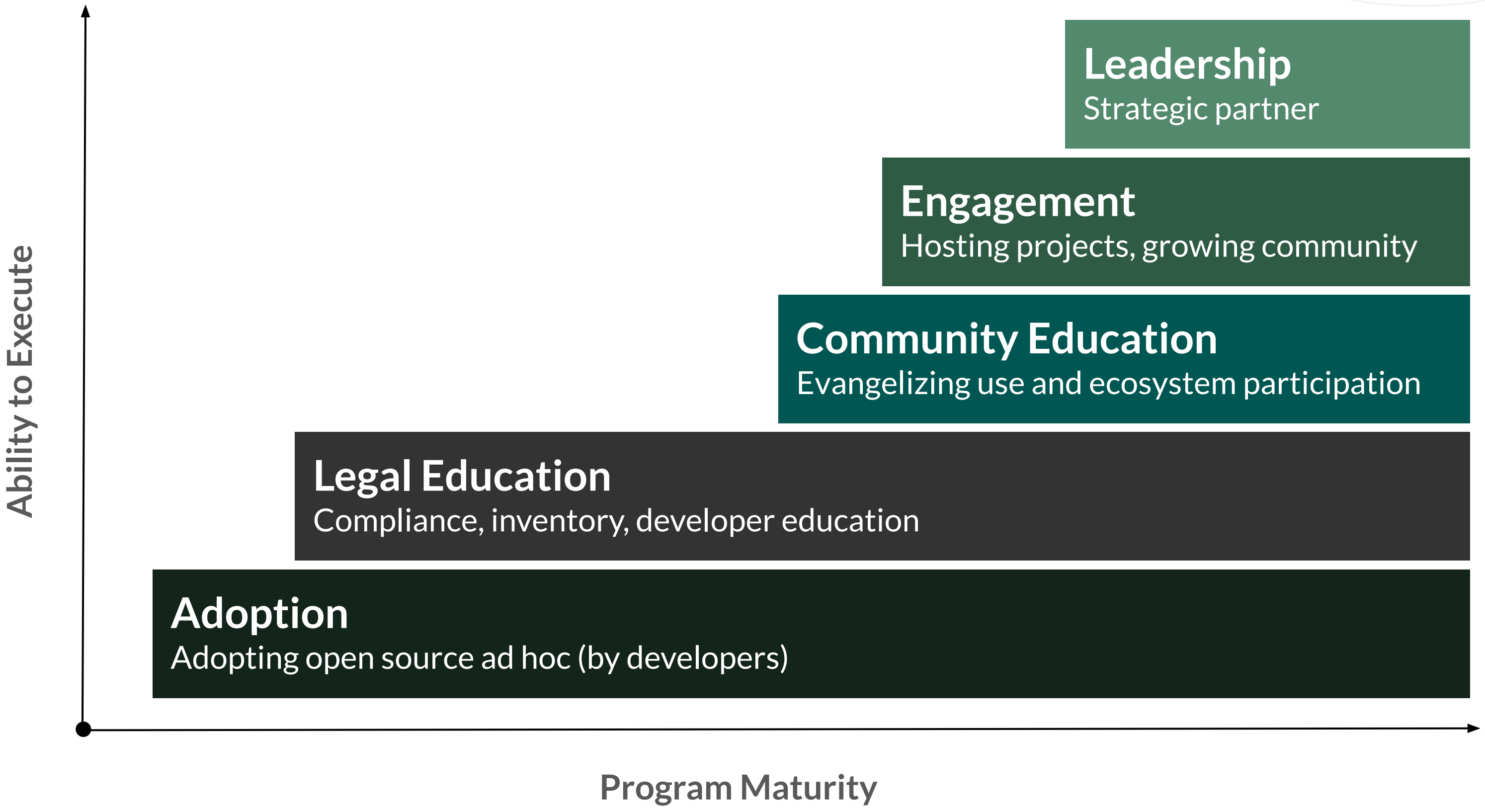
Container Security

Open Source Governance

Open source software (OSS) has risks, including longstanding common problems around vulnerable components and other considerations like quality, maintainer support, and so on. Endor Labs pioneered the OWASP Top 10 Risks for Open Source Software, which includes lagging indicators of risk such as known vulnerabilities, along with leading indicators of risk such as compromises of legitimate packages, name confusion attacks, unmaintained software and untracked dependencies, and other considerations.

There are various elements to OSS governance, ranging from selection controls to vulnerability management. The creation of an Open Source Program Office (OSPO) is an established way for an organization to govern and control the consumption (and production) of OSS across the entire organization in a uniform or standardized way. Addressing OSS governance may also help you with long term compliance goals related to risk: After all, if you can prevent unhealthy projects from entering your ecosystem, you're likely to have fewer vulnerabilities to remediate in the future.

OSPO Maturity Model


Leadership
Strategic partner
Engagement
Hosting projects, growing community
Community Education
Evangelizing use and ecosystem participation
Legal Education
Compliance, inventory, developer education
Adoption
Adopting open source ad hoc (by developers)
Ability to Execute
Program Maturity

OSPO Model courtesy of The Open Source Initiative.

The Endor Labs Solution for Open Source Governance

Endor Labs helps organizations govern use of OSS components without slowing down developers by providing monitoring, package scoring, governance, ai-assisted OSS selection, and customization.

Monitoring: Continuously monitor OSS components in your software projects for vulnerabilities and other risks, with powerful focus tools and policies that help developers identify and fix the issues that matter most to your organization.

Package Scoring: Our AI-enhanced, enriched OSS component and vulnerability database includes over 170 types of risk signals for leading indicators of risk from a package or the project it supports. This information is consolidated into an Endor Score for each package, which provides an easy-to-understand metric of how well a package does based on factors of security, activity, popularity, and code quality.With this data, you can build admission policies that prevent risky packages from entering your projects, identify suitable alternatives, and identify emerging risks before they become critical.

Governance: Integrate governance components with GRC tools, risk aggregators (including SIEMs), and developer workflow tools to streamline research and remediation. Use our risk database and Endor Score to build admission policies that prevent risky packages from entering your projects, identify suitable alternatives, and identify emerging risks before they become critical.

AI-Assisted OSS Selection: With DroidGPT system, is a GPT-enhanced recommendation and research engine, that helps developers can use natural language to identify OSS components that meet their needs while minimizing risks introduced. It's backed by our enriched OSS component database that monitors over 170 risk indicators for each component (summarized to 4 individual and one overall Endor Score), along with known specific risks like vulnerabilities, malicious behaviors, and evidence of supply chain attacks like typosquatting. The API for this capability makes it simple to leveraging the data for admission control decisions -- such as "disallow any component that has a Security Score of less than 6, or an Activity score less than 4", or even against individual signals like "warn about any component that does not have significant high-reputation contributor". During the CI/CD pipeline runs, this data is combined with our SCA analysis (which includes function-level reachability data; see Code Security below) and policy to allow for notification or deployment-blocking behaviors should there be packages in use that don't meet policy, providing a detective control even if developers bypass the preventive control.

Customization: Our API-driven design philosophy and our integrations allow for a mix of out-of-the-box and easily developed custom routing of issues to places developers work (like their IDE, within pull requests, issue tracking systems, etc.) and risk is managed (like GRC tools, SIEM tools, and compliance/risk aggregators).

Code Security

A core part of SSCS is code security, which analyzes code projects for vulnerabilities and other weaknesses. Code security tools generally fit into one of two categories:

- **SAST** (Static Application Security Testing) tools examine source or binary code to discover coding errors that represent actual or potential security risks.
- **SCA** (Software Composition Analysis) tools examine a software project's dependencies for known vulnerabilities and related risks.

This tool divide is essential because the remediation process for each use case is different. Novel flaws detected by SAST tools typically require changes to the design or implementation; that is, a developer must rethink how a problem is being solved and make changes to the code. Risks discovered in OSS or other dependencies usually require a version upgrade to the affected component. This latter may sometimes also require code changes to accommodate (or to mitigate flaws that cannot be repaired through an update), because sometimes updates change how the consuming code must be written. One key difference in outcomes is that while it may be very reasonable to break a build where first-party code is vulnerable, with OSS code your teams will want more flexibility because a fix might not be available from the OSS producer, or because the required upgrade may cause unrelated problems over which your teams have very little control.

Use of SAST tools enables developers to scan code earlier in the software development lifecycle (SDLC) prior to runtime production environments, meaning they can identify vulnerabilities before the affected code is released to production environments where it can be attacked. SAST can identify not just existing vulnerabilities, but also poor coding practices that later manifest as vulnerabilities.

SAST tools typically enable direct feedback to developers with details on the vulnerabilities identified as well as guidance on how to resolve the vulnerabilities. The challenge here is that SAST tools can also be noisy, often producing false positives and findings that can sink developers' time investigating and discussing with security peers, despite not being actual vulnerabilities or posing real risks. We hear from many customers that a noisy SAST tool actually directly leads to developers not remediating risks. This makes it key for organizations to adopt high quality SAST tooling that can produce high-fidelity findings, minimize false positives, and enrich findings to help developers focus on real risks.

The Endor Labs Solution for Code Security

Endor Labs helps organizations address Code Security through integration and management, and reachability-based SCA.

Integration & Management: Simplifying tool management by providing out-of-the box integrations with existing security tools; this is a growing list that includes SAST tools like GitHub Advanced Security's CodeQL, risk aggregators like Vanta, and so on. And we help our customers build and integrate reusable security pipelines that make swapping out specific security tools a much simpler process.

Reachability-Based SCA: Providing a next-generation SCA and vulnerability management platform that identifies security and operational risks in OSS components with strong focus and execution tools.

Endor Labs takes a "get safe, stay safe" approach with our next-generation SCA. We identify the OSS components your applications rely on, as well as the components those components rely on (transitive or indirect dependencies), and perform static call path analysis on your applications and the whole of your dependency tree to determine which risks genuinely affect you through context data and Function-Level Reachability Analysis.

This prioritization system helps security organizations identify relevant and actionable risks and prioritize them for remediation, allowing an organization to "get safe" by reducing the existing risk footprint. Additionally, the comprehensive, high-accuracy vulnerability data combined with our reachability and context analyses and our enriched OSS package and risk database allow precision policies for identifying the most important and actionable risks in your dependencies. This looks like being able to set policies to help development teams "stay safe", such as:

- "Leave a PR comment if a pull request results in a new risk in a dependency, where a patch is available, the vulnerable function is reachable from first-party code, and the chance of exploit in the next 30 days is > 1% based on EPSS data"
- "Open a Jira ticket if a direct dependency in a project is no longer maintained"
- "Send a Slack message if the released version of an application has a new critical or high severity vulnerability affecting it"

Artificial Intelligence (AI) Security

Security practices related to AI are still in nascent stages, but with both industry and government focused on the topic, we can expect this field to grow quickly. Like a human, an AI model can be tricked into divulging sensitive information (such as API keys), and like any other OSS project, it can be the victim of bugs and malicious code. In recognition of these risks, in July 2023 OWASP released the first Top 10 for LLM list. The risks range from the malicious (prompt injections, training data poisoning, DDoS, supply chain) to accidental (overpermissioning, data leakage).

October 2023, the United States released an Executive Order on the Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence. This E.O. has many requirements that echo what we saw with the Cybersecurity E.O. Most relevant to an organization's SSCS strategy include the requirements regarding:

- Development of standards, tools, and tests by NIST to help ensure that AI systems are safe, secure, and trustworthy.
- Establishment of an advanced cybersecurity program (at the government level) to develop AI tools to find and fix vulnerabilities in critical software
- The safe, ethical, and effective use of AI by the United States military and intelligence community.

As a result of the E.O., we can expect frameworks to be created and help with secure usage, and of course vendors will start to fill gaps. In the meanwhile, most AI security is somewhat do-it-yourself.

Most organizations started by setting policies regarding the use of AI, which typically addresses the types of data (e.g. proprietary vs. generic) that can be fed into different types of models (e.g. public vs. private instances). This is a good first step that covers some risks, but there will still be gaps that will require processes and tools. When designing a SSCS program to address AI, it's helpful to think about the ways your organization is using - or plans to use - AI. For example, will they use AI models to write application code? Are they developing proprietary AI models?

When developers use AI to write code (and we bet yours already are), there are interrelated concerns:

- *Is the developer using an approved model and does it adhere to good security practices?*
  This may be the easiest to address if your organization has already vetted a model and has a contract with the vendor (e.g. OpenAI). But the large vendors can be expensive, so your organization may resort to open source LLMs that can bring in more risk.
- *Is the AI-generated code bringing in any risks to your supply chain?*
  Code generated by a model should be subjected to SAST (including SCA) just like your other first-party and OSS code.

Continued on following page.

Artificial Intelligence (AI) Security

When your organization is designing AI-based tools, including your own models, then you need a program in place to evaluate whether the components meet your security requirements. Building a foundational model is very hard, but building on top of a foundational model is much easier. As with application code, AI models often have numerous dependencies, many of which are open source. You can reduce risk by:

- Reviewing how the model was constructed
- Reviewing the organization that created the model
- Taking advantage of HuggingFace's security features
- Being careful about downloading dependencies from models in HuggingFace

The Endor Labs Solution for AI Security

Python is the language of choice for creating AI models, but because of this popularity it's likely to be targeted by malicious parties. Unfortunately, detecting vulnerable dependencies is difficult because many Python dependencies aren't in the manifest so they aren't caught by traditional SCA tools. Endor Labs' SCA tool scans Python source code to find hidden dependencies and uses reachability analysis to prioritize just the risks that can actually impact your application. This involves a four-step process that results in better visibility of risk:

1. **Source Code as a Ground Truth:** As the primary "source of truth", Endor Labs uses the source code offers the clearest insight into which dependencies are called upon and used.
2. **Correlate with Package Manager Data:** After establishing dependencies, the data is cross-referenced with package manager information, identifying phantom and unused dependencies.
3. **Correlate with the File System:** By comparing dependencies declared by package management manifests with those used in the code and those available in the file system, you get a complete picture of the dependencies used.
4. **Highlight Discrepancies:** Any variations between the actual code and package manager definitions are clearly marked, alerting developers to potential issues like missed vulnerabilities or unnecessary packages.

Software Bill of Materials

As the industry looks to grapple with software supply chain threats, the SBOM has become a key component of enabling software transparency, a longstanding cybersecurity best practice cited in sources such as CIS's Critical Security Controls list. An SBOM is a nested inventory of components that make up a piece of software. This enables both suppliers and consumers to have an understanding of what components make up a specific product, service, or application. An accurate SBOM enables discovery of known vulnerabilities within software components that make up an application system.

The ability to exchange SBOM data reliably has given rise to standards for SBOM representations. The industry currently has two predominant SBOM representation format standards: The Linux Foundation's Software Package Data Exchange (SPDX) and OWASP's CycloneDX. There's been significant advances in both OSS and proprietary tooling to create, enrich, analyze, store, and report on SBOMs, helping both suppliers and consumers understand their software asset inventory.

Some highly regulated industries, such as the United States government (including the Department of Defense and the Food & Drug Administration) have begun to call for the use of SBOM's between software suppliers and consumers as well as for critical activities and environments.

A comprehensive SBOM solution must, at minimum, consider both SBOM Producers (generate SBOMs for your components) and SBOM Consumers (ingest SBOMs from applications you consume), which of course can both occur at the company or even within the same team.

Your customers and external auditors who require SBOMs will need assurance that you have accurate and complete SBOMs tied to each application, service, and component they consume from you; and that you have appropriate processes to maintain the accuracy of these as new versions are released. In many cases, in order to provide an accurate SBOM to this audience, you will also need to consume SBOMs generated by 3rd-party providers you use to deliver your applications and services.

Consuming SBOMs also enhances your own 3rd-party risk program, as it enables close monitoring of components in your environment, and rapid identification of affected 3rd-party applications and services when new vulnerabilities are announced in components they consume. For example, if you have consumed accurate SBOMs for your services, and a vulnerability like log4shell is announced, you can rapidly determine which 3rd-party services and applications use the affected version, and take steps to mitigate.

Continued on following page.

Software Bill of Materials

The National Telecommunications and Information Administration (NTIA), part of the U.S. government, issued a document describing the minimum requirements for a SBOM. They are:

- **Supplier Name:** The name of an entity that creates, defines, and identifies components.
- **Component Name:** Designation assigned to a unit of software defined by the original supplier.
- **Version of the Component:** Identifier used by the supplier to specify a change in software from a previously identified version.
- **Other Unique Identifiers:** Other identifiers that are used to identify a component, or serve as a look-up key for relevant databases.
- **Dependency Relationship:** Characterizing the relationship that an upstream component X is included in software Y.
- **Author of SBOM Data:** The name of the entity that creates the SBOM data for this component.
- **Timestamp:** Record of the date and time of the SBOM data assembly.

Unfortunately, Chainguard found that only 1% of SBOMs were entirely conformant with these requirements. This emphasizes the importance of selecting a comprehensive SBOM tool.

The Endor Labs Solution for SBOM

Endor Labs supports SBOM programs for both producers and consumers.

SBOM and VEX Generation: Our SCA tool creates an accurate inventory of OSS components (aligned with NTIA guidance) necessary to generate accurate SBOMs for your first-party applications and components. SBOMs for your packages can be exported in standard formats via the UI, or in an automated fashion via an API call. This allows your release pipelines to generate a valid SBOM as an artifact in an automated and repeatable fashion. Additionally, Endor Labs supports the VEX (Vulnerability Exploitability eXchange) format, an OASIS standard format that follows CycloneDX conventions to produce a sidecar file that describes the vulnerabilities in the components within your SBOM, along with analysis data detailing their impact. For example, we can produce a VEX document for a component that lists the disclosable vulnerabilities in OSS components, and annotates those that are not exploitable because the vulnerable function in that dependency is not reachable.

SBOM Ingestion: Our SBOM Hub product addresses the Consumer role by allowing import of common SBOM format documents from suppliers via simple API calls or use of our command-line tool. Ingesting an SBOM connects the components identified with our enriched OSS risk database, allowing you to quickly identify vulnerabilities and other OSS component risks in your 3rd-party software.

Secret Management

In cloud-native environments, the use of "secrets" has become commonplace for DevOps and operational activities. A classic example is the username and password that allows an application access to its database, but a secret can also be API keys, credentials, certificates, and private keys, among other types of information. When secrets are unintentionally leaked, they can allow malicious actors initial access, providing the ability to move laterally across enterprise environments and gain administrative or elevated access to systems and infrastructure. Secrets can be leaked when a developer "hardcodes" the secret into the application or stored in repositories, even if they're not in application code directly, but in supporting artifacts such as configuration files. Additional factors leading to the sprawl of secrets include the growing popularity of source code, container images, and infrastructure-as-code (IaC) manifests where secrets may be exposed.

Poor (or non-existent) secrets management has been associated with several incidents. For example, Samsung's 2022 source code leak included over 6,000 secret keys. Leaked secrets account for an average of $1.2M revenue loss, yet most organizations report that they do not have a mature secret management program.

To mitigate these risks, organizations need to have a robust secrets management policy and process in place. This program should include secrets scanning tools, which can identify secrets which may inadvertently be exposed and abused.

The Endor Labs Solution for Secret Detection

Endor Labs Secrets helps you avoid secret leaks in your git-based repositories, in any text file, at three control points:

- **Deep scan**: Scan your entire repository commit history for secrets in order to identify latent secrets-related risks.
- **Branch scan**: Scan a given branch, either from within the pipeline (enabling you to block a PR if a valid secret is detected) or in a supervisory fashion without needing to alter your pipeline.
- **Pre-commit hook:** Developers can include a rapid, "changes only" secrets check in a pre-commit hook in their development environment to stop new secrets from entering the git history in the first place .

Our solution addresses the noise problem present with many secrets scanners by testing secrets whenever possible, allowing you to set policies to handle secrets known to be valid credentials -- and thus high risk -- while ignoring those that are known to be invalid (while still being able to track and analyze them as resources permit).

CI/CD Pipeline Security

It isn't just the security of the code going through a CI/CD pipeline that is of concern, but also the pipeline itself. If malicious actors can compromise the pipeline and platforms facilitating the delivery of code to production environments, they can compromise the code and downstream consumption. There are also ample examples of notable software supply chain incidents that involved abusing flaws in the CI/CD environment, such as SolarWinds, CodeCov and the PHP breach. So while it is critical to conduct activities such as SAST and secrets scanning in pipelines, the underlying pipeline, processes, and infrastructure supporting the flow of code must also be addressed.

This risk is identified in industry resources such as OWASP's Top 10 CI/CD Security Risks. This guidance specifically calls out insufficient flow control, inadequate identity and access management, and insecure system configurations. It also points out that CI/CD environments and their associated processes and systems are fundamental to modern code delivery.

The Endor Labs Solution for CI/CD Pipeline Security

Endor Labs CI/CD helps address three of the main considerations in hardening CI/CD pipelines:

- **Tool Discovery:**The tools, pre-build actions, templates, etc. your pipeline consumes are a type of dependency. Our scanner understands common pipeline configuration files and identifies the components in use, identifying risks they or their dependencies pose to your pipelines, as well as highlighting excessive privileges given to pipeline jobs. As with all our findings, you can then apply policy to prohibit or require certain tools, route important issues to DevOps teams for repair, etc.
- **Repository Security Posture Management (RSPM)**: Your CI/CD environment relies on effective controls around the code commit and management process. Endor Labs can scan your repository configuration to determine if it complies with your organizational policies, making sure that your controls function appropriately across your enterprise. This isn't one-size-fits-all: our flexible, open policy system allows you to have different requirements for different sets of projects across your repositories.
- **Artifact Signature Verification**: Our artifact signing solution (see more on Page 22) can be used as part of CI/CD security posture controls by ensuring that components and tools pulled into your pipeline have been signed and verified before being executed. This allows mature security/DevSecOps teams to implement adoption controls, sign consumable dependencies to attest that they've been appropriately tested and approved, and ensure that components that don't flow through that path are not used during the build pipeline.

This approach ultimately gives confidence that you have control over what enters the repository (by ensuring a strong configuration using RSPM), confidence in what's running in your pipelines (using tool discovery), and confidence that critical components that enter your pipeline have been properly vetted (using artifact signing).

Attestation and Code Signing

As malicious actors increasingly look to carry out software supply chain attacks, including tampering and injection of malicious code, organizations have begun to push for not just transparency of software components, through artifacts such as SBOMs, but also assurances around integrity. This includes integrity of the software development and build processes as well as final outputs and software components.

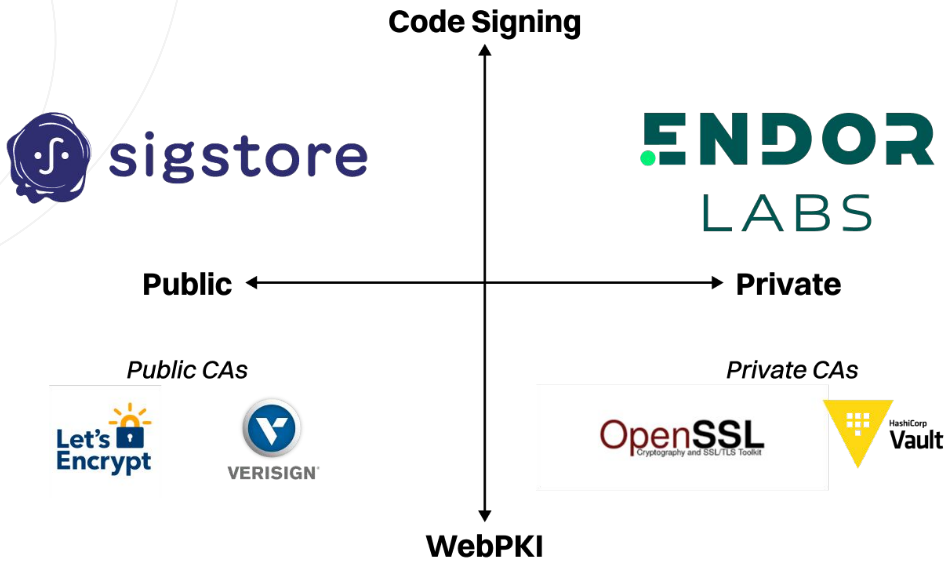
In fact, some organizations and industries, (especially highly regulated industries) are increasingly calling for the use of artifacts such as SBOM’s and attestations to provide assurances around the integrity of the software development process and its associated outputs. To meet these requirements, it’s necessary to use build integrity verification, which refers to the process of ensuring the integrity and authenticity of software builds or container images. This is achieved through signing, verification, and provenance.

Signing: Developers use tools like Cosign (Sigstore) to digitally sign their container images or software builds. This involves attaching a cryptographic signature to the image, which proves that the image has not been tampered with and was indeed built by the claimed author or organization.

Verification: Users or other stakeholders can then use the same tool to verify the signatures on these container images. This verification process involves checking the digital signatures against public keys which provides a trust mechanism to ensure the authenticity and integrity of the image.

Provenance: Sigstore aims to provide transparency and a clear provenance for software builds, the public keys and signatures are stored in a tamper-resistant, publicly auditable log, making it possible to trace back and verify the history of a container image, including who signed it and when.

Artifact Signing Options



The Endor Labs Solution for Artifact Signing

Endor Labs provides an alternative to Sigstore's Cosign for organizations that value signature privacy (unlike the public sigstore) and no-new-infrastructure deployment (unlike rolling out your own private sigstore).

The low-code/no-code artifact signing solution is part of Endor Labs CI/CD. This solution provides a robust path to signing any artifact along with relevant metadata, using your existing identity system (such as Azure AD or Okta) and our signature and append-only log platforms as the base. This means deploying signing is as easy as:

- Setting up a SAML or OIDC link between your identity system and Endor Labs; do this once for your entire organization
- Deploying an Endor Labs binary into your pipeline (as a CLI tool, GitHub Action, etc.) where you want signatures to occur, and configuring the pipeline to call the signing function -- your pipeline uses its verified identity in your system to authenticate to Endor Labs, and signing proceeds
- Setting up your admission controller/other points of verification to use Endor Labs to sign (typically one command-line call)

Container Scanning & Hardening

Increasingly, modern applications are run in containers, which are defined as "standard units of software that packages up code and all of its dependencies". This standardized packaging allows for applications to be lightweight, portable and standalone, enabling scalability, replication, and re-use. The challenge, as discussed in the previous section regarding dependencies, is that those dependencies also may have vulnerabilities.

This makes the concept of container hardening, and utilizing hardened and secure containers that have a minimized vulnerability footprint critical. Otherwise, developers end up building on top of vulnerable containers full of often unnecessary dependencies, typically referred to as "bloated containers". Those vulnerabilities in container images now place any applications running on top of them at risk as malicious actors can exploit vulnerabilities in the underlying containers.

Malicious actors have shown that they will look to exploit vulnerabilities in underlying containers, as well as compromise widely used publicly available containers from popular container repositories such as Docker Hub in efforts to impact downstream consumers and their associated applications using the vulnerable containers.

Frequently Asked Questions

Why is software supply chain security such a major concern?

Software supply chain security is a major concern because vulnerabilities introduced anywhere in the complex network of components and processes used to develop software can be exploited by attackers to gain access to systems or data. This makes it difficult to trust even well-known software, and highlights the need for strong security practices throughout the entire development lifecycle.

What are some different types of attacks that can target a software supply chain?

Two prevalent types of attacks are infiltrating a vendor's network and injecting malicious code into software before it's distributed (e.g., the SolarWinds attack) and tampering with open source code or app stores to introduce vulnerabilities.

How does the concept of "pull" style attacks differ from traditional security threats?

Traditional attacks target a system's own flaws, while "pull" attacks target trusted sources like vendors to inject malicious code that users unknowingly install.

What are some trends that influence how we secure the software supply chain?

The rise of containerized applications and the widespread adoption of open source components (OSS) introduce new attack surfaces. At the same time, Agile and CI/CD practices, while accelerating development, can leave less time for security checks. Finally, the integration of AI models, while powerful, brings its own security considerations.

What role do industry standards and compliance play in software supply chain security?

Industry standards can enable better communication and vulnerability management, for example SPDX and CycloneDX provide a common format for SBOMs. Compliance with regulations like PCI DSS can also drive adoption of security best practices throughout development.

What are the different components that make up a software supply chain?

The software supply chain includes all the elements involved in creating and delivering software: *code* (internal, open source, vendor-developed), *infrastructure* (development environments, build servers, deployment platforms), people (developers, security, operations), and *tools* (code repositories, build tools, configuration management, containerization).ential vulnerabilities in pipelines.

What are direct and transitive dependencies, and their associated risks?

Direct dependencies are components you explicitly choose for your software, while transitive dependencies are those indirectly required by your direct choices. Both can introduce risk: both can have vulnerabilities, and transitive dependencies are hidden and potentially unmanaged, making them a blind spot for security issues.

What are some security considerations when using open source software (OSS) in development?

Security considerations for OSS include evaluating the source's reputation, license terms, and vulnerability history. Organizations can leverage tools to detect OSS dependencies, determine if they carry risk, and maintain an updated inventory of used components.

What is an Open Source Program Office (OSPO)?

An OSPO is a cross-functional team or program tasked with managing an organization's open source activities, strategies, policies, and best practices. They may set guidelines for using, distributing, selecting, and auditing open-source components, as well as educating employees and fostering engagement with the open source community.

What is the difference between SAST and SCA tools, and how do they work together for better code security?

SAST (Static Application Security Testing) tools analyze source code for vulnerabilities and coding flaws, while SCA (Software Composition Analysis) tools identify known vulnerabilities within an application's dependencies. Together, they provide a comprehensive view of potential security risks, allowing developers to fix flaws in their code (SAST) and upgrade vulnerable dependencies (SCA).

How can organizations leverage AI models securely within their software supply chain?

AI models can be vulnerable to attacks that manipulate training data or the model itself. Organizations should establish policies for using AI models, considering aspects like data types and vendor security practices. Secure practices include using well-vetted training data, monitoring model behavior, and implementing controls to prevent unauthorized access or modification.

What are the benefits of using SBOMs, and what are the challenges associated with them?

SBOMs offer transparency by detailing all software components within an application, enabling identification of known vulnerabilities and other risks. However, ensuring accurate and complete SBOMs can be challenging. Industry standards exist (SPDX, CycloneDX) to address this, and some tools can help generate and analyze SBOMs.ulnerabilities in pipelines.

How can organizations prevent accidental leaks of sensitive information like passwords stored as secrets?

Robust secrets management involves implementing policies and using scanning tools to identify secrets unintentionally stored in code repositories, configuration files, or other artifacts. These tools can differentiate between valid and invalid secrets, allowing for focused remediation.

Beyond code security, what other aspects of CI/CD pipelines require attention to ensure overall security?

Securing CI/CD pipelines involves hardening the underlying infrastructure, managing access controls, and controlling the tools and configurations used within the pipeline. Security tools can discover potential risks in pipeline configurations and dependencies, while artifact signing can verify the integrity of components used in the build process.

How does artifact signing improve software supply chain security, and what are some considerations for implementing it?

Artifact signing involves attaching a digital signature to software builds, verifying their authenticity and integrity. This helps prevent tampering during the build process and allows organizations to implement controls based on the signer's identity. Considerations include choosing a signing solution and establishing trust mechanisms for verifying signatures.

Why is it important to harden containers used in development, and how can organizations achieve this?

Container hardening minimizes the attack surface by removing unnecessary dependencies and functionalities. This reduces the risk of vulnerabilities within the container impacting applications built on top of them. Container scanning tools can identify vulnerabilities in container images, allowing developers to address them before deployment.

What are some tools organizations use for software supply chain security?

There are several types of scanners that are used, including application security testing (SAST, DAST, SCA, etc), secret scanners, and container scanners. Organizations usually use a tool to generate SBOMs (which may be combined with a scanning tool). And a new category of tools is emerging to address CI/CD security issues, which can discover tools and potential vulnerabilities in pipelines.

ENDOR LABS

Secure Everything Your Code Depends On

Without the Productivity Tax

Start Trial