boost security.io

CISO & CTO Guide to
Supply Chain Security

Securing The Software Factory

Guide

2025

Developer Risk 06

Risk #1 Developer account compromise targeted through phishing, social engineering, malware 3 06

Risk #2 Insider Threat 07

1st Party Code 08

Risk #3 Writing insecure code or designing insecure systems - unintentionally 08

Risk #4 Inserting malicious code into the project - intentionally 08

Risk #5 IP theft 6 08

Risk #6 Credential Theft 08

3rd Party Code 10

Risk #7 License Risk 10

Risk #8 Known Vulnerability Risk 10

Risk #9 Disappearing, unmaintained, end-of-life, or poor quality 3rd party software 12

Risk #10 Malware in OSS packages 13

The Development Infrastructure 14

Risk #11 Compromising a misconfigured source control system 14

Risk #12 Compromise the source control system via its extension capabilities 14

Risk #13 Compromising a misconfigured Build or Deployment System 15

Risk #14 Compromising the Build & Deployment Systems through their extension mechanisms 16

Risk #15 Compromise the artifact registry for IP Theft or uploading modified artifacts 16

Additional Layers of Risk

So, what's a Cx0 to do?

Appendix: Security Standards

CISO & CTO Guide to
Supply Chain Security

Securing The Software Factory

Introduction

In 2020, SolarWinds unknowingly distributed malware-infected software to its customers. The malware, inserted by Cozy Bear, a group linked to the Russian Foreign Intelligence Service, infiltrated over 18,000 networks and sent sensitive data to a remote server. This was a very high profile, very advanced software supply chain attack.

18,000+ Networks compromised in SolarWinds

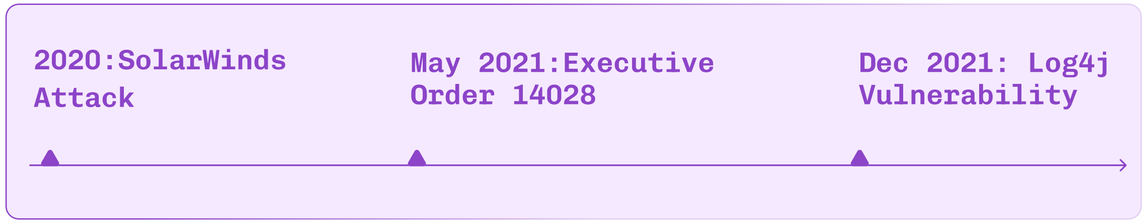
In May 2021, the Biden administration issued Executive Order (EO) 14028 on Improving the Nation's Cybersecurity, which included a strong focus on supply chain security. The order aimed to enhance software supply chain transparency, integrity, and security following major cyber incidents like the SolarWinds breach. It mandated new security standards for software sold to the federal government, requiring vendors to :

- → Provide Software Bill of Materials (SBOM)
- → Adhere to secure development practices based on guidance from NIST (National Institute of Standards and Technology)

The EO also called for pilot programs to evaluate trusted software suppliers and improve incident response coordination across federal agencies and critical infrastructure. This move signaled a significant shift toward zero-trust architecture and supply chain risk management as national security priorities.

Log4Shell:Affected millions of devices in 24 hours

In December 2021, the Log4J vulnerability, also known as Log4Shell, exposed a critical flaw in the widely-used Apache Log4j Java logging library. This zero-day vulnerability allowed remote code execution, enabling attackers to gain control of affected systems. The impact was widespread, affecting numerous applications and services across the globe, leading to significant security breaches and prompting urgent remediation efforts by organizations worldwide. Given the activity of the last few years, the software supply chain has been a popular topic of discussion among practitioners, and an increasing target among cyberattackers.



Despite the attention & importance, the "software supply chain" is one of the many terms in cybersecurity whose meaning is overloaded . When having conversations with different cybersecurity leaders, we realize that for some, talk of the securing the software supply chain implies having SBOM (Software Bill of Materials) capabilities, for others, it implies using attestations & verification steps to ensure that code and artifacts have not been tampered with, and yet for

others, the image in their head is that of ensuring that malware does not enter their development and production environments through compromised open source packages.

You are only as strong as your weakest link

In 2025, the software supply chain is one of the weakest links

Guide Roadmap

- Explore how BoostSecurity thinks of the software supply chain
- Define the unique components that make up your software supply chain
- Examine the risks and exploits in each area
- Share emerging industry standards

Remember, perfect security never exists. The goal is to first understand the risks, and then decide on the approach to handle each of them.

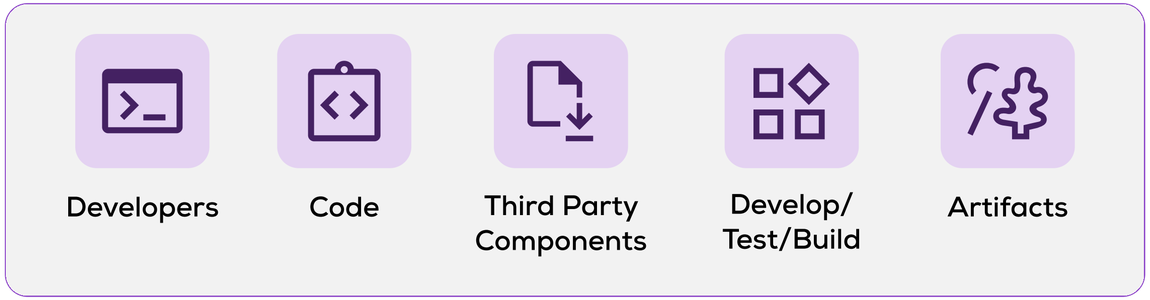
Defining the Software Supply Chain

Let's start by defining the software supply chain:

> The software supply chain is everyone and everything that is involved in the development, building, testing, and deployment of your (software) artifacts.

Note that in this definition we excluded the entire consumer side of the supply chain. In the case of a SaaS service or API service, this would be the runtime environment. This could also be the software running on a chip in a vehicle. While we do think that the consumer side is indeed part of the software supply chain - and indeed it does face many of the same risks, we keep it out of this paper for now, for brevity's sake, and to keep things focused on the software factory vs. the operational nature of running a software product.

Let's start breaking this down.
The big picture:



Exploring Categories of Risk

→ Developer Risk

An often overlooked, but nonetheless critical part of your supply chain is the developer, and the machine & services they use to develop code. In some cases, the developer is an employee of the organization. In other cases, they're external, either contracted directly by the organizations or through an outsourced development shop.

Risk #1 - Developer account compromise targeted through phishing, social engineering, malware

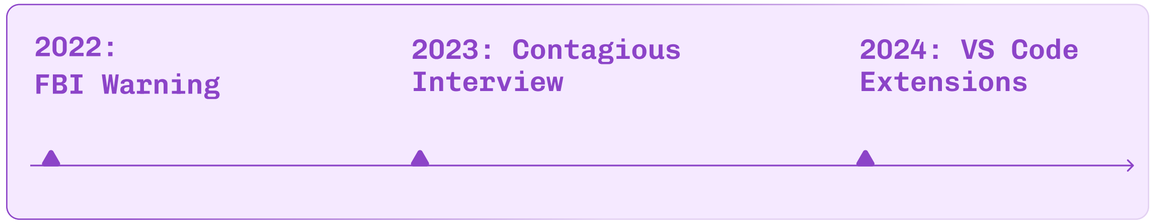
Developers are now direct targets of cyber attacks: the Contagious Interview (PaloAlto Networks research) campaign labeled as such in 2023, continued to occur in 2024, and even in 2025 (SentinelOne research).

In these types of campaigns, attackers pose as interviewers looking to employ developers, the developers do an interview, and then will be asked to download a video conferencing app.In this scenario, the app contains malware. We have seen other variations of this, where the developer is expected to do some work on a sample software project. Unbeknownst to the developer, by working on that project, the developer ends up installing malware, which among other things, can steal credentials and tokens required to access their current employer's source code.

Their machine can be compromised through a variety of other means, among which is installing tools that contain malware (and we have seen many examples in open source and in closed source products as well). Infostealers will lead to the compromise of the session cookies for services such as GitHub.

Recently, malicious VSCode Extensions with millions of downloads were pulled from the marketplace due to them having malware. Machine Learning Models on HuggingFace were also found containing malware. These models are used directly by developers and data scientists.

Attack Evolution



Risk #2 - Insider Threat

Then there's the insider threat factor; a developer (contractor or employed) with access to source, build, and potentially even runtime access may be malicious.In fact, in 2022 the FBI warned about North Korean state workers posing as other nationals, working in Russia, China, Southeast Asia, and Africa as remote workers to dozens of Fortune 100 companies.

A CISO recently told us that they hired people based on interview results in a different country, to do remote work, and learned later that other developers showed up to do the work.

Of course, your organization may have a more simple version of this - developers with access to source/IP may act with malicious intent for a variety of reasons (hacktivism, personal gain, etc). This happened in 2021 in the Crypto DeFi project SushiSwap, where a malicious commit by a contractor resulted in the theft of $3M from the chain for the personal benefit of the contractor. In another example, a developer working for Eaton corporation inserted malware into their production systems. In the event he was ever let go and his active directory account was terminated, the systems would lock up across the organization.

EXECUTIVE SNAPSHOT: DEVELOPER SECURITY THREATS

⚠️ **ACCOUNT COMPROMISE (Risk #1)**

→ **Real threat:** Contagious Interview campaign targeting your developers

→ **Attack vector:** Malicious "video conferencing apps" during fake job interviews

→ **Recent Examples:** Compromised VSCode Extensions (millions of downloads)

⚫ **INSIDER THREAT (Risk #2)**

→ **Real threat:** Malicious employees or contractors with system access

→ **Attack vector:** North Korean state workers posing as remote employees

→ **Recent Examples:** Eaton corporation malware, SushiSwap $3M theft

1st Party Code

Risk #3 - Writing insecure code or designing insecure systems - unintentionally

Just as “all software has bugs”, all software will have security flaws, too. Various forms of testing in addition to threat modeling are needed to uncover these unintentional vulnerabilities introduced in the normal course of development. Think AppSec, Static or Dynamic Analysis (SAST/ DAST) or other forms of security testing to uncover these bugs and eliminate them.

Risk #4 - Inserting malicious code into the project - intentionally

We stated earlier that there are several scenarios in which a developer account can be compromised (fake interview campaigns, malware) or in which there is an insider threat. Regardless of the origin event, the impact is that malicious code may be inserted in the project, perhaps in the form of a backdoor.

CASE STUDY: $1.4B ByBit Hack (2025)

In February, the Bybit Crypyto hack started with the compromise of a developer, with what is believed to be either a phishing or a social engineering tactic. The hackers used the developer’s compromised machine to inject malicious source code into the code repository, conducted the hack, then removed the code.

Risk #5 - IP theft

Compromised developer tokens can lead to source code theft. When a malicious insider discovers they have access to source code repositories that they shouldn’t, source code theft is often the result.

Risk #6 - Credential theft

We know that stolen secrets are often the cause of a security breach, and unfortunately, hardcoded secrets in the development process remain more common than they should. In 2023, close to 13M secrets were detected in public GitHub commits. At BoostSecurity, we can confirm that the same problem exists in private repositories.

EXECUTIVE SNAPSHOT: 1st PARTY CODE RISKS

⚠ UNINTENTIONAL SECURITY FLAWS (Risk #3)

Reality: All software inevitably contains security vulnerabilities.
Solution needed: AppSec, Static/Dynamic Analysis (SAST/DAST)

Best practice: Implement security testing in development workflow

⚠ MALICIOUS CODE INSERTION (Risk #4)

Impact: Backdoors that can lead to major breaches

Example: Bybit wallet hack ($1.4B loss) in February 2025

Attack Vector: Developer account compromise led to malicious code injection

⚠ UNINTENTIONAL SECURITY FLAWS (Risk #3)

Problem: 13M secrets detected in public GitHub commits in 2023
Consequence: Source code theft when attackers access repositories
IP & CREDENTIAL THEFT: Hardcoded secrets in development remain too common

How AI Impacts 1st Party Code Risks

EXECUTIVE INSIGHT: AI CODING RISKS

→ AI assistants increase code volume but introduce more security vulnerabilities

→ Developers overestimate the security of AI-generated code (Stanford study)

→ LLMs lack system-level context and threat modeling

→ Proprietary IP can be unintentionally exposed (Samsung case)

Research Finding: Stanford University

A recent Stanford University study found that participants with AI assistant were:

- More likely to introduce security vulnerabilities for the majority of programming tasks
- More likely to rate their insecure answers as secure compared to those in the control group

- Producing code with security issues that they failed to recognize
- Proprietary IP was placed into LLM's
- This raised questions about whether this IP can be served to other users
- This happened at Samsung in 2023
- Demonstrates real risks of IP exposure when using AI coding tools

Key Issues with AI Coding Assistants:

Lack of Application Context:

- LLMs lack context about the overall application
- Cause code snippets to incorporate bold assumptions
- Cannot understand the broader system requirements

Missing Threat Model Thinking:

- AI tools lack system threat model thinking in their design
- Generate code without security considerations
- Higher volume of code + more security issues per line = more vulnerabilities overall

Recommendation

Implement additional security scanning for AI-generated code and establish clear AI usage policies to prevent IP exposure.

3rd Party OSS packages & container images

This area is what most people tend to think of as the collective "Software Supply Chain". We all use OSS packages and container images and given that prevalent use, the risks inherent in these pieces of 3rd party software are more or less understood these days.

EXECUTIVE INSIGHT: 3RD PARTY RISKS

- → License risks may create legal exposure during M&A or through GPL violations
- → Most reported vulnerabilities are not exploitable in your specific context
- → Focus on vulnerabilities with high EPSS scores and confirmed reachability

Risk #7 - License Risk

OSS packages come in all sorts of license flavors, including restrictive licenses such as GPL. In many organizations, checking for license issues is done frequently, even at every commit. This activity also occurs in M&A transactions, where the acquiring company wants to understand any legal risk involved in the acquired company.

Risk #8 - Critical Risk

3rd party components, be they open source or not, may have known vulnerabilities associated with them. Software Composition Analysis and Container Scanning solutions are used to detect these types of vulnerabilities. SBOM's can be used to understand what CVE's exist in a particular artifact as well.

THE VULNERABILITY CHALLENGE

The typical challenge companies face when looking at known vulnerability risk is primarily around:

- Triaging the sheer volume of reported vulnerabilities
- Understanding which are exploitable (the vast majority are not)
- Resolving these issues

Hence, the current best practice revolves around the idea of narrowing in on exploitable vulnerabilities and providing clear remediation guidance to the right person.

Understanding Exploitability

What is Reachability Analysis?

Reachability analysis is the type of analysis performed that tries to distinguish between CVE presence and CVE exploitability. Reachability aims to determine whether an application is actually vulnerable to a particular CVE; the idea being that if a vulnerability exists in a library, but the application using the library does not make use of the vulnerability path, then the application is not necessarily vulnerable.

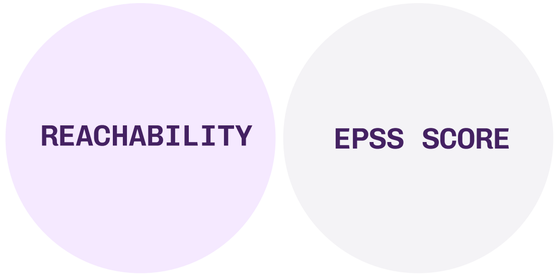
TYPES OF REACHABILITY ANALYSIS:

There are many different types of reachability analysis: using static analysis to determine if the vulnerable code is called, using runtime analysis to determine if the module with the vulnerable function is loaded into memory, or even called in certain applications. There are even approaches that leverage LLM's to try to determine reachability.

Reachability is only one metric to help with inferring exploitability, the other is looking at whether or not a particular CVE is being actively exploited in the wild. The EPSS score is such a metric.

CISO KEV

Another one would be the CISO KEV - which tracks known frequently exploited vulnerabilities, although the emphasis is less on open source packages, and rather more on commercial off-the-shelf applications and larger systems. If you find such a package in your applications you have to investigate the possibility of an exploit that has already happened.



KEY ASSESSMENT FACTORS

What is the EPSS score? EPSS stands for exploit prediction scoring system, and is a percentage that is assigned to a vulnerability, meant to indicate the likelihood that the vulnerability will be used by attackers in the next 30 days. The formula to calculate the percentage takes into account factors such as:

UNDERSTANDING THE EPSS SCORE

- Reported exploits in known threat feeds
- Availability of public exploit code
- Whether or not the vulnerability is discussed in mailing lists or websites such as CISA KEV, Google's Project Zero, Trend Micro's Zero Day Initiative
- Age of the vulnerability

PRIORITIZATION BEST PRACTICES

EPSS helps you prioritize which vulnerabilities the team should work on first, as ones with a high EPSS score can be considered to be riskiest. However, EPSS should NOT be the only criteria of prioritization: whether the vulnerability is actually reachable by attackers, reached in the code, and its impact if exploited, are among many factors that have to be weighed when determining the priority.

Risk #9 - Disappearing, unmaintained, end-of-life, or poor quality 3rd party software

In 2021, a Ruby change in a library called mimemagic resulted in the breaking of hundreds of thousands of applications. Something similar happened in the Javascript ecosystem when a developer deleted a project in protest. That project was used by many projects in turn, and this act broke large portions of the internet.
The open source world is full of wonderful projects. However, not all of them are properly maintained, or have high quality standards. To infer the quality of a project, one can look at the download stats as well as the volume of development activity in the project - for example, are bugs reported

To infer the quality of a project, one can look at the download stats as well as the volume of development activity in the project - for example, are bugs reported being addressed, the size and activity of the contributors to the project, to name a few.
Another consideration is whether or not the different projects are configured with basic security best practices. The OpenSSF scorecard is a great project that helps to achieve some of these goals. Anyone can can easily view the scorecard of a project on https://deps.dev

Certain versions of a project become end-of-life. When this occurs, they stop receiving support and security fixes. It is almost always better to only use software that is still maintained. Hence you want to ensure that you are aware of any projects that are already, or are soon going to be, end of life.

Collaborative Defense

There is wide collaboration in both industry and the open source ecosystem to address these types of flaws. There are features being built directly into the open source registries to ensure that such malicious entries are either prevented, or detected early.

Specific examples helping to address these risks include:

1. **Package Repository Security**
   Package repositories implementing built-in security scanning capabilities
2. **Cryptographic Verification**
   Cryptographically signing and verifying packages and making it relatively easy to check for validity of package
3. **Best Practices Publications**
   OpenSSF's Software Development Best Practices publications
4. **Alpha-Omega Project**
   Alpha-Omega project scans and and monitors the most critical open source software projects and ecosystems

Attackers have figured out that inserting malware into the open source ecosystem is an effective way of compromising a lot of developer accounts and even production environments.

Attackers are able to insert malware into open source packages in a variety of ways:

Malicious Uploads

Simply uploading a malicious package and tricking users into installing via a variety of means. Examples of this include warbeast2000 and kodiak2k which was outright theft of developer ssh keys.

Trust Exploitation

Gaining the trust of the project maintainers, getting rights to contribute to the code base, and then acting maliciously → the near miss of xz-utils which could have been the biggest cyber breach in history had it gone unnoticed. Jia Tan, the "name" of the developer (that was never caught) - spent 2 years working positively on an important, and widely used project, before inserting malicious code into it.

Credential Theft

Stealing a project developers credentials Abusing a vulnerability in the build environment of the package

Typosquatting

Leveraging typosquating opportunities - attackers can add clone projects of well known projects, with only slightly different names. All the developer has to do to fall victim to this type of attack is to reference the typosquat version of the package.

Recreating Deleted Projects

Re-registering de-commissioned project names - Open source packages are created every day, but every now and then, a project will get deleted. When this happens, the project is removed from the central registry (such as pypi and npmjs) however, there may be many applications that still depend on it. This attack occurs when an attacker re-registers a project name, recreating it with malware included.

Dependency Confusion

Dependency Confusion - many software teams have their own packages that they develop and re-use across an organization. For example, a company may develop a "company-authentication" package that is to be used across the various teams. More often than not, these packages are hosted on internal package registries. However, in certain circumstances, if a package with a similar name exists on public registries, such as npm or pypi, hose external packages can be used during the build process.

The Development Infrastructure

SCALE OF THREAT

This part of the software supply chain is the big blindspot of the software factory. The vast majority of software being developed today, leverages source code management systems, continuous integration tools, and continuous deployment tools. In most cases that we see, additional infrastructure such as artifact registries are also used.

From a security standpoint, in our opinion **this is the least understood, and most insecure part of the software supply chain.** For the past two decades, agile development and devops practices took over the software development world, leading to an enormous amount of infrastructure being used to automate the software development process. Awareness around insecure configuration and usage of this infrastructure is still very low. Attackers, on the other hand, are becoming more and more aware of the various ways in which they can attack this particular surface.

Risk #11 - Compromising a misconfigured source control system

The source code management system you use, whether GitHub, GitLab, Azure DevOps, BitBucket, etc, can be improperly configured from a security standpoint. Why does this happen?

- Lack of knowledge
- These systems develop new security capabilities over time; and people don't always find the time to go back and leverage the security enhancements
- No consistency around configuration; Employees move around and bring different ways of setting up projects and systems

Risk #12 - Compromising the source control system via extension capabilities

The SCM systems are all extendable via some form of plugin or app mechanism. For example, at the time of this writing, there are over 6000 GitHub Apps (over 1000 listed on the marketplace). These apps vary in quality - some are by verified publishers, while others are not. What we typically see in organizations is many apps are installed by developers, because they offer useful functionality - without consideration to the security risk they bring. For example, many apps will require reading and/or writing source code permissions where they are installed.

Risk #13 - Compromising a Misconfigured Build or Deployment System

Continuous Integration and Continuous deployment systems, such as GitHub Actions, GitLab pipelines, CircleCI, Jenkins, ArgoCD, and Terraform, automate the software development process, but their configuration settings can inadvertently introduce security vulnerabilities. Misconfigured CI systems can expose sensitive information, allow unauthorized code changes, or even provide a pathway for attackers to compromise the entire software supply chain. These misconfigurations can arise from various reasons, including a lack of understanding of security best practices, failure to update configurations as security capabilities evolve, or inconsistencies introduced by employee turnover. Consequently, a seemingly efficient CI system can become a significant security risk if not properly secured.

Some examples of such misconfigurations include:

- → inadequate access controls, which allow unauthorized users to modify pipelines or deploy code
- → insufficient logging and monitoring, which makes it hard to detect and respond to incidents
- → Insecure pipelines that are not validated, allowing attackers to inject malicious steps or code

GitHub Verified Publisher

What is a GitHub Verified Publisher?

A GitHub app by a verified publisher just means that the domain name has been verified, that the developer of that app uses Two-Factor Authentication, and that there is a way to contact the publisher. It does not say anything about the security or safety of using that app.

Another way to extend these systems is through leveraging their webhook capabilities. An attacker (malicious insider, or attacker that managed to get access to developer account) can register a webhook that sends code base updates to a server under their control, enabling them to receive IP well after the account access is removed.

Pipeline workflow code is just code. It is the application that builds your application. Since it is just code, it can contain vulnerabilities. Just like application code, it needs to be tested for vulnerabilities. Just like your applications, it includes:

- → First party code: code your developers wrote
- → 3rd party code: code your developers brought in

Both can contain vulnerabilities.

Risk #14 -Compromising the Build & Deployment SystemsThrough Their Extensions

CI/CD systems have plugin and extension mechanisms, which developers take advantage of. For example, GitHub workflows are written using many of the 12,000 3rd party GitHub actions. CircleCI workflows leverage many of the 3,700 3rd party orbs. These extensions can have vulnerabilities, or could be compromised with malware as well. On the Continuous Deployment side,this risk includes systems like ArgoCD, and Terraform for cloud infrastructure.

An example of this is the Codecov supply chain attack from a few years ago. The Codecov bash uploader script was modified by attackers to exfiltrate CI environment variables which often contain secrets. This allowed attackers to steal secrets from Codecov customers

Risk #15 - Compromising the Artifact Registry

Most organizations with sufficient software activity have their own internal artifact registries, or use SaaS versions of them. These registries store various artifacts for building or deployment. Examples of such registries include:

Registry Examples:

- DockerHub, Quay.io
- JFrog artifactory
- Amazon ECR, Google Artifact Registry
- GitHub Packages, GitLab Package Registry

Registry Risks:

These registries will contain either the building blocks of the application (container base images, open source packages), and/or the finished artifacts for deployment. You need to ensure that only trusted software is deployed into these repositories.

Misconfiguration Risks:

If these repositories are misconfigured, or if the tokens required to access them are in the wrong hands, attackers may be able to:

ATTACK SCENARIOS

→ **Directly upload artifacts modified with malware**

→ **Steal IP by downloading the artifacts**

Additional Layers of Risk

In addition to viewing risks across your supply chain as we described above, you'll also want to consider code based specific risk nuances. For example, the risks of writing an API interface are very different from the risks of embedding a binary (firmware) into your product.

Hence, to truly get a complete understanding of the risks of the supply chain, you will need to understand **"What type of code is this? What kinds of risks is it exposed to?"**

Here are some examples of risks that are specific to certain types of code:

API

If your codebase exposes an API, and chances are that if it is a newer application, then it does (because of API first architectures, or wanting an integration plane) - then you want to consider risks described in the OWASP API Top 10.

SaaS

Applications that use 3rd party SaaS services (think payment services, authentication services, and so on) - could be vulnerable to . At a minimum, you would probably want to know which 3rd party services are used by your applications, to be able to determine if you are affected in any way by a breach. A recent case in point is the Snowflake breach.

AI Components

If your application leverages AI technologies and services (be it Generative/LLM or Predictive/ML), then you will want to consider AI specific risks. There are many AI risk models out there, but commonly referenced ones would be the OWASP LLM Top 10 and OWASP ML Top 10. Risks vary from attackers inserting bad data into the model, to prompt injection, to supply chain attacks on these models, and much, much more. We have a lot more to say about this topic, but we will leave that for a future blog.

Binary Artifacts

This may be less common in pure software applications, but is seen more in manufacturers (computers, toys, medical devices, security devices, lab equipment, phones, etc). The supply chain will include certain binary components (firmware, an entire OS image, etc). These binary components carry their own risk; they can include malware (intentional, or not), include known vulnerabilities, code with restrictive licensing, or even end-of-life (and unmaintained) software.

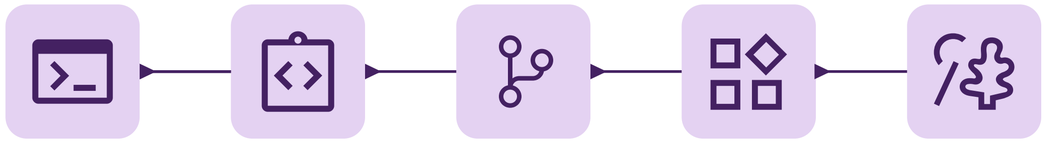
So, what's a CxO to do?

We hope that this guide helps you understand in practical terms what sorts of supply chain risks your software development organization is exposed to.
Far too frequently, leaders believe that having SBOMs in place and OSS package or container scanning is sufficient for securing the software supply chain.
To truly secure your development organization against these emerging threats, one must take a much more comprehensive view.

Start by recognizing the risks facing different parts of the supply chain, from the developers, through to the source code they write, to the open source they use, to the development infrastructure that is used to develop, test, and release, and finally ending with the artifacts. This guide gives you a list to work with your teams on.

A number of best practices and standards are outlined below that address different parts of securing the software supply chain. Furthermore, solutions like BoostSecurity offer a comprehensive solution to mitigate these risks.

Software Supply Chain Risk Areas



This guide gives you a list to work with your teams on.
A number of best practices and standards are outlined below that address different parts of securing the software supply chain. Furthermore, solutions like BoostSecurity offer a comprehensive offering to mitigate these risks.

Don't stop at the SBOM. Instead, work to identify a purpose built software supply chain security solution that extends beyond application code to address risks across the entire Software Development Lifecycle (SDLC). One that encompasses third-party dependencies, libraries, tools and infrastructure for a more comprehensive view of your risks, leverages advanced technology and applications to reduce alert noise through reachability analysis, and plugs seamlessly into your developer tech stack and workflow.

Appendix

Additional References: Security Standards

Over the past few years, several security standards emerged to address the challenges facing our industry in securing the software development process. The following is a list of a few of the most well known:

CIS Supply Chain Benchmark:
List of controls for securing your development infrastructure.

NIST 800-218 (Secure Software Development Framework - SSDF):
A very comprehensive set of recommended practices designed to help organizations reduce the risk of vulnerabilities in software and improve the security of the software supply chain.

SLSA:
A security framework with a list of controls to prevent tampering and secure packages and infrastructure.

CNCF Secure Software Factory:
A reference architecture for securing the software supply chain

CNCF Supply Chain Security Best Practices:
Set of best practices that encompass securing the software development process

Secure Supply Chain Consumption Framework:
A framework for securely consuming OSS projects.

Secure Software Attestation Form:
A form that has to be completed by the CEO of an organization attesting to the fact that the software produced in the organization is built using what is considered to be essential secure software best practices.

NSA & CISA's Cybersecurity Information Sheet on Defending CI/CD Environments:
Information sheet describing the emerging CI/CD attack surface, and the best practices to secure it.