The Practical Guide for Understanding & Preventing Software Supply Chain Attacks

Table of Contents

Introduction

Our world runs on software. Code drives everything we use, from the apps to our connected devices to the industrial systems that manage our utilities. All of these products depend on huge quantities of code to get the job done, with most of it originating from outside their organization and their direct control.

In order to keep up with the pace of development and focus their talents on the pieces of their products that make them shine, organizations are making increasing use of open source software and commercial components.

By some estimates, 99% of all software products use open source software – comprising 85-97% of the total code base.

(source: GitHub)

While this widespread use of third-party (open source and commercial software) should be celebrated for making development faster, cheaper, and of generally higher quality, it is not without its challenges. Just as organizations are expected to take responsibility for the code that they produce themselves, their customers depend on their vendors to vet and secure all software components that they integrate from other sources as if it was their own.

The challenge that software developers, product producers, and asset owners must contend with is in ensuring that their supply chain is secure.

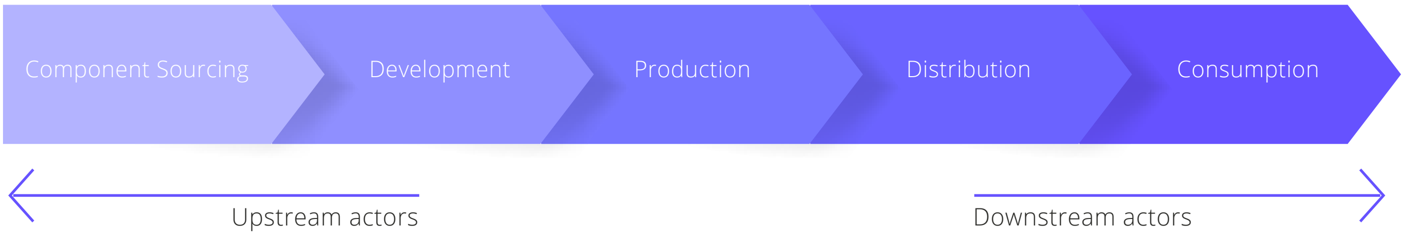
This document presents a guide to understanding the threats facing your software supply chain and will offer actionable recommendations on how to effectively mitigate risks.

Defining Supply Chain Risk

Software today is not composed, it is aggregated.

Developing software in modern organizations depends heavily on software components from third-party sources, using them to form the foundation of your code base. These software components and products are drawn from open source software projects (ex. Apache, Linux) and commercial products (Windows, Oracle).

Organizations that utilize these third-party software components have a responsibility to know the provenance and security of the code that they are putting into their products. Customers care little if they are exploited through a vulnerability that your team wrote or if it came from a bug in an open source library.



Basic Software Supply Chain Flow

The buck stops with you.

At the tail end of 2020, the world received a sharp reminder of the risks facing the software supply chain. State-based attackers successfully compromised the SolarWinds' Orion IT performance monitoring system's update server to push their malicious code out to SolarWinds' customers through an otherwise trusted path. This allowed them to bypass their targets' defensive tools and make their breach undetected. Victims included Microsoft, the U.S. Government, and many other high profile targets that were otherwise well defended from external threats.

Supply chain cybersecurity risks result from weaknesses or vulnerabilities that are intentionally, or unintentionally, introduced into the organization through the acquisition of software and hardware. In cases where this is done intentionally, it constitutes a supply chain attack.

Supply chain attacks provide hackers with an enticing alternative to traditional perimeter attacks which first target the organization's network ingress points and spread inwards through the breach. Instead of breaching the fortified walls, these attacks are similar to a Trojan horse which enters the city through the main gates. The attack works by introducing a malicious component into benign-looking software or an appliance. This code is then installed inside the organization's network by its unwitting administrator. Because they bypass the perimeter defenses and most network-based monitoring systems, supply chain attacks can be very hard to detect.

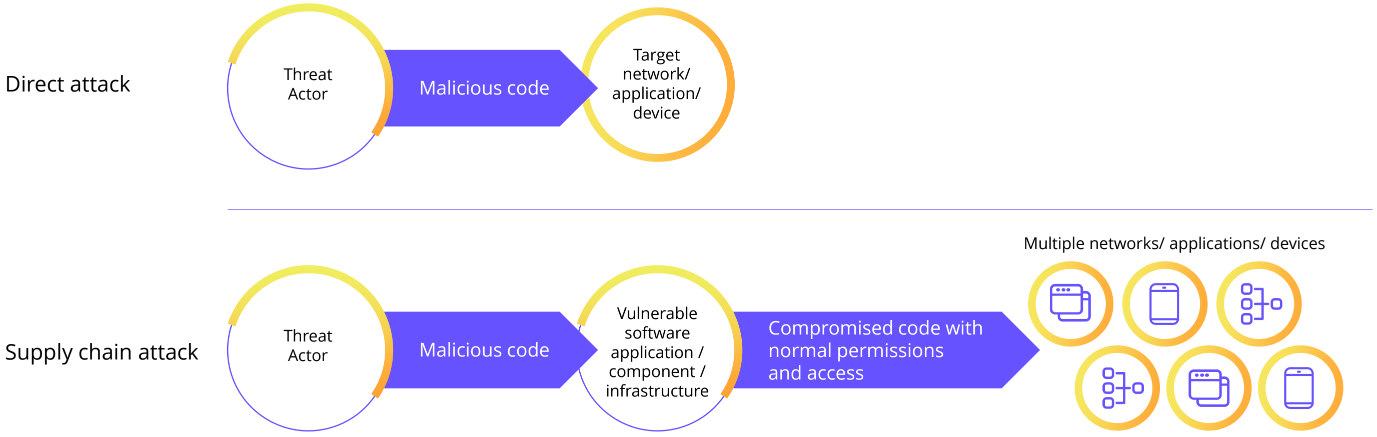
Supply Chain Risk Challenges

Supply chain attacks pose additional risks and challenges for product owners and security teams to contend with beyond the standard threat of a more traditional cyber attack. The concerns can be divided between their potentially expanded scope of impact and the difficulty in detecting them.

Expanding the Scope

A directed attack against an organization via social engineering or by exploiting a vulnerability in their code can be devastating. However, the impact of such attacks is generally limited to the targeted organization.

In the case of a supply chain attack though, there is the potential to exploit any organization that is using the compromised software component downstream. These attacks can have widespread effects, especially in cases when a popular open source project used in many different organizations is compromised.


Direct attack
Threat Actor
Malicious code
Target network/ application/ device
Supply chain attack
Threat Actor
Malicious code
Vulnerable software application / component / infrastructure
Compromised code with normal permissions and access
Multiple networks/ applications/ devices

Difficulties in Detection

Organizations often lack the ability to inspect code from third-parties, making it more challenging to detect potential threats. They simply do not have the necessary visibility to protect themselves from malicious code tucked away in the package. No ingredient lists or nutrition facts are available. Definitely no allergy warnings.

Black Boxes

One major challenge is that significant amounts of the code installed by organizations is closed source that they receive as opaque binaries. This code is like a black box, difficult to inspect and prohibitively costly to reverse-engineer.

Lack of Visibility

The software development industry has improved many processes but still has a ways to go when it comes to creating a comprehensive Software Bill of Materials (SBOM) that lists what is inside the product. Unfortunately, most companies lack the tools and the processes to establish the exact provenance of a software package down to its sub-components.

Hidden Dependencies

A common issue when it comes to open source components is in understanding which dependencies are lying beneath the surface of the code. A single library can sit on top of dozens of dependencies, both direct and transitive. If your code is making calls to a functionality in one of these underlying dependencies that happens to have a vulnerability, then it can put your product at risk.

Package Managers

Often overlooked by many organizations but actively exploited by attackers, package managers are a powerful vector for the unintentional installation of malicious code. Package managers can abstract away many of the actions taken behind the scenes when installing third-party software. This includes the decision to use a local package from the organization's internal repository or an identically-named package from a public (and potentially malicious) source. This can lead to the automated and recursive import of dependencies, any of which could be compromised.

False Positives

Organizations will commonly fork open source projects, adding their own code to fit their requirements and backport patch any vulnerabilities that they identify. However, they will sometimes fail to alter the indicated package version. This makes it hard to understand whether the package is actually vulnerable. It is especially challenging if your vulnerability detection is based on the software package version alone.

A second issue in this category relates to whether or not you are dealing with effective vulnerabilities. Since we use a plethora of third-party components, which themselves are developed on top of other components, there is a high probability that more than a few real vulnerabilities may show up in your AST scans. However, just because you have vulnerabilities in your product does not mean that your product is at high risk of being exploited.

In many cases, a developer will write an API that only makes calls to specific functionalities within a third-party (most likely open source) component in order to access the features that they need. Those functionalities that receive calls are considered to be effective. All the rest are not given access to your product, essentially sitting there in the background as ineffective functionalities.

The upside here is that those ineffective functionalities do not pose a direct threat to your product. They can be easily mitigated or simply deprioritized for remediations. That said, they may still appear as vulnerabilities in your product, leading your team to spend time addressing them unnecessarily.

The "Shift Left" Attack

These attacks can elude defenders because they can be inserted into the SDLC before security measures like hashing (checksum), encryption, and many AST tools have a chance to detect them. As more security solutions continue to move further to the earlier stages of development, canny attackers will look for ways to stay a step ahead of the curve.

Who Targets the Supply Chain?

Successfully exploiting a software supply chain can be a lucrative win for an adversary, granting them an inside path to spread throughout multiple targets with a single breach.

The types of adversaries vary along the spectrum from the lower level malicious developers to the elite state-actor hacking teams and everyone in between.

694 organizations were affected by supply chain attacks in 2020.

(Source: ITRC)

Individual Actors

Compromising an open source project that might be used in a cryptocurrency wallet application or inserting a backdoor to steal PII data from an unsuspecting company can be short work for a talented individual hacker.

A successful attack for this level of actor might mean that they convinced the project owner of an open source project to give them commit rights, allowing them to inject malicious code that then makes its way into the product of a valuable target somewhere down stream. Perhaps an inside threat actor working as a developer inserts malicious code into an update server that goes out to a client. More often than not, many of these criminal acts revolve around crypto mining or malvertising to make them a few extra bucks.

Criminal Gangs

In the middle position, more organized criminal groups may look to target the supply chain in order to carry out a more substantive attack that will yield them a bigger payday.

By accessing the supply chain, a more resourced gang can penetrate their victims' defenses with trojan horse malware, infecting them with ransomware. Breaching an update server can also be an ideal tactic for larger scale activities like building botnets for activities like DDoS attacks or even spam networks.

State Actors

For government run hacking operations that are tasked with breaking into hardened targets like rival government networks or for industrial espionage, a supply chain attack may be highly appealing. Governments and large corporations that either have information that a state actor might want or can help them gain access to the rival government (think breaching Microsoft to reach other targets), can be difficult to breach if you face them head on.

The SolarWinds incident, which has widely been contributed to Russian state-run hackers, is a masterclass in how to successfully carry out a wide reaching supply chain attack that breaks open your adversary from the inside. Over the months that followed the initial news of the breach, more U.S. Government departments and companies that work with the Federal system were added to the list of those compromised.

With their achievements from this operation, they have likely inspired others to carry out similar actions against the supply chain in the coming months and years. Their tactics and methods developed over time by well resourced intelligence organizations eventually have a way of trickling down to lower level attackers who can leverage them for their criminal activities. Think Eternal Blue and the generations of ransomware attacks that followed and you get the picture.

How Attackers Target the Supply Chain?

In many ways, attacks on the software supply chain are like any other cyber attack in that they succeed in exploiting a vulnerability in a victim's systems.

Common Vulnerability Types

Zero Day Vulnerabilities – Supply chain software components, much like any other hardware or software component, include unknown vulnerabilities. These unknown vulnerabilities may lead to remote code execution, information leak, or denial of service, just to list a few of the potential consequences. They are especially concerning because there are no protections against them since they are, well, unknown.

Once discovered, these vulnerabilities can be weaponized into exploits that will be used by a malicious actor to attack further down the systems or networks that use these components.

Known Vulnerabilities – Once zero-day vulnerabilities are publicly disclosed or caught in use by a threat actor, they become known and are usually assigned a Common Vulnerabilities and Exposures (CVE) identification number.

As more technical details are publicly available on these vulnerabilities, they can be more easily exploited by different threat actors. They need only to develop an exploit or use one released by the infosec community, making for some strong ROI. It is for this reason that these vulnerabilities should be tracked and mitigated through patching, upgrades and/or configuration or code changes.

Unmaintained / Abandoned Packages – Using third-party software packages, whether open source or closed source, that are no longer supported by their maintainer poses significant risks.

Without the tender loving care of a maintainer community, these packages are unlikely to have their code refreshed and security issues remediated. From our experience in disclosing more than 400 vulnerabilities, this is a very common problem and many times consumers of software packages are left without any support and have to rely on custom patches or to replace their vulnerable packages with a different package altogether.

Backdoors – A backdoor is an intentional – and usually covert –– access method to a system that bypasses the standard access methods. There are many forms of backdoors and even more ways to implement them via code or configuration.

A common example seen in embedded devices for a configuration-based authentication-bypass backdoor in Linux-based systems is for the attacker to simply add a hidden private key. They can configure the SSH daemon to use this private key in addition to any keys that will later be configured by the legitimate user. This key will allow an attacker with the corresponding public key to later bypass the SSH daemon authentication and spawn a shell to the device or server.

A common code-based backdoor performs an authentication bypass on custom authentication mechanisms that are implemented in code. Suppose an Access Point (AP) has a custom web management interface that requires a user to login by supplying their username and password as credentials. The code that performs the authentication process hashes the provided password and compares it against hashes that are stored in a database. This code can be modified to check for specific hardcoded hashes and provide access if they and/or other inputs are received. It can do this regardless of the legitimate list of users defined by the real user or administrator of the AP. A different form of backdoor utilizes code auto-update techniques that allow it to try and download arbitrary code packages (shellcode, shared objects or DLLs) from a remote server. It will then load and execute them. This allows an attacker to execute code on the device without the need to try and access it directly through its open management interfaces, instead having the backdoor piggyback to the attacker.

Bugdoors – A specific type of backdoor or a specific type of vulnerability, a bugdoor is a software bug that was planted as a backdoor with malicious intent. This bug can later be exploited by whoever introduced it into the code or hardware. They can use it as a backdoor to change the behavior of a certain software or hardware component.

The obvious advantage of a bugdoor is plausible deniability. Everybody has bugs. It is very hard to prove whether security-related bugs were introduced by mistake or with malicious intent.

Compare this to obvious backdoors like hardcoded credentials or dropper techniques like we have seen with some recent supply chain attacks, and it is clear why a bugdoor would be a much smarter move for the threat actor.

Vectors of Attack

A defining feature of supply chain attacks is that they occur upstream of the intended target that the adversary is hoping to exploit. This specifically expresses itself via the points of ingress.

These include:

Vulnerable Open Source Components

Insufficient vetting of open source components is the most commonly cited scenario for supply chain exploitations.

Working securely with open source software means first and foremost determining if they contain vulnerabilities before integrating them into your products. Open source components, like all software, contain vulnerabilities because it is written by humans – for now.

One of the challenges that open source, and commercial software for that matter, must contend with in contrast to proprietary software, is that it is likely widely used in multiple products across different organizations. When a vulnerability is discovered and reported to the CVE database, it becomes visible for all.

In one sense, this is a good thing because it allows organizations to learn which third-party components are vulnerable and helps them to patch their products. The downside is that the bad actors can also view the same database and use this information to exploit their victims. As soon as a CVE is published, it becomes a race to see who can patch first before an attacker can discover that they are using the vulnerable component.

This is far from a fair fight.

As noted above, organizations are generally unaware of which third-party components they are using in their software. Commercial software intentionally hides the code from view while open source has layers upon layers of dependencies. Most developers do not track their open source usage, let alone research the direct or transitive dependencies.

More often than not, they research which open source project has the feature that they need, integrate it into their code, and move on to the next task. This lack of awareness can leave the organization to be easy pickings for attackers who know what to look for. It may also lead to a situation where a hacker may compromise one of those dependencies' repositories, slipping in vulnerabilities into components that developers are not even aware of that they are using to support their proprietary product.

Installation or Deployment of Hardware Devices and Appliances

Our connected devices can serve as a point of ingress for attackers if they are not properly secured.

End-users and asset managers often install or add their devices to the organization's network, allowing them easier access to make the most out of their devices. One of the most common examples is a printer.

The buyer may not even be fully aware of the full range of the network capabilities the device comes with. If such a connected device is vulnerable, and exposes its functionality too widely over the network, it can present an entry point into the organization.

Attackers commonly scan the public network for exposed and unsecured connected devices, and try to use them as a foothold to spread further.

Compromised Update Servers

Staying up to date with patches and updates is at the top of the best practices list.

Automating the process through update servers is probably the most effective way to ensure that you are working with the latest, most secure version of your vendor's software.
Unfortunately, in some rare cases, software updates can be used to carry malicious code into the organization.

Attackers who compromise an update server can plant their payload into update packages, preferably before they are cryptographically signed so they can have valid signatures, and have that payload delivered inside the target organization. This was the scenario in the SUNBURST attack which used SolarWinds' Orion update servers to gain access to their targets.

White-Label Devices

This is common practice in the embedded industry. Hardware components and entire devices get re-packaged, rebranded and resold under the name of the integrator or distributor. Often without attribution to the source vendor.

The software inside these devices can be proprietary or open source, but any vulnerabilities introduced into these devices by the vendor spread reliably to the rest of the supply chain.

Vendors need to remember that their customers do not care who wrote the vulnerability. If they buy it from you, then they will hold you responsible for its security.

Mitigating Your Supply Chain Risk

- ✓ Establish Supply Chain Vetting Processes
- ✓ Implement Binary Code Integrity Verification
- ✓ Require and Analyze SBOM
- ✓ Perform Network-based Risk Assessments
- ✓ Prioritize Remediations with Contextual Threat Analysis
- ✓ Remediate Continuously
- ✓ Enforce MFA (multi-factor-authentication) on Critical Sections of the Software Development Lifecycle (SDLC)
- ✓ Avoid Dependency Confusion/ Namesquatting and Typosquatting Attacks Against Developers
- ✓ Use Version Pinning
- ✓ Disable Execution on Install

Given the potential for a supply chain attack to undermine your product's security, there are steps that you can take to reduce your risk of compromise and exploitation.
We have compiled a list of nice best practices that should form the foundation of every security stakeholder's supply chain security strategy.

Establish Supply Chain Vetting Processes

Start with making supply chain risk a core element of your organization's overall security process.
This means having conversations with your vendors to ensure that they are upholding the same standards that you expect from your own organization. Inquire about their practices and track record on working securely. Ask them what they are doing to improve their security processes and work together to bring them into your process. Share your requirements with your partners and resolve incidents together. Supply chain security management needs to be an ongoing conversation with frequent check ins to ensure that everyone is keeping up with their obligations.

Implement Binary Code Integrity Verification

Verify downloaded software by hashing the binary file and comparing it to the hash provided by the vendor.
This method can be highly effective against Man in the Middle (MITM) attacks and some supply chain attacks.
Note that some package managers, such as npm (for Node.js), NuGet (for .NET) and pip (for Python) allow for automatic integrity verification of downloaded packages prior to their installation.

Require and Analyze SBOM

Identifying supply chain risks requires a high degree of transparency into the security state of the acquired software. Identifying known or potential vulnerabilities requires compiling a Software Bill of Materials (SBOM).
This is only possible with full cooperation from the vendor who provides the SBOM, or with the use of automated scanning tools which compile the SBOM from a binary image or artifact provided by the vendor.

Perform Network-based Risk Assessments

Network scanning can be used in some situations to detect vulnerable software components by performing network fingerprinting and detecting the existence of the vulnerable code.

Some commercial tools offer these capabilities, but most of the time, FOSS tools are released by the InfoSec community or the researchers who discover the vulnerabilities to detect such specific, high-profile and network-facing vulnerabilities.

Alas, it is often hard to accurately detect the existence of the vulnerabilities or vulnerable versions through network probing alone. Network monitoring can also help detect actual break-ins when they take place. For example, the SolarWinds SUNBURST attack was caught by detecting the malicious behavior following the initial breach.

Prioritize Remediations with Contextual Threat Analysis

Researchers are constantly uncovering and reporting on new vulnerabilities, attacks and CVEs that can impact your products. It is thus critical to be able to focus on the actual threats that matter for a specific environment and its specific conditions.

Use vulnerability management and Software Composition Analysis solutions that are able to continuously monitor new threats and prioritize by analyzing vulnerabilities within the context of their environment. Regardless of whether it is a device, server, network or in the cloud.

This will help in reducing false positives and allow your teams to focus on the most pressing of threats.

Remediate Continuously

Remediation against vulnerable software needs to be applied regularly. Where possible, software should be upgraded to new and fixed versions in line with best practices.

Where that is not feasible, software should be patched; failing that, it's possible to mitigate some vulnerabilities by isolating the vulnerable software, either by operating system security mechanisms such as restrictive containers, or by external services such as firewalls, which can stop some malicious traffic. Finally, there are cases where affected software and/or devices should be replaced with more secure counterparts when necessary as keeping the old, vulnerable ones become untenable.

Enforce MFA (Multi-Factor Authentication) on Critical Sections of the Software Development Lifecycle (SDLC)

All of the components of the development pipeline that require or support authentication should be configured to use MFA. This could prevent attacks where attackers are able to acquire username and password credentials, whether hardcoded in scripts or from a targeted spear-phishing attack on a developer.

While each pipeline has its own characteristics and components and deserves a dedicated threat assessment, some elements of your developer ecosystem are musts to run MFA on.

Start with these at the top of your list and build out from there.

- Source code version control systems (e.g., Gitlab or Github)
- Package registries (e.g. npmjs)
- Container registries (e.g., DockerHub, AWS ECR, Azure Container Registry)
- Artifact repositories (e.g., Artifactory or Azure Artifacts)
- CI/CD and pipelines (e.g., Jenkins)
- Build and developer machines
- Secure internal dependencies

Avoid Dependency Confusion/Namesquatting and Typosquatting Attacks Against Developers

Reduce the chances that your dev team might get hoodwinked with one of these pesky yet risky attacks by setting a default registry. Organizations that have an internal, private registry should configure the projects in their DevOps infrastructure to use their registry by default. There are different configurations depending on the language and package manager that you are using.

- **npm** – Use the scoped packages feature to include private packages under the organization name. Scoped packages can be categorized as private and updated only by the user or organization associated with that scope.
- **Python** – Consider using pipproxy, an open source solution by Vdoo that prevents namesquatting attacks on pip (the Python package manager) packages. It is also possible to use the index-url option with pip to override the default package registry in favor of a private one.
- **Java** – When using Gradle, there is not a default registry. However, Maven does pose a risk which is possible to mitigate by forcing Maven to use an internal private registry by setting the <mirrorOf/> section in the settings.xml configuration file to "*".
- **Golang** – Skip this one as packages names contain the full URL of the package registry source.
- **.NET**– Adding an internal package registry to the Nuget package manager configuration is done under the <packageSources/> section in the XML-based nuget.config file. It is also possible to disable the public official registry by adding it under the <disabledPackageSources/> section.

Use Version Pinning

It is prudent to be as precise as possible when specifying packages to be installed in terms of specific version to avoid downgrade attacks.

Disable Execution on Install

With some package managers it is possible to prevent any source distributions from being installed (pip's --only-binary flag) or to disable arbitrary install commands (npm's --ignore-scripts flag). It will deny execution during installation time.

This might not prevent code execution if the package is deployed to production and then run. It will however prevent immediate code execution on the developer machine, which we have seen for example in the Birsan attack mentioned above.

It is also possible that the malicious package will not be deployed to production as it will fail tests as it does not necessarily perform all of the behavior of the internal package that it was masquerading as.

The Next Step in the Chain

In 2017, NotPetya attack hit Ukrainian targets through a mandated accounting program, knocking not only Ukraine offline but spreading rapidly across the globe and causing billions of dollars in destruction.

In the years since, security professionals have been waiting for the next big supply chain attack to emerge. The attack on SolarWinds has highlighted how vulnerable major entities like the U.S. Government and large corporations can be when the software they consume from external vendors is not properly secured.

The good news is that the damage from SUNBURST appears to have been blessedly narrow in scope.

That said, nobody wants to be the next SolarWinds or one of the thousands of organizations that found themselves compromised or even breached in the event.

Unfortunately, there will most certainly be a next time. And it may come sooner than we think.

In the months to come, we can likely expect to see a rise in attempts to carry out these sorts of attacks. Having proved its effectiveness, more actors will try their hand at compromising valuable supply chains.

The proliferation of embedded devices has expanded the threat surface, leading to a potential jump in attacks targeting hardware. While these sorts of attacks against hardware are more difficult to execute, they are also quite challenging to detect and the payoffs could be significantly enticing.

Given the challenges, where should security professionals be looking in order to better protect their products?

One area that is showing considerable promise is the community-driven CycloneDX SBOM format standards. It is believed that it will soon simplify the process of producing and consuming digital Bills of Materials between organizations, thereby improving the communication and coordination between entities.

Following this path, we believe that the next step will be to move towards embedding this digital SBOM in the (signed) binary itself so it will be easy to consume it as metadata. This will increase transparency and will make this complex, distributed process much simpler. However, in our opinion, for this vision to materialize, much effort would be needed from all of the involved parties in the supply chain cycle.

As organizations of every size and kind are involved in the supply chain, suppliers are very often easier to compromise than the actual target. As long as that holds true, supply chains are going to remain a major source of risk, and a dangerous vector for the attackers.

For more information on preventing supply chain attacks in your organization download the Vdoo solution overview

About Vdoo

Vdoo is a global leader in the complex and increasingly-critical product security space. With Vdoo, organizations can identify, prioritize, and mitigate a vast range of security issues. As the only automated platform that provides end-to-end product security, Vdoo helps development and security teams reduce time and effort while ensuring optimal product security. The platform addresses a diverse variety of security risks including supply chain threats, configuration risks, standard compliance, zero-day vulnerabilities, and more. Founded in 2017 by a team of seasoned cybersecurity entrepreneurs and product security experts, Vdoo is now a global company with offices in Israel, US, Germany, Singapore, Japan, and dozens of Fortune 500 customers representing the most security-diligent companies from various industries.

For additional information, please contact us at info@vdoo.com or visit vdoo.com