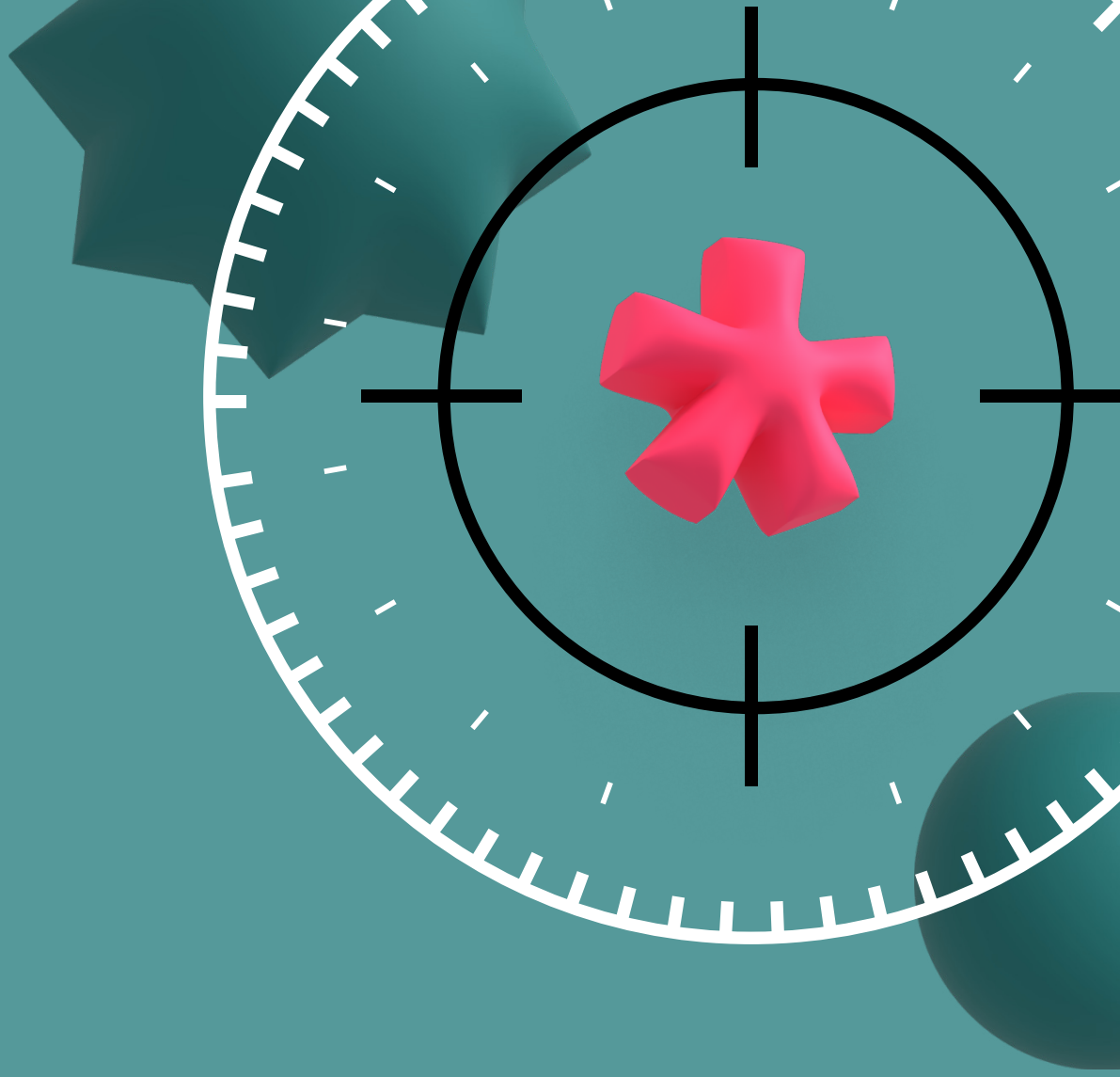
MYRROR



Demystifying the Software Supply Chain's Known & Unknown Risks

A Myrror Security Cheat Sheet

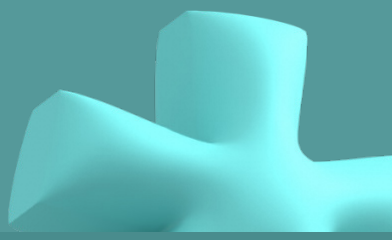


Table of Contents

Introduction

The software you rely on is built on a complex network of connected components, each a potential weak link. This "software supply chain" presents a dynamic challenge: a constantly evolving web of known and unknown security vulnerabilities. Ignoring these risks is a gamble. This cheat sheet is intended to empower application security personnel to:

- Understand the concrete threats lurking within the software supply chain
- Grasp the critical importance of addressing unknown risks alongside the documented ones
- Discover actionable strategies to fortify applications against both known and unknown risks

Before delving into these a bit deeper, let’s explore a few of the major supply chain attacks that have occurred in the past.

Major Supply Chain Attacks

Revenue Loss:

NotPetya
2017

$10 billion

Estimated damages exceeding $10 billion, targeting critical infrastructure and causing widespread data loss and system disruptions.

SolarWinds
2020

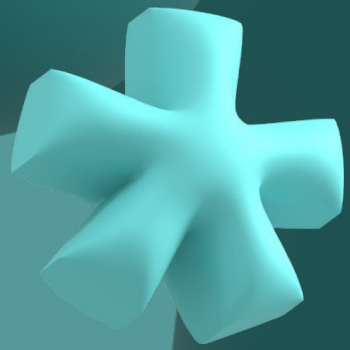
$1 billion

Widespread supply chain attack compromising government agencies and private companies, incurring damage estimated at $1 billion.

Kaseya VSA
2021

up to $1 billion

Ransomware attack affecting thousands of businesses globally, leading to estimated losses of $500 million to $1 billion.



Reputation Loss:

Equifax 2017

147 million americans

Massive data breach exposing the personal information of 147 million Americans, severely damaging brand reputation and trust.

Marriott International 2018

500 million guests

Estimated damages exceeding $10 billion, targeting critical infrastructure and causing widespread data loss and system disruptions.

Yahoo 2014

3 billion accounts

Massive data breach compromising 3 billion accounts, eroding user trust and leading to a steep decline in market value.

Target 2013

40 million credit cards

Data breach of 40 million credit and debit cards, triggering investigations and damaging customer confidence.

Recent Additions (2023):

- **Codecov attack:**
  Potential for widespread downstream impact on organizations relying on affected dependencies.
- **UCSF research software attack:**
  Breaching trust in the integrity of scientific research and data.
- **Airbus supplier attack:**
  Highlighting the importance of securing even indirect links in the supply chain.

Sophisticated Malicious Code Attacks:

Malicious code attacks sneak in through trusted third-party tools, like hidden weapons within seemingly harmless packages.

This approach makes them incredibly difficult to detect, and — once triggered - the damage can be significant, impacting not just your organization but all those relying on your software. A few examples of such attacks:

- SolarWinds
- Kaseya
- Codecov
- 3CX
- Ledger dApp
- CyberLink

The Software Supply Chain Security Landscape

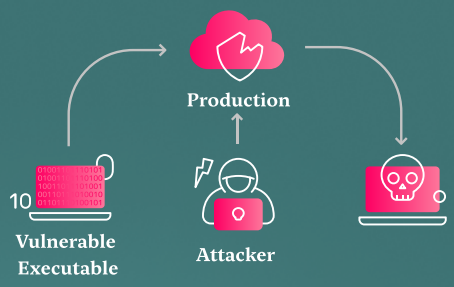
The Software Supply Chain Security Threat Landscape encompasses the various risks and threats that pose a danger to the integrity and security of your third-party code or your own CI/CD process.

To assemble it, one must identify and understand the potential vulnerabilities and attacks that can compromise software integrity throughout every step of the chain. These threats can include attacks - where malicious actors inject malware and tamper open source packages which are later added into the supply chain, as well as vulnerabilities in third-party dependencies.

Before moving on, let's focus on distinguishing the difference between two important terms:

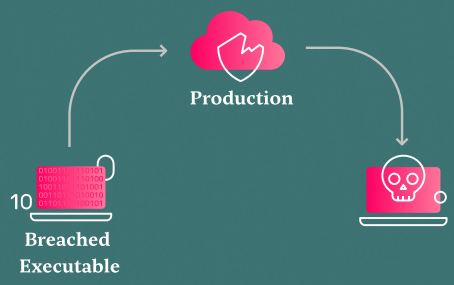
The Difference Between Vulnerabilities & Supply Chain Attacks

Vulnerability Exploit



A vulnerability is (with a few edge cases) a non-deliberate mistake, which is identified by a CVE and recorded in public databases. It's usually possible to defend against it before it has been exploited.

Malicious Code Attack



An attack is a deliberate malicious activity that lacks specific CVE identification number, and this is untracked by standard SCAs and public DBs. Once discovered, an attempt to exploit it has usually already taken place.

The Difference Between Vulnerabilities & Supply Chain Attacks

To paint things in a little more picturesque way, comparing supply chain attacks to cunning thieves and vulnerabilities to broken locks feels apt.

Attacks infiltrate seemingly trustworthy third-party software components and the pipelines that get your software where it needs to go and allow for malicious activity. This makes detection extremely difficult as the code is hidden and disguised. When triggered, the attack can harm not only your organization but potentially your customers as well as any other entities using the same components. To prevent such attacks, strong vigilance and thorough analysis of third-party code and binaries are essential.

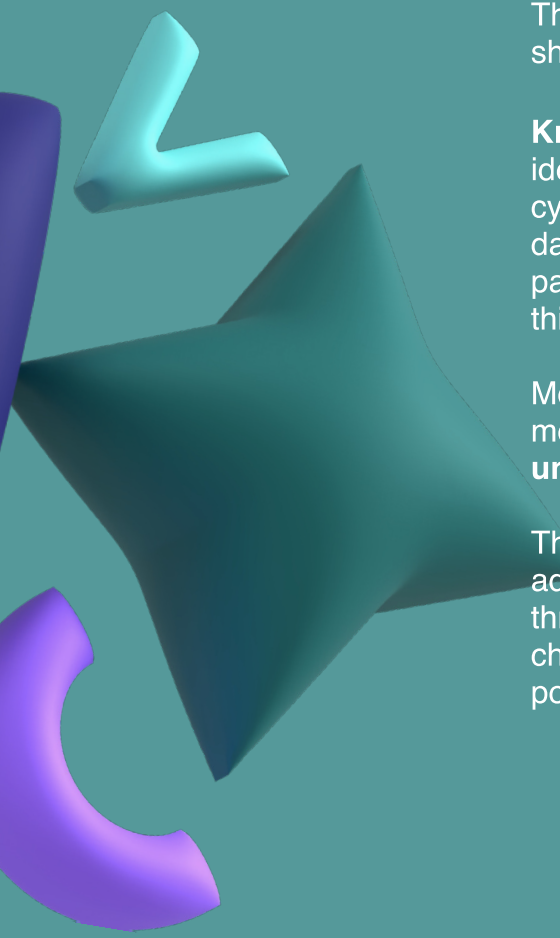
Vulnerability exploits, on the other hand, target weaknesses present in your own systems, packages or your own code. Some are known vulnerabilities, like cracks in your defenses, while others might be zero-days or too complex to exploit readily.

The sheer number of vulnerabilities can overwhelm security teams, leaving attackers plenty of potential entry points if patching efforts are not done properly. This phenomenon, often referred to as "**alert fatigue**", can cause severe oversights and result in major vulnerabilities remaining unpatched for long periods of time.

Organizations need to be aware of this threat landscape and implement proactive measures such as binary-to-source analysis (read more about it in the "Security Best Practices" section of this guide), supply chain verification, continuous monitoring, and threat intelligence sharing to mitigate these risks and protect their software supply chain from potential security breaches.

By understanding the risks and implementing robust security practices, we can build a more resilient software supply chain and protect our valuable digital assets.

Understanding the Known and Unknown Risks in the Supply Chain



This guide distinguishes between known and unknown risks, shedding light on their unique characteristics and implications.

Known risks are those vulnerabilities and threats that have been identified, documented, and are well-understood by the cybersecurity community. While these known risks can be daunting, they can also be effectively addressed through regular package version updates, vulnerability management, and smarter third-party software choices.

More importantly, though, this guide chooses to shine a light on the more challenging (and less discussed) side of things - the **unknown risks** inherent to the software supply chain.

These are the risks that are harder to discover and harder to address; They can originate from various sources, such as insider threats, malicious actors injecting malicious code into the supply chain or a host of techniques used to compromise the distribution portion of your software's delivery journey.

We would like to stress again that unknown risks pose a significant challenge due to their elusive nature, making them harder to identify and mitigate effectively.

Before we dive deeper into what they look like in practice, though, let's take a look at the known risks and touch on how to mitigate them. This will help illustrate the difference between the two, and make it easier to discuss the unknown risks.

Handling Known Risks with SCA Tools

Software composition analysis (SCA) tools act as security scanners for your software stack, hunting down vulnerabilities lurking in your third-party code.
They usually work in two steps:

1 Dependency Mapping

SCA tools scan your code to identify all the third-party libraries and open-source components used. This comprehensive map reveals the hidden connections and potential risk points in your software. This is usually done by scanning the manifest or BOM - Bill of Materials of your software.

2 Vulnerability Database Lookup

Armed with the dependency map, SCA tools cross-reference each component against known vulnerability databases. They instantly flag any vulnerable library, alerting you to take action.

While SCA helps detect known threats, complete mitigation requires additional steps:

3 Continuous Monitoring

Stay ahead of newly discovered vulnerabilities by setting up ongoing scans and alerts to address emerging threats promptly.

4 Updating Vulnerable Dependencies

Patching outdated libraries with fixes or switching to secure alternatives is crucial to eliminate vulnerabilities.

5 Planning remediation

Just updating a library might cause unexpected effects - all patching should be planned and take into consideration all the potential side effects the update might have.

By actively using SCA and implementing these combined strategies, you can effectively address the known threats lurking within your software supply chain.

4 Reasons Why SCA Tools Aren’t Enough

Although SCA tools are needed, the lack in a few different ways:

* **SCAs Unnecessarily Focus On Every Vulnerability**

  Not every vulnerability is exploitable and important. This causes the aforementioned phenomenon of “alert fatigue”, where security personnel are bombarded with alerts without being able to properly address all of them.

* **By Extension, SCAs Do Not Automatically Prioritize Vulnerabilities**

  It might seem as if they do at first glance, since every vulnerability has a score, but that is just the “raw” score given to it by the vulnerability databases or the governing body - it has no correlation with its exploitability, reachability or business impact.

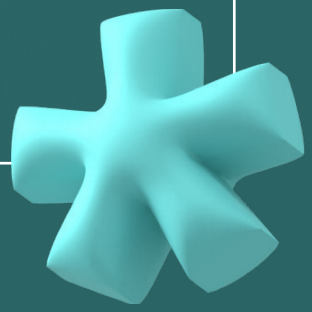
* **SCAs Don’t Detect Unknown Risks**

  As mentioned above, SCA tools cross-reference each component against known vulnerability databases. Hence, unknown risks remain unknown and pose a greater danger to organizations.

* **SCAs Offer Problematic Remediation Advice**

  Often, SCA tools provide remediation advice that is not targeted, and also doesn't take into account the vulnerabilities that might be introduced by the remediation effort.

Now that we’ve explored the ways known risks can be mitigated, let’s jump back into the unknown risks side of the equation.



Unknown Risks and Emerging Threats

Generally speaking, we'll refer to an unknown risk as something that traditional SCA tactics will not capture. This is mostly due to the fact that these attacks are not clearly reported on in major vulnerability databases - since they're not vulnerabilities - but also, and more importantly, because they are harder to detect and identify than traditional vulnerabilities due to their heterogeneous nature.

These attacks take a few different forms, which can be roughly categorized as follows:

Typosquatting

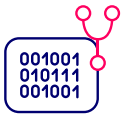
A malicious package masquerading to be another package by "squatting" its namespace, often using similar words in its name or typos. A few examples:

- **Symfony -** The popular PHP framework symfony had a typosquatting attack performed against the symfony/process package - it was replaced with symfont/process.
- **Dozens of packages at once in PyPi -** Over 450 malicious PyPI python packages were found installing malicious browser extensions to hijack cryptocurrency transactions made through browser-based crypto wallets and websites.

Dependency Confusion

A package masquerading to be another package by being uploaded, with the same name, to a different distributor (relying on people downloading it by mistake). A few examples:

- **PyTorch -** A malicious dependency package (torchtriton) that was uploaded to the Python Package Index (PyPI) code repository with the same package name as the one PyTorch ships on the PyTorch nightly package index. Since the PyPI index takes precedence, this malicious package was being installed instead of the version from the official repository.
- **Apple, Microsoft & More -** The attacker publishes malicious packages to public code repositories, mimicking names of internal packages used in corporate software. Unsuspecting developers may download these malicious packages, thinking they are legitimate internal packages. This can lead to code execution within a company's internal software systems.



Malicious Code In the Repo

An attacker modifies the popular dependency to contain malicious activities as part of the normal operation of the software. A few examples:

- **node-ipc Protestware** — node-ipc is a popular package to help with inter-process communication in Node. In protest of Russia's invasion of Ukraine, the author of the package intentionally added malware on March 16th that targets Russian and Belarusian IPs.The code attempts to geo-locate where it's running, and if it discovers it is running with in Russia or Belarus, then it attempts to replace the contents of every file on the system with a unicode heart character: ❤️. In a more recent version, it instead just drops a file with a peace message on the desktop.Vue.js and many other projects are affected.
- **Discord.dll** — The discord.dll is an npm component which conducts sinister activities that are hard to spot upfront. It also uses the legitimate Discord.js npm dependency to potentially distract researchers from its otherwise nefarious activities, which might cause it to appear like a legitimate Discord API package. It is not.

CI/CD Attacks

Compromising the build process of specific pieces of software in order to affect any downstream user of that software. A few examples:

- **SolarWinds** - The attackers managed to modify a SolarWinds plug-in called SolarWinds.Orion.Core.BusinessLayer.dll that is distributed as part of Orion platform updates. The trojanized component is digitally signed and contains a backdoor that communicates with third-party servers controlled by the attackers. FireEye tracks this component as SUNBURST and has released open-source detection rules for it on GitHub.
- **3CX -** The attackers compromised a trading software's binary file, an employee downloaded it and then the attackers moved laterally through the network.

Distribution Server Attacks

An attack on the server in charge of delivering a piece of software to its customers. A key example:

- **Jumpcloud -** The threat actor compromised the cloud provider's commands framework and used it for malicious data injection.

Supply Chain Security Best Practices Cheat Sheet

Use an SCA Platform

By now you should understand why a software composition analysis is not only a suggested - but required - portion of your security stack, and how it can be used to automatically identify and track open-source components and their dependencies. You also should understand the main disadvantages of traditional SCAs, and what you should look for when acquiring such a platform.

Use Automated Prioritization

Using a tool dedicated to automating alert prioritization, ensuring the AppSec team knows what to deal with first, makes sure you are working on the right thing at any given point in time and remediating the most risky issues first. Great SCAs, by the way, often take care of this part of the equation for you.

Rely on Package Attestations

Which will provide assertions about open source software development practices at the package level, ensuring that the package is well-developed (see NIST's TACOS as an example framework). While attestations are great for making sure your vendors are truthful with you, they are only a verification system for something your provider says is true - not a bulletproof solution to package tampering.

Rely on Packages that Provide Provenance Data

Which lets users ensure that the package comes from the correct, reputable source (see GitHub's support for npmjs registry provenance here, based on SLSA, as an example). This approach, like attestations, is a great way to learn that the software came from where it said it did - but still does not guarantee that the original source wasn't somehow compromised as well.

Perform Software Integrity Validation Based on Binary-to-Source Analysis

Binary-to-Source code analysis refers to the practice of comparing the source code and the binary artifacts of a piece of code, looking for any deviation between the two. This process helps reveal trojanized packages and malicious code insertions “hidden in plain sight”, making sure the integrity of the open-source packages you use has been maintained.

The advantages of the binary-to-source analysis are numerous:

- Earlier Detection: This analysis acts as an early warning system, flagging potential threats before they reach production, saving valuable time and resources.
- Complete Code Visibility: Analyzing binaries sheds light on the inner workings of compiled code and exposes malicious code attacks that might evade detection.
- Upstream Assurance: It verifies package integrity throughout the supply chain, building trust in the packages you use.
- CI/CD Assurance: Running that same analysis on your own pipelines, can ensure your build process integrity.

By integrating binary-to-source analysis into your security toolkit, you gain a powerful ally in the fight against sophisticated software supply chain attacks, ensuring the resilience and trustworthiness of your code.

Actively Manage Vendors and Third-Party Risk

While ensuring your developers do not use compromised packages is one step of the way, you still need to carefully vet every software supplier before letting them into your secure environments.

This also includes performing a diligent **Dependency Management** process, Maintain a detailed inventory of third-party dependencies and libraries used in your software. Regularly update them to include security patches and monitor for known vulnerabilities.

Use Traditional DAST Tools

This will ensure - during the CI/CD process usually - that your web application’s endpoints can “handle” various attack use cases (normally enumerating OWASP and other frameworks’ cases on every single exposed web endpoint). By doing so, you’re making sure that one of the most intuitive attack vectors is covered against the majority of known attack methods, strengthening your posture significantly.

Use Traditional Open-Source SAST Scanning Tools

Traditionally, these tools are not intended to scan your 3rd-party code and will be tweaked for your own codebase. Looking for a platform that also scans 3rd-party code is a great idea, and will enable you to check if the software you use by extension (read: your 3rd-party code) also complies with the strict security standards enforced on your own, 1st-party code.

We must note however that this approach still creates a lot of noise and false positives due to the nature of the scan, which is often calibrated to detect problematic coding practices in your own code, which might not apply to 3rd-party code.

Plan for Incident Response

Incident response planning prepares you for the worst-case scenario, outlining clear steps for containment, investigation, and recovery in case of a security breach. By having a playbook ready, you can minimize damage and bounce back attacks.

Secure your Build Environments

Ensure that your build environments are secure, stateless and isolated. Only trusted personnel should have access, and regularly scan for malware and vulnerabilities in your build tools and infrastructure.

Sign your Artifacts

Sign your software executables and updates to verify their authenticity. This prevents tampering with the artifact during transit.

Focus on Container Security

If using containers, ensure container images are scanned for vulnerabilities as well and follow best practices for securing container runtimes. In addition, make sure to treat your container images just like the rest of your third-party code - just because it’s packed into a container, doesn’t make it risk-free.

Create Secure Packaging Procedures

When packaging your software for distribution, use trusted and verified package managers and repositories. Ensure packages are signed and verified before deployment.

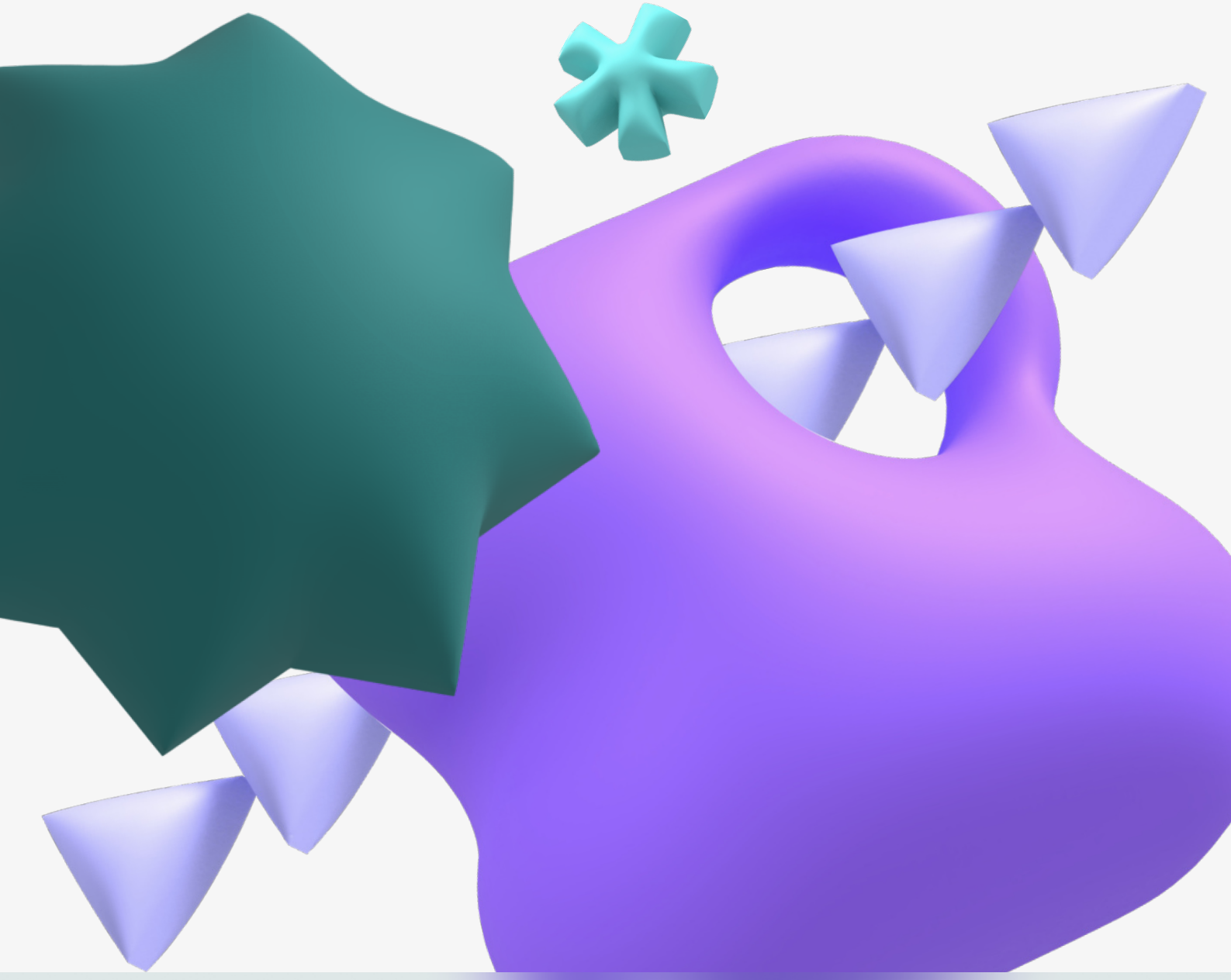
Use Continuous Monitoring

Point in time is not enough. Continuously monitor your software supply chain for anomalies, unauthorized access, and suspicious activities.

By addressing the known risks effectively and only looking at the real critical vulnerabilities, you achieve a higher pace of remediation and streamline the workflow between your engineering and security teams, eventually decreasing technological debt.

In addition, while “freeing up the CPU” on the known risks side of things, you end up getting more time to properly deal with the unknown risks - which, as a security professional, allows you to prevent more security risks and do your job better.

By implementing these best practices, you can build a resilient software supply chain security practice, one that's resistant to known and unknown threats, safeguarding your precious digital assets and protecting your users from harm.



Conclusion

The ever-shifting realm of the software supply chain demands both vigilance and strategic action. We've explored the landscape of known risks as well as delved into the murky waters of the unknown risks in your supply chain security.

By embracing robust security practices like binary-to-source analysis, third-party risk management and incident response planning, we can ensure that our software remains safe and secure - from all angles.

Remember, security is not a one-time endeavor, but an ongoing journey. By consistently prioritizing proactive defense and embracing the wisdom of "security first," we can build a future where software development is synonymous with trust, resilience, and unwavering protection.