Software Supply Chain Security

Understanding Software Supply Chain Attacks

White Paper

xygeni.

Table of Content

Introduction

Software supply chain attacks are becoming increasingly prevalent and devastating, with Gartner predicting that 45% of all businesses will experience a breach by 2025. Cybersecurity Ventures further underscores the gravity of this threat, projecting a staggering $138 billion in annual damages caused by software supply chain attacks by 2031. These alarming forecasts highlight the urgent need for organizations to prioritize software supply chain security and implement robust measures to protect their sensitive data, operations, and reputations. The rise of third-party components, accelerated software development cycles, complex supply chains, lack of visibility, evolving attack techniques, SaaS adoption, and limited resources are all contributing factors driving the surge in software supply chain attacks. Organizations need to adopt a comprehensive and proactive approach to address these challenges and safeguard their software supply chains.

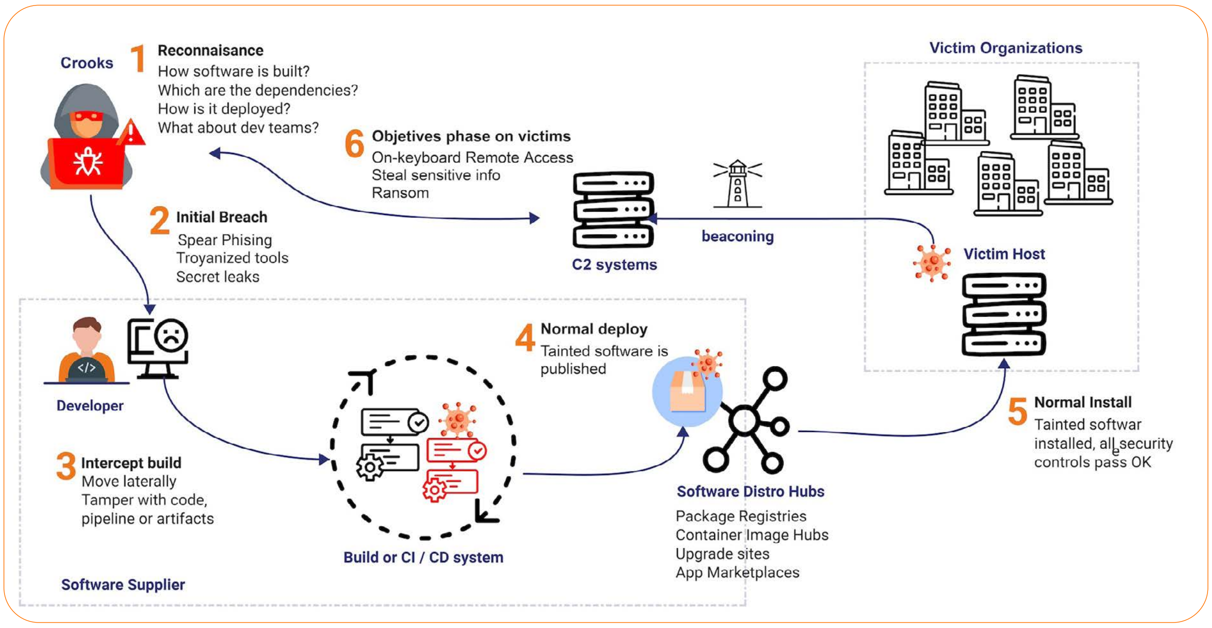
$138bn
annual damages caused by software supply chain attacks by 2031

45%
of all businesses will experience a breach by 2025

What is a Software Supply Chain Attack?

ENISA defines SSCA as "a compromise of a particular asset, e.g. a software provider's infrastructure and commercial software, with the aim to indirectly damage a certain target or targets, e.g. the software provider's clients." In other words, a Software Supply Chain Attack refers to a malicious activity targeting the software supply chain, aiming to compromise and introduce vulnerabilities or malicious elements into the software development and distribution process. This attack capitalizes on the interconnected and often complex network of processes, tools, and entities involved in creating and delivering software.

Anatomy of a SSCS Attack



The academic cyber threat intelligence and infosec literature had segmented software supply chain attacks into distinct categories for a more comprehensive understanding. We provide an introduction to these concepts based on the MITRE Attack Pattern Catalog. This catalog - describes supply chain attack patterns to facilitate analysis using various sources, including the adversarial threats compiled by NIST.

Attack Act: The What.

An action that causes a malicious payload or intention to be delivered to or directed at a system to adversely affect that system

- **Example 1:** Malware is inserted into system software during the build process
- **Example 2:** System requirements or design documents are maliciously altered.

Attack vector: The How

The route or method used by an adversary to exploit system design vulnerabilities or process weaknesses to cause adverse consequences. (Attack vectors are how adversaries can access attack surfaces, which can be thought of as reachable and exploitable vulnerabilities). The term Tactics, Techniques and Procedures (TTPs) was coined for describing the adversarial behavior at different levels of detail.

- **Example 1:** An adversary with access to software development tools and processes during the software integration and build process
- **Example 2:** An adversary gains unauthorized access to system technical documentation

Attack Origin: The Who

The source of an attack. Information to identify the adversary's role, status, and/or relationship to the system development and acquisition (e.g. inside or outside the acquiring organization and/or supply chain, type of job performed, etc.).
Types of adversaries include cyber criminals, hack-tivists, state-sponsored threat groups (aka APTs, Advanced Persistent Threats), and malicious insiders.

Attack Goal: The Why

The adversary's reason for the attack. Typical objectives include ransom, espionage, sabotage, intellectual property theft, abuse of resources or plain theft. More than one may apply.

Attack Impact: The Consequences

What the attack accomplishes, its impact on the affected organizations. Common consequences are financial loss due to system downtime, lost revenue, extra costs from abused cloud resources, reputational damage, loss of customers or partners.

Taxonomy of a SSCS Attacks

Numerous types of software supply chain attacks (SSCAs) exist. The "Software Supply Chain Attacks: An Illustrated Typological Review" delineates two prominent frameworks for classifying and analyzing SSCAs.

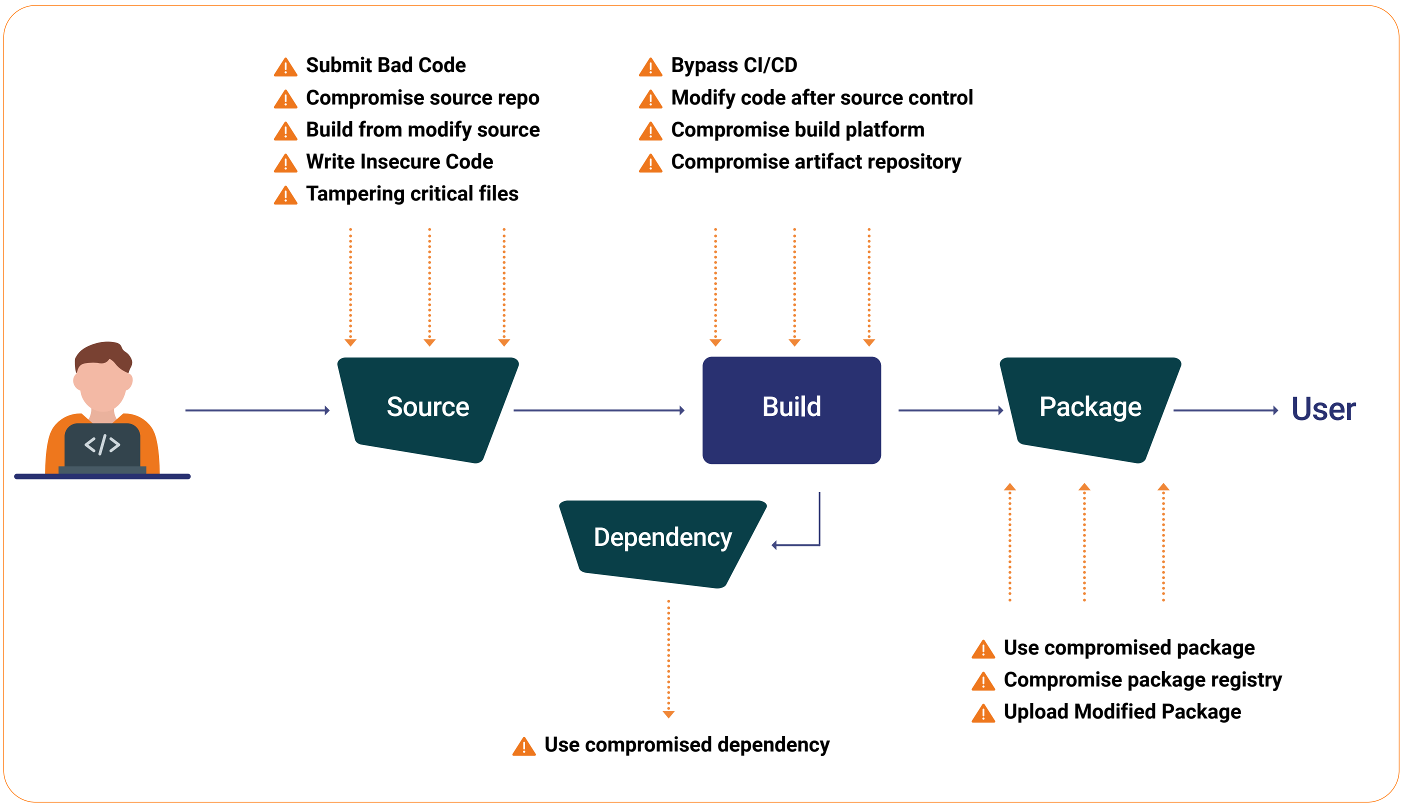
The first type of frameworks, analogous to **attack technique catalogs,** provides a comprehensive assessment of TTPs employed in SSCAs. Examples of these types of frameworks are the MITRE ATT&CK framework, the Common Attack Pattern Enumeration and Classification (CAPEC) or the ENISA framework. A similar catalog for generic cybersecurity countermeasures is the D3FEND Ontology vulnerabilities from MITRE.

The second category of frameworks related to Software Supply Chain Attacks (SSCAs) focuses on the threat/vulnerability landscape encompassing the entire software supply chain perimeter. These frameworks highlight the diverse risks of compromise at each lifecycle stage, emphasizing the interconnectedness of the supply chain and the potential for compromise to propagate across different stages. Examples such as The SLSA framework, The US National Institute of Standards and Technology (NIST), or the Cybersecurity and Infrastructure Security Agency (CISA) discern between different stages.

In this white paper, we will focus on the most common software supply chain security threats at the different stages of the software supply chain lifecycle, with a particular focus on the second type of framework mentioned above.

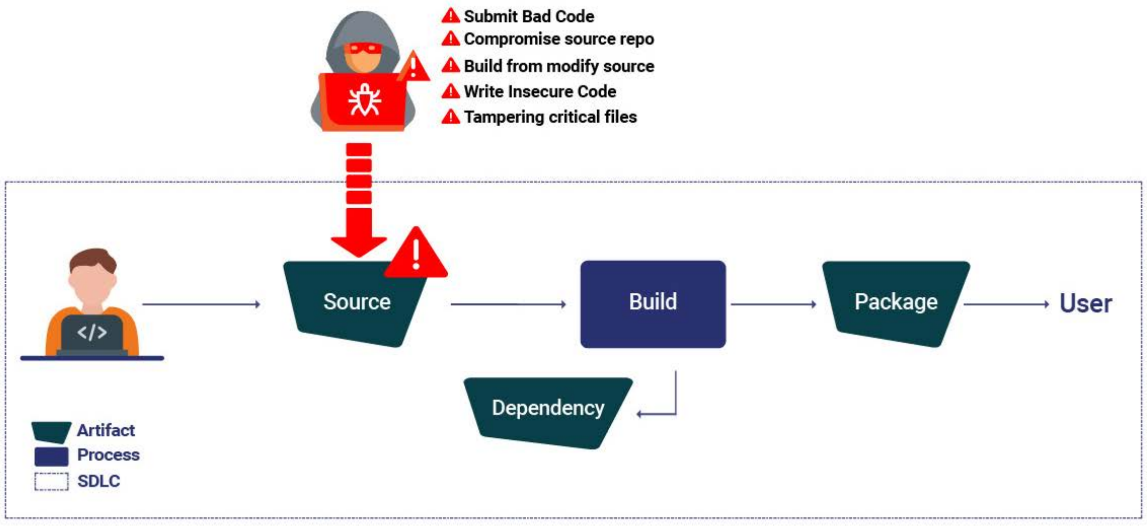
We distinguish between the following phases of the software lifecycle: Source, build, package, and dependency threats.

Core Strategies for Protecting Our Software Factory



Software Supply Chain Security Threats in the Source Stage

The source stage of the software supply chain lifecycle encompasses the initial phases of software development, from ideation to the creation of source code. This stage involves the selection of tools, libraries, and components, as well as the development and implementation of the software's core functionalities. Software supply chain security threats in the Source Stage refer to security vulnerabilities that can be exploited to introduce unauthorized or malicious changes to the source code. This includes the threat of both unauthorized individuals and authorized individuals introducing unauthorized changes.



Examples of Source Threats

1. Submit bad code

Submitting bad code refers to the practice of committing code to a source repository that contains defects, errors, or vulnerabilities. This can range from malicious code intentionally introduced to compromise the integrity or security of the software, to unintentional code that introduces bugs or vulnerabilities due to poor coding practices or lack of testing. An example of this vector attack was the NPM attack. In 2022, a hacker infiltrated the source code repository of a popular open-source software library called npm. The hacker inserted malicious code into the library's code that allowed them to gain unauthorized access to the systems of organizations that installed the library. The malicious code allowed the hacker to steal data from the affected systems, install malware, and disrupt operations. The attack affected a wide range of organizations, including government agencies, businesses, and individuals.

Software Supply Chain Security Threats in the Source Stage | Examples of Source Threats

2. Build from a modified source

An adversary obtains a copy of the source code from a source other than the official source code repository and uses it to build and deploy the software. This modified source code may contain malicious code, backdoors, or other harmful alterations that can compromise the integrity, functionality, or security of the software. An example of this vector attack was the Webmin Attack. An attacker gained unauthorized access to Webmin's build infrastructure, which is responsible for compiling and packaging the Webmin software. The attacker modified the build infrastructure to use source files that were not present in the official Webmin source repository.

3. Compromise source repo

An adversary gains unauthorized access to a source code repository (SCM) and introduces malicious changes or removes legitimate code. This can be achieved through various methods, such as exploiting vulnerabilities in the SCM, compromising the credentials of a developer with access to the repository, or gaining access to the underlying infrastructure hosting the SCM. An example of this vector attack was the PHP Attack. An attacker compromised PHP's self-hosted Git server, which is a secure repository for storing and managing the source code for the PHP programming language. The attacker was able to inject two malicious commits into the main codebase of PHP. These commits added backdoors that allowed the attacker to gain unauthorized access to PHP installations. The backdoors allowed the attacker to execute arbitrary code on any PHP installation, which could be used to steal data, install malware, or disrupt operations. The attack also caused a great deal of reputational damage to PHP, as it raised concerns about the security of the programming language.

4. Write insecure code

Insecure coding practices, either intentional or unintentional, can introduce vulnerabilities into software. These vulnerabilities can be exploited by attackers to gain unauthorized access, modify or steal data, or disrupt operations. An example of this vector attack was the Apache Struts Attack. In 2003, a hacker infiltrated the source code repository of the open-source software library called Apache Struts. The hacker introduced a vulnerability into the library that allowed them to gain unauthorized access to the systems of organizations that installed the library. The vulnerability allowed the hacker to execute arbitrary code on the affected systems, which could be used to steal data, install malware, and disrupt operations. The attack affected a wide range of organizations, including government agencies, businesses, and individuals.

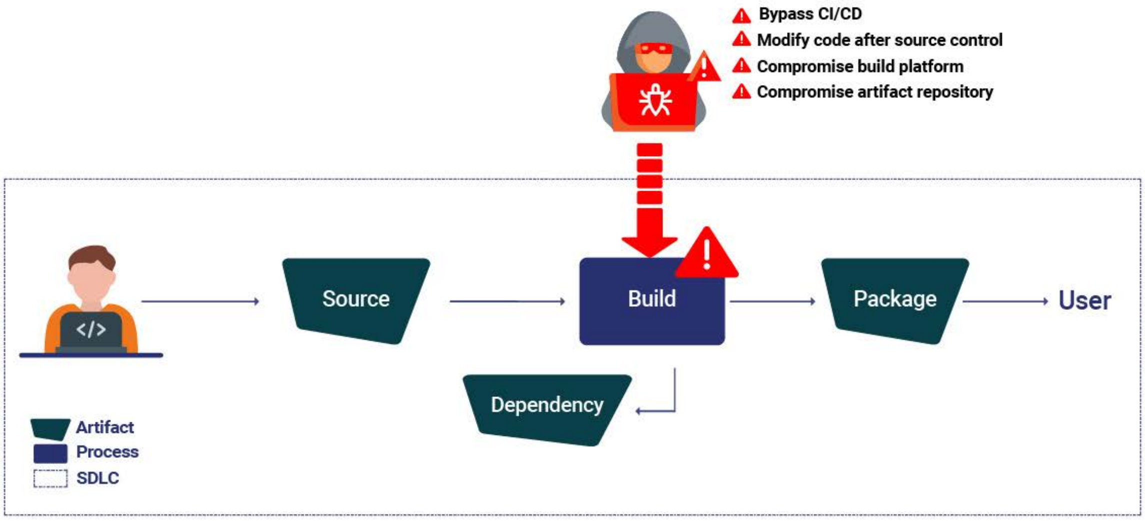
5. Tampering critical files

Altering or modifying critical files in the software development lifecycle can have severe consequences, including the introduction of malicious code, the compromising of sensitive data, and the disruption of software operations. An example of this tampering vector attack was the Maven Attack. In 2020, hackers infiltrated the source code repository of a popular open-source software library called Maven. The hackers inserted malicious code into the library's pom.xml file, which is used to configure the build process. The malicious code allowed the hackers to inject their dependencies into the build process, which were then included in the compiled software. These dependencies contained backdoors that allowed the hackers to gain unauthorized access to the systems of organizations that installed the software.

Software Supply Chain Security Threats in the Build Stage

The build stage of the software supply chain lifecycle encompasses the process of transforming source code into executable software artifacts. This stage involves compiling, linking, and packaging the source code, as well as generating installation packages and configuration files.

Build integrity threats are vulnerabilities that could allow an adversary to introduce unauthorized changes to the software during the build process without altering the source code. These threats can be introduced through various methods, such as compromising the build environment or exploiting vulnerabilities in build tools.


Bypass CI/CD
Modify code after source control
Compromise build platform
Compromise artifact repository
Source
Build
Package
User
Dependency
Artifact
Process
SDLC

Examples of Source Threats

1. Bypass CI/CD

This refers to the practice of circumventing the established CI/CD (continuous integration and continuous delivery) pipeline to directly build and publish software without undergoing the rigorous testing, verification, and auditing processes that are typically enforced by the official pipeline. This can be done by manually building the software outside of the CI/CD environment or by using tools or scripts that allow for unauthorized modifications

to the build process. An example of this type of vector attack was the Jenkins Attack. In 2022, hackers infiltrated the build pipeline of a popular open-source software project called Jenkins. The hackers injected malicious code into a Jenkinsfile, which is a script that defines the build process. The malicious code allowed the hackers to bypass the CI/CD pipeline's security checks and inject their code into the build process. This code is then executed on the systems of organizations that installed the software.

2. Modify code after source control

This practice involves making unauthorized changes to source code after it has been committed to a trusted source control system (SCS) and then building the software using this modified code. This can be done by directly modifying the code on a developer's workstation or by using external tools or scripts to inject malicious code into the build process. An example of this vector attack was the GitLab Attack in 2022. Hackers infiltrated the build pipeline of GitLab. The hackers injected malicious code into the GitLab CI/CD pipeline, which is a tool that automates the build process. The malicious code allowed the hackers to modify the code after it had been checked into source control. This allowed them to inject their code into the software, which was then executed on the systems of organizations that installed the software.

.3. Compromise build process

This involves manipulating or altering the build process itself, either through direct access to the build environment or by exploiting vulnerabilities in build tools or third-party dependencies. This can be done to introduce malicious code into the build output, tamper with build provenance, or disrupt the build process altogether. The most famous example of this vector attack was the SolarWinds Attack. An attacker had gained unauthorized access to SolarWinds' build platform, a system used to compile and package SolarWinds Orion software. This script injected malicious code into the compiled SolarWinds Orion software. When users installed the compromised software, the malicious code was executed on their systems, giving the attacker unauthorized access to their systems. The attacker was also able to steal sensitive data from their systems, such as credentials, intellectual property, and customer information.

.4. Compromise artifact repository

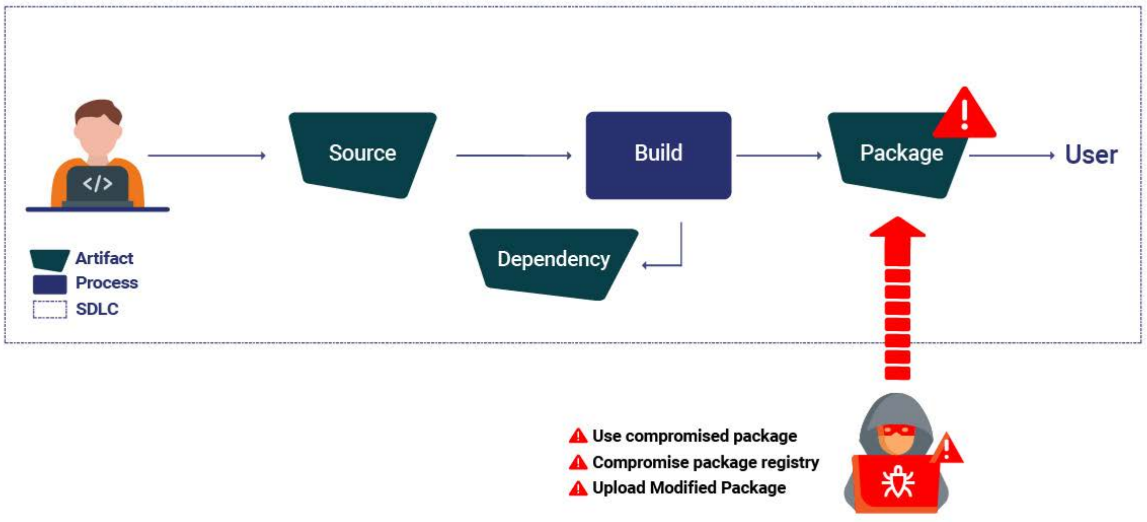
This refers to the unauthorized access or manipulation of an artifact repository, where software packages and binaries are stored for distribution to internal or external users. Attackers can exploit this vulnerability to introduce malicious code, tamper with the authenticity of the software, or disrupt the deployment process. An example of this vector attack was The RubyGems in 2022. Hackers infiltrated the artifact repository of RubyGems. The hackers replaced a legitimate artifact with a malicious one, which was then downloaded by thousands of organizations building software with Ruby on Rails. The malicious artifact allowed the hackers to execute arbitrary code on the systems of organizations that installed the software. This could potentially allow them to steal data, install malware, or disrupt operations.

Software Supply Chain Security Threats in the Package Stage

The package stage of the software supply chain lifecycle encompasses the process of packaging and preparing software for distribution to users. This stage involves creating installation packages, managing dependencies, and generating metadata for the software.

Build integrity threats are vulnerabilities that could allow attackers to introduce unauthorized changes to the software during the packaging process. These threats can be introduced through various methods, such as compromising the package registry, exploiting vulnerabilities in packaging tools, or manipulating third-party dependencies.

The total dependency on open-source components in modern software made this stage the most frequent SSCA target. Introducing stealth malware in a popular open-source component is a dream for many cyber criminals. This is why more than 245,000 malicious packages were found during 2023.



Examples of Source Threats

1. Use compromised package

This refers to the act of deploying or using a software package that has been tampered with or modified by an adversary. This can happen after the package has left the official package registry, either through direct access to the user's system or through social engineering tactics that trick the user into downloading or installing a malicious package. An example of this vector was the Browserify Typosquatting Attack. An attacker, seeking to compromise Linux and Mac systems, infiltrated the development process of a popular Node.js library called Browserify. The attacker slipped malicious code into the project's source code, intending to distribute it through the NPM package registry. Once the tainted Browserify package was uploaded to NPM, unsuspecting developers would download and install it, believing it to be the legitimate version. The malicious code, embedded within the package, would run silently, compromising the integrity of the systems it infected. This could lead to data theft, system instability, or even remote access for the attacker.

2. Compromise package registry

A compromised package registry is a software repository that has been infiltrated by an adversary who has gained unauthorized access to the registry's administrative interface or infrastructure. This allows the adversary to modify or replace legitimate software packages with malicious ones, which can then be distributed to users who unsuspectingly install them. An example of this type of threat was the Attack on Package Mirrors: A researcher, with the intent of promoting open-source software, compromised several popular package registries, including Maven Central, NPM, and RubyGems. By gaining access to these registries, the researcher was able to create mirrors, and replicas of the original repositories, which provided a convenient alternative for developers to download packages. However, these mirrors harbored a sinister purpose. The compromised mirrors served as conduits for the researcher to distribute malicious packages. These packages replaced legitimate ones, undetected by the primary registries, and unsuspecting developers unknowingly downloaded and installed them. Once installed, these malicious packages unleashed their payload, executing arbitrary code, stealing sensitive data, or disrupting operations.

3. Upload Modified Package

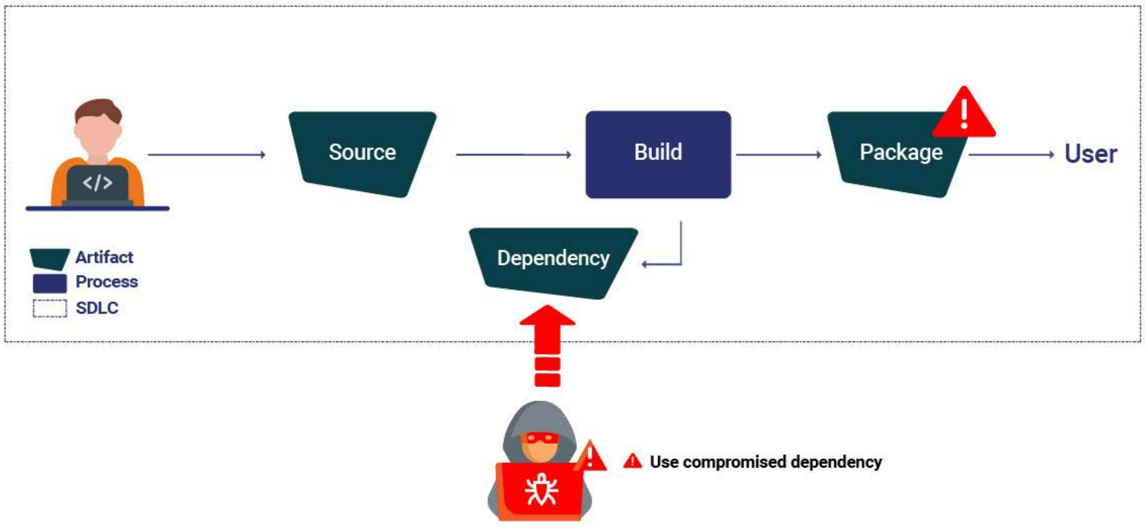
An adversary uploads a modified package to a repository or distribution channel that contains malicious code or payloads. This can be done by modifying the package's source code, packaging, or metadata. One of the most notorious of this type of threat was the CodeCov Attack in 2021. An attacker, seeking to compromise software projects using CodeCov, a popular continuous integration and continuous delivery (CI/CD) tool, utilized leaked credentials to gain unauthorized access to a project's Google Cloud Storage (GCS) bucket. Once the attacker gained access to the GCS bucket, they uploaded a malicious artifact, a modified version of the CodeCov package, which was then distributed to users through the CodeCov service. Unsuspecting developers, relying on the automatic updates feature, would download and install the malicious package, believing it to be the legitimate one. Once installed, the malicious code would run silently, compromising the integrity of the systems it infected. This could lead to data theft, system instability, or even remote access for the attacker.

Attacks to the package registries are so common that some attack patterns received a name:

- In **Typosquatting**, the bad actor uploads to the registry multiple malicious packages with slight typo errors or similar names to legit, popular ones, with the hope that developers will misspell the intended package name with a malicious one. Often the malicious package masquerades the legit one to pass undetected, augmenting the probability of being hit with stargazing.
- **Dependency Confusion** leverages the way some package managers resolve the requested packages from multiple registries. When an organization uses internal components published in an internal registry, an attacker that knows the fact may publish a malicious component with the same name in a public registry. If the name used for the internal component is not scoped, some package managers will fetch the malicious component instead of the internal one.
- With **Troyan Packages**, the cybercriminal disguises malware among useful valid code. This could be used by the real author, or by a contributor that offers himself for maintaining the package. This is also known as Package Hijacking. And attackers used many techniques to hijack an existing package, like Domain Takeover where an abandoned expired domain was taken by the attacker that re-created the old maintainer email and performed password recovery to take over the maintainer account.

Dependency Threats in the Software Supply Chain

Dependencies in the software supply chain lifecycle are third-party libraries, tools, or other components that are used by a software application to function correctly. These dependencies can be included directly in the software code or downloaded and installed separately. Dependency threats are vulnerabilities that can be exploited by adversaries to introduce malicious code into a software application through its dependencies.



Use Compromised Dependency

A compromised dependency is a situation where an adversary has inserted malicious code or payloads into a third-party library or dependency that is used by the software artifact. This can happen during the development of the third-party library or dependency, or it can happen after the artifact has been deployed to production.

An example of this attack was the Event-Stream attack in 2018. An attacker, seeking to compromise software projects using event-stream, a popular Node.js library for handling streams of data, added an innocuous dependency to a project. This innocuous dependency was not malicious in itself, but it served as a placeholder for a future malicious update. Once the innocuous dependency was added to the project, the attacker waited for the project to be released and for users to download and install the updated project. Once the project was updated, the attacker then released a malicious update to the dependency. This malicious update would then be downloaded and installed by unsuspecting users, who believed it to be the legitimate version. Once installed, the malicious code would run silently, compromising the integrity of the systems it infected. This could lead to data theft, system instability, or even remote access for the attacker.

SSCS Threats that Can Attack the Haul SDLC

As we've discussed, software supply chain attacks often target vulnerabilities that span the entire software development lifecycle (SDLC), posing a significant threat to organizations at every stage. Aside from the previously mentioned attack vectors, vulnerabilities and misconfigurations are crucial to consider. These vulnerabilities and misconfigurations can infiltrate the SDLC from design and development to deployment and operation, leaving organizations vulnerable to unauthorized access, data breaches, or operational disruptions.

Vulnerabilities arise from flaws or weaknesses in the software itself, while misconfigurations stem from inadequate setup or configurations. Both vulnerabilities and misconfigurations can be introduced at any stage of the SDLC, providing adversaries with opportunities to exploit them. The NotPetya ransomware attack, which affected thousands of organizations worldwide in 2017, serves as a stark example of how vulnerabilities in the update process can lead to widespread disruption. The attack originated from a misconfiguration in the Ukrainian tax software MeDoc update process. The attacker accessed MeDoc's update server and uploaded a malicious version of the software update, which was then distributed to MeDoc's customers. When users of MeDoc installed the malicious update, it installed the ransomware on their systems, encrypting files and disrupting operations. This widespread outbreak highlighted the importance of patching vulnerabilities promptly and of properly configuring update processes throughout the software development lifecycle.

Common software supply chain attack techniques

Although there are numerous types of techniques that can be employed in the cybersphere, the Cybersecurity & Infrastructure Security Agency (CISA), the National Institute of Standards and Technology (NIST) and the U.S. Department of Commerce have summarized them in three categories in their publication
"Defending Against Software Supply Chain Attacks".

1. Hijacking Updates:

Software vendors regularly release updates to address bugs and security vulnerabilities. Threat actors can hijack the update process by infiltrating the vendor's network and either inserting malware into the update itself or modifying the update to grant them control over the software's normal functionality. This is how the infamous NotPetya attack in 2017 spread to Ukraine and beyond, causing widespread disruption to various industries, including international shipping, financial services, and healthcare.

2. Undermining Codesigning:

Codesigning is a security measure that ensures the authenticity and integrity of code. Threat actors can undermine code signing by self-signing certificates, breaking signing systems, or exploiting misconfigured account access controls. By doing so, they can impersonate trusted vendors and insert malicious code into updates, making it more likely to succeed. The China-based threat actor APT 41 has been known to exploit this technique to infiltrate software supply chains.

3. Compromising Open-Source Code:

Open-source code compromises happen when malicious code is inserted into publicly accessible code libraries that are commonly used by developers. Unsuspecting developers often incorporate these libraries into their third-party code, inadvertently introducing the malicious code into their software. In 2018, researchers discovered 12 malicious Python libraries uploaded to the official Python Package Index (PyPI) using typosquatting tactics. These libraries impersonated the popular "Django" Python library but contained additional functionality, such as the ability to obtain boot persistence and open a reverse shell on remote workstations. Open-source code compromises can also affect privately owned software because developers of proprietary code often include blocks of open-source code in their products.

How to mitigate Software Supply Chain Attacks

To mitigate these threats, organizations must adopt a comprehensive software supply chain security strategy that encompasses the SDLC's entirety. This strategy should include proactive vulnerability assessments, regular security testing, and robust authentication and authorization mechanisms. Additionally, organizations should prioritize security measures during the design and development phases, ensuring that security is embedded into the software from its inception.

Mitigate Software Supply Chain Attacks

1 Efficient Risk Assessment

- Prioritize findings based on severity to enable DevSecOps teams to efficiently inspect and remediate security issues.
- Automate the inventory and assessment of potential vulnerabilities in open-source, proprietary, and third-party components used in software projects.
- Ensure compliance with regulatory requirements and standards throughout the software development process.

2 Continuous Compliance & SBOM

- Systematically enforce standards like CIS Software SSC, OWASP, OpenSSF, and SLSA.
- Continuously monitor delivery systems, applications, tools, and teams to ensure compliance with corporate software delivery security policies and practices.
- Track the provenance of all components in your software product and inspect their detailed information for continuous application risk assessment.

3 Enhanced Security Measures for the SSC

- Identify hard-coded secrets in software components and offer actionable recommendations for remediation.
- Protect Infrastructure as Code (IaC) by providing code analysis, policy enforcement, and vulnerability scanning.
- Detect and resolve misconfigurations across the DevOps ecosystem.

4 Advanced Threat Detection

- Proactively monitor and detect unusual activity in the software supply chain.
- Utilize behavior analytics and anomaly detection to establish baselines of normal activity.
- Flag suspicious activities and serve as a mechanism for insider threat detection.

5 Seamless Integration

- Out-of-the-box integrations with several SDLC systems.
- REST API and Web UI for customized integration into the customer's ecosystem and processes.
- Seamless integration with collaboration tools and DevOps workflow.

6 Secure Build & Attestation

- Establish trust with your clientele using software attestations.
- Establish trust with your clientele using software attestations.
- Ensure component security upon installation.

7 Comprehensive Visibility

- Implement asset discovery and comprehensive asset inventory to gain complete visibility into all software artifacts, including components, dependencies, pipelines, systems, tools, and user involvement in software projects.
- Continuously assess and monitor the security posture of every asset in the SDLC.

8 Security Posture

- Offer a unified tool for policy management, configuration scanning, and vulnerability management.
- Automatically enforce security policies and standards across the DevOps ecosystem.
- Provide actionable recommendations for remediation of identified vulnerabilities.

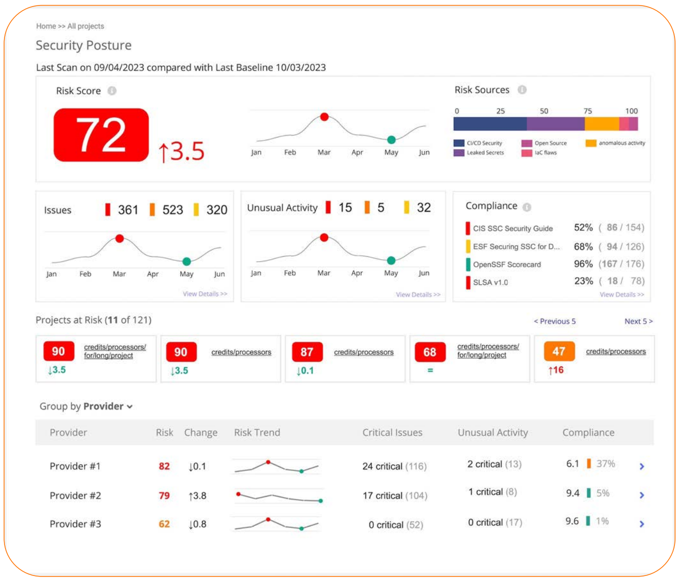
9 Code Tampering Prevention & Anomaly

- Proactively identify risky or suspicious user actions and provide automated real-time alerts.
- Ensure the integrity of critical files. Enforce security and build procedures.

10 Continuous Remediation

- Automate the remediation of identified vulnerabilities.
- Rapid fix deployment to minimize downtime and disruption.

Your Comprehensive Solution for Robust Software Supply Chain Security



Home >> All projects

Security Posture

Last Scan on 09/04/2023 compared with Last Baseline 10/03/2023

Risk Score 72 ↑3.5

Issues 361 523 320

Unusual Activity 15 5 32

Compliance

CIS SSC Security Guide	52%	(86 / 154)
ESF Securing SSC for D...	68%	(94 / 126)
OpenSSF Scorecard	96%	(167 / 176)
SLSA v1.0	23%	(18 / 78)

View Details >>

Projects at Risk (**11** of 121) < Previous 5 Next 5 >

90	↓3.5	credits/processors/for/long/project
90	↓3.5	credits/processors
87	↓0.1	credits/processors
68	=	credits/processors/for/long/project
47	↑16	credits/processors

Group by **Provider**

Provider	Risk	Change	Risk Trend	Critical Issues	Unusual Activity	Compliance
Provider #1	82	↓0.1		24 critical (116)	2 critical (13)	6.1 37%
Provider #2	79	↑3.8		17 critical (104)	1 critical (8)	9.4 5%
Provider #3	62	↓0.8		0 critical (52)	0 critical (17)	9.6 1%

Xgeni's comprehensive platform goes beyond traditional security solutions that focus on isolated phases of the SDLC or specific threat vectors. We take a holistic approach to security, providing a unified defense against all types of threats that can compromise your software supply chain.

Our approach starts with a thorough analysis of your current security posture. We assess your SDLC's vulnerabilities, identify potential risks, and determine your organization's specific needs and security priorities. This in-depth understanding allows us to tailor a customized solution that aligns perfectly with your unique requirements.

Whether you need to mitigate specific threats or secure your entire SDLC, Xygeni offers a comprehensive range of solutions to address your security challenges effectively

Our suite of features includes:

- **Comprehensive visibility:** Gain a unified view of your entire SDLC, including components, dependencies, pipelines, systems, tools, and user involvement.
- **Automated risk assessment:** Leverage Xygeni's automated tools to identify and assess vulnerabilities in open-source, proprietary, and third-party components, ensuring you stay ahead of the latest threats.
- **Enforced security policies:** Uphold regulatory compliance and industry best practices by enforcing security policies and standards across your DevOps ecosystem.
- **Proactive threat detection:** Employ advanced threat detection techniques to proactively identify and prevent unauthorized access, malicious code, and other threats from infiltrating your SDLC..
- **Seamless integration:** Seamlessly integrate Xygeni with your existing SDLC tools and processes, eliminating the need for siloed security solutions.
- **Automated remediation and compliance:** Automate the remediation of identified vulnerabilities and generate evidence of compliance with security policies and standards, streamlining your security operations.

Secure Your Software Supply Chain: From Design to Delivery with Xygeni's Approach

REDUCED RISK OF SUPPLY CHAIN ATTACKS

Proactively identify and mitigate potential threats before they can cause damage.

STREAMLINED COMPLIANCE

Meet regulatory requirements and industry standards across the entire SDLC.

ENHANCED EFFICIENCY

Automate security processes, gain insights into security posture and reduce manual effort, accelerating software delivery.

PROTECTED ASSETS

Safeguard your organization's sensitive data, operations, and reputation from evolving threats.

End To End
Software Supply Chain Security

Protects the integrity and security of your software assets, pipelines and infrastructure of the entire software supply chain.

Contact

Get in touch today!

www.xygeni.io

https://www.linkedin.com/company/xygeni

https://twitter.com/xygeni