Software Supply Chain Security

Jingyue Li

31 March 2025

NTNU

Agenda

- Software supply chain
- Software supply chain threats
- Software supply chain security countermeasure strategies and technologies

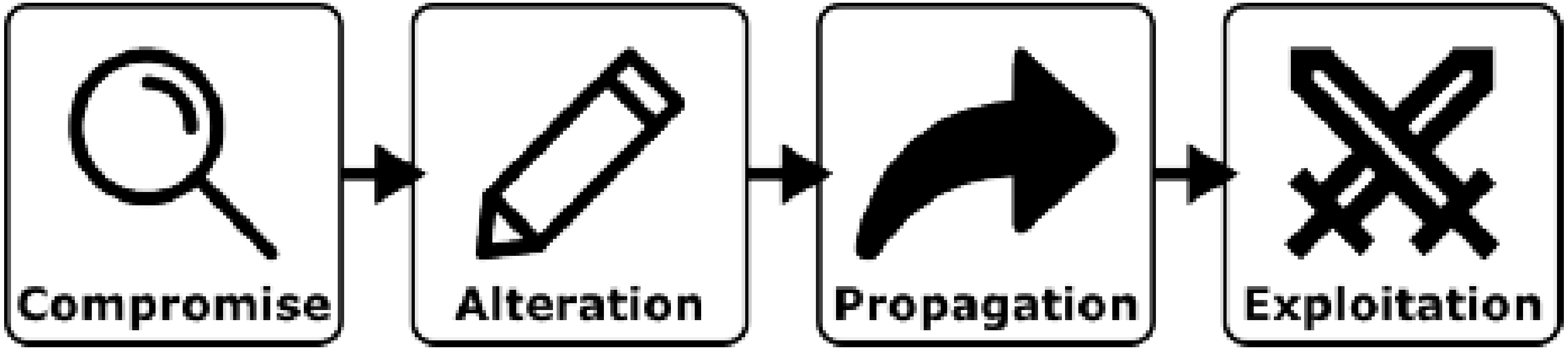
Software supply chain

- Organization's use of externally supplied software (open source or commercially purchased) in products

* https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWT0Nl

Understanding software supply chain threats

Software supply chain attack



- **Compromise:** First, an attacker finds and compromises an existing weakness within a supply chain.
- **Alteration:** Second, an attacker leverages the initial compromise to alter the software supply chain.
- **Propagation:** Third, the change introduced by the attacker propagates to downstream components and links.
- **Exploitation:** The attacker exploits the alterations in a downstream link.

Difference between supply chain attack and vulnerable components

- A06-2021 (OWASP top 10): Vulnerable and Outdated Components
  - Could be the consequence of careless or unintended use/integration of vulnerable components by downstream users

- Supply chain attacks always have malicious attackers in the loop who purposely inject vulnerabilities and plan to exploit them in the future.

An example of software supply chain security incident

- On the 27th of June 2017, a new cyberattack hit many computer systems in Ukraine, as well as in other countries. That attack was spearheaded by the malware detected as Diskcoder.C. This malware is a typical ransomware: it encrypts the data on the computer and demands $300 in bitcoins for recovery. The malware authors intended to cause damage, so they did all they could to make data decryption very unlikely (https://www.welivesecurity.com/2017/07/04/analysis-of-telebots-cunning-backdoor/).

- More information about Diskcoder.C ransomware can be found at https://support.eset.com/en/ca6489-diskcoderc-trojan-outbreak

Other supply chain attacks

Top 5 supply chain attacks of 2023

There has been a notable surge in supply chain cyber-attacks affecting numerous vendors, underscoring a concerning trend in cybersecurity. These incidents emphasize the critical need for robust security measures to protect against evolving threats in the software supply chain. Let's examine some of the major incidents that occurred in 2023.

1. Okta (October 2023):

Okta, a leading provider of identity and authentication management services, disclosed a significant breach where threat actors gained unauthorized access to private customer data through its support management system. Despite security alerts, the breach went undetected for weeks, highlighting the vulnerability of widely used services like Okta to third-party supply chain risks.

2. JetBrains (September/October 2023):

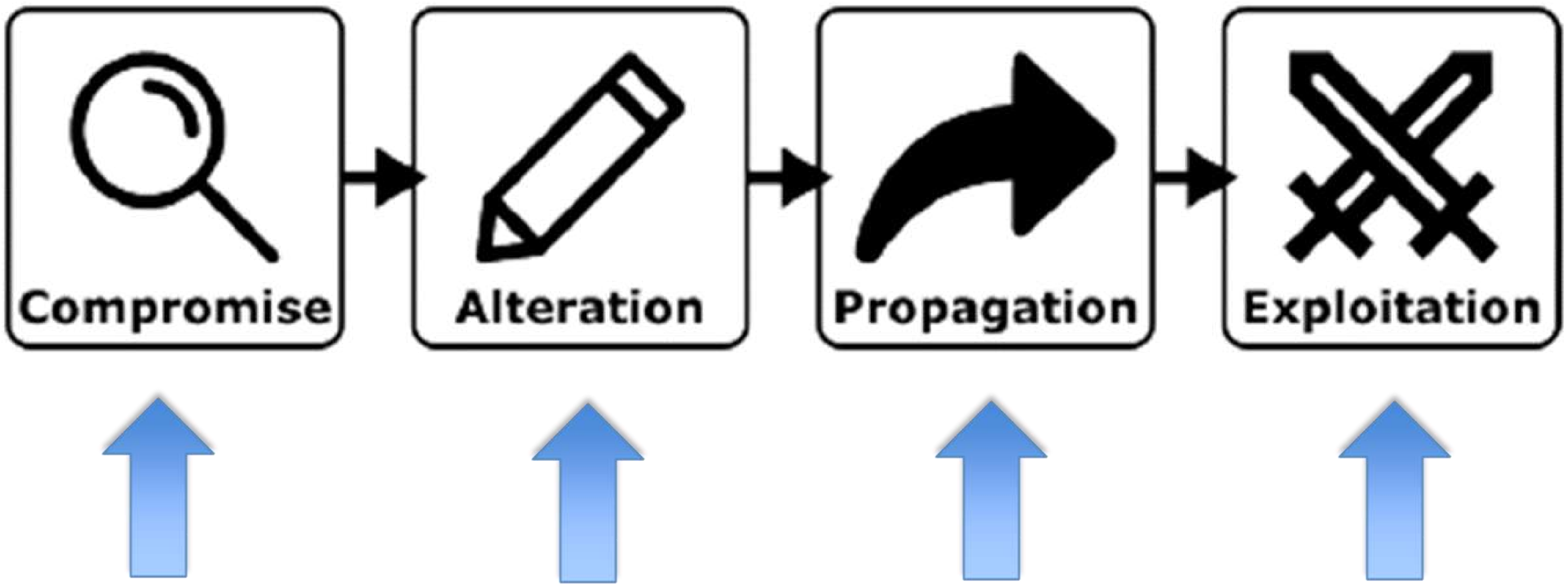
In a concerning development, the SolarWinds hackers exploited a critical vulnerability in JetBrains TeamCity servers, potentially enabling remote code execution and administrative control. This incident underscores the severity of supply chain attacks, as even trusted tools like JetBrains can be compromised, posing significant risks to organizations relying on their software.

3. MOVEit (June 2023):

The MOVEit Transfer tool, renowned for securely transferring sensitive files, was targeted in a supply chain attack affecting over 620

More can be found on https://outshift.cisco.com/blog/top-10-supply-chain-attacks

Countermeasure strategies



Transparency

Validity

Separation

Recovery

Transparency

- Transparency builds trust and security.
- Enables perfect vision of all actors, operations, and artifacts across the supply chain.
- Allow supply chain managers to identify link weaknesses before they are compromised.
- By identifying weaknesses first, managers prevent attackers from completing the first stage.

Validity

- By maintaining
  - integrity of artifacts
  - perfect integrity of operations
  - authentication of actors
- No unauthorized changes can be made to the supply chain.

Separation

- Compartmentalize and moderate interactions between entities.

- Connections between artifacts, operations, and actors are managed so malicious changes cannot affect other supply chain components.

Countermeasure techniques

*

Techniques	Transparency			Validity			Separation		
	Artifacts	Operations	Actors	Artifacts	Operations	Actors	Artifacts	Operations	Actors
SBOM	✓	✓							
npm-audit [55]	✓			✓					
Code scanning [1]	✓			✓					
Dependabot features [29]	✓			✓					
GitHub Actions [28]		✓		✓	✓			✓	
Git Commit Signing [27]			✓	✓					
Scope [54]				✓			✓		✓
Multi-Factor Authentication						✓			
In-toto [73]	✓	✓		✓	✓			✓	✓
Containerization							✓	✓	✓
Version Locking							✓		
Sigstore [51]	✓	✓	✓	✓	✓				
Mirroring and Proxies [53]	✓			✓			✓	✓	

* Okafor et al. SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties. In Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED'22).

Software Bill of Materials (SBOM)

- A SBOM is a nested inventory, a list of ingredients that comprise software components.
- “Minimum elements” for an SBOM*

Minimum Elements	
Data Fields	Document baseline information about each component that should be tracked: Supplier, Component Name, Version of the Component, Other Unique Identifiers, Dependency Relationship, Author of SBOM Data, and Timestamp.
Automation Support	Support automation, including via automatic generation and machine-readability to allow for scaling across the software ecosystem. Data formats used to generate and consume SBOMs include SPDX, CycloneDX, and SWID tags.
Practices and Processes	Define the operations of SBOM requests, generation and use including: Frequency, Depth, Known Unknowns, Distribution and Delivery, Access Control, and Accommodation of Mistakes.

* https://www.ntia.doc.gov/report/2021/minimum-elements-software-bill-materials-sbom

An example of SBOM

- In Software Package Data eXchange **(**SPDX) format

Description *

```
content
├── build
│   └── hello
└── src
    ├── Makefile
    └── hello.c
```

One C source file with a simple "hello world" program, compiled into a single binary with no dependencies via a Makefile. (Assumed dependencies such as the operating system kernel, C standard library, etc. are not addressed here.)

* https://github.com/swinslow/spdx-examples/blob/master/example1/spdx/example1.spdx

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: hello
DocumentNamespace: https://swinslow.net/spdx-examples/example1/hello-v3
Creator: Person: Steve Winslow (steve@swinslow.net)
Creator: Tool: github.com/spdx/tools-golang/builder
Creator: Tool: github.com/spdx/tools-golang/idsearcher
Created: 2021-08-26T01:46:00Z

##### Package: hello

PackageName: hello
SPDXID: SPDXRef-Package-hello
PackageDownloadLocation: git+https://github.com/swinslow/spdx-examples.git#example1/content
FilesAnalyzed: true
PackageVerificationCode: 9d20237bb72087e87069f96afb41c6ca2fa2a342
PackageLicenseConcluded: GPL-3.0-or-later
PackageLicenseInfoFromFiles: GPL-3.0-or-later
PackageLicenseDeclared: GPL-3.0-or-later
PackageCopyrightText: NOASSERTION

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-hello
```

```
FileName: /build/hello
SPDXID: SPDXRef-hello-binary
FileType: BINARY
FileChecksum: SHA1: 20291a81ef065ff891b537b64d4fdccaf6f5ac02
FileChecksum: SHA256: 83a33ff09648bb5fc5272baca88cf2b59fd81ac4cc6817b86998136af368708e
FileChecksum: MD5: 08a12c966d776864cc1eb41fd03c3c3d
LicenseConcluded: GPL-3.0-or-later
LicenseInfoInFile: NOASSERTION
FileCopyrightText: NOASSERTION

FileName: /src/Makefile
SPDXID: SPDXRef-Makefile
FileType: SOURCE
FileChecksum: SHA1: 69a2e85696fff1865c3f0686d6c3824b59915c80
FileChecksum: SHA256: 5da19033ba058e322e21c90e6d6d859c90b1b544e7840859c12cae5da005e79c
FileChecksum: MD5: 559424589a4f3f75fd542810473d8bc1
LicenseConcluded: GPL-3.0-or-later
LicenseInfoInFile: GPL-3.0-or-later
FileCopyrightText: NOASSERTION

FileName: /src/hello.c
SPDXID: SPDXRef-hello-src
FileType: SOURCE
FileChecksum: SHA1: 20862a6d08391d07d09344029533ec644fac6b21
FileChecksum: SHA256: b4e5ca56d1f9110ca94ed0bf4e6d9ac11c2186eb7cd95159c6fdb50e8db5a823
FileChecksum: MD5: 935054fe899ca782e11003bbae5e166c
```

NPM audit

- It automatically checks all your dependencies and its dependency tree for packages that are vulnerable to security flaws
- Command: npm audit

<table>
<tr><th>Moderate</th><th>Prototype pollution</th></tr>
<tr><td>Package</td><td>hoek</td></tr>
<tr><td>Patched in</td><td>> 4.2.0 < 5.0.0 || >= 5.0.3</td></tr>
<tr><td>Dependency of</td><td>numbat-emitter</td></tr>
<tr><td>Path</td><td>numbat-emitter > request > hawk > boom > hoek</td></tr>
<tr><td>More info</td><td>https://nodesecurity.io/advisories/566</td></tr>
</table>

Dependabot

- Discovers insecure dependencies in your project.
- When GitHub detects a vulnerable dependency in the default branch, dependabot creates a pull request to fix it.

- Pull request will upgrade the dependency to the minimum possible secure version needed to avoid the vulnerability.

GitHub Actions

- Threat model
  - The attack can modify the build process.

- Countermeasures
  - The build steps should be precise and repeatable.
  - You know exactly what was running during the build process
  - Ensure each build starts in a new environment to reduce the likelihood of attackers persisting in a build environment.

Git Commit Signing

- Transparency (Actors) and Validity (Artifacts)
- Generate a private and public key pair.
- Use your private key to sign your commit.
- Use another person's public key to verify the author of a commit.

* https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work

Sigstore (1/2)

- Making software signing part of an invisible and ubiquitous infrastructure.

- Using existing identity providers to issue short-lived certificates for individual package signing workflows.

- Users can sign using ephemeral keys (“keyless signing”), which allows developers to sign packages without managing their cryptographic material.

Sigstore (2/2)

- Hosts more than two million (as of April 2022) different package signatures over more than 450 GitHub repositories

Scope

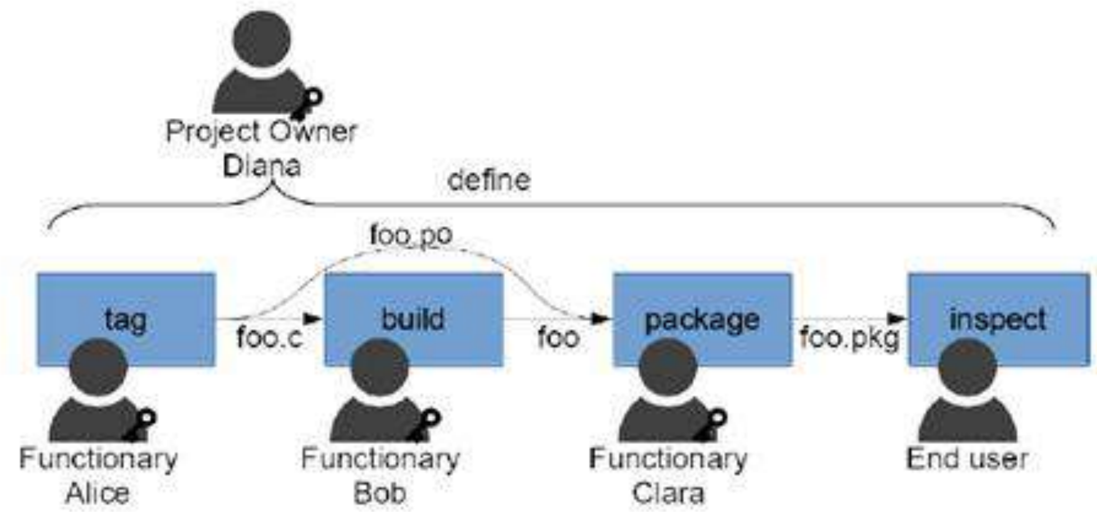
- Threat model
  - Dependency confusion risks where an internal package name is claimed by an attacker on the public registry.
- Countermeasure
  - Restricting the package’s namespace to an organization or user using scope (@somescope/somepackagename).
  - Scopes can be associated with a given registry, which ensures that all requests for packages under the scope will be routed to the given registry

```
$ npm login --scope=@myorg
    --registry=http://registry.myorg.com
```

In toto (Latin for “as a whole”) (1/3)

- One of the key element: Layout

Layout is a recipe that identifies which steps will be performed, by whom, and in what order



*

* Torres-Arias et al. In-toto: providing farm-to-table guarantees for bits and bytes. In Proceedings of the 28th USENIX Conference on Security Symposium (SEC’19).

In toto (2/3)

- Another key element: Link metadata
  - Each link serves as a statement that a given step was carried out.
  - Functionaries executing a step within the supply chain must share information about these links.
  - Sharing such information ensures no artifacts are altered in transit.
  - One-to-one relationship between the step definitions in the supply chain layout and the link metadata.
  - The intended entity must cryptographically sign link metadata

In toto (3/3)

- The third key element: the delivered product
  - To verify the delivered product, the end user will utilize the supply chain layout and its corresponding pieces of link metadata.
  - The end user will use the link metadata to verify that the software provided has not been tampered with and that all the steps were performed as the project owner intended.

Containerization

- Threat model
  - Attackers can propagate the attack or attack consequence via unintended connections.
- Countermeasure
  - Remove unnecessary connections and separate internal operations, artifacts, and actors.

Version Locking

- Threat model
  - Malicious changes upstream may be automatically propagated to downstream links.

- Countermeasure
  - Version locking ensures that a link includes a particular version of an upstream component.
  - However, it relies on actors to accurately set and manage version numbers.

Proxy

- Threat model
  - An attacker might publish a malicious package to the public repository with the same name as a package hosted on a private registry but with a higher semantic version.
  - If a custom setting for an internal registry is omitted, the package manager would default to the public registry and download the latest (malicious) packages from there.
- Countermeasure
  - Configuring the proxy never to allow an upstream request to the public registries protects against fetching arbitrary packages in place of the legitimate package.

Mirroring (1/2)

- Threat model
  - The package manager may download the malicious packages from the public registry.
- Countermeasure
  - Organizations create private package feeds to mitigate the risk of pulling dependencies from public sources.

Mirroring (2/2)

*

Maven example

```
<settings>
  ...
  <mirrors>
    <mirror>
      <id>other-mirror</id>
      <name>Other Mirror Repository</name>
      <url>https://other-mirror.repo.other-company.com/maven2</url>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>
  ...
</settings>
```

* https://maven.apache.org/guides/mini/guide-mirror-settings.html#using-a-single-repository

Way forward

- Most approaches focus on managing artifacts.
- More approaches are needed to focus on operations and actors.
- More empirical studies on using the proposed approaches in practice.

Summary

- Supply chain attacks
  - **Compromise**
  - **Alteration**
  - **Propagation**
  - **Exploitation**

- Countermeasure strategies
  - **Transparency**
  - **Validity**
  - **Separation**
  - **Recovery**