

# Dancing on Thin Ice Leveraging Software Bill of Materials in Java



César Soto Valero

cesarsotovalero.net



Source: DALLE



November 2018







# **Software Supply Chain**

"A software supply chain is composed of the components, libraries, tools, and processes used to develop, build, and publish a software artifact."



"The software supply chain is made up of everything and everyone that touches your code in the software development lifecycle (SDLC), from application development to the CI/CD pipeline and deployment." Red Hat

"The process and components involved in the creation, distribution, and maintenance of software."

OWASP

"The sequence of steps resulting in the creation of an artifact."

SI SA

# 🕺 Software Supply Chain



"The software supply chain is made up of everything and everyone that touches your code in the software development lifecycle (SDLC), from application development to the CI/CD pipeline and deployment."

"The process and components involved in the creation, distribution, and maintenance of software."

"The sequence of steps resulting in the

SI SA

<mark>creation</mark> of an <mark>artifact</mark>.'

## **§** Software Supply Chain



Source: https://slsa.dev/spec/v1.0/terminology

## Software Supply Chain Threats



Source: https://slsa.dev/spec/v1.0/terminology

### Software Development Lifecycle



Source: "Journey to the Center of Software Supply Chain Attacks," in IEEE Security & Privacy, 2023



- 1. Open-source software is safe
- 2. Dependency risks are low
- 3. Vendors are secure



Source: https://xkcd.com/2347



#### SoK: Taxonomy of Attacks on Open-Source Software Supply Chains

Piergiorgio Ladisa<sup>†‡</sup>, Henrik Plate<sup>\*</sup>, Matias Martinez<sup>†</sup>, and Olivier Barais<sup>†</sup>, "SAP Security Research <sup>†</sup>Université Polytechnique Hauts-de-France <sup>†</sup>Université de Rennes 1, Inria, IRISA {piergiorgio.ladisa, henrik.pheil @ @sap.com, matias.matrinez@uphff. {piergiorgio.ladisa, olivier.barais]@irisa.fr

1500

Abstract—The widespread dependency on open-source software makes it a fruiful larget for malicious actors, as demonstrated by recurring attacks. The complexity of today's opensource supply chains results in a significant attack surface, giving attackers nunerous opportunities to reach the goal of injecting malicious code into open-source artifacts that is then downloaded and executed by victims.

This work proposes a general taxonomy for attacks on opensource supply chains, independent of specific programming languages or ecosystems, and covering all supply chain stages from code centributions to package distribution. Taking the form of an attack tree, it covers 107 unique vectors, linked to 94 realworld incidents, and mapped to 33 mitigating safeguards.

User surveys conducted with 17 domain experts and 134 software developers positively validated the correctness, comprehensiveness and comprehensibility of the taxonomy, as well as its suitability for various use-cases. Survey participants also assessed the utility and costs of the identified safeguards, and whether they are used.

Index Terms—Open Source, Security, Software Supply Chain, Malware, Attack

#### I. INTRODUCTION

Software supply chain antacks aim at injecting malicious code into software components to compromise downstream users. Recent incidents, like the infection of SolarWind's Orion platform [1], downloaded by approx. 18,000 customers, including government agencies and providers of critical infrastructure, demonstrate the reach and potential impact of such attacks. Accordingly, software supply chain attacks are among the primary threats in today's threat landscape, as reported by ENISA [2] or the US Executive Order on Improving the Nation's Cobersecutiv 13].

This work focuses on the specific instance of attacks on Open-Source Software (OSS) supply chains, which exploit the widespread use of open-source during the software development lifecycle as a means for spreading malware. Considering the dependency of the software industry on open-source – across the technology stack and throughout the development tifecycle, from libraries and frameworks to development, test and build tools. Ken Thompson's reflections [4] on trust (in code and its authors) is more relevant than ever. Indeed, attackers abuse trust relationships existing between the different open-source stakeholders [5]. [6]. The appearance and significant increase of attacks on OSS throughout the last few years, as reported by Sonatype in their 2021 report [7], demonstrate that attackers consider them a viable means for spreading malware.

© 2023, Piergiorgio Ladisa, Under license to IEEE. DOI 10.1109/SP46215.2023.00010 Recently, industry and government agencies increased their efforts to improve software supply chain security, both in general and in regards to open-source. MITRE, for instance, proposes an end-to-end framework to preserve supply chain integrity [8], and the OpenSSF develops the SLSA framework, which groups several security best-practices for open-source projects [9]. Academia contributes an increasing number of scientific publications, many of which get broad attention in the developer community, e.g., [10] or [11].

Nevertheless, we observed that existing works on opensource supply chain security lack a comprehensive, comprehensive, and general description of how attackers inject malicious code into OSS projects, that is independent of specific programming languages, ecosystems, technologies, and stakeholders.

We believe a taxonomy classifying such attacks could be of value for both academia and industry. Serving as a common reference and clarifying terminology, it could support several activities, e.g., developer training, risk assessment, or the development of new safeguards. As such, we set out to answer the following research questions:

#### RQ1 - Taxonomy of attacks on OSS supply chains

- RQ1.1 What is a comprehensive list of general attack vectors on OSS supply chains?
- RQ1.2 How to represent those attack vectors in a comprehensible and useful fashion?

#### RQ2 - Safeguards against OSS supply chain attacks

- RQ2.1 Which general safeguards exist, and which attack vectors do they address?
- RQ2.2 What is the utility and cost of those safeguards?
   RO2.3 Which safeguards are used by developers?

To answer those questions, we first study both the scientific and grey literature to compile an extensive list of attack vectors, including ones that have been exploited, but also nonexploited vulnerabilities and plausible proofs-of-concept. We then outline a taxonomy in the form of an attack tree. From the identified attacks, we list the associated safeguards. Finally, we conduct two user surveys aiming to validate the attack taxonomy and to collect qualitative feedback regarding the utility, costs, awareness, and use of safeguards.

To this extent, the main contributions of our work are as follows:

 A taxonomy of 107 unique attack vectors related to OSS supply chains, taking the form of an attack tree and validated by 17 domain experts in terms of complete-



Source: https://sap.github.io/risk-explorer-for-software-supply-chains

### Software Supply Chain Attacks



### Software Supply Chain Attacks



Third-party components can be compromised



Software is outdated or unpatched



# The focus of attackers has shifted from "push" to "pull"

### 🐛 Log4Shell Vulnerability (2021)



### 🐛 Log4Shell Vulnerability (2021)



#### Source: Swiss Government Computer Emergency Response Team

## 🐛 Log4j Vulnerability (2021)

- Hidden risks of open-source dependencies
- Complex dependency trees



It could have been detected using SBOMs

Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2021/log4j.md

#### 🐛 GCP Golang Buildpacks Old Compiler Injection (2022)



Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2022/golang-buildpacks-compiler.md

#### 🐛 GCP Golang Buildpacks Old Compiler Injection (2022)





#### Source: https://zt.dev/posts/gcp-buildpacks-old-compiler

#### 🐛 GCP Golang Buildpacks Old Compiler Injection (2022)

- Hidden risks of build systems
- Never assume transparency



• It could have been detected using SBOMs!

Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2022/golang-buildpacks-compiler.md

#### 🐛 NPM Package mathjs-min Credential Stealer (2023)



Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2023/mathjs-min.md

#### 🐛 NPM Package mathjs-min Credential Stealer (2023)



#### Malicious GitHub user



#### Malicious package

E Readme	Code (Beta)	9 Dependencies	🚯 0 Dependents	3 Versions
matk	lic			
IIIau	12			
dath is is an extensive mati	h library for JavaScript and Node is	It features a flexible expression	github.com/lee	k066/mathjs
parser with support for sym	bolic computation, comes with a la	rge set of built-in functions and	Homepage	
constants, and offers an integrated solution to work with different data types like numbers, big			𝔗 mathjs.org	
npm v11.7.0 downloads 2.6M/m	senth 🔿 Node.js CI passing maintained yes	license Apache 2.0	2 weekty Download	
license scan passing @codecov	90% Tests 24.7x [ + 0 ] + 0 Sponsor r	ne on GitHub	147	
Sponsors			Version	License
sponsors			11.7.2	Apacne-2.0
Foresight: Increase CI/CD Health & Test Performance Foresight provides full visibility and deep insights into the health and performance of your tests and CI pipelines.			5.41 MB	2401
		Observability		Pull Requests
		Power-Up for		
Assess the risk of changes		GITHUD ACTIONS		
	times, and deliver high-quality software at speed			
Assess the risk of changes build times, and deliver h	s, resolve bottlenecks, reduce igh-quality software at speed	foresight	Last publish	
with Foresight			z days ago	

Source: https://blog.phylum.io/phylum-discovers-npm-package-mathis-min-contains-discord-token-grabber

#### 🐛 NPM Package mathjs-min Credential Stealer (2023)

• Hidden risks of package managers



- New attackers tactics to deceive developers
- It could have been detected using SBOMs

Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2023/mathjs-min.md

# What is an SBOM, btw?



"A Software Bill of Materials (SBOM) is a formal, machine-readable inventory of software components and dependencies, information about those components, and their hierarchical relationships."

Source: https://www.ntia.gov/sites/default/files/publications/sbom\_at\_a\_glance\_apr2021\_0.pdf

### Inventory of Software Components

#### Köttbullar, färdigstekta Ingredienser: Gris\*- och nötkött\*\* 68%, lök, potatisstärkelse, potatisflingor, animaliskt nötprotein, salt, potatisgranulat, potatisfiber, druvsocker, stabiliseringsmedel E450, E451, kryddor (svart- och vitpeppar, kryddpeppar), köttbuljong. Stekta i rapsolja. Köttursprung: \*Tyskland, \*\*Irland. Förpackad i skyddande atmosfär. Näringsvärde per 100g Sorteras som plastförpackning Energi 1000kJ/240kcal Kylvara: Högst +8°C, Öppnad Fett 18g förpackning hållbar i ca 3 dagar. varav mättat fett Bäst före gäller oöppnad förpackning, Kolhydrat 10g varav sockerarter Nettovikt: 0.8g Protein 11a 1000g Salt 1,7g Bäst före: 04.05.2022 099 Snahht Konsumentkontakt: Färskvaruhuset AB 0303-24 64 50 www.farskvaruhuset se





Make software transparent

Safeguard against software supply chain attacks



EU Cybersecurity Strategy (2020)





DARPA on Integrity of Open Source (2022)

White House Executive Order (2021)



Source: "Software Bill of Materials (SBOM) and Cybersecurity Readiness," The Linux Foundation, 2022

### **Components of an SBOM**

## Components of an SBOM

- 1. SBOM metadata
- 2. Project metadata

. . .

3. Inventory of dependencies



<u> https://spdx.dev</u>



https://cyclonedx.org



#### 🕥 CycloneDX

```
"bomFormat" : "CycloneDX",
"specVersion" : "1.4",
"metadata" : {
    "timestamp" : "2024-01-26T15:04:45Z",
    "tools" : [
        { "name" : "CycloneDX Maven plugin",
        "version" : "2.7.5" }
   ],
....
```

#### **F**SPDX

```
"SPDXID": "SPDXRef-DOCUMENT",
"spdxVersion": "SPDX-2.3",
"creationInfo": {
    "created": "2024-01-26T15:04:45Z",
    "creators": [
        "Tool: GitHub.com-Dependency-Graph"
    ],
    "comment": "..."
},
```

### 📃 Project Metadata

```
"component" : {
    "group" : "org.asynchttpclient",
    "name" : "async-http-client-project"
    "version" : "2.12.3",
     "hashes" : [ { "alg" : "SHA-256",
         "content" : "70997d52db...3b6172ebb5"}, ... ],
     "licenses" : [...].
     "externalReferences" : [ {
       "url" : "http://github.com/AsyncHttpClient/async-http-client" }
     "bom-ref" :
"pkg:maven/org.asynchttpclient/async-http-client-project@2.12.3?type=pom"
. . .
```







70997d52dbffd4e797c467b2062 c6ba9b8b5a7380ff2884779f274 3b6172ebb5



### Third-Party Dependencies



### **Fird-Party Dependencies**



Figure made with: https://github.com/ferstl/depgraph-maven-plugin

# Use cases of SBOMs



### **Vulnerability Scanning**



### Compliance Checking

- Meet legal standards
- Check license compliance
- Better visibility

📒 An official website of the United States government Here's how you know 🗸		
NIST	Search NIST	ୟ ≡ Menu
Information Technology Laboratory		
EXECUTIVE ORDER 14028, IMPROVING TH	E NATION'S CYBER	RSECURITY
Software Supply Chain Security Guidance		+
Cybersecurity Labeling for Consumers		+
Workshops & Call for Papers		
News & Updates		
Engage		
Fact Sheet		
Resources		
FAQs		
Cybersecurity @ NIST		

#### Improving the Nation's Cybersecurity: NIST's Responsibilities Under the May 2021 Executive Order

Overview Completed Assignments Latest Updates

#### OVERVIEW

The President's Executive Order (EO) 14028 on Improving the <u>Nation's Cybersecurity</u>. issued on May 12, 2021, charges multiple agencies – including NIST – with enhancing cybersecurity through a variety of initiatives related to the security and integrity of the software supply chain.

Section 4 directs NIST to solicit input from the private sector, academia, government agencies, and others and to identify existing or develop new standards, tools, best practices, and other guidelines to enhance software supply chain security. Those guidelines, which are ultimately aimed at federal agencies but which also are available for industry and others to use, include:

- · criteria to evaluate software security,
- · criteria to evaluate the security practices of the developers and suppliers, and
- innovative tools or methods to demonstrate conformance with secure practices.

NIST is to consult with other agencies in producing some of its guidance; in turn, several of those agencies are directed to take steps to ensure that federal procurement of software follows that guidance.

The EO also assigns NBT to work on two labeling efforts related to consumer internet of Things (IoT) devices and consumer software with the goal of **encouraging manufacturers to produce – and purchasers to be informed about** – products created with present consideration of cybersecurity risks and capabilities.

Source: https://www.nist.gov/itl/executive-order-14028-improving-nations-cybersecurity

### Dependency Management

- Understand dependencies
- Detect outdated dependencies
- Reduce code bloat





#### César Soto-Valero, Martin Monperrus, and Benoit Baudry, KTH Royal Institute of Technology

Ethereum is the single largest programmable blockchain platform today. Ethereum nodes operate the blockchain, relvina on a vast supply chain of third-party software dependencies. In this article, we perform an analysis of the software supply chain of Java Ethereum nodes and distill the challenges of maintaining and securing this blockchain technology.

thereum is a spearhead of the blockchain para- and financial transactions.<sup>4</sup> There are several public digm, with its smart contract infrastructure sup- blockchains running today (for example, Bitcoin, Etheporting a vibrant decentralized finance ecosys- reum, Litecoin, and NEO), each one of them serving a em<sup>1</sup> and a blooming art scene.<sup>2</sup> Since the release particular purpose and solving specific problems. In this of Bitcoin in 2008.3 the adoption of blockchain-based article, we focus on the single case of Ethereum as it is solutions has grown significantly, mainly driven by the the largest blockchain platform by most notable metrics. promise of secure, reliable, and decentralized monetary Ethereum is a feature-rich platform, considered by some as the avant-garde of blockchain technologies.<sup>5</sup> It has its own cryptocurrency (Ether), its own consensus protocol, and its own smart contract platform. Ethereum

Digital Object Identifier 10.2109/MC.2022.3275542

Source: "The Multibillion Dollar Software Supply Chain of Ethereum" in IEEE Computer, 2022

## Java dependencies



### Package Managers (Maven)



#### \$ mvn dependency:tree

[INFO]	Scanning for projects
[INFO]	
[INFO]	< org.springframework.samples:spring-petclinic >
[INFO]	Building petclinic 3.2.0-SNAPSHOT
[INFO]	from pom.xml
[INFO]	[ jar ]
[INFO]	
[INFO]	dependency:3.6.1:tree (default-cli) @ spring-petclinic
[INFO]	org.springframework.samples:spring-petclinic:jar:3.2.0-SNAPSHOT
[INFO]	+- org.springframework.boot:spring-boot-starter-actuator:jar:3.2.1:compile
[INFO]	+- org.springframework.boot:spring-boot-starter:jar:3.2.1:compile
[INFO]	+- org.springframework.boot:spring-boot-starter-logging:jar:3.2.1:compile
[INFO]	+- ch.qos.logback:logback-classic:jar:1.4.14:compile
[INFO]	\-  ch.qos.logback:logback-core:jar:1.4.14:compile
[INFO]	+- org.apache.logging.log4j:log4j-to-slf4j:jar:2.21.1:compile
[INFO]	\- org.apache.logging.log4j:log4j-api:jar:2.21.1:compile
[INFO]	\- org.slf4j:jul-to-slf4j:jar:2.0.9:compile
[INFO]	+- jakarta.annotation:jakarta.annotation-api:jar:2.1.1:compile
[INFO]	\- org.yaml:snakeyaml:jar:2.2:compile
[INFO]	+- org.springframework.boot:spring-boot-actuator-autoconfigure:jar:3.2.1:compile
[INFO]	+- org.springframework.boot:spring-boot-actuator:jar:3.2.1:compile
[INFO]	\- com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.15.3:compile
[INFO]	<pre>+- io.micrometer:micrometer-observation:jar:1.12.1:compile</pre>
[INFO]	\- io.micrometer:micrometer-commons:jar:1.12.1:compile
[INFO]	\- io.micrometer:micrometer-jakarta9:jar:1.12.1:compile
[INFO]	<pre>  \- io.micrometer:micrometer-core:jar:1.12.1:compile</pre>
[INFO]	+- org.hdrhistogram:HdrHistogram:jar:2.1.12:runtime
[INFO]	<pre>  \- org.latencyutils:LatencyUtils:jar:2.0.3:runtime</pre>

\$ mvn dependency:tree -DoutputType=dot -DoutputFile=deps.dot \$ dot -Txdot\_json -o deps.json deps.dot

"name": "org.springframework.samples:spring- <u>petclinic</u> :jar:3.2.0-SNAPSHOT",
"_subgraph_cnt": 0,
"objects": [
{
" qvid": 0.
"name": "org.springframework.samples:spring-petclinic:jar:3.2.0-SNAPSHOT".
"label": "\\N"
4
"name": "ong enningfnamework host enning-host-stanten-actuator jar 3 2 1 commile"
Laber . (W
"_ <u>gvi</u> g": 2,
"name": "org.spring+ramework.boot:spring-boot-starter-cache:jar:3.2.1:compile",
"label": "\\N"
l F,
{
"_gvid": 3,
"name": "org.springframework.boot:spring-boot-starter-data-jpa:jar:3.2.1:compile",
"label": "\\N"

### Package Managers (Gradle)



#### \$ gradle dependencies --configuration runtimeClasspath





JSON output of gradle dependencies #21894 h4sh5 opened this issue on Sep 7, 2022 · 17 comments •



tsjensen commented on Nov 9, 2023

I think we should not invent a custom format, but instead output a proper <u>SBOM</u>. The easiest format for that is <u>CycloneDX</u>. It's still JSON, but in a way that other tools natively understand. There's a <u>Java library</u> to create it.

 $\odot$ 

Source: https://github.com/gradle/gradle/issues/21894





- Extra metadata from dependencies:
  - Direct / Transitive / Inherited
  - Used / Unused
  - JAR size

. . .

• Debloated pom.xml

🔳 🖸 🖂 ASSERT-KTH /	depclean		0   + • 💿 n 🖻 🌡
<> Code 💿 Issues 💈 🕅	Pull requests 1 💿 Action	ns 🕮 Wiki 🕸	Settings
depclean Public	🖒 Edit Pins 👻 💿 Unv	watch 8 👻 😵 F	ork 27 🔹 🌟 Starred 223 👻
°¢ master ▼ °¢ S	Go to file +	<> Code •	About 🕸
(a) cesarsotovalero Update F	R 🗸 dfb6773 · 8 hours ago	🕙 508 Commits	DepClean automatically detects and removes unused
.github	Update actions/cache act	2 weeks ago	(https://dx.doi.org/10.1007/
img .	Improve README.md (#72)	3 years ago	s10664-020-09914-8)
🖿 art	Rename header	2 years ago	java bytecode maven-plugin dependencies bloatware
beclean-core	Update dependency org	2 days ago	debloating
bepclean-gradle-plugin	Update dependency org	3 weeks ago	🖽 Readme
epclean-maven-plugin	Update dependency org	2 days ago	ৰ্বায় MIT license
<ul> <li>.gitattributes</li> </ul>	ref: licence	4 years ago	Cite this repository → -√ Activity
<ul> <li>.gitignore</li> </ul>	Implemented createResul	3 years ago	E Custom properties
CITATION.cff	Enhance CITATION.cff	3 years ago	☆ 223 stars
IICENSE.md	Update LICENSE.md	4 years ago	♀ 27 forks
README.md	Update README.md	8 hours ago	Releases 9
<> checkstyle.xml	Static analysis check base	2 years ago	S 2.0.5 (Latest) on Jan 1, 2023
ି codecov.yml	Update codecov.yml	last month	+ 8 releases
<> pom.xml	Update dependency org	2 weeks ago	Contributors 14
💘 renovate.json	Config perpetual autome	6 months ago	🎒 🌏 🍪 😫 😤
다 README 화 MIT license	e	∅ :≡	🔁 🗍 🕘 🚇
-			a 🔁 🔁 🛨

https://github.com/ASSERT-KTH/depclean

### **SBOM producers for Java projects**

### **GitHub UI**

cesarsotovalero / spring-petclinic	Q. Type [] to search	<b>6</b>
> Code 🖞 Pull requests 🕞 Actions	⊟ Projects ① Security └── Insights 診 Settings	
Pulse	Dependency graph	
Contributors	Eurort CPON	
Community	Dependencies Dependents Dependabot	VI
Traffic	Detect additional dependencies with GitHub Actions	
Commits	Not all dependencies are automatically detected for ecosystems like Gradle.	
Code frequency	GitHub Actions adds your dependencies using the <u>dependency submission</u> API so you can receive Dependabot alerts for known vulnerabilities.	
Dependency graph	View in Marketplace	
Network		
Forks	Q Search all dependencies	
	actions/checkout 4 Detected automatically on Jan 26, 2024 (GitHub Actions)github/workflows/maven-build.yml	

actions/setup-java 4 Detected automatically on Jan 26, 2024 (GitHub Actions) · .github/workflows/maven-build.yml

#### com.github.ben-manes.caffeine:caffeine

Detected automatically on Jan 26, 2024 (Maven) - pom.xml

#### com.gitlab.haynes:libsass-maven-plugin 0.2.29

Detected automatically on Jan 26, 2024 (Maven) · pom.xml · MIT

#### com.h2database:h2

Detected automatically on Jan 26, 2024 (Maven) - pom.xml

1	
2	"SPDXID": "SPDXRef-DOCUMENT",
3	"spdxVersion": "SPDX-2.3",
4	"creationInfo": {
5	"created": "2024-01-27T20:03:37Z",
6	"creators": [
7	"Tool: GitHub.com-Dependency-Graph"
8	],
9	"comment": "Exact versions could not be resolved for some packages. For more information
10	$\{\cdot,\cdot\}_{i}$
11	"name": "com.github.cesarsotovalero/spring-petclinic",
12	"dataLicense": "CC0-1.0",
13	"documentDescribes": [
14	"SPDXRef-com.github.cesarsotovalero-spring-petclinic"
15	],
16	documentNamespace": "https://github.com/cesarsotovalero/spring-petclinic/dependency_graph/
17	"packages": [
18	{
19	"SPDXID": "SPDXRef-com.github.cesarsotovalero-spring-petclinic",
20	"name": "com.github.cesarsotovalero/spring-petclinic",
21	"versionInfo": "",
22	"downloadLocation": "git+https://github.com/cesarsotovalero/spring-petclinic",
23	"licenseDeclared": "Apache-2.0",
24	"filesAnalyzed": false,
25	"supplier": "NOASSERTION",
26	"externalRefs": [
27	{
28	"referenceCategory": "PACKAGE-MANAGER",
29	"referenceType": "purl",
30	"referenceLocator": "pkg:github/cesarsotovalero/spring-petclinic"
31	}
32	]
33	};
34	{
35	"SPDXID": "SPDXRef-maven-com.github.ben-manes.caffeine-caffeine",
36	"name": "maven:com.github.ben-manes.caffeine:caffeine",
37	"versionInfo": "",
38	"downloadLocation": "NOASSERTION",
39	"filesAnalyzed": false,
40	"supplier": "NOASSERTION"

### **GitHub Actions**



28	# =====================================
29	#
30	# Step 1: Build your artifacts.
31	#
32	#
33	– name: Build artifacts
34	run:
35	# These are some amazing artifacts.
36	echo "artifact1" > artifact1
37	echo "artifact2" > artifact2
38	
39	#
40	#
41	# Step 2: Add a step to generate the provenance subjects
42	# as shown below. Update the sha256 sum arguments
43	# to include all binaries that you generate
44	# provenance for.
45	#
46	# =====================================
47	– name: Generate subject for provenance
48	id: hash
49	run:
50	set —euo pipefail
51	
52	# List the artifacts the provenance will refer to.
53	files=\$(ls artifact*)
54	# Generate the subjects (base64 encoded).
55	echo "hashes=\$(sha256sum \$files   base64 -w0)" >> "\${GITHUB_OUTPUT}"
56	
57	provenance:
58	needs: [build]
59	permissions:
60	actions: read # To read the workflow path.
61	id-token: write # To sign the provenance.
62	contents: write # To add assets to a release.
63	uses: slsa-framework/slsa-github-generator/.github/workflows/generator_generic_slsa3.yml@v1.4.0
64	with:
65	<pre>base64-subjects: "\${{ needs.build.outputs.digests }}"</pre>
66	upload-assets: true # Optional: Upload to a new release

Source: https://github.com/slsa-framework/slsa-github-generator

### **Dedicated Plugins**

- cyclonedx-maven-plugin
- cyclonedx-gradle-plugin
- spdx-maven-plugin
- spdx-gradle-plugin

Maven <build> <plugins> <plugin> <groupId>org.cyclonedx</groupId> <artifactId>cvclonedx-maven-plugin</artifactId> <version>2.7.11</version> <executions> <execution> <goals> <goal>makeBom</goal> </goals> <phase>package</phase> </execution> </executions> </pluain> </plugins> </build>

Example: https://github.com/CycloneDX/cyclonedx-maven-plugin

### GraalVM







Source: https://www.graalvm.org/22.2/reference-manual/native-image/debugging-and-diagnostics/InspectTool/#software-bill-of-materials-sbom

# Challenges with SBOMs

### **Tool Consistency**



#### SOFTWARE SUPPLY CHAIN SECURITY

#### **Challenges of Producing Software Bill of Materials for Java**

Musard Balliu<sup>10</sup>, Benoit Baudry<sup>10</sup>, Sofia Bobadilla<sup>10</sup>, Mathias Ekstedt<sup>10</sup>, Martin Monperrus<sup>10</sup>, Javier Ron<sup>(1)</sup>, Aman Sharma<sup>(1)</sup>, Gabriel Skoglund<sup>(1)</sup>, César Soto-Valero<sup>(1)</sup>, and Martin Wittlinger<sup>(D)</sup> KTH Royal Institute of Technology

Software bills of materials (SBOMs) promise to become the backbone of software supply chain hardening. We deep-dive into six tools and the SBOMs they produce for complex open source lava projects, revealing challenges regarding the accurate production and usage of SBOMs.

a major challenge for both security and reliability.<sup>2</sup> For the artifact. example, to compromise a high-value application, malicious actors can choose to attack a less well-guarded malicious intent, bugs can propagate through the software supply chain and cause breakages in applications.4 Gathering accurate, up-to-date information about all vital importance.

#### Introduction

emerged as a key concept to enable principled engineering of software supply chains. This takes the well-known concept of "bill of materials" for manufacturing physical

Digital Object Identifier 10.1109/MSEC.2023.3302956 Dete of current version: 29 August 202

odern software applications are virtually never goods into the world of software development. The built entirely in-house. As a matter of fact, they purpose of an SBOM is to capture relevant informareuse many third-party dependencies, which form the tion about the internals of a software artifact. First and core of their software supply chain.<sup>1</sup> The large num- foremost, an SBOM is expected to include a complete ber of dependencies in an application has turned into inventory of all of the third-party dependencies of

Accurate SBOMs are essential for software supply chain management,5 vulnerability tracking, build dependency of the project.<sup>3</sup> Even when there is no tampering detection,<sup>6</sup> and high software integrity. For example, software developers leverage SBOMs to identify vulnerable software components in a timely manner. This is usually done by matching software component dependencies included in an application is, therefore, of versions against vulnerability databases and reporting a warning whenever a vulnerable component is part of an application. For example, in 2021, a serious vulnerability present in the popular Java logging component The software bill of materials (SBOM) has recently Log4J was discovered. This component was extensively used by a large number of open source and proprietary projects, and consequently, it was a tedious and costly endeavor to identify all impacted projects.7 Had all of these Java projects published an SBOM, it would have facilitated the precise identification and remediation of vulnerable applications.

Source: "Challenges of Producing Software Bill of Materials for Java," in IEEE Security & Privacy, 2023

### **Sharing & Distribution**



#### An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead

Boming Xia<sup>\*†</sup>, Tingting Bi<sup>+‡</sup>, Zhenchang Xing<sup>+‡</sup>, Qinghua Lu<sup>\*†</sup>, Liming Zhu<sup>\*†</sup> <sup>\*</sup>CSIRO's Data61, Sydney, Australia <sup>†</sup>University of New South Wales, Sydney, Australia <sup>\*</sup>Manash University, McIbourne, Australia <sup>\*</sup>Materialian National University, Conberra, Australia

2630

Abstract-The rapid growth of software supply chain attacks has attracted considerable attention to software bill of materials (SBOM), SBOMs are a crucial building block to ensure the transparency of software supply chains that helps improve software supply chain security. Although there are significant efforts from academia and industry to facilitate SBOM development, it is still unclear how practitioners perceive SBOMs and what are the challenges of adopting SBOMs in practice. Furthermore, existing SBOM-related studies tend to be ad-hoc and lack software engineering focuses. To bridge this gap, we conducted the first empirical study to interview and survey SBOM practitioners. We applied a mixed qualitative and quantitative method for gathering data from 17 interviewees and 65 survey respondents from 15 countries across five continents to understand how practitioners perceive the SBOM field. We summarized 26 statements and grouped them into three topics on SBOM's states of practice. Based on the study results, we derived a goal model and highlighted future directions where practitioners can put in their effort

Index Terms—software bill of materials, SBOM, bill of materials, responsible AI, empirical study

#### I. INTRODUCTION

Modern software products are assembled through intricate and dynamic supply chains [1], while recent attacks against software supply chains [3, 58] while recent attacks against assembled attack [2]. According to Sonatype's report [3], there was a 650% year-over-year increase in SSC attacks mainly aim at the upstream open source software/components (SSS) [4], yet OSS is heavily refu apon in software development [5]. The reliance on OSS leads to additional risks, and as the lack OS is heavily release and support compared to the strength of trainable maintenance and support compared to the strength of trainable maintenance in adaptor to compared to the SSC, with which integly and accounts identification of the impacted software/components could be carried out in case of a vulnerability on a SSC attack.

A software bill of materials (BROM) is a formal machinereadable inventory of the components (and their dependency relationships) used for producing a software product [7]. SBOMs enhance the security of both the proprietary and open source components in SSCs [8] through improved transparnecy. According to Linux Foundation's SBOM and Cybersecurity Readiness report (SBOM readiness report for short) [5]. SBOMs are eritical for enhancing SSC security. 90% of the surveyed organizations have started or are planning their SBOM journey, with 54% already addressing SBOMs. The report also estimated 66% and 15% growth in SBOM production or comsumption in 2022 and 2023, respectively. Nonetheless, some organizations are still concerned about how SBOM adoption and application will evolve (e.g. 40% are uncertain about industrial SBOM commitment and 39% seek consensus on SBOM data fields).

Despite SBOMs' essentiality for software and SSC security, there remain questions to answer and problems to solve. Motivated by the value of SBOMs and the existing gaps, with the overarching goal of investigating the SBOM status quo from practitioners' perspectives, this paper aims to answer the following research questions (RQs).

#### RQ1: What is the current state of SBOM practice?

Despite the benefits of SBOMs and the SBOM readiness report showing an overall 90% of SBOM readiness, how practitioners perceive SBDMs and how SBOMs are being addressed in practice needs further investigation. To answer this question, we analyzed the SBOM practice status from SBOM generation, distribution and sharing, validation and verification, and vulnerability and exploitability management. We summarized the current SBOM practices and what practitioners expect.

#### RQ2: What is the current state of SBOM tooling support?

Despite the proliferation of the SBOM tooling market, this RQ focuses on the SBOM tooling status from the practitioners' perspective. We investigated the practitioners' attitudes towards existing tools from the following aspects: the necessityavailabilityasibilityingerity of SBOM tools. While exploring the current SBOM tooling state, we also looked into practitioners' expectations of SBOM tools.

#### RQ3: What are the main concerns for SBOM?

This RQ investigates the most outstanding concerns SBOM practitioners have. Although the prospect of SBOMs is promising, there are still challenges to resolve. With this RQ, we aim to provide a reference for the most imminent issues for future research and development on SBOMs.

Our research aims to unveil the state of the SBOM field, investigating what practitioners have and how they are addressing SBOMs, versus what they expect. Our work on

1558-1225/23/\$31.00 @2023 IEEE DOI 10.1109/ICSE48619.2023.00219

# Wrapping up...



Source: DALLE



- SBOMs are essential for software supply chain security
- SBOMs are increasingly getting attention and adoption
- SBOMs are well supported in the Java ecosystem

