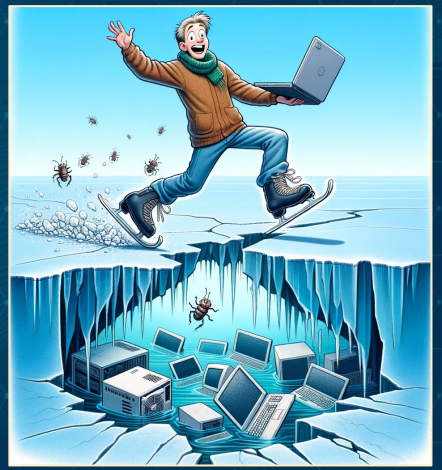
Dancing on Thin Ice ⛸️

Leveraging Software Bill of Materials in Java

César Soto Valero

cesarsotovalero.net



Source: DALL·E

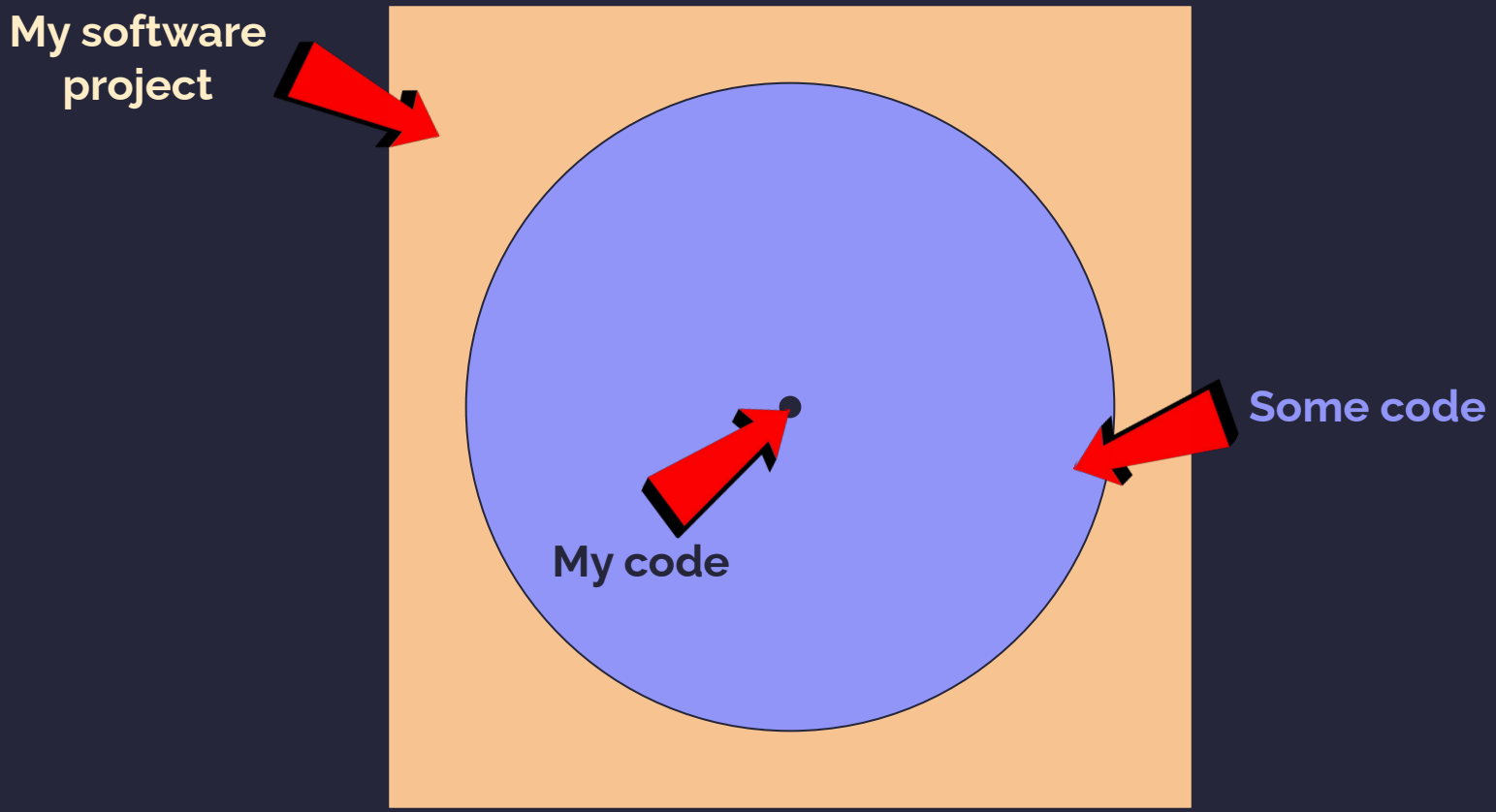
$ whoami

2023 Data Engineer @ SEB

2018 PhD Student @ KTH

2013 Software Engineer @ UCLV

November 2018


My software project
Some code
My code

(Not this) Supply Chain

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."

Wikipedia

"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."

SLSA

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."

Wikipedia

"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."

RedHat

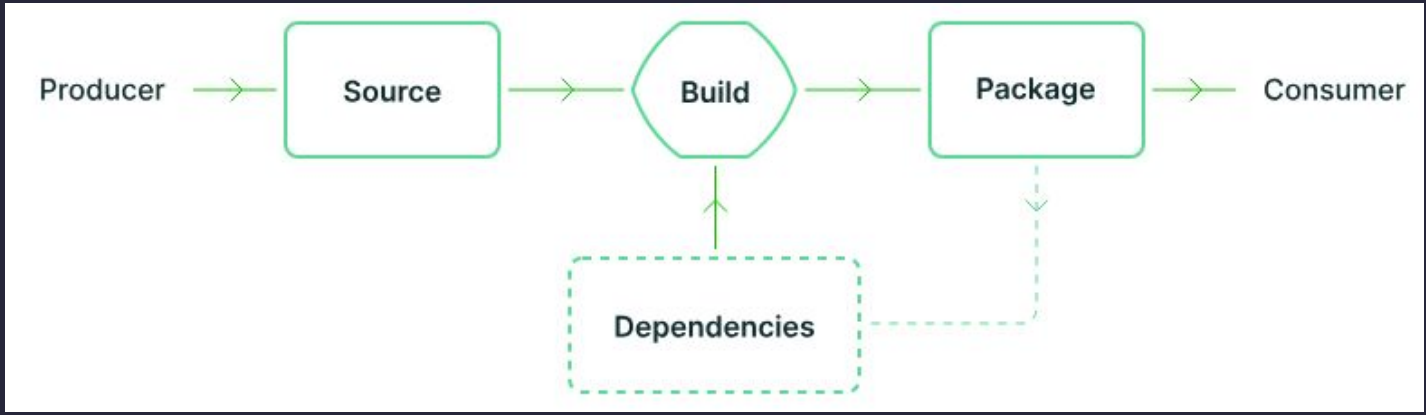
"The process and components involved in the creation, distribution, and maintenance of software."

OWASP

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

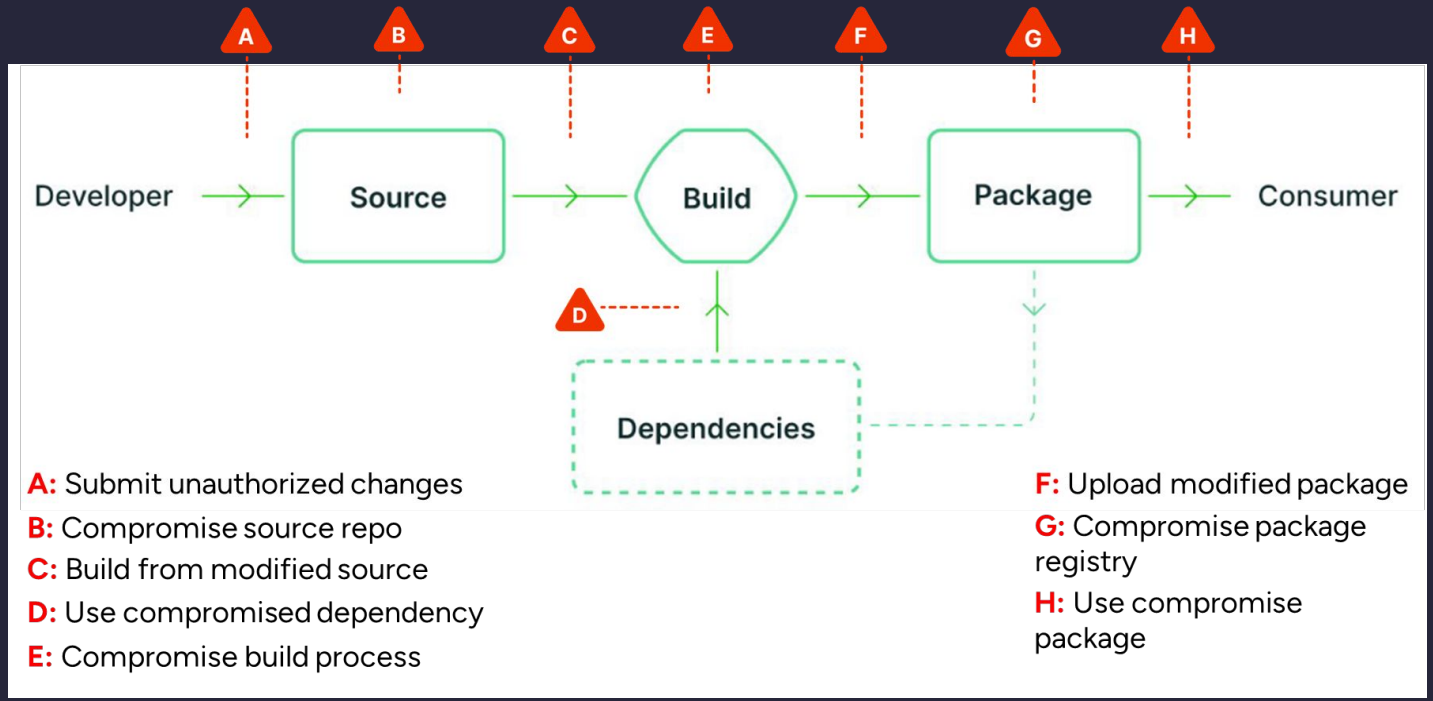
SLSA

Software Supply Chain



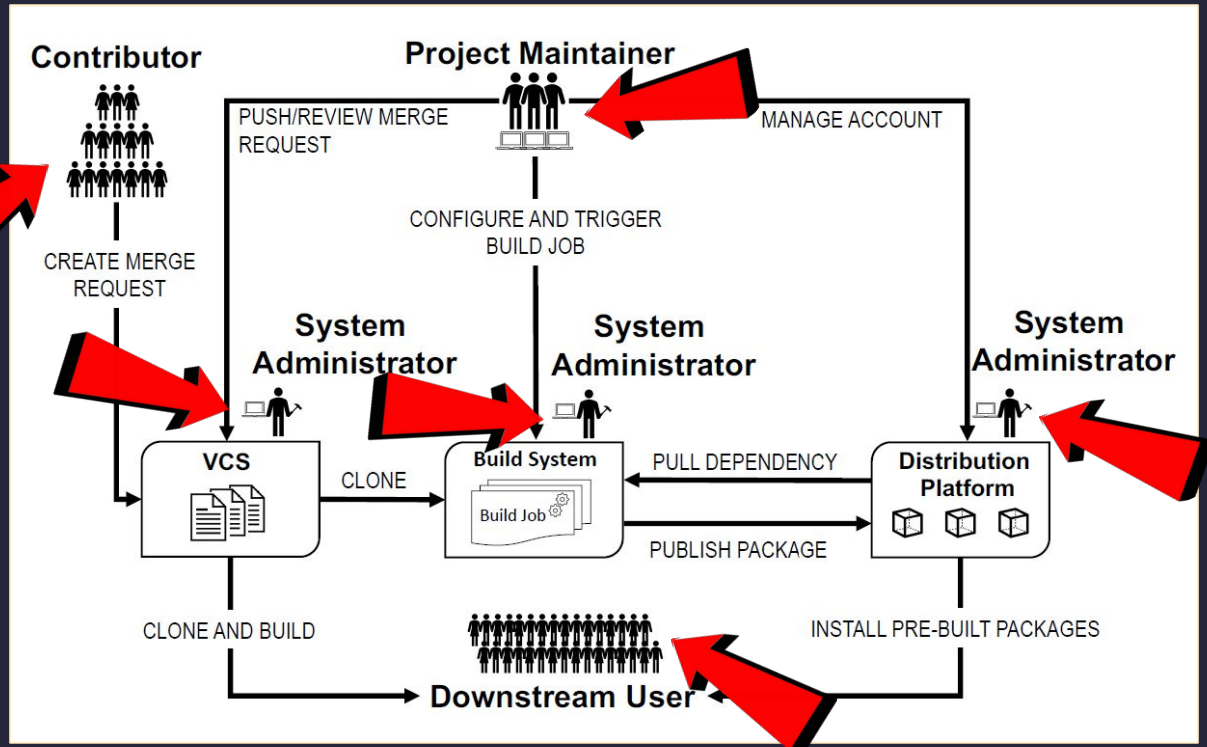
Source: https://slsa.dev/spec/v1.0/terminology

Software Supply Chain Threats



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

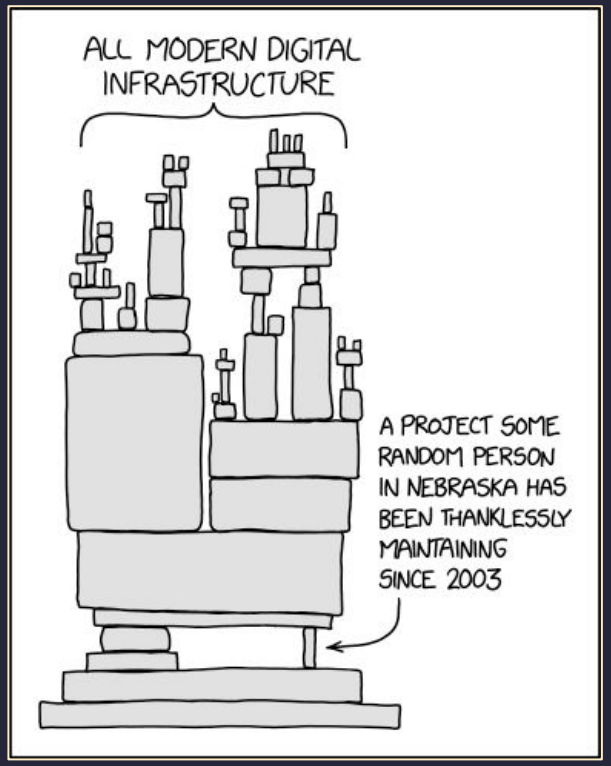
💻Software Development Lifecycle



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

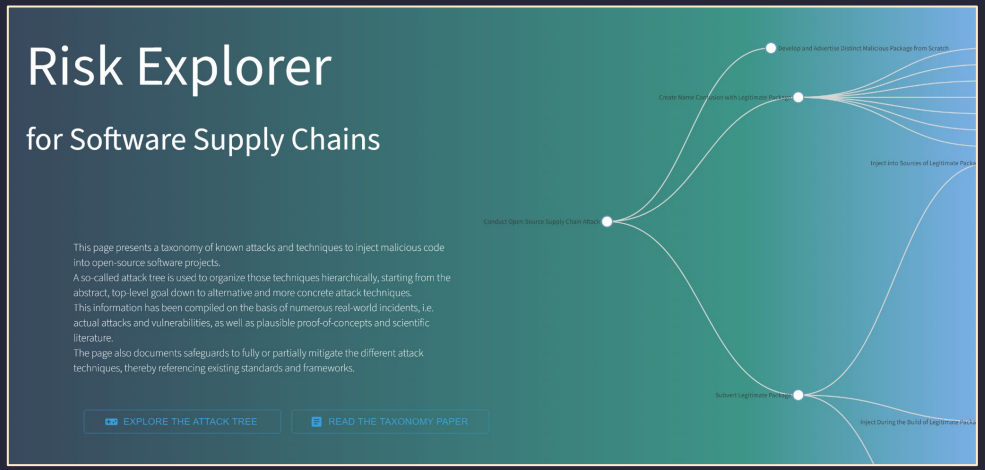
😐Misconceptions

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

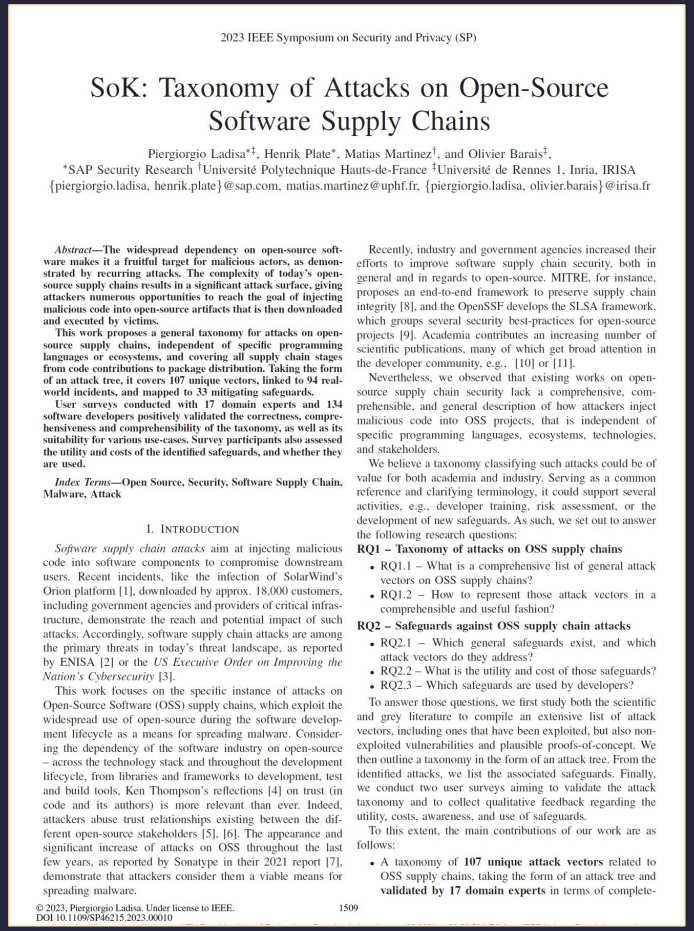


Source: https://xkcd.com/2347

🤔 Attack Vectors



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



2023 IEEE Symposium on Security and Privacy (SP)

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

Piergiorgio Ladisa*‡, Henrik Plate*, Matias Martinez†, and Olivier Barais‡,
*SAP Security Research †Université Polytechnique Hauts-de-France ‡Université de Rennes 1, Inria, IRISA
{piergiorgio.ladisa, henrik.plate}@sap.com, matias.martinez@uphf.fr, {piergiorgio.ladisa, olivier.barais}@irisa.fr

***Abstract*—The widespread dependency on open-source software makes it a fruitful target for malicious actors, as demonstrated by recurring attacks. The complexity of today's open-source supply chains results in a significant attack surface, giving attackers numerous 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 open-source supply chains, independent of specific programming languages or ecosystems, and covering all supply chain stages from code contributions to package distribution. Taking the form of an attack tree, it covers 107 unique vectors, linked to 94 real-world 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 attacks 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 Cybersecurity* [3].

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 lifecycle, 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.

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 open-source supply chain security lack a comprehensive, comprehensible, 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?
- RQ2.3 – 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 non-exploited 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.

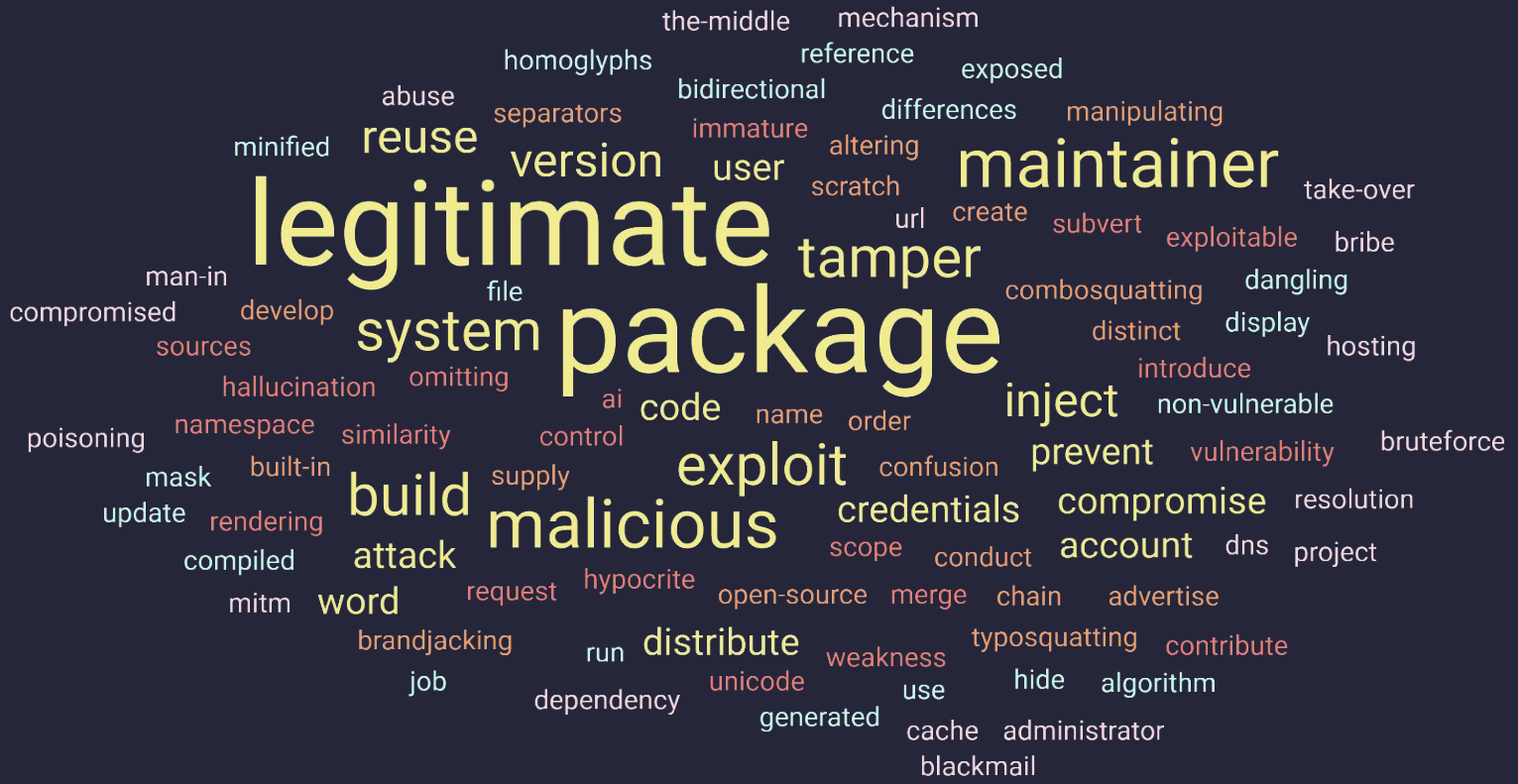
To this extent, the main contributions of our work are as follows:

- 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-

DOI 10.1109/SP46215.2023.00010

Source: "**Sok: Taxonomy of Attacks on Open-Source Software Supply Chains**" in *IEEE Symposium on Security and Privacy (SP), 2023*

🤦 Software Supply Chain Attacks



🤦 Software Supply Chain Attacks

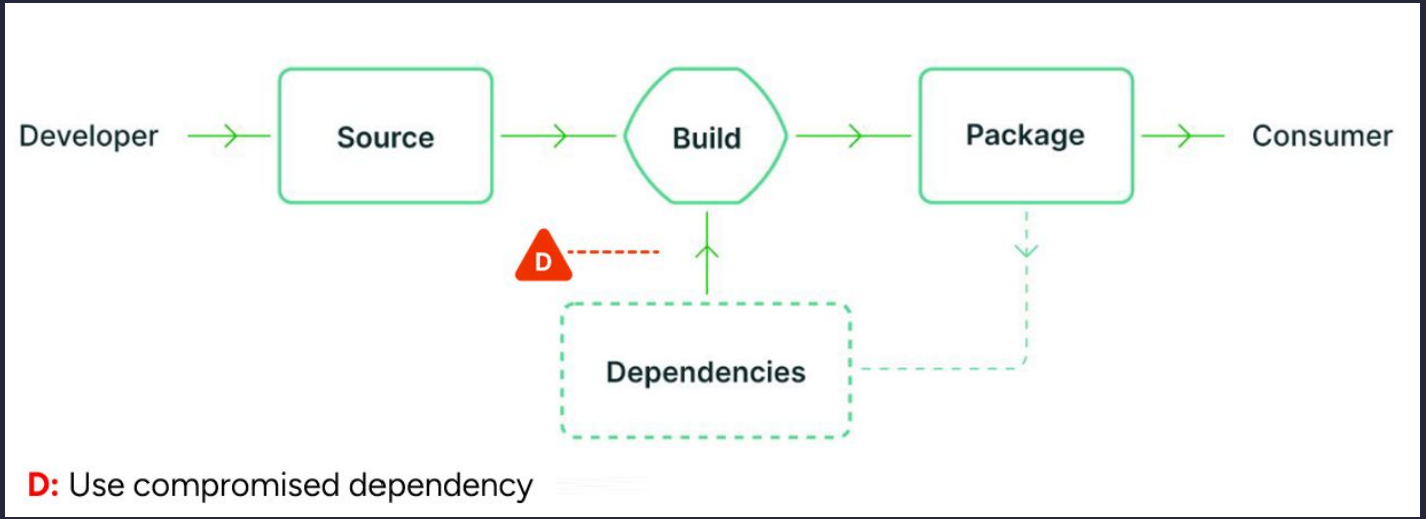
Third-party components can be compromised

Software is outdated or unpatched

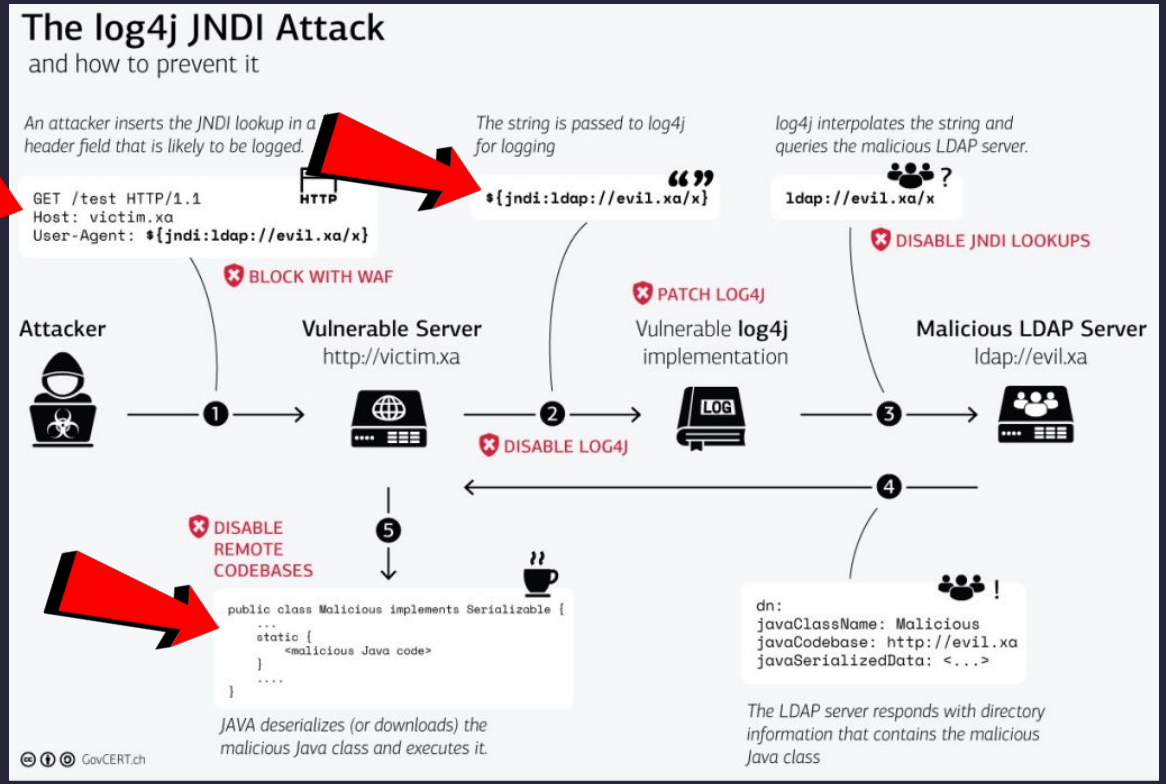
Lack of transparency

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

Log4Shell Vulnerability (2021)



Log4Shell Vulnerability (2021)



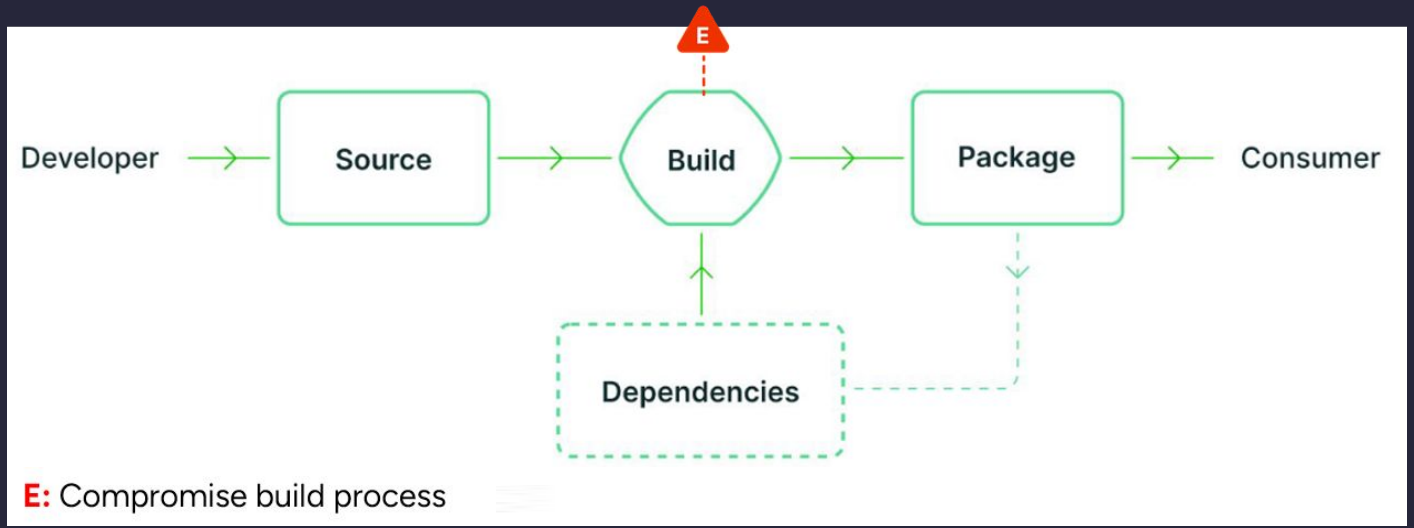
Source: [Swiss Government Computer Emergency Response Team]

Log4j Vulnerability (2021)

- Hidden risks of open-source dependencies
- Complex dependency trees
- It could have been detected using SBOMs

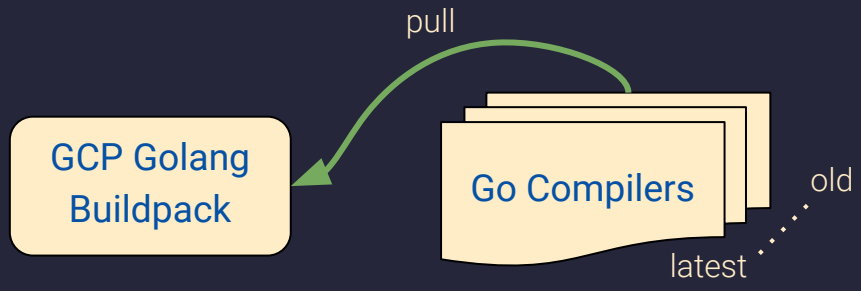
Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2021/log4j.md

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)




```
gcloud alpha builds submit --pack image=gcr.io/fkautz-dev/sample-go
===> ANALYZING
[analyzer]           image with name "gcr.io/571261452737/sample-go" not found
===> RESTORING
===> BUILDING
[builder] === Go - Runtime (google.go.runtime@0.9.1) ===
[builder] Using runtime version from go.mod: 1.14
[builder] Installing Go v1.14
[builder]
----------------------------------------------------------------------------
[builder] Running ...
```

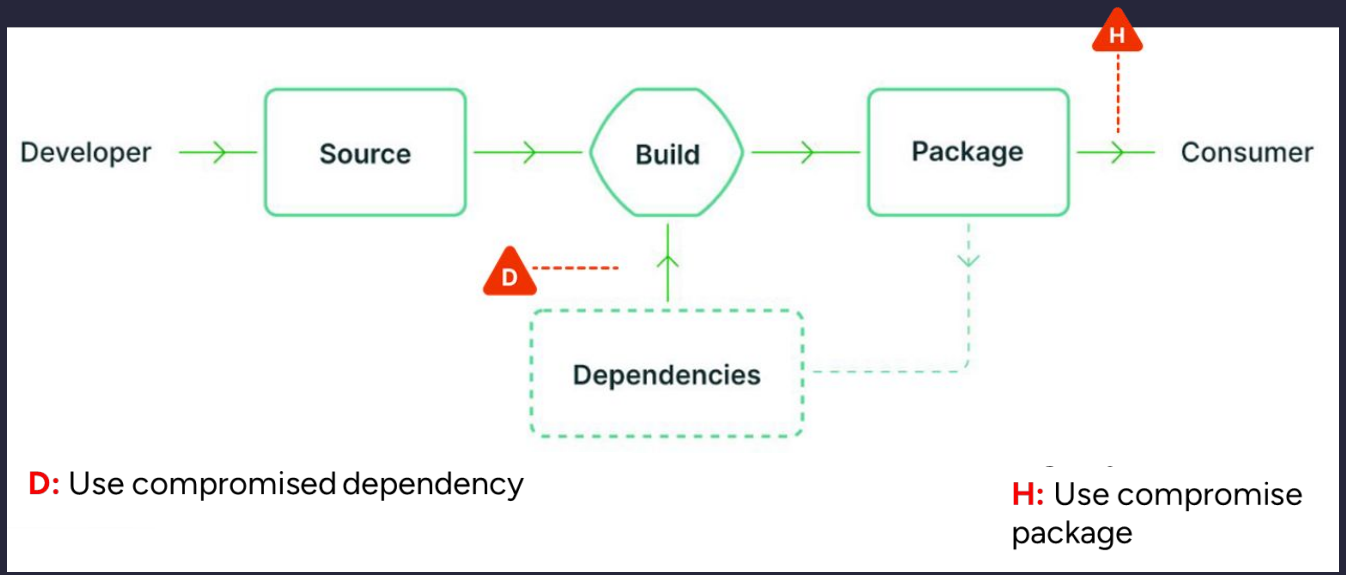
Source: https://zt.dev/posts/gcp-buildpacks-old-compiler

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)

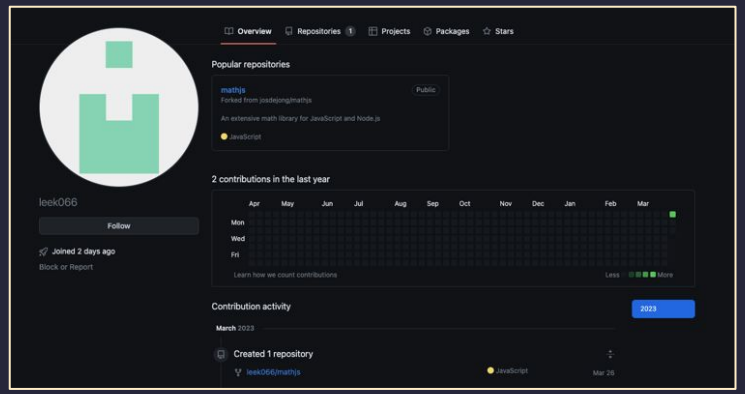


Source: https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2023/mathjs-min.md

NPM Package mathjs-min Credential Stealer (2023)

Brandjacking attack

Malicious GitHub user



publish

Malicious package

Source: https://blog.phylum.io/phylum-discovers-npm-package-mathjs-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?

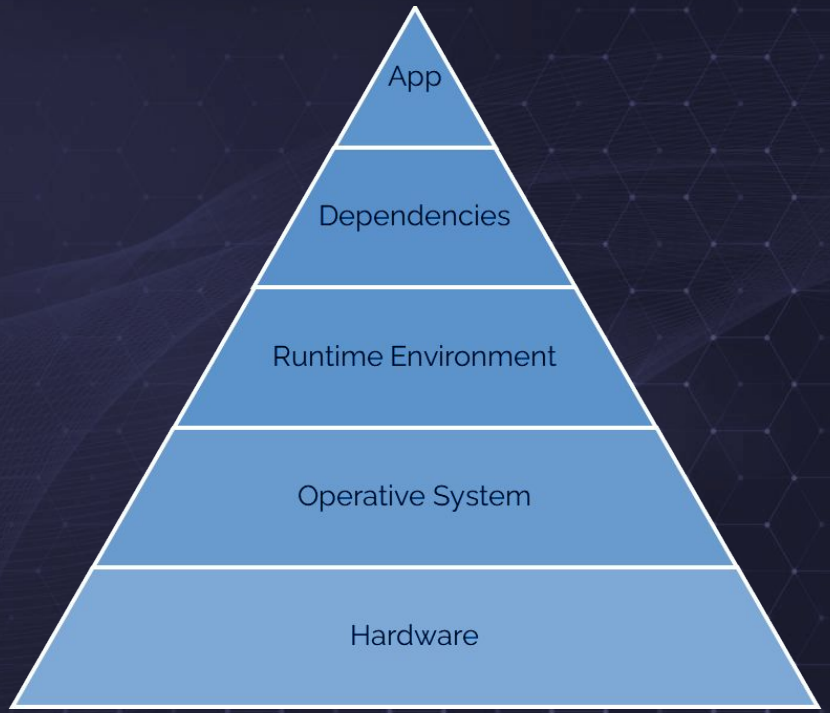
📜 SBOMs

"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."

National Telecommunications and Information Administration

Source: https://www.ntia.gov/sites/default/files/publications/sbom_at_a_glance_apr2021_0.pdf

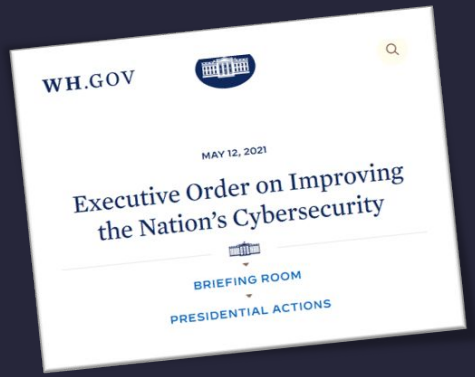
📜 Inventory of Software Components



🎯 Purpose of SBOMs

- Make software transparent
- Safeguard against software supply chain attacks

EU Cybersecurity Strategy (2020)



White House Executive Order (2021)

DARPA on Integrity of Open Source (2022)

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

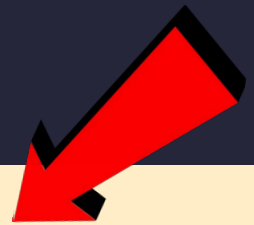
. . .

https://spdx.dev

https://cyclonedx.org

SBOM Metadata

CycloneDX



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

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"
   }
...
```

🔍 Checksums

Reproducible builds

async-http-client-project
v2.12.3
(JAR)

SHA-256

70997d52dbffd4e797c467b2062c6ba9b8b5a7380ff2884779f2743b6172ebb5

✓ Integrity

✗ Tampering

Third-Party Dependencies

```
"components" : [
   { "group" : "com.sun.activation",
     "name" : "jakarta.activation",
     "version" : "1.2.2",
     "bom-ref" :
"pkg:maven/com.sun.activation/jakarta.activation@1.2.2?type=jar"
   } ...
 ],
"dependencies" : [
   {
     "ref" :
"pkg:maven/org.a      ttpclient/async-http-client-project@2.12.3?type=pom",
     "dependsOn" : [
       "pkg:maven/com.sun.activation/jakarta.activation@1.2.2?type=jar"
       ....
     ]
   } ... ] }
```

📦Third-Party Dependencies

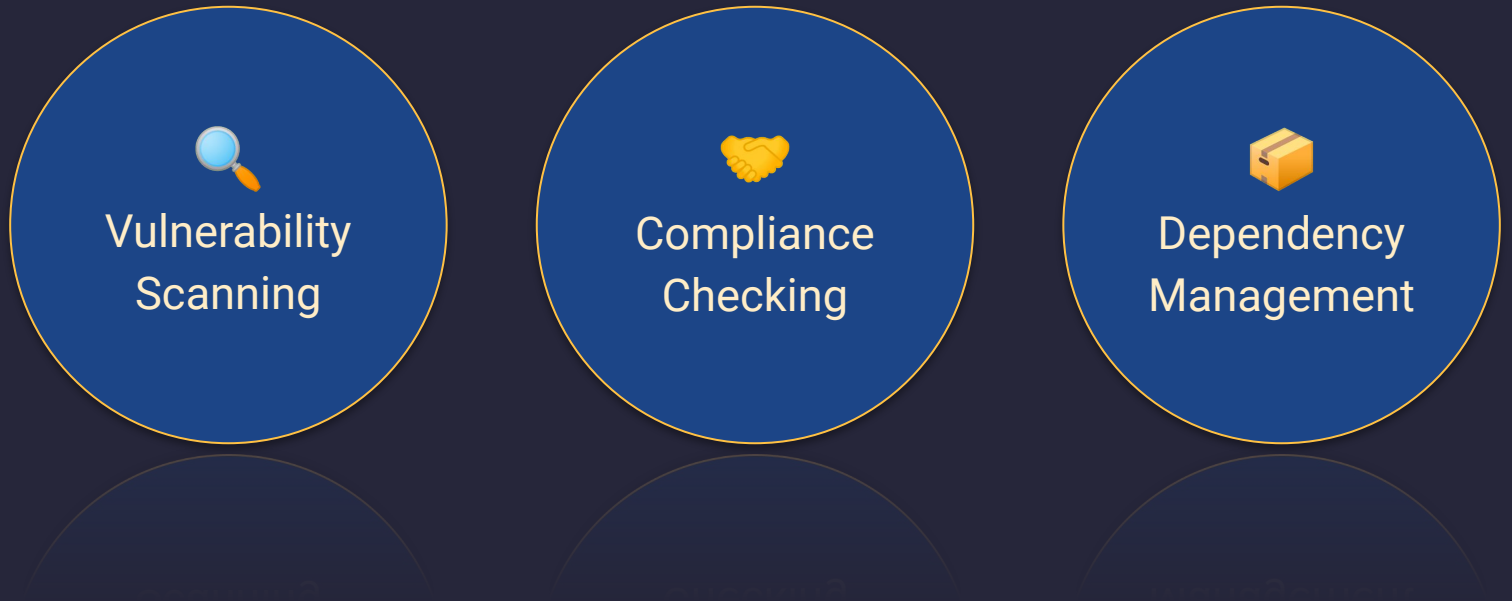
🤔Vulnerable?

🤔Do I have a licence?

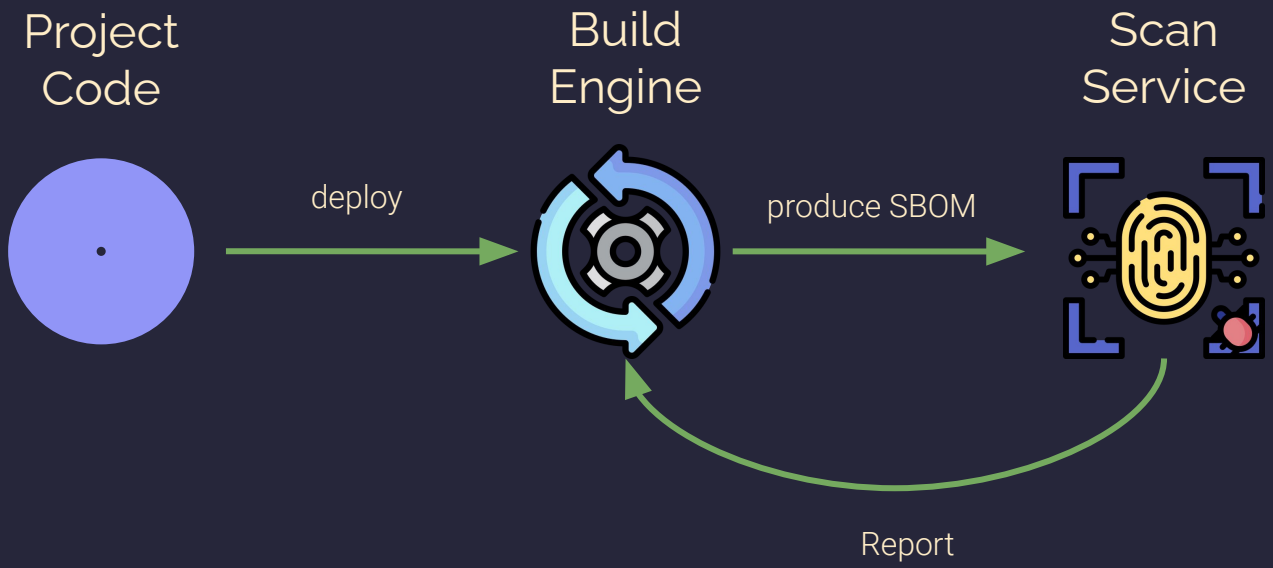
🤔Do I depend on this?

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

Use cases of SBOMs

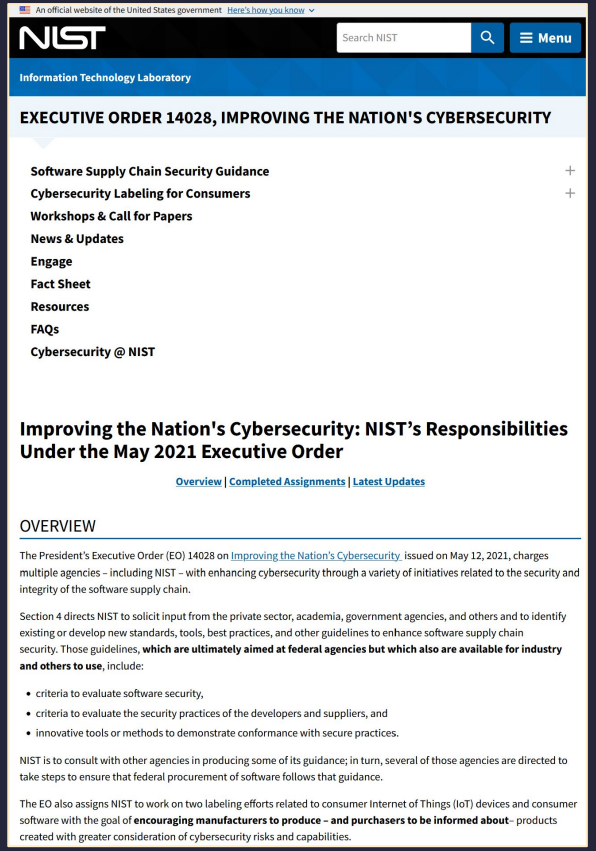

Vulnerability Scanning
Compliance Checking
Dependency Management

🔍 Vulnerability Scanning



🤝Compliance Checking

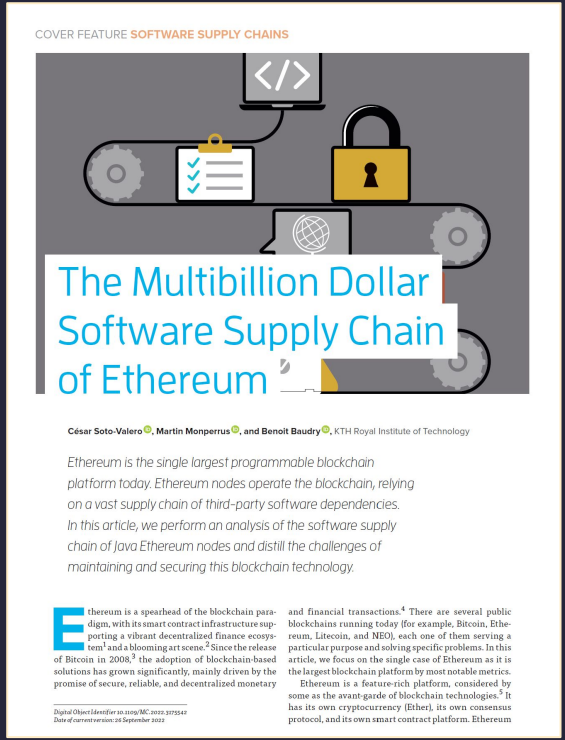
- Meet legal standards
- Check license compliance
- Better visibility



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

📦 Dependency Management

- Understand dependencies
- Detect outdated dependencies
- Reduce code bloat



COVER FEATURE **SOFTWARE SUPPLY CHAINS**

The Multibillion Dollar Software Supply Chain of Ethereum

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, relying 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.

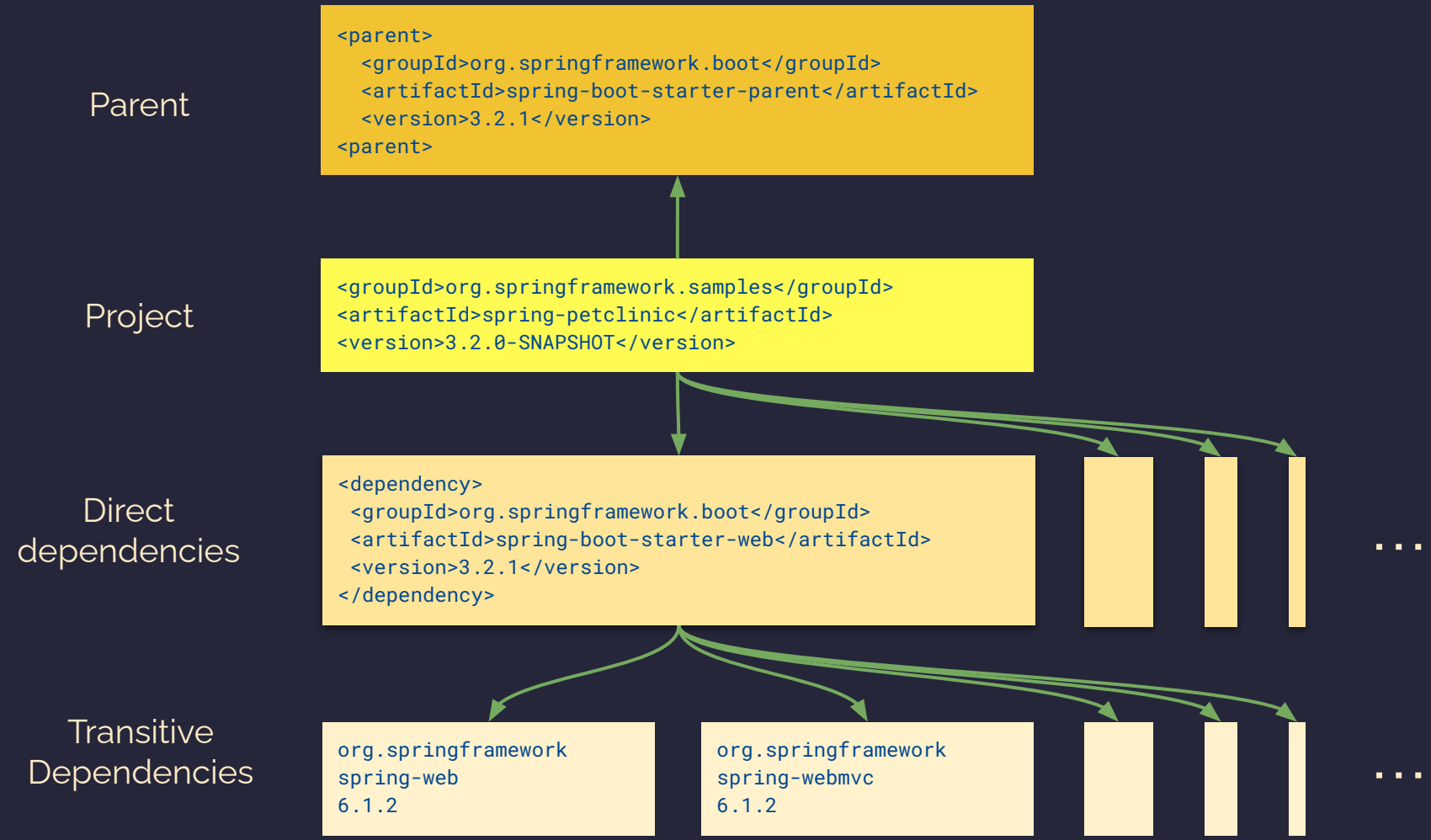
Ethereum is a spearhead of the blockchain paradigm, with its smart contract infrastructure supporting a vibrant decentralized finance ecosystem[1] and a blooming art scene.[2] Since the release of Bitcoin in 2008,[3] the adoption of blockchain-based solutions has grown significantly, mainly driven by the promise of secure, reliable, and decentralized monetary and financial transactions.[4] There are several public blockchains running today (for example, Bitcoin, Ethereum, Litecoin, and NEO), each one of them serving a particular purpose and solving specific problems. In this article, we focus on the single case of Ethereum as it is the largest blockchain platform by most notable metrics.

Ethereum is a feature-rich platform, considered by some as the avant-garde of blockchain technologies.[5] It has its own cryptocurrency (Ether), its own consensus protocol, and its own smart contract platform. Ethereum

Digital Object Identifier 10.1109/MC.2022.3175542
Date of current version: 26 September 2022

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

Java dependencies


Parent
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.2.1</version>
<parent>
Project
<groupId>org.springframework.samples</groupId>
<artifactId>spring-petclinic</artifactId>
<version>3.2.0-SNAPSHOT</version>
Direct dependencies
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>3.2.1</version>
</dependency>
Transitive Dependencies
org.springframework
spring-web
6.1.2
org.springframework
spring-webmvc
6.1.2

Package Managers (Maven)


```
$ mvn dependency:tree
```

```
$ mvn dependency:tree -DoutputType=dot -DoutputFile=deps.dot
$ dot -Txdot_json -o deps.json deps.dot
```

```
[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] |  +- io.micrometer:micrometer-observation:jar:1.12.1:compile
[INFO] |  |  \- io.micrometer:micrometer-commons:jar:1.12.1:compile
[INFO] |  \- io.micrometer:micrometer-jakarta9:jar:1.12.1:compile
[INFO] |     \- io.micrometer:micrometer-core:jar:1.12.1:compile
[INFO] |        +- org.hdrhistogram:HdrHistogram:jar:2.1.12:runtime
[INFO] |        \- org.latencyutils:LatencyUtils:jar:2.0.3:runtime
```

Terminal output


```
{
  "name": "org.springframework.samples:spring-petclinic:jar:3.2.0-SNAPSHOT",
  "directed": true,
  "strict": false,
  "_subgraph_cnt": 0,
  "objects": [
    {
      "_gvid": 0,
      "name": "org.springframework.samples:spring-petclinic:jar:3.2.0-SNAPSHOT",
      "label": "\\N"
    },
    {
      "_gvid": 1,
      "name": "org.springframework.boot:spring-boot-starter-actuator:jar:3.2.1:compile",
      "label": "\\N"
    },
    {
      "_gvid": 2,
      "name": "org.springframework.boot:spring-boot-starter-cache:jar:3.2.1:compile",
      "label": "\\N"
    },
    {
      "_gvid": 3,
      "name": "org.springframework.boot:spring-boot-starter-data-jpa:jar:3.2.1:compile",
      "label": "\\N"
    },
```


JSON file from the previous output

Package Managers (Gradle)


```
$ gradle dependencies --configuration runtimeClasspath
```

```
> Task :dependencies

------------------------------------------------------------
Root project 'spring-petclinic'
------------------------------------------------------------

runtimeClasspath - Runtime classpath of source set 'main'.
+--- org.springframework.boot:spring-boot-devtools -> 3.2.0
|    +--- org.springframework.boot:spring-boot:3.2.0
|    |    +--- org.springframework:spring-core:6.1.1
|    |    |    \--- org.springframework:spring-jcl:6.1.1
|    |    \--- org.springframework:spring-context:6.1.1
|    |         +--- org.springframework:spring-aop:6.1.1
|    |         |    +--- org.springframework:spring-beans:6.1.1
|    |         |    |    \--- org.springframework:spring-core:6.1.1 (*)
|    |         |    \--- org.springframework:spring-core:6.1.1 (*)
|    |         +--- org.springframework:spring-beans:6.1.1 (*)
|    |         +--- org.springframework:spring-core:6.1.1 (*)
|    |         +--- org.springframework:spring-expression:6.1.1
|    |         |    \--- org.springframework:spring-core:6.1.1 (*)
|    |         \--- io.micrometer:micrometer-observation:1.12.0
|    |              \--- io.micrometer:micrometer-commons:1.12.0
|    \--- org.springframework.boot:spring-boot-autoconfigure:3.2.0
|         \--- org.springframework.boot:spring-boot:3.2.0 (*)
+--- org.springframework.boot:spring-boot-starter-cache -> 3.2.0
|    +--- org.springframework.boot:spring-boot-starter:3.2.0
|    |    +--- org.springframework.boot:spring-boot:3.2.0 (*)
|    |    +--- org.springframework.boot:spring-boot-autoconfigure:3.2.0 (*)
|    |    +--- org.springframework.boot:spring-boot-starter-logging:3.2.0
|    |    |    +--- ch.qos.logback:logback-classic:1.4.11
|    |    |    |    +--- ch.qos.logback:logback-core:1.4.11
|    |    |    |    \--- org.slf4j:slf4j-api:2.0.7 -> 2.0.9
```

Terminal output

Closed

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 SBOM.
The easiest format for that is CycloneDX. It's still JSON, but in a way that other tools natively understand.
There's a Java library to create it.

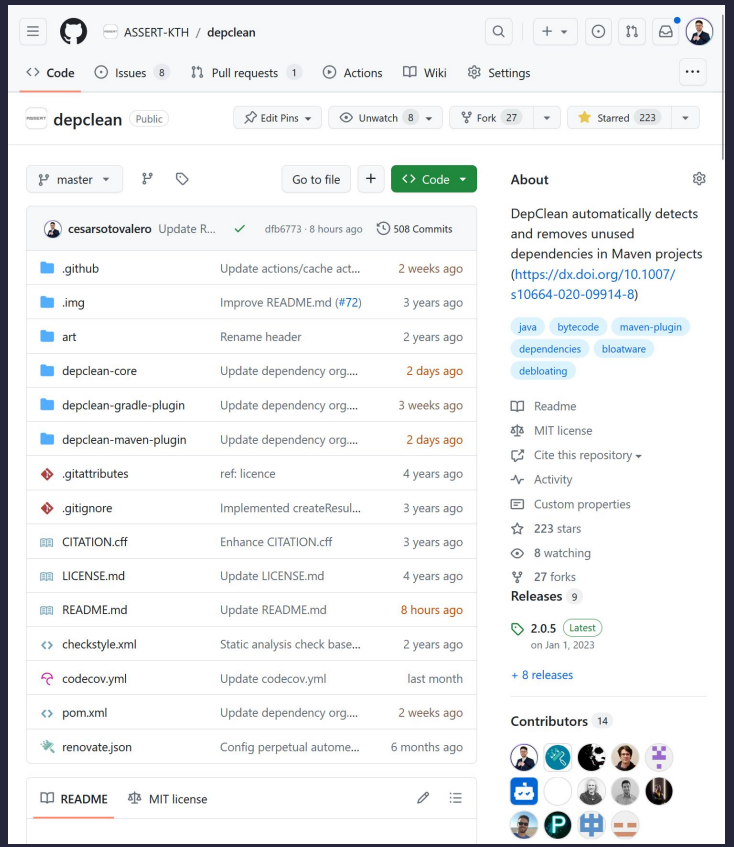
Source: https://github.com/gradle/gradle/issues/21894

📦 Dedicated Tools

DepClean

- Extra metadata from dependencies:
  - Direct / Transitive / Inherited
  - Used / Unused
  - JAR size
- Debloated **pom.xml**

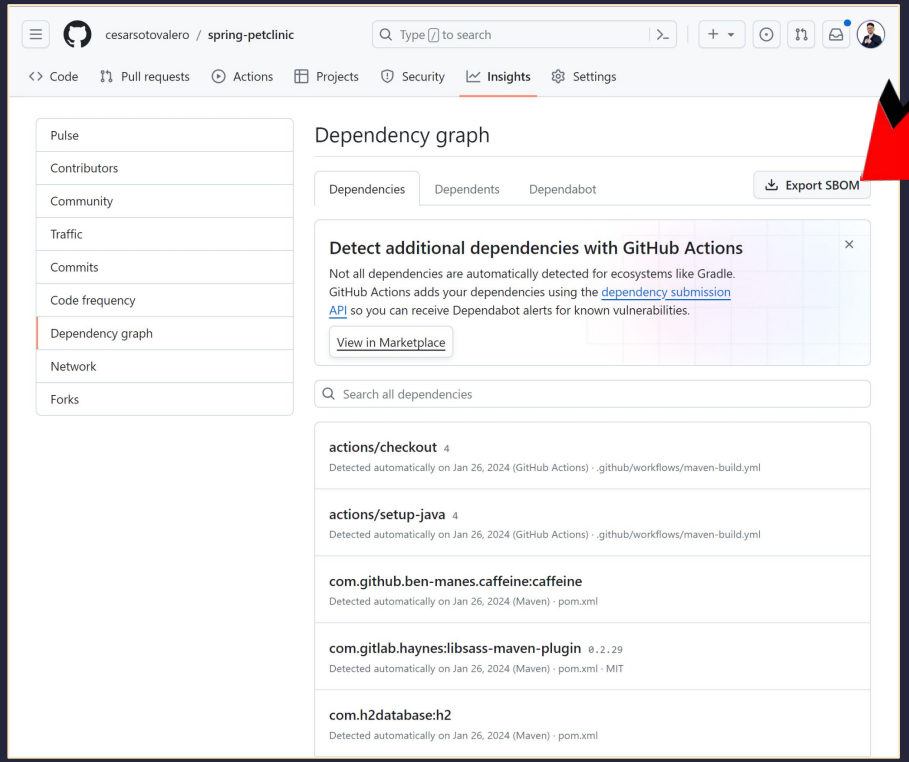
...



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

SBOM producers for Java projects

GitHub UI

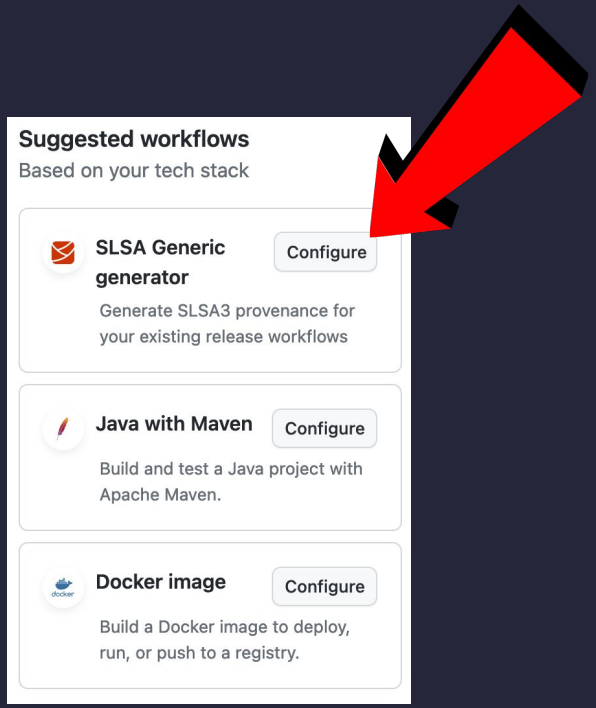


Output

```
{
  "SPDXID": "SPDXRef-DOCUMENT",
  "spdxVersion": "SPDX-2.3",
  "creationInfo": {
    "created": "2024-01-27T20:03:37Z",
    "creators": [
      "Tool: GitHub.com-Dependency-Graph"
    ],
    "comment": "Exact versions could not be resolved for some packages. For more information
  },
  "name": "com.github.cesarsotovalero/spring-petclinic",
  "dataLicense": "CC0-1.0",
  "documentDescribes": [
    "SPDXRef-com.github.cesarsotovalero-spring-petclinic"
  ],
  "documentNamespace": "https://github.com/cesarsotovalero/spring-petclinic/dependency_graph/
  "packages": [
    {
      "SPDXID": "SPDXRef-com.github.cesarsotovalero-spring-petclinic",
      "name": "com.github.cesarsotovalero/spring-petclinic",
      "versionInfo": "",
      "downloadLocation": "git+https://github.com/cesarsotovalero/spring-petclinic",
      "licenseDeclared": "Apache-2.0",
      "filesAnalyzed": false,
      "supplier": "NOASSERTION",
      "externalRefs": [
        {
          "referenceCategory": "PACKAGE-MANAGER",
          "referenceType": "purl",
          "referenceLocator": "pkg:github/cesarsotovalero/spring-petclinic"
        }
      ]
    },
    {
      "SPDXID": "SPDXRef-maven-com.github.ben-manes.caffeine-caffeine",
      "name": "maven:com.github.ben-manes.caffeine:caffeine",
      "versionInfo": "",
      "downloadLocation": "NOASSERTION",
      "filesAnalyzed": false,
      "supplier": "NOASSERTION"
```

Source: https://github.blog/2023-03-28-introducing-self-service-sboms

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          base64-subjects: "${{ needs.build.outputs.digests }}"
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

. . .


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

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

GraalVM

⚠️ Only on GraalVM Enterprise

```
$ native-image . . . -H:IncludeSBOM=cyclonedx
```

Build executable

Native image with SBOM

Get SBOM

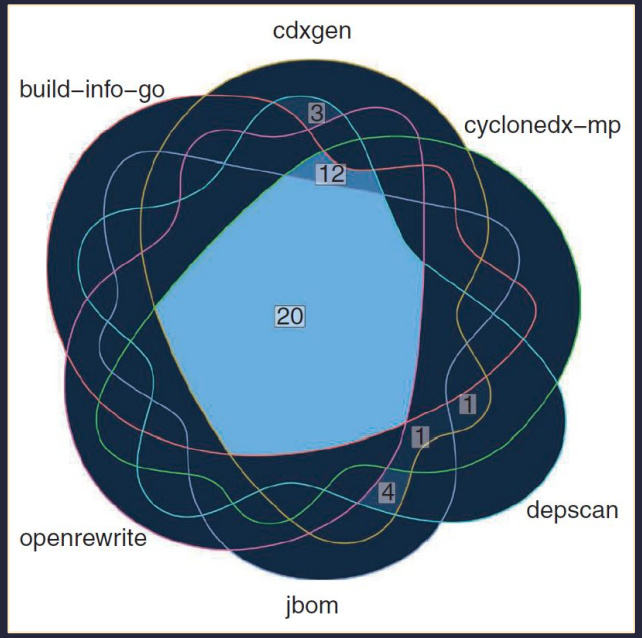
```
$ native-image-inspect --sbom <path_to_binary>
```

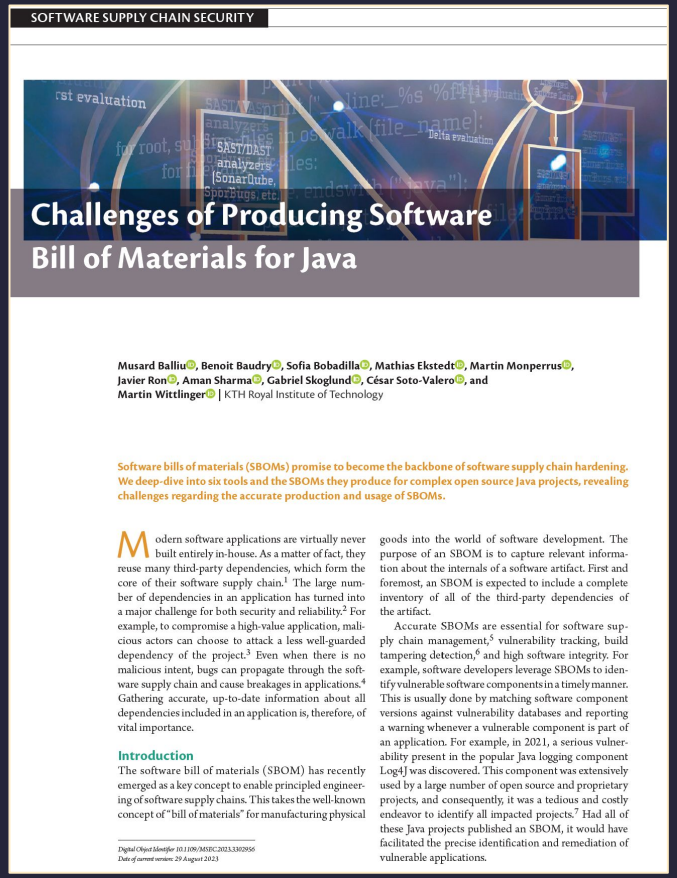
```
<plugin>
  <groupId>org.graalvm.buildtools</groupId>
  <artifactId>native-maven-plugin</artifactId>
  <version>${graalvm.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>build</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <buildArgs>
      <buildArg>-H:IncludeSBOM=cyclonedx</buildArg>
    </buildArgs>
  </configuration>
</plugin>
```

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

Challenges with SBOMs

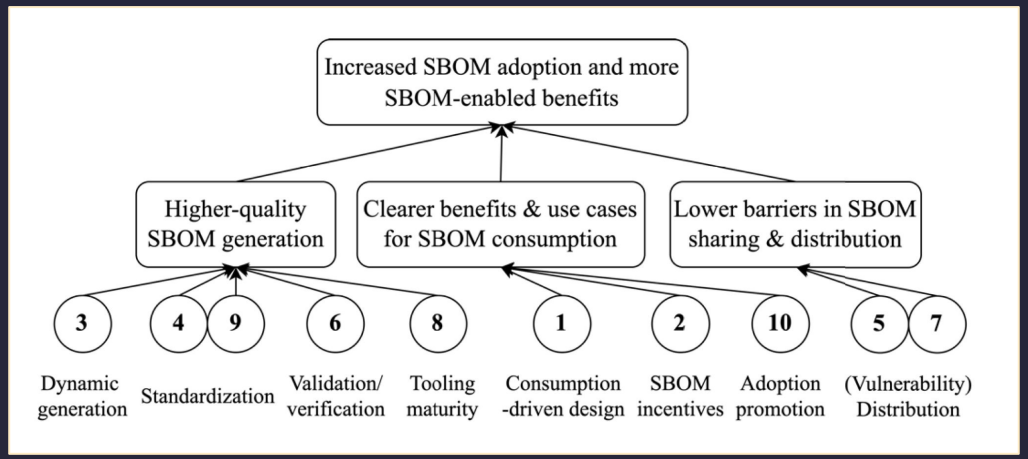
Tool Consistency





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

Sharing & Distribution



2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)

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

Boming Xia*†, Tingting Bi*‡, Zhenchang Xing*§, Qinghua Lu*†, Liming Zhu*†
*CSIRO's Data61, Sydney, Australia
†University of New South Wales, Sydney, Australia
‡Monash University, Melbourne, Australia
§Australian National University, Canberra, Australia

***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 (SSC) have increased significantly (e.g., SolarWinds attack [2]). According to Sonatype's report [3], there was a 650% year-over-year increase in SSC attacks in 2021, and the number was 430% in 2020. SSC attacks mainly aim at the upstream open source software/components (OSS) [4], yet OSS is heavily relied upon in software development [5]. The reliance on OSS leads to additional risks, such as the lack of reliable maintenance and support compared to proprietary software/components [6]. The security risks of software and its supply chain call for improved visibility into the SSC, with which timely and accurate identification of the impacted software/components could be carried out in case of a vulnerability or an SSC attack.

A software bill of materials (SBOM) is a formal machine-readable 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 transparency. According to Linux Foundation's SBOM and Cybersecurity Readiness report (SBOM readiness report for short) [5], SBOMs are critical 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 13% growth in SBOM production or consumption 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 SBOMs 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 necessity/availability/usability/integrity 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

Source: "**An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead**," in *ICSE, 2023*

Wrapping up...

Source: DALL·E

👋Summary

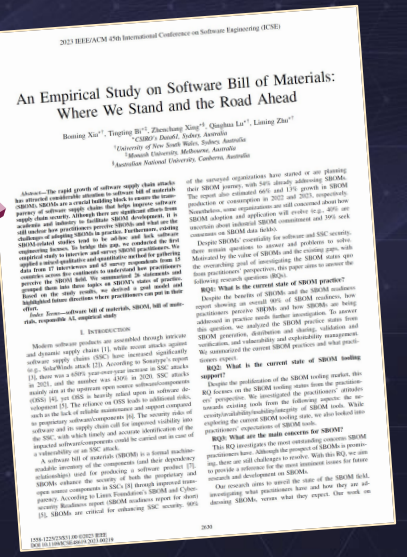
- SBOMs are essential for software supply chain security
- SBOMs are increasingly getting attention and adoption
- SBOMs are well supported in the Java ecosystem

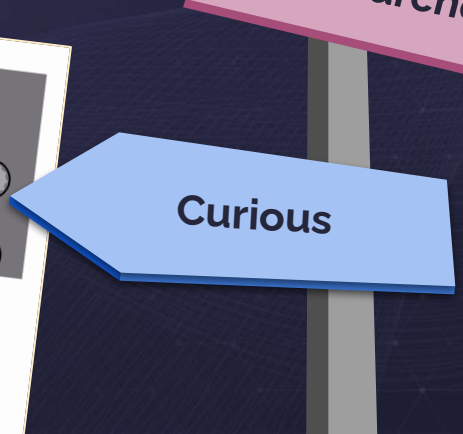
What to read next ❓

Business

Practitioners

Researchers





Curious