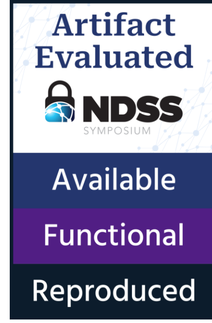
NDSS 2025, San Diego



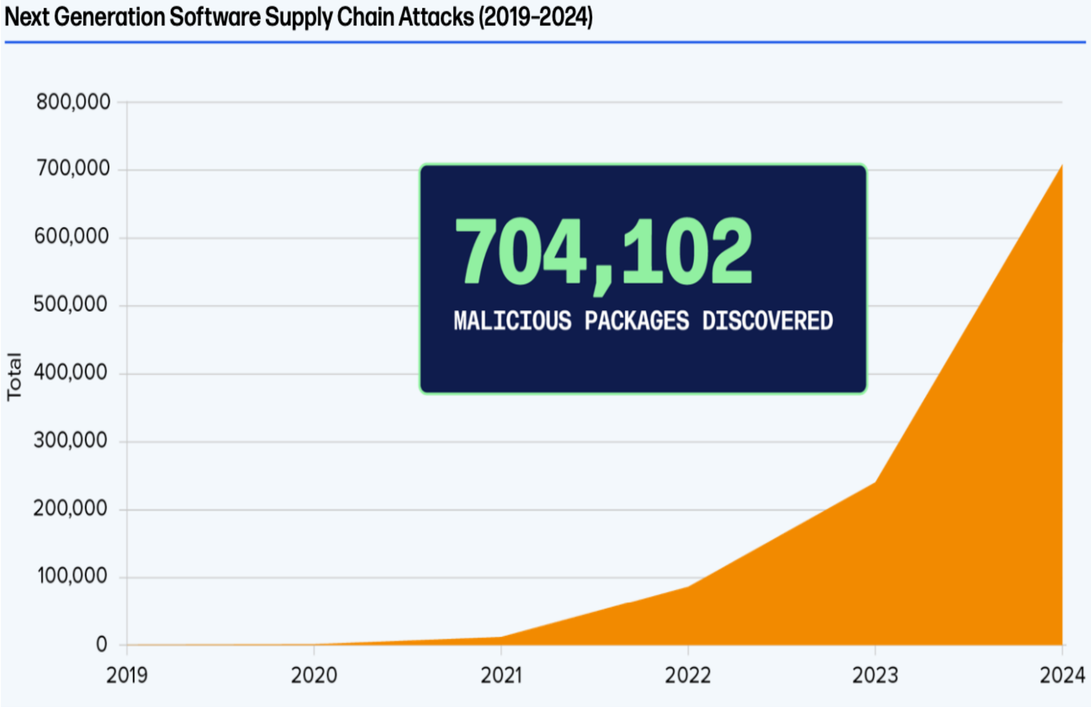
JBomAudit: Assessing the Landscape, Compliance, and Security Implications of Java SBOMs

Yue Xiao, Dhilung Kirat, Douglas Lee Schales, Jiyong Jang, Luyi Xing, Xiaojing Liao

IBM Research & Indiana University Bloomington

The Growing Threats of Supply Chain Attacks

Rising Trend of Supply Chain Attacks



Recent Severe Incidents

EQUIFAX

2017: The Equifax Breach and the Rise of Targeted Supply Chain attacks

solarwinds

2020: SolarWinds and the Expansion of Supply Chain Attacks

LOG4J

2021–2022: Log4Shell, the Vulnerability that Set the Internet on Fire

XZ Utils

2024: The Attempted XZ-Utils Supply Chain Attack

https://www.wired.com/story/hacker-lexicon-what-is-a-supply-chain-attack/
https://www.comparitech.com/software-supply-chain-attacks/
https://www.sonatype.com/state-of-the-software-supply-chain/2023/open-source-supply-and-demand

Supply Chain Transparency and SBOM Motivation

Lack of transparency

Two fundamental questions

How can we enhance transparency within software systems?

What strategies can effectively detect and prevent software supply chain attacks?

The SBOM Solution

Executive Order 14028 — Improving The Nation's Cybersecurity

National Cyber Security Centre

IMPROVING EFFICIENCY AND RESILIENCY IN CANADA'S SUPPLY CHAINS

SBOMs & Use Cases

Listing 1: SBOM Example of `flink-json`

```
{
  "bomFormat" : "CycloneDX",
  "specVersion" : "1.4",
  "serialNumber" : "urn:uuid:3db22509-0440-45f0-9ebd...",
  "version" : 1,
  "metadata" : {
    "timestamp" : "2024-03-06T14:55:56Z", ... }
  "components": :[{
      "group" : "com.ibm.icu",
      "name" : "icu4j",
      "version" : "73.2",
      "purl":"pkg:maven/com.ibm.icu/icu4j@73.2?type=jar"...]
  "dependencies": [
    {
      "ref" : "pkg:maven/org.apache.flink/flink-json@1.19.0?
           type=jar",
      "dependsOn" : [
        "pkg:maven/org.apache.flink/flink-table-common@1
             .19.0?type=jar", ...]
    }, {
      "ref": "pkg:maven/org.apache.flink/flink-table-
           common@1.19.0?type=jar",
      "dependsOn": [
        "pkg:maven/com.ibm.icu/icu4j@67.1?type=jar", ... ]
    }, ... ]
}
```

Vulnerability Management

dependency track

JFrog XRAY

aqua trivy

Non-Complaint SBOMs in Java Ecosystem

The Minimum Elements For a Software Bill of Materials (SBOM)

“An SBOM should contain all primary (top-level) components, along with all their transitive dependencies (second-level)”

```
ClassLoader loader = VfsUtils.class.getClassLoader();
try {
    Class<?> vfsClass = loader.loadClass("org.jboss.vfs.VFS");
    vfsMethodGetRootUrl = vfsClass.getMethod("getChild", URL.class);
    vfsMethodGetRootUri = vfsClass.getMethod("getChild", URI.class);
    Class<?> virtualFile = loader.loadClass("org.jboss.vfs.VirtualFile");
    virtualFileMethodExists = virtualFile.getMethod("exists", new Class[0]);
    virtualFileMethodGetInputStream = virtualFile.getMethod("openStream", new Class[0]);
    virtualFileMethodGetSize = virtualFile.getMethod("getSize", new Class[0]);
    virtualFileMethodGetLastModified = virtualFile.getMethod("getLastModified", new Class[0]);
    virtualFileMethodToUri = virtualFile.getMethod("toURI", new Class[0]);
    virtualFileMethodToUrl = virtualFile.getMethod("toURL", new Class[0]);
    virtualFileMethodGetName = virtualFile.getMethod("getName", new Class[0]);
    virtualFileMethodGetPathName = virtualFile.getMethod("getPathName", new Class[0]);
    virtualFileMethodGetPhysicalFile = virtualFile.getMethod("getPhysicalFile", new Class[0]);
    virtualFileMethodGetChild = virtualFile.getMethod("getChild", String.class);
    virtualFileVisitorInterface = loader.loadClass("org.jboss.vfs.VirtualFileVisitor");
    virtualFileMethodVisit = virtualFile.getMethod("visit", virtualFileVisitorInterface);
    Class<?> visitorAttributesClass = loader.loadClass("org.jboss.vfs.VisitorAttributes");
    visitorAttributesFieldRecurse = visitorAttributesClass.getField("RECURSE");
} catch (Throwable th) {
```

VfsUtils.class in JAR

Missing

org.jboss.vfs Aa .ab .* No results

```
{
  "bomFormat" : "CycloneDX",
  "specVersion" : "1.4",
  "serialNumber" : "urn:uuid:d54dbe6d-2dd9-3283-a4b9-585f752934bd",
  "version" : 1,
  "metadata" : {…
  },
  "components" : […
  ],
  "dependencies" : [
    {
Click to collapse the range.  :maven/nl.basjes.parse.useragent/yauaa@7.26.0?type=jar",
      "dependsOn" : [
        "pkg:maven/org.antlr/antlr4-runtime@4.13.1?type=jar",
        "pkg:maven/org.yaml/snakeyaml@2.2?type=jar",
        "pkg:maven/org.apache.commons/commons-text@1.11.0?type=jar",
        "pkg:maven/org.apache.commons/commons-lang3@3.14.0?type=jar",
        "pkg:maven/com.github.ben-manes.caffeine/caffeine@3.1.8?type=jar",
        "pkg:maven/org.apache.commons/commons-collections4@4.4?type=jar",
        "pkg:maven/nl.basjes.collections/prefixmap@2.0?type=jar",
        "pkg:maven/org.projectlombok/lombok@1.18.30?type=jar",
        "pkg:maven/org.apache.logging.log4j/log4j-core@2.23.1?type=jar",
        "pkg:maven/com.esotericsoftware/kryo@5.6.0?type=jar",
        "pkg:maven/com.google.code.findbugs/jsr305@3.0.2?type=jar"
      ]
```

SBOM

Our Work

- 25,882 SBOMs and JARs
- June 2023 to April 2024

A Formalized Consistency Model

We identified six specific types of dependency inconsistencies that do not satisfy NTIA requirements, each reflecting a certain granularity of discrepancy between the declared dependency relationships in SBOMs and the actual dependency relationships in code.

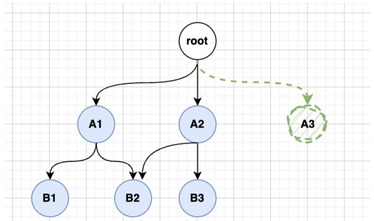
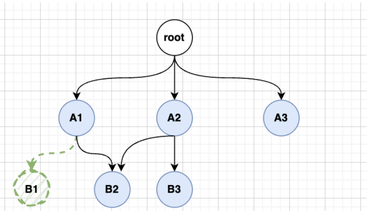
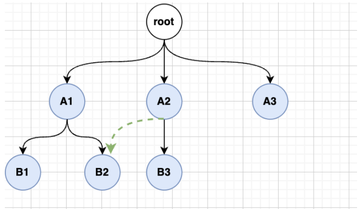
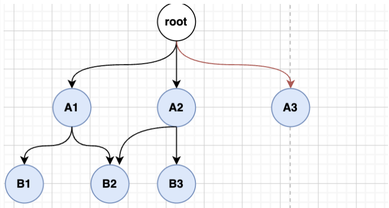
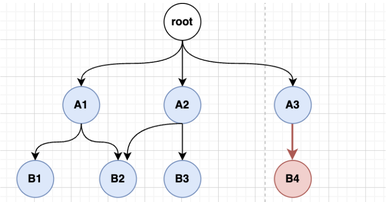
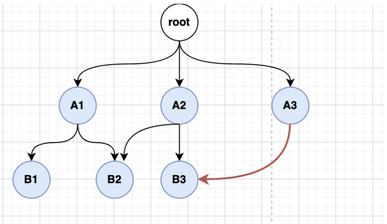
An End to End Detection Tool

We propose an end-to-end implementation, called JBomAudit, that automatically assesses the correctness and completeness of dependencies in SBOMs

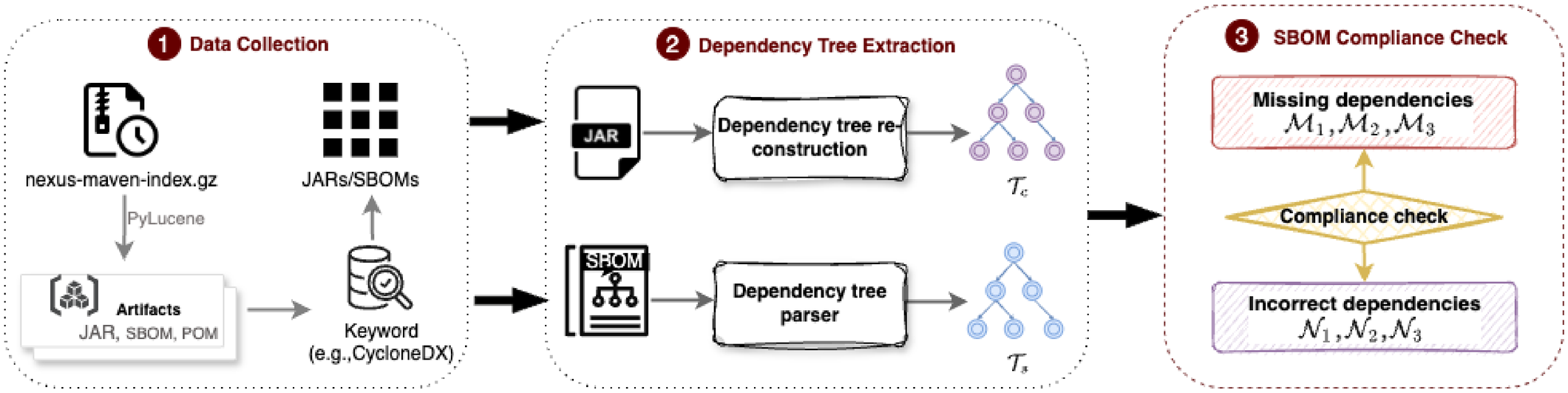
A Large-scale Non-compliance measurement

We conducted comprehensive measurement studies on SBOM non-compliance issues, investigating root causes, and analyzing their security implications.

A Formalized Consistency Model

Type	Description	Security Implication	Virtualization for Different Fine-grained Types
Missing Dependencies	*Missing dependencies* are components used in the distributed code but not listed in the SBOM, leading to potential security oversights.	If missing dependencies contain vulnerabilities that are not tracked due to their absence in the SBOM, the resulting security gaps can remain unpatched, leaving the software open for exploitation.	 **M1: Missing direct dependency**  **M2: Missing transitive dependency**  **M:3 Missing transitive relationship**
Incorrect Dependencies	Incorrect dependencies are listed in the SBOM but not used in the distributed code	Resources may be wasted while trying to investigate and fix vulnerabilities for non-existent components. This not only wastes time and resources but also diverts attention and efforts away from real threats and potentially delays the mitigation of critical security risks.	 **N1: Incorrect direct dependency**  **N2: Incorrect transitive dependency**  **N3: Incorrect transitive relationship**

An End-to-End Detection Tool



JarPkgTags

Utilized Java Classes Extraction

- External classes
- Method parameters
- Annotations
- Dynamic features

Dependency Tree Construction

1. Class Name to Package Name Transformation
2. Package Name to Dependency Mapping
3. Iteratively Construct Dependency Tree

Non-Compliant SBOMs Detected by *JBOMAudit*

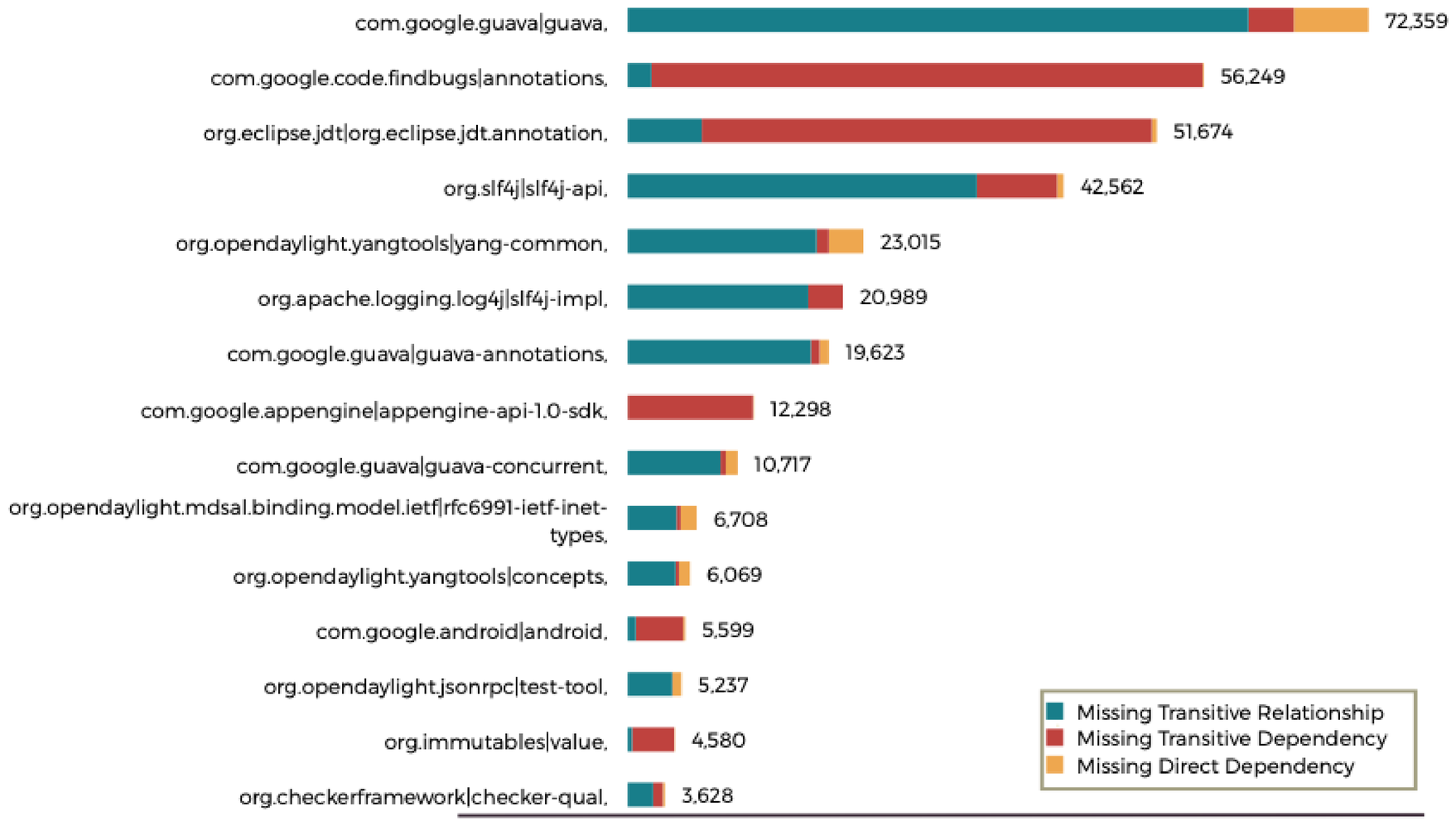
TABLE VI: Overall results of *JBomAudit*

Type	#SBOMs	#Dependencies	Average	Std
$\mathcal{M}_1$	7,907	48,931	6.18	10.95
$\mathcal{M}_2$	23,362	309,286	13.23	20.65
$\mathcal{M}_3$	21,665	365,897	16.88	36.33
$\mathcal{N}_1$	19,404	86,483	4.45	3.81
$\mathcal{N}_2$	6,140	14,830	2.41	2.02
$\mathcal{N}_3$	11,168	101,745	9.11	18.25

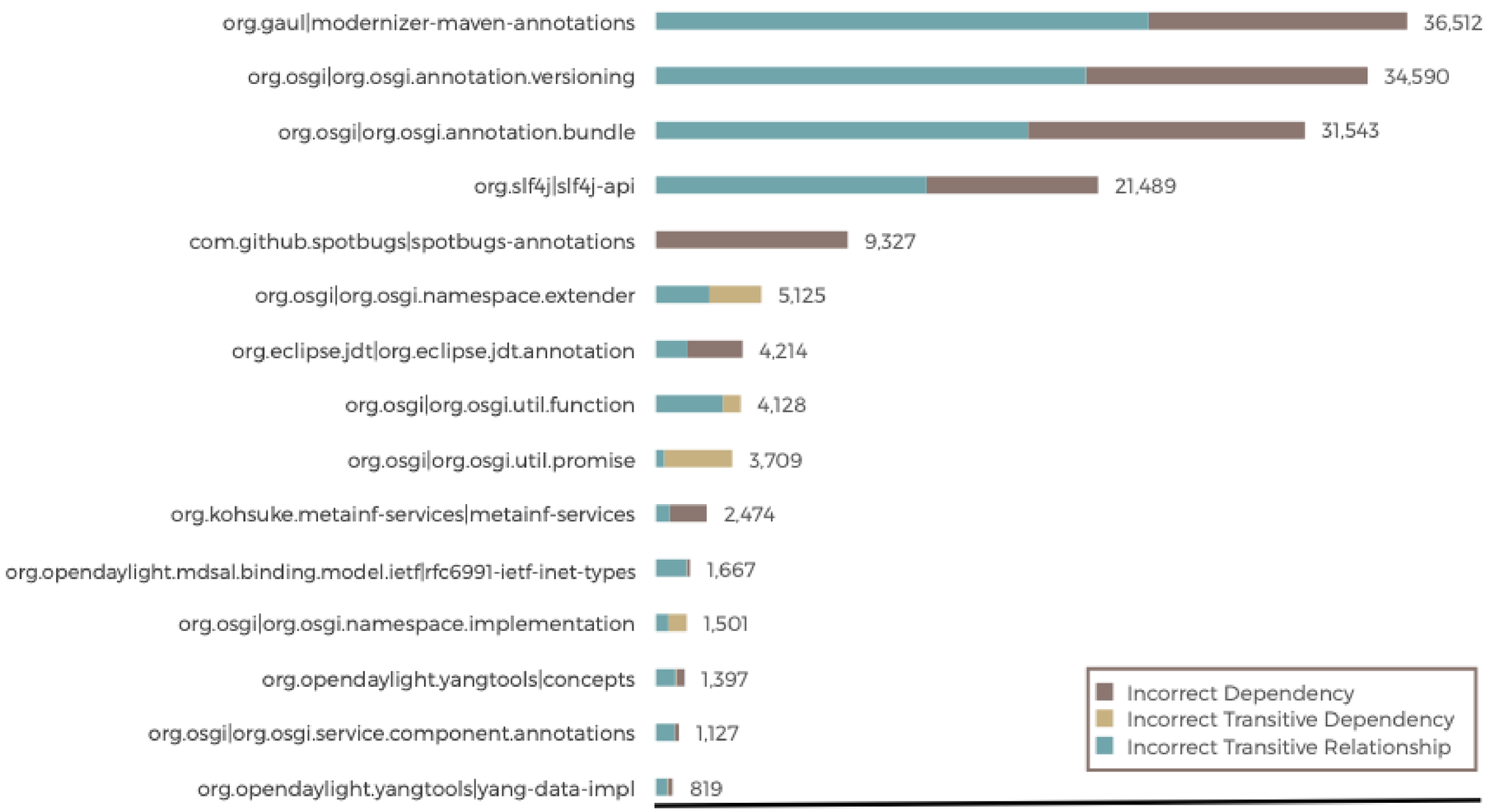
The prevalence of errors in direct dependencies is particularly concerning, as it directly compromises SBOM usability and affects the performance of dependent downstream tools.

The widespread nature of these issues underscores the critical need for rigorous compliance checks and quality audits of SBOMs to ensure their reliability.

Top Missing Dependencies

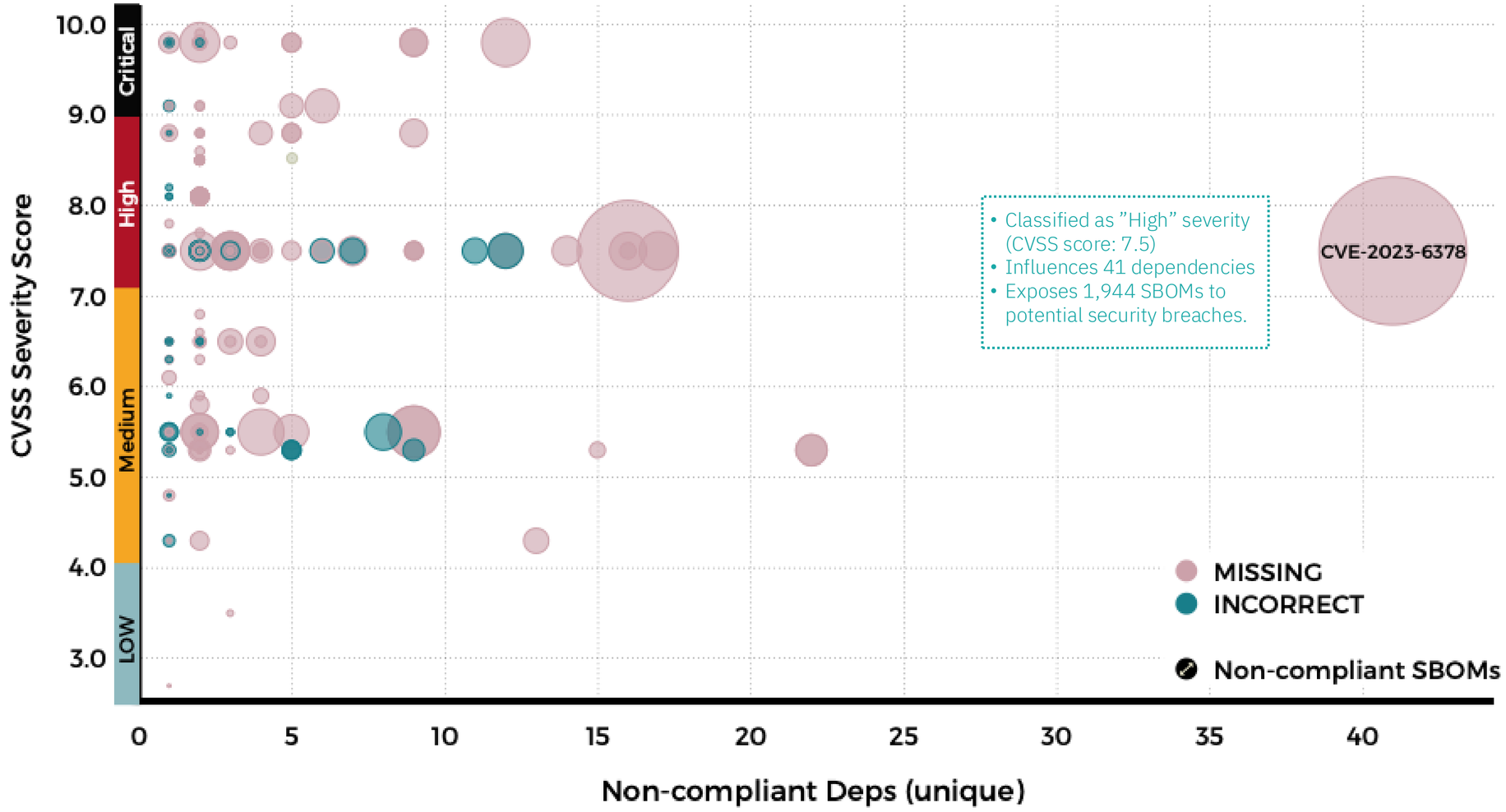


Top Incorrect Dependencies



Security Implications

The distribution of vulnerabilities severity within non- compliant dependencies/SBOMs



Root Cause Analysis

- Incomplete & Incorrect Metadata (i.e., pom.xml)
- Insufficient Design in SBOM Generators
- Improper Usage of SBOM generators
- Different Interpretations of SBOM Requirements

Artifact Evaluated
NDSS SYMPOSIUM
Available
Functional
Reproduced

Takeaways

- ❑ Proposes a consistency model
- ❑ Designs and implements JBomAudit
- ❑ Measure non-compliance SBOMs at scale
- ❑ Artifact Evaluation Badges & Responsible Disclosure

github.com/code-genome/jbomaudit

Thank you!