Preprint

Build and Runtime Integrity for Java

Aman Sharma
amansha@kth.se

KTH Royal Institute of Technology, Sweden

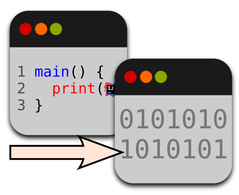
Software Supply Chain Attack

① Write source code

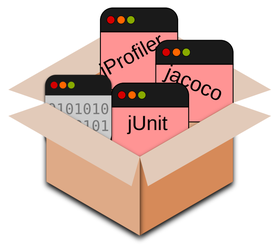
```
1 main() {
2   print(IJ)
3 }
```

Build

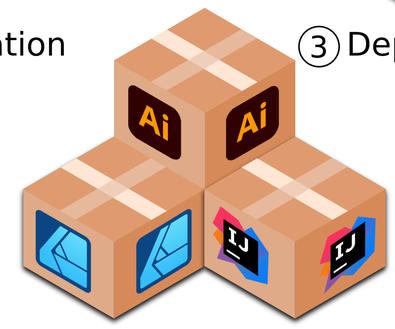
② Compilation



③ Dependencies

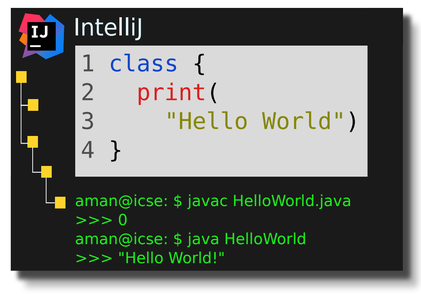


④ Packaging



Runtime

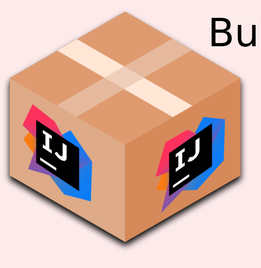
⑤ Running software



What if the compilation, dependencies, or packaging introduce some malware?

What if this software is stealing data while running?

Problems



Built by developer (builder)

Built by verifier (rebuilder)

Ideally, should be bit-by-bit identical

so many differences ...

```
{  // SBOM
   "bomFormat": "CycloneDX",
-  "serialNumber": "a96d",
+  "serialNumber"; "43de",
}
```

```
// MANIFEST.MF
  Main-Class: se.kth.Main
- Built-By: builder
+ Built-By: rebuilder
```

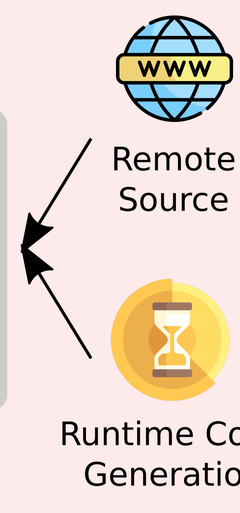
```
// file permissions
--rw-r--r-- Main.class
+-rw-rw-r-- Main.class
```

```
// constant pool
#1 = Utf8 "hello"
#2 = Class se/kth/Main
#3 = Utf8 "hello"
```

```
// JVM bytecode
- invokeinterface
    Object.equals()
+ invokevirtual
    Main.equals()
```

Problem 1: Spurious differences between artifacts make comparison hard.

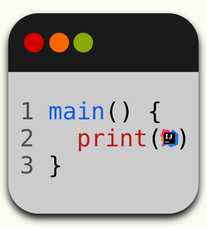
Loading	Linking	Initializing
return	getstatic	101010110010
ldc	ldc	110000101010
getstatic	invokestatic	101010100000
invokestatic	return	000001001111
steal	steal	010000100010



`steal` instructions are a result of dynamic classloading!
They were not present in the JAR.

Problem 2: Java can trigger download or generation of code.

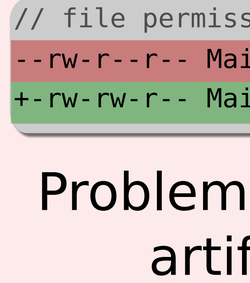
Our solutions



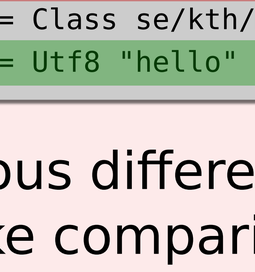
Fix build script

1) Changing JDK version for rebuild

2) Setting umask



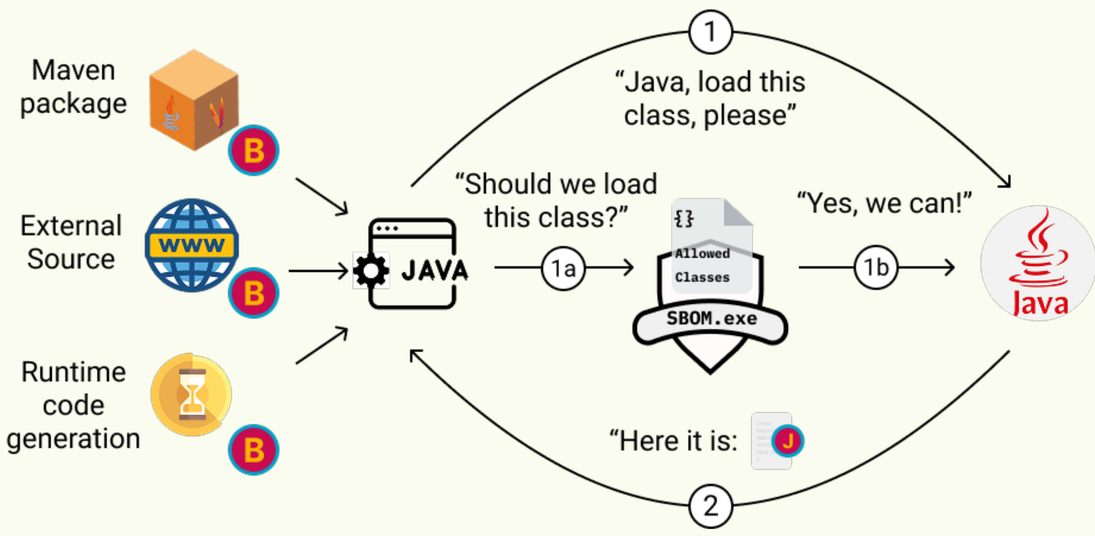
Strip non-deterministic changes



3) Delete attributes in MANIFEST

4) Setting file permissions to fixed value

Contribution 1: Fixing build scripts and/or canonicalizing artifacts before reproducibility..



This approach mitigates Log4shell and 2 other CVEs!

Contribution 2: Limit execution of classes that are not part of the application.

[1] C. Lamb and S. Zacchiroli, ‘Reproducible Builds: Increasing the Integrity of Software Supply Chains’, IEEE Software, vol. 39, no. 2, pp. 62–70, Mar. 2022.
[2] J. Xiong, Y. Shi, B. Chen, F. R. Cogo, and Z. M. (Jack) Jiang, ‘Towards build verifiability for Java-based systems’, in Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice, in ICSE-SEIP ’22. Oct. 2022, pp. 297–306.
[3] P. C. Amusuo et al., ‘ZTD$_{JAVA}$: Mitigating Software Supply Chain Vulnerabilities via Zero-Trust Dependencies’, presented at the ICSE ’25: 47th International Conference on Software Engineering, Apr. 2025.
[4] A. Sharma, M. Wittlinger, B. Baudry, and M. Monperrus, ‘SBOM.EXE: Countering Dynamic Code Injection based on Software Bill of Materials in Java’, Jun. 28, 2024, arXiv: arXiv:2407.00246. Available: http://arxiv.org/abs/2407.00246