Open Research Webinars

On-going research to fuel and enhance Eclipse Steady

Serena Elisa Ponta
SAP Security Research
Software products embed open-source components

- 80% to 90% of software products on the include OSS components
- 80+% of the codebase of a typical Java application is open-source

Dependency Graph
- Direct/transitive dependencies
- Duplicates and version conflicts
- Automated build systems handle the complexity transparently
Heartbleed, Equifax...

Using components with **known vulnerabilities**:
- Included in OWASP Top 10 since 2013
- Root cause of major data breaches

---

**Equifax** confirmed that their high profile, high impact data breach was due to an exploit of a vulnerability in an open source component, **Apache Struts CVE-2017-5638**. **Apache Struts** is a mainstream web framework, widely used by Fortune 100 companies in education, government, financial services, retail and media.

**Behind the Equifax Breach: A Deep Dive Into Apache Struts CVE-2017** ...

Log4Shell

● 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)
● Not only user-facing apps are affected (but any app that receives and logs untrusted input)
● Three other vulnerabilities have been found afterwards (CVE-2021-45046, 45105 and 44832)
● Latest non-vulnerable release is 2.17.1
Known vulnerabilities... Patch Exists!

Simply update?

- Depends on lifecycle phase and deployment model
- May include breaking changes
- Majority of vulnerabilities in transitive dependencies
- Re-bundles can also result in vulnerable apps (3233 artifacts on Maven Central contain the problematic Log4j class JndiLookup)
Open Source Vulnerability Scanners
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-centric**
- Detect the presence of code (no matter the package)
- Supports impact assessments (static and dynamic analyses), esp. important for later lifecycle phases and non-cloud
- Supports update metrics to avoid regressions
- Example: Eclipse Steady (heavy-weight, requires fix-commits) [https://eclipse.github.io/steady/](https://eclipse.github.io/steady/)
Metadata-based (Some) Limitations

- Short CVE descriptions and varying quality of referenced information
- Error-prone (2.3.5 and 2.3.6 were also affected)
- Coarse-granular reference of entire projects, ignoring reusable components and code (800+ artifact versions contain the resp. classes)
- CPE identifier != package identifier (30 search hits for “mojarra” on Maven Central don’t include org.glassfish:javax.faces)

CVE-2018-14371

The `getLocalePrefix` function in `ResourceManager.java` in Eclipse Mojarra before 2.3.5 is affected by Directory Traversal via the `loc` parameter. A remote attacker can download configuration files or Java bytecodes from applications.
Eclipse Steady
Code-centric detection and application-specific assessment

Validate if vulnerable code is (1) contained and (2) executed by the application

- Applications typically include large pieces of OSS code where only a fraction of it is used
- Combination of static analysis (call graph construction) and dynamic analysis (test/runtime instrumentation)
### Vulnerable Archives (distinct SHA1):

Vulnerabilities: 62

#### Include historical vulnerabilities

#### Include unconfirmed vulnerabilities (hourglass)

*Note: Analyze and assess ALL vulnerabilities, no matter the CVSS score. The severity of open-source vulnerabilities significantly depends on the application-specific context (in which the open-source component is used). Thus, the actual severity can differ significantly from the (context-independent) CVSS base score provided by 3rd parties such as the NVD.*

<table>
<thead>
<tr>
<th>Ass...</th>
<th>Dependenc...</th>
<th>Archive Filename (Digest)</th>
<th>Vulnerability (CVSS Score*)</th>
<th>Inclusion of vulnerable code</th>
<th>Static Analysis...</th>
<th>Dynamic Analysis...</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>cf1.2.2-cc1.4-xz1.0.jar</td>
<td>CVE-2012-2098 5.0 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>7778C34114BF620EAF9DFF6770C458234FDBC</td>
<td>CVEN-2013-2186 7.5 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>cf1.2.2-cc1.4-xz1.0.jar</td>
<td>CVE-2014-0050 7.5 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>7778C34114BF620EAF9DFF6770C458234FDBC</td>
<td>CVE-2016-3092-FU 7.8 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>COMPIL...</td>
<td>direct</td>
<td>commons-collections-3.2.1.jar</td>
<td>COLLECTIONS-580 8.0 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>

Programming constructs of the change list of the OSS patch

Repository: http://sun.apache.org/repo/axlf/poi

Revisions fixing the vulnerability: 1569991, 1560759, 1615720, 1616509, 1617849

<table>
<thead>
<tr>
<th>Change</th>
<th>Revision</th>
<th>Type</th>
<th>Qualified Construct Name (Path)</th>
<th>Contained</th>
<th>Reachable</th>
<th>Traced</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD</td>
<td>1569991</td>
<td>Method</td>
<td>org.apache.poi.openxml4j.opc.InternalContentTypesManager.parseContentTypesFile(InputStream) /poi/srksrрооxмйJava.org.apache.poi.openxml4j.opc/monas/ContentTypesManager.java</td>
<td>true</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>
Complementarity of dynamic and static analysis

- Due to **missing test case**, dynamic analysis does not find path starting from `ArchivePrinter.compressExploitability(Path, Path)`
- Due to the use of **reflection**, static analysis does not find path starting from `Thread.run()`
Mitigation options supported by reachability analyses

Calls from application to archive:

<table>
<thead>
<tr>
<th>Caller</th>
<th>Caller type</th>
<th>Callee</th>
<th>Potential</th>
<th>Traced</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.acme.foo ArchivePrinter.openSpreadsheet(Path)</td>
<td>CONS</td>
<td>org.apache.poi.xssf.usermodel.XSSFWorkbook(InputStream)</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>com.acme.foo ArchivePrinter.openSpreadsheet(Path)</td>
<td>METH</td>
<td>org.apache.poi.xssf.usermodel.XSSFSheet.getPhysicalNumberOfRows()</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Finding non-vulnerable library releases

Only libraries that are not vulnerable and newer than the one in use are shown.

<table>
<thead>
<tr>
<th>Library Id</th>
<th>Count c…</th>
<th>Callee stability</th>
<th>Dev. effort (calls to modify)</th>
<th>Reachable body stability</th>
<th>Overall body stability</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.poi:poi-ooxml:3.17</td>
<td>0</td>
<td>5 out of 5 (100%)</td>
<td>276 out of 288 (95%)</td>
<td>3569 out of 4509 (79%)</td>
<td></td>
</tr>
<tr>
<td>org.apache.poi:poi-ooxml:4.0.0</td>
<td>0</td>
<td>4 out of 5 (80%)</td>
<td>273 out of 288 (95%)</td>
<td>3169 out of 4509 (70%)</td>
<td></td>
</tr>
</tbody>
</table>

- Exclude dependency
- Update (non-breaking)
- Fork and down-port security fix
- Implement application-specific safeguards
Vulnerability Data about Open-source Software Should Be Open Too!

Today
• Information about open source vulnerabilities is scattered
• Mining is labor-intense despite advances in AI-based commit classification
• Providers step-in (and compete) with proprietary databases

This does not scale, and has the paradoxical consequence that data about open-source software is not open!
Open, collaborative, and trustworthy knowledge base of vulnerabilities (+fixes) that affect open-source software

Git repositories used to store vulnerability statements

Plain-text data format, machine-readable and human-readable

Tool-support

- Create, aggregate and validate statements
- Find fixes in open-source code repositories
Reducing the attack surface removing **bloated code**

- Unused by the application
- Potentially usable by attackers
- Needs maintenance
Can existing debloating tools minimize the dependencies of an industrial grade Java application?

- 260 application classes, 62 test classes yielding 446 test cases
- 2725 compile dependency classes

Reduced bloated code containing a potential security vulnerability but did not handle a service loader definition

<table>
<thead>
<tr>
<th>Execution</th>
<th>Classes</th>
<th>Size (KB)</th>
<th>Test success</th>
<th>Vulnerable classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>2725</td>
<td>15033</td>
<td>446</td>
<td>1</td>
</tr>
<tr>
<td>DepClean</td>
<td>11</td>
<td>57.26</td>
<td>446</td>
<td>-</td>
</tr>
<tr>
<td>Maven Shade</td>
<td>12</td>
<td>57.63</td>
<td>446</td>
<td>-</td>
</tr>
<tr>
<td>ProGuard®</td>
<td>1</td>
<td>4</td>
<td>446</td>
<td>-</td>
</tr>
<tr>
<td>ProGuard®c</td>
<td>11</td>
<td>57.26</td>
<td>446</td>
<td>-</td>
</tr>
</tbody>
</table>

Ponta, S., et al.: The Used, the Bloated, and the Vulnerable: Reducing the Attack Surface of an Industrial Application (2021)
Conclusion

- Need for **precise analysis** techniques for effective vulnerability management
- Code-based approaches reduce FP and FN and support impact assessment
- Code-level information about vulnerabilities and their fixes is key
- Gathering and maintaining this information is best done in a collaborative fashion
- Open formats and tools to enable **publishing, sharing** and **aggregating vulnerability data** in an efficient, flexible, trustworthy fashion
- Reducing bloated code may dramatically reduce the attack surface of applications
Eclipse Steady & project "KB"
Dealing with vulnerabilities of open-source software
the open-source way

Links
https://github.com/eclipse/steady
https://eclipse.github.io/steady
https://github.com/SAP/project-kb
https://sap.github.io/project-kb

Acknowledgements
Sparta (EU-funded project)
https://www.sparta.eu/
AssureMOSS (EU-funded project)
https://assuremoss.eu/