Security of Open Source Software Dependencies

Serena Elisa Ponta
SAP Security Research
Outline

• OSS Components in Commercial Applications and Security
• Open Source Supply Chain Attacks
• Dependencies with Known vulnerabilities
• Code-centric detection assessment and remediation of known vulnerabilities
• Open problem, direction
Commercial software products embed open-source components

- 80% to 90% of software products on the market include OSS component
- 80+% of the codebase of a typical Java application is open-source
- Direct dependencies yield to dozens of transitive ones
- Dependency graph includes duplicates and version conflicts
- Automated build systems handle the complexity transparently
Quality of the overall application depends on the quality of each component:

- Performance?
- Issues?
- Level of support?
- Security?
Using Open-Source in Enterprise Contexts

Security Threats

• Open-Source Supply Chain Attacks
  *Attack an application via its open source dependencies by sneaking malware*

• Dependencies with Known vulnerabilities
  *Attack an application via its open source dependencies by exploiting their publicly known vulnerabilities*
Open Source Supply
Chain Attacks

https://xkcd.com/2347/
Attacks on Open-Source Supply Chains

• Supply chain attack: Compromise a software vendor’s infrastructure and infect legitimate software such that malware is delivered through trusted distribution channels

• Flavor: Inject malicious code into an open-source component that (many) other legitimate components depend on, such that malware is executed at dependents’ dev, build or runtime
Attacks on Open-Source Supply Chains

• Supply chain attack: Compromise a software vendor’s infrastructure and infect legitimate software such that malware is delivered through trusted distribution channels

• Flavor: Inject malicious code into an open-source component that (many) other legitimate components depend on, such that malware is executed at dependents’ dev, build or runtime
Attacks on Open-Source Supply Chains

- Supply chain attack: Compromise a software vendor’s infrastructure and infect legitimate software such that malware is delivered through trusted distribution channels
- Flavor: Inject malicious code into an open-source component that (many) other legitimate components depend on, such that malware is executed at dependents’ dev, build or runtime
Successful attack on NPM package `event-stream`

- 1.5+ million downloads/week, 1600 dependent packages
- Contacted by mail, original developer handed-over ownership to “right9control”
- Added dependency on a malicious package
- Malicious code (and encrypted payload) only present in published NPM package
- Malware and decryption only ran in context of a release build of the bitcoin wallet `copay`

https://www.theregister.co.uk/2018/11/26/npm_repo_bitcoin_stealer/
https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502
Backstabber’s Knife Collection
Review of Open Source Software Supply Chain Attacks

- Analysis of temporal aspects, injection technique, trigger/conditional execution, primary objective, targeted OS, use of obfuscation, and clusters/campaigns
- **Dataset** released as open-source, 1100+ packages
  (write to ohm[at]cs.uni-bonn.de or henrik.plate[at]sap.com to get access)

https://github.com/cybertier/Backstabbers-Knife-Collection
Interactive Attack Tree & Associated Safeguards

https://survey.opensourceunchained.eu/expertsurvey.html
Interactive Attack Tree & Associated Safeguards

https://survey.opensourceunchained.eu/expertsurvey.html
Open Source Supply Chain Attacks

Take-Aways

• Number of dependencies and actors + complexity of build processes and infrastructures result in a considerable attack surface

• All dependencies matter (not only compile/runtime)

• The truth is in downloaded packages (source code visible in GitHub etc. does not matter)

• Noticeable increase in supply chain attacks targeting open source ecosystems

• Python, Node.js and Ruby ecosystems are the primary targets (but some ecosystems like Java have not been analyzed in a systematic fashion)
Dependencies with known vulnerabilities

https://xkcd.com/2347/
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 ...
https://www.brighttalk.com/.../behind-the-equifax-breach-a-deep-dive-into-apache-struts...

Source: Google
Known vulnerabilities... Patch Exists! Simply update?

- Depends on lifecycle phase and deployment model
- Breaking changes
- Majority of vulnerabilities are found in transitive dependencies
The hard reality of vulnerability management

• **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 *ridiculously expensive*)
• **Release patch** (cloud 😊 on-premise ☹️)
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/
Eclipse Mojarra and 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.

• CVSS Base Score: 7.5 (high)

• References: [fix-commit](#) and [issue](#)

• Affected products:

  cpe:2.3:a:eclipse:mojarra:*  
  up to (excluding) 2.3.5
Eclipse Mojarra and CVE-2018-14371

- The `getLocalePrefix` function in `ResourceManager.java` in Eclipse Mojarra before 2.3.7 is affected by Directory Traversal via the `loc` parameter. A remote attacker can download configuration files or Java bytecodes from applications.
- CVSS Base Score: 7.5 (high)
- References: [fix-commit](#) and [issue](#)
- Affected products:
  cpe:2.3:a:eclipse:mojarra:*
  up to (excluding) 2.3.7
The `getLocalePrefix` function in `ResourceManager.java` in Eclipse Mojarra before 2.3.7 is affected by Directory Traversal via the `loc` parameter. A remote attacker can download configuration files or Java bytecodes from applications.

- CVSS Base Score: 7.5 (high)
- References: [fix-commit](#) and [issue](#)
- Affected products:
  
  org.glassfish:javax.faces

Maven coordinates:

```
cpe:2.3:a:eclipse:mojarra:* up to (excluding) 2.3.7
```

https://github.com/eclipse-ee4j/mojarra/commit/1b434748d9239f42eae8aa7d37d7a0930c061e24
Maintain fine-grained mapping?

- 700+ packages contain the resp. classes
- Number of indexed packages keeps on increasing

https://mvnrepository.com/repos/central
Code-centric detection, assessment and remediation of known vulnerabilities
Vulnerable Code

Repository: https://github.com/some/project
Commits: 236AF0619, 1234E4D5C1A, ...

Patch Information

Method x.y.z ADDED
Method a.b.c MODIFIED
Constructor i.j.k DELETED

Vulnerable Code
Code-centric detection

$S_a, S_i$ All programming constructs of application $a$ and its library $i$

$C_j$ Code changed to fix vulnerability $j$
Eclipse Steady vs OWASP DC

300 enterprise applications, 78165 dependencies

Manual review of (subset of) findings

- Steady-only findings are true positives (misconfigurations covered with workaround)
- 88.8% (36.7% for misconfigurations) of OWASP DC-only are false positives
- True positive of a single tool are false negatives for the other!

<table>
<thead>
<tr>
<th>Category</th>
<th>Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SO_C$</td>
<td>2099</td>
</tr>
<tr>
<td>$SO_{NC}$</td>
<td>843</td>
</tr>
<tr>
<td>$SC$</td>
<td>1829</td>
</tr>
<tr>
<td>$SN_C$</td>
<td>27</td>
</tr>
<tr>
<td>$OC$</td>
<td>17062 (2909)</td>
</tr>
<tr>
<td>$ON_{NC}$</td>
<td>1728 (237)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
<th>Proportion of TP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SC$</td>
<td>30/30 (0.943 ± 0.0568)</td>
</tr>
<tr>
<td>$OC$</td>
<td>3/30 (0.145 ± 0.111)</td>
</tr>
<tr>
<td>$SN_C$</td>
<td>27/27 (1.0)</td>
</tr>
<tr>
<td>$ON_{NC}$</td>
<td>19/30 (0.618 ± 0.163)</td>
</tr>
</tbody>
</table>

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)
<table>
<thead>
<tr>
<th>Associate</th>
<th>Dependency</th>
<th>Archive Filename (Digest)</th>
<th>Vulnerability</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</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>7F7798C34114BF620EFA9DFF6770C458234FDBC</td>
<td>CVE-2013-2186</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>7F7798C34114BF620EFA9DFF6770C458234FDBC</td>
<td>CVE-2014-0050</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>direct</td>
<td>7F7798C34114BF620EFA9DFF6770C458234FDBC</td>
<td>CVE-2016-3092-FU</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>COMPILE</td>
<td>direct</td>
<td>commons-collections-3.2.1.jar</td>
<td>COLLECTIONS-580</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>COMPILE</td>
<td>direct</td>
<td>761EA405B9B37CED573D2DF0D1E3A4E9F9EDC668</td>
<td>CVE-2013-0248</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>COMPILE</td>
<td>direct</td>
<td>commons-fileupload-1.2.1.jar</td>
<td>CVE-2013-2186</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
<tr>
<td>COMPILE</td>
<td>direct</td>
<td>384FA82E193D4E4B0546059CA09572654BC3970</td>
<td>CVE-2014-0050</td>
<td>![Bullet Point]</td>
<td>![Bear Shape]</td>
<td>![Bear Shape]</td>
</tr>
</tbody>
</table>

https://eclipse.github.io/steady/
Vulnerable Archives (distinct SHA1): 11
Vulnerabilities: 62

<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>cf1.2.2-cc1.4-xz1.0.jar</td>
<td>CVE-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>cf1.2.2-cc1.4-xz1.0.jar</td>
<td>CVE-2016-3092-FU 7.8 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
<tr>
<td>COMPILE</td>
<td>direct</td>
<td>commons-collections-3.2.1.jar</td>
<td>COLLECTIONS-580 7.8 (v2.0)</td>
<td>![ ]</td>
<td>![ ]</td>
<td>![ ]</td>
</tr>
</tbody>
</table>

Programming constructs of the change list of the OSS patch
Repository:http://svn.apache.org/repos/asf/poi
Revisions fixing the vulnerability:1569991,1589759,1615720,1616509,1617849

<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>
</table>

https://eclipse.github.io/steady/
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()`

https://eclipse.github.io/steady/
Combination of dynamic and static analysis

Example

• Ground truth
Combination of dynamic and static analysis

Example

• Reachability analysis starting from the app (5)
Combination of dynamic and static analysis

Example

• Collected traces through dynamic instrumentation of tests
Combination of dynamic and static analysis

Example

- Reachability analysis starting from all traced methods
Mitigation options supported by reachability analyses

- Exclude dependency
- Update (non-breaking)
- Fork and down-port security fix
- Implement application-specific safeguards

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>0 out of 5 (0 %)</td>
<td>276 out of 288 (96 %)</td>
<td>3566 out of 4509 (79 %)</td>
</tr>
<tr>
<td>org.apache.poi:poi-ooxml:4.0.0</td>
<td>0</td>
<td>4 out of 5 (80 %)</td>
<td>1 out of 5 (20 %)</td>
<td>273 out of 288 (95 %)</td>
<td>3169 out of 4509 (70 %)</td>
</tr>
</tbody>
</table>
• CVE/NVD has issues with quality, timeliness and coverage
• No other public database with comprehensive and high-quality information about OSS vulnerabilities is available
• Code-based approaches
  • Reduce false positives
  • Support impact assessment
  • Needs workaround for no-code patches
  • Requires an open and comprehensive knowledge base with code-level information about vulnerabilities
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
github.com/SAP/project-kb

- **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
Open problem, direction
How do developers include OSS?

Unmodified
com.google:guava:23.0

Patched
(re-compiled source-code) com.google:guava:23.0_fix3

Uber-JAR
(re-bundle multiple OSS) com.my:servicebundle:1.0

bare Uber-JAR
(Uber-JAR without metadata)

re-packaged Uber-JAR
(Uber-JAR with re-packaged classes, i.e., prepended string)

Prevalence of modified OSS

Investigated modification for 254 classes from 249 CVEs affecting the most-used OSS components at SAP

<table>
<thead>
<tr>
<th></th>
<th>Patched</th>
<th>Uber-Jar</th>
<th>Bare-Uber-Jar</th>
<th>Re-packaged Uber-Jar</th>
</tr>
</thead>
<tbody>
<tr>
<td># classes subject to</td>
<td>143</td>
<td>222</td>
<td>222</td>
<td>17</td>
</tr>
<tr>
<td># affected GAV</td>
<td>5,919</td>
<td>36,609</td>
<td>24,500</td>
<td>168</td>
</tr>
<tr>
<td># affected GA</td>
<td>360</td>
<td>6,728</td>
<td>3,882</td>
<td>89</td>
</tr>
</tbody>
</table>
Vulnerability Scanners Comparison

<table>
<thead>
<tr>
<th></th>
<th>Unmodified</th>
<th>Type 1 (patched)</th>
<th>Type 2 (Uber-JAR)</th>
<th>Type 3 (bare Uber-JAR)</th>
<th>Type 4 (re-pack. Uber-JAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>precision</td>
<td>recall</td>
<td>F1</td>
<td>precision</td>
<td>recall</td>
</tr>
<tr>
<td>OWASP*</td>
<td>0.34</td>
<td>0.92</td>
<td>0.50</td>
<td>0.35</td>
<td>0.92</td>
</tr>
<tr>
<td>Eclipse Steady*</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.38</td>
<td>0.75</td>
</tr>
<tr>
<td>Security Alerts</td>
<td>0.60</td>
<td>0.50</td>
<td>0.55</td>
<td>0.60</td>
<td>0.50</td>
</tr>
<tr>
<td>C1</td>
<td>0.64</td>
<td>0.58</td>
<td>0.61</td>
<td>0.32</td>
<td>0.58</td>
</tr>
<tr>
<td>C2</td>
<td>0.75</td>
<td>0.75</td>
<td>0.75</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C3*</td>
<td>0.71</td>
<td>0.83</td>
<td>0.77</td>
<td>0.71</td>
<td>0.83</td>
</tr>
</tbody>
</table>

- highly divert in performance even for unmodified OSS
- struggle to identify vulnerabilities in modified OSS

Reducing the attack surface by removing bloated code

- Unused by the application
- Potentially usable by attackers
- Needs maintenance
Case Study

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

<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**</td>
<td>11</td>
<td>57.26</td>
<td>446</td>
<td>-</td>
</tr>
</tbody>
</table>

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

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

- Open source software is omnipresent
- Open source supply chain attacks are on the rise
- Dependencies with known vulnerabilities are wide-spread
- Current scanners struggle to identify vulnerabilities
- Code-level information about vulnerabilities and their fixes is key
- Gathering and maintaining this information is best done in a collaborative fashion
- Reducing bloated code may dramatically reduce the attack surface of applications