Software Composition Analysis

OWASP Stammtisch
19.11.2019
München

Stanislav Sivak
Agenda

• Introduction
• Challenges
• Approaches
• Integration
• Q & A
Agenda

• Introduction
• Challenges
• Approaches
• Integration
• Q & A
Introduction

Disclaimer:

This is my personal presentation and represents neither my current employer nor any other organization.
Introduction

• Senior security consultant at Synopsys
• Working on various AppSec related projects mainly in Germany in the areas of
  • How to secure SDLC with focus on:
    • Threat modelling
    • Application security testing
    • Security in CI/CD
• Previously worked as web developer, security administrator, pentester
Open Source Software

• How did the open source usage evolve?

1998: 10% Open Source
2005: 20% Open Source
2010: 50% Open Source
TODAY: Up to 90% Open Source
It enters your code through many channels...

...and open source vulnerabilities can come with it.
Black Duck On-Demand audits found open source components in 96% of the applications scanned, with an average 257 components per application.

The number of open source vulnerabilities per codebase grew by 134%.

Open source represented 60% of the code analyzed in 2018, up from 57% in 2017.

17% of the codebases contained a highly publicized vulnerability such as Heartbleed, Logjam, Freak, Drown, and Poodle.


Based on over 1,200 commercial applications analyzed by Black Duck On-Demand in 2018.
State of open source 2018 (2/2)
Most seen open-source components

jQuery  
Bootstrap  
jQuery UI  
Font Awesome  
Moment
What software is in scope?
FOSS – Free and open-source software

License type approved by Open Source Initiative

License type approved by Free Software Foundation
What software is in scope?

- FOSS – Free and open-source software

- **Open Source Code**
  - Snippets
  - Modules
  - SDK

- **Open Source Binary Software**
  - Libraries
  - Executables

- Proprietary 3rd-party components
Agenda

• Introduction
• Challenges (No challenge-> no fun)
• Approaches
• Integration
• Q & A
Challenge No.1 - Assets

- I need further information to our application inventory...
- How much open source do we use?
- How is the use of open source governed in our company?
Challenge No.2 - Security

• Which our projects have known open-source vulnerabilities?
• Do we have any components with critical and high vulnerabilities?
• Do our projects have the XXX vulnerable component?

CISO/Security Manager
### Challenge No.2 - Security

<table>
<thead>
<tr>
<th>Challenge No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7:2017- Cross-Site Scripting (XSS)</td>
<td>XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.</td>
</tr>
<tr>
<td>A8:2017- Insecure Deserialization</td>
<td>Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.</td>
</tr>
<tr>
<td>A9:2017- Using Components with Known Vulnerabilities</td>
<td>Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.</td>
</tr>
<tr>
<td>A10:2017- Insufficient Logging &amp; Monitoring</td>
<td>Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.</td>
</tr>
</tbody>
</table>
Challenge No.3 - Licensing

- Are we allowed to share/distribute my software in its current form?
- Do we have any licenses non-compliant with our internal FOSS policy?
- Do we distribute any software with a copyleft license?

License count

- Apache License 2.0
- Eclipse Public License 1.0
- MIT License
- Unknown License
- BSD 3-clause "New" or "Revised" License
- Others
# Beware of these license families

<table>
<thead>
<tr>
<th>Licensing scheme</th>
<th>License Family</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Copyleft</strong></td>
<td>Affero General Public License (AGPL)</td>
<td>GNU Affero General Public License v3 or later</td>
</tr>
<tr>
<td><strong>Copyleft</strong></td>
<td>Reciprocal</td>
<td>GNU General Public License (GPL) 2.0 or 3.0 Sun GPL with Classpath Exception v2.0</td>
</tr>
<tr>
<td><strong>Copyleft</strong></td>
<td>Weak Reciprocal</td>
<td>Code Project Open License 1.02 Common Development and Distribution License (CDDL) 1.0 or 1.1 Eclipse Public License GNU Lesser General Public License (LGPL) 2.1 or 3.0 Microsoft Reciprocal License Mozilla</td>
</tr>
<tr>
<td><strong>Non-commercial use</strong></td>
<td>Non-commercial</td>
<td>AFPL JRL</td>
</tr>
</tbody>
</table>

**Full source code available to any network user**

**Full source code available if distributed**

**The modified/used OSS source code (mostly) must be shared.**

**For non-commercial use only**
License breach – is it really suable?
2017 - Artifex Software, Inc. versus Hancom, Inc.

- 1. Developed open-source PDF interpreter
- 2. The interpreter has a dual license: either GPL or commercial
- 3. Used the interpreter in the commercial Office software
- 4. Hancom neither paid for the commercial license nor published the custom software as open-source -> license infringement

5. GPL can be treated like a legal contract

https://www.linux.com/blog/artifex-v-hancom-open-source-now-enforceable-contract
Challenge No.4 – Operational risks

• How well is the component maintained?
• Is there any support?
• Are security vulnerabilities/bugs fixed within tolerable time?
• How large is the community?
• What is plan B if there is no new update?
Challenge No.5 – Date protection

Does any of my open-source components access sensitive data and if yes, what happens with that data?

• User tracking
• Data collection
• GDPR
Who wins?

FOSS advantages
Agenda

• Introduction
• Challenges
• Approaches
• Integration
• Q & A
Approach
How to deal with our 4 challenges?

Improve existing approaches?

Add a new approach?
Common manual approaches

MANUAL DISCOVERY
- Cumbersome processes
- Occurs at end of SDLC
- High effort and low accuracy
- No ongoing controls

SPREADSHEET INVENTORY
- Requires consistent developer input
- Difficult to maintain and scale
- Not a full/accurate list of actual usage

SPORADIC VULNERABILITY TRACKING
- No single responsible entity
- Labor intensive manual effort
- Unmanageable (~11 new vulns/day)

PERIODIC VULNERABILITY SCANNING
- Monthly/quarterly vulnerability assessments
- Not aimed at open source vulnerabilities
- Integrated later in the SDLC

#FAIL
Common automated approaches

Shift left!
Common automated SCA approaches (1/5)

Source code repository checks

+ Examines open source components automatically – no triggered scan needed
+ Known FOSS security vulnerabilities with CVE are reported
+ Visualisation
+ Often easy remediation in the repository -> replacement of the vulnerable component
+ Alerts sent and displayed for new vulnerabilities
+ Continuous analysis

- Focus on dependencies but no code snippets or modified files/directories
- Often no licenses overview
- Reporting
Common automated SCA approaches (1/5)
Using GitHub source code repository checks

We found a potential security vulnerability in a repository for which you have been granted security alert access.

Known moderate severity security vulnerability detected in sprockets >= 2.6.0 defined in Gemfile.lock.

Gemfile.lock update suggested: sprockets ~> 2.7.1.

Always verify the validity and compatibility of suggestions with your codebase.
Common automated SCA approaches (2/5)
Binary repository manager checks
+ Examines all binary components known for open-source vulnerabilities
+ Easy access to artifacts
+ Can be triggered on-demand or automatically when new artifacts appear
+ Easy implementation of approved artifacts only (due to licensing, whitelisting,...)
+ Dependency graph
+ Easy integration
+ Continuous analysis
- Only successful if all artifacts stored there -> single source of truth
- Can miss references in Source Code repositories
- Licensing information?
- Reporting
Common automated SCA approaches (2/5)
Binary repository manager checks - example
Common automated SCA approaches (3/5)

**SAST**

<table>
<thead>
<tr>
<th>Static Application Security Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Analyzes any source code, not only FOSS specific</td>
</tr>
<tr>
<td>• Finds common vulnerability patterns:</td>
</tr>
<tr>
<td>• SQL injection</td>
</tr>
<tr>
<td>• Cross-site scripting</td>
</tr>
<tr>
<td>• Buffer overflows, etc.</td>
</tr>
</tbody>
</table>

+ Finds both publicly known and unknown security vulnerabilities in the source code
+ No additional tool/stage needed
+ SAST can be performed in various pipeline stages
+ SAST tools can have a separate module that inspects software composition

- Limited insight into Software Composition Analysis
- No Bill of Material
- No licensing information
- Results represent a point in time
Common automated SCA approaches (4/5)

**DAST**

Dynamic Application Security Testing

- Tests running apps
- Finds vulnerable app behavior:
  - Misconfigurations
  - Authentication issues

+ Finds both publicly known and unknown security vulnerabilities
+ No additional tool/stage needed
+ Fewer false positives than SAST

- Limited insight into Software Composition Analysis as it examines running software from outside
- Runs later in a later pipeline stage
- Very incomplete Bill of Material
- No licensing information
- Results represent a point in time
Common automated SCA approaches (5/5)

**SCA Testing**

+ Focused on Open Source Components
+ Few false positives due to several ways of identifying FOSS components
+ Both compiled and uncompiled code can be analysed
+ Usually faster in scanning FOSS components
+ Public and private vulnerability databases
+ Can integrate with other application security testing metrics

**Software Composition Analysis (Testing)**

- Scans for open source
- Provides Bill of Material
- Finds Open Source licenses
- Finds open source vulnerabilities:
  - Detects known vulns
  - Works through full SDLC
  - Monitors for new vulns

- Yet another stage/tool to implement
- Does not find publicly unknown vulnerabilities, so need to be complemented with SAST/DAST
Software composition analysis

SCA is a process that can determine all underlying components of a software and identify at least the public known (open-source) components.

A well defined process is consistent, automated and measurable.
Commercial SCA tools (1/2)

### Security Risk
- High: 2
- Medium: 1
- Low: 6

### License Risk
- High: 1
- Medium: 6
- Low: 14

### Operational Risk
- High: 1
- Medium: 5
- Low: 3

<table>
<thead>
<tr>
<th>Component</th>
<th>Source</th>
<th>Metric Type</th>
<th>Usage</th>
<th>License</th>
<th>Security Risk</th>
<th>Operational Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Germinio Bidas Revig - Assembly</td>
<td>4 Matches</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>Germinio Framework, Modules</td>
<td>1 Match</td>
<td>Exact Directory</td>
<td>Dynamically Linked</td>
<td>Apache 2.0</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>OpenSSL - 1.5.1h</td>
<td>12 Matches</td>
<td>Exact Directory, Files Modified</td>
<td>Dynamically Linked</td>
<td>OpenSSL License</td>
<td>🟢</td>
<td>🟢</td>
</tr>
<tr>
<td>OpenSSL - 1.8.10</td>
<td>12 Matches</td>
<td>Exact Directory, Files Modified</td>
<td>Dynamically Linked</td>
<td>OpenSSL and others</td>
<td>🟢</td>
<td>🟢</td>
</tr>
</tbody>
</table>
Commercial SCA tools (2/2)

Cortana_Android-4.4.apk

Vulnerability analysis

<table>
<thead>
<tr>
<th>Components</th>
<th>27 Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerable</td>
<td>3</td>
</tr>
<tr>
<td>No known vulnerabilities</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vulnerabilities</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>3</td>
</tr>
<tr>
<td>Major</td>
<td>18</td>
</tr>
<tr>
<td>Minor</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Licenses</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissive</td>
<td>7</td>
</tr>
<tr>
<td>LOPL</td>
<td>1</td>
</tr>
<tr>
<td>Proprietary</td>
<td>2</td>
</tr>
</tbody>
</table>

CVSS v2 >= 7.0  CVSS v2 >= 4.0  CVSS v2 < 4.0  Clean  Triaged  Historical

Identified 3rd party components (27)

Component
- expat 2.1.0
- sqlite3 3.11.0
- openssl 1.1.0
- android
- opensal 1.0.1
- p0r 1.0

Vulnerabilities
- 2
- 4
- 0
- 0
- 0
- 5
- 1
- 2
- 0
- 1
- 11
- 0
- 12
- 1
- 0
- 182
- 0
- 0
- 0
- 0
- 0
- 1780
- 0
- 0
- 0
- 198
- 0
- 0
- 0
- 47
Open Source SCA tools (1/5)
RetireJS – JavaScript dependencies

Integration:
- A command line scanner
- A grunt plugin (NPM)
- A Chrome extension
- A Firefox extension
- Burp and OWASP Zap plugin
- Eclipse plugin

https://github.com/retirejs/retire.js/
Open Source SCA tools (2/5)

NPM Audit

- A command line scanner
- Focuses on NPM packages
- Suggest fixes -> easy remediation
- Package signing checks in the future?

https://blog.npmjs.org/post/173719309445/npm-audit-identify-and-fix-insecure
Open Source SCA tools (3/5)

OWASP Dependency Check

OWASP Dependency Track

https://jeremylong.github.io/DependencyCheck/

https://www.owasp.org/index.php/OWASP_Dependency_Track_Project
Open Source SCA tools (4/5)
Dependency Track – THE open source tool for SCA

https://www.owasp.org/index.php/OWASP_Dependency_Track_Project
Open Source SCA tools (5/5)
Open-Source tools examples for finding licensing issues

- Fossology  https://www.fossology.org/
- SW360  https://sw360.github.io/
## SCA decision table

<table>
<thead>
<tr>
<th>Profile</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer startup with JS frameworks</td>
<td>Use technology-specific tools such as RetireJS, npm audit,…</td>
</tr>
<tr>
<td>SMB with multiple technologies and powerful development teams</td>
<td>Use binary repository manager add-ons or source control versioning mechanisms</td>
</tr>
<tr>
<td>SMB with multiple technologies at SCA beginning with focus on security</td>
<td>Use or start with OWASP Dependency Track</td>
</tr>
<tr>
<td>SMB with multiple technologies at SCA beginning with focus on compliance</td>
<td>Use open-source tools such as Fossology/OSS Review Toolkit</td>
</tr>
<tr>
<td>Enterprises with clear SCA requirements and multiple stakeholders: CISO, Legal, Developers, Open-Source Officers</td>
<td>Start with OWASP Dependency Track and/or Evaluate commercial SCA tools</td>
</tr>
</tbody>
</table>
KEYS TO open source security management

1. Contextual identification
2. Complete vulnerability and legal data
3. Zero-day notification
4. Timely remediation
5. Efficient policy management
6. Integrate and automate
Agenda

• Introduction
• Challenges
• Approaches
• Integration in SDLC
• Q & A
Requirements

CI/CD

• Automatable
• User-friendly
• Actionable
• Flexible/Open
• Easy to integrate
Application security pipeline

- Code
- Build
- Test
- Deploy
- Production

SAST  SCA  IAST  DAST

Development

Operations
CI/CD Pipeline

2018-09-14 14:25:13 INFO [main] --- Starting the Hub signature scans
2018-09-14 14:25:13 INFO [pool-2-thread-1] --- Starting the signature scan of /var/lib/jenkins/workspace/test_pipeline1

2018-09-14 14:25:13 INFO [pool-2-thread-1] --- -v
2018-09-14 14:25:13 INFO [pool-2-thread-1] --- -statusWriteDir

INFO] Sensor Black Duck Hub Plugin for SonarQube [hubsonarqube]
INFO] Successfully connected to https://hubsig.blackducksoftware.com
INFO] Gathering local component files...
INFO] Gathering Hub component files...
INFO] Getting matched files for Apache Ant...
INFO] Getting matched files for Apache Commons Compress...
INFO] Getting matched files for Apache Maven 2...
INFO] Getting matched files for Apache Tomcat...
INFO] Getting matched files for Bouncy Castle...
INFO] Getting matched files for XStream...
INFO] Getting matched files for XStream...
INFO] Getting matched files for XStream...
INFO] Getting matched files for Spring Transaction...
INFO] Getting matched files for Spring TestContext Framework...
INFO] Getting matched files for Spring Security...
INFO] Getting matched files for Spring Framework...
INFO] Getting matched files for Spring Data Commons...
INFO] Getting matched files for Spring Security...
INFO] Getting matched files for Spring Security...
INFO] Getting matched files for Spring Security...
INFO] Getting matched files for Spring Security...
INFO] Getting matched files for Spring Security...
INFO] Getting matched files for Spring Security...
INFO] --> Number of local files matching inclusion/exclusion patterns: 8
INFO] --> Number of vulnerable Hub component files matched: 8
Interesting Links

• Copyright trolling [https://blog.fossa.io/patrick-mchardy-and-copyright-profiteering-44f7c28c0693](https://blog.fossa.io/patrick-mchardy-and-copyright-profiteering-44f7c28c0693)

• GitHub and SCA [https://www.dev-insider.de/security-alerts-auf-github-nutzen-a-758877/](https://www.dev-insider.de/security-alerts-auf-github-nutzen-a-758877/)

• Open Source Metadata [https://clearlydefined.io/about](https://clearlydefined.io/about)
Q&A

Stanislav.Sivak@synopsys.com