Abstract

Today, nearly all developers rely on third party components for building an application. Thus, for most software vendors, third party components in general and Free/Libre and Open Source Software (FLOSS) in particular, are an integral part of their software supply chain.

As the security of a software offering, independently of the delivery model, depends on all components, a secure software supply chain is of utmost importance. While this is true for both proprietary and as well as FLOSS components that are consumed, FLOSS components impose particular challenges as well as provide unique opportunities. For example, on the one hand, FLOSS licenses contain usually a very strong "no warranty" clause and no service-level agreement. On the other hand, FLOSS licenses allow to modify the source code and, thus, to fix issues without depending on an (external) software vendor.

This talk is based on working on integrating securely third-party components in general, and FLOSS components in particular, into the SAP's Security Development Lifecycle (SSDL). Thus, our experience covers a wide range of products (e.g., from small mobile applications of a few thousands lines of code to large scale enterprise applications with more than a billion lines of code), a wide range of software development models (ranging from traditional waterfall to agile software engineering to DevOps), as well as multiple deployment models (e.g., on premise products, custom hosting, or software-as-a-service).
Secure Software Development

Start of development
Preparation
Development
Risk Identification
Plan Security Measures
Secure Development
Security Testing
Security Validation
Security Response
Transition
Utilization
Training

- Security awareness
- Secure programming
- Threat modeling
- Security static analysis
- Data protection and privacy
- Security expert curriculum

- Security Risk Identification and Management (SECURIM)
- Data Privacy Impact Assessment
- Threat Modeling
- Secure programming
- Dynamic testing
- External security assessment
- Secure product compliance
- Plan security features
- Plan security tests
- Plan security response
- Independent security assessment
- Execute the security response plan

How We Used To Develop Software

- Very few external dependencies
- Full control over source code

How We Develop Software Today

- Many external dependencies
- Only control over a small part of the source code

The Maintenance Challenge

- > 90% of customers are using the latest two releases
- > 50% of customers are using releases older 10 years

Product | Release | EoL | Ext. EoL
---|---|---|---
Windows XP | 2001 | 2009 | 2014
Windows 8 | 2012 | 2018 | 2023
SAP SRM | 2006 | 2018 | 2023
Red Hat | 2012 | 2020 | 2023
Tomcat | 2007 | 2016 | n/a

Source: SAP’s Security Development Lifecycle (S2DL)
### Types of Third-Party Software

<table>
<thead>
<tr>
<th>Commercial Libraries</th>
<th>Outsourcing</th>
<th>Freeware</th>
<th>Free/Libre Open Source Software (FLOSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outsourcing</td>
<td>SAP HANA</td>
<td></td>
<td>Jabra Device Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NVIDIA Device Driver</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Apache Tomcat</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>JQuery</td>
</tr>
</tbody>
</table>

| Upfront costs        | High        | Low       | Low                                  |
| Ease of access (for developers) | Hard      | Medium    | Easy                                 |
| Modification of Source Code | Depends on contract | Impossible | Possible                             |
| Support contract     | Easy        | Hard      | Medium                               |

### Data Sources

**Public**
- FOSS information repositories
  - Open Hub (formerly Ohloh)
  - Core Infrastructure Initiative (CII) Census project
- Public databases of vulnerabilities
  - National Vulnerability Database (NVD)
  - Exploit Database website (ExploitDB)
  - Open Sourced Vulnerability Database (OSVDB)
- Project data
  - Coverity FOSS scan service
  - Source code repositories

**Internal**
- Software inventory (e.g., Black Duck Code Center as used by SAP)

### FLOSS Usage At SAP

**Programming Languages**
- Java
- C
- JavaScript
- PHP
- C++
- Other

**Vulnerabilities (CVEs)**
- DoS
- Code execution
- Overflow
- Bypass something
- Gain information
- XSS
- Gain privileges
- Directory traversal
- Memory corruption
- CSRF

Based on the 166 most used FOSS components (as of autumn 2015)
Part II:
Security of Open Source Enterprise Frameworks
or
Assessing Risks and Planning Efforts of the Secure Consumption of FLOSS

What We Want

1. How many vulnerabilities will be published next year for component X?
2. How often do I need to ship a patch to fix a vulnerability caused by component X?

Vulnerability Prediction

Tomcat 6.x publicly known vulnerabilities (CVEs)

Vulnerability Prediction: Problems

- There is not enough data
- Number of vulnerabilities depends on:
  Age of the project
  Number of users
- Sometimes you simply have no choice...
Understanding Factors Is More Critical Than Predictions

- When will a vulnerability appear in a FOSS component?
  - We do not know

- Can we distinguish features of projects causing "problems" for consuming software?
  - We use maintenance effort of proprietary consumers to denote "problems"
  - Does the "security culture" of FOSS developers make a difference?
  - Does is make a difference which main language/technology is used?

Which Factors Are Interesting?

- Collect all possible data, build a regression model to assess the impact of each factor

- Can we use all data that is available?
  - Actual Total #LoCs of a component
  - Added Total #LoCs of a component
  - Removed Total #LoCs of a component
  - Changed Total #LoCs (added, removed, etc.)...

Relationships Between Factors

Different Maintenance Models

- 60 products are using Apache Tomcat
  - Requires a lot of expertise to resolve security issues
  - It makes more sense to have a team of Apache Tomcat experts around

- 2 products are using a small JavaScript library
  - This does not require any major expertise
  - However, if a company ends up using large number of products for which only the "local" expertise exists, it may be problematic
Centralized Security Maintenance

- Policy: dev. teams must select only components widely used and supported within a company
- A central team resolves vulnerabilities in all FOSS components and pushes changes to all consumers
- The security maintenance effort scales logarithmically with the number of products consuming a component

\[ \text{effort} \propto \log(|\text{vulns}| \times |\text{products}|) \]

Distributed Security Maintenance

- Policy: each dev. team is free of selecting appropriate components
- Each team has to take care of security issues individually
- While this model should decrease the effort for organizational aspects (not considered by us), it adds up for the technical part of the effort

\[ \text{effort} \propto |\text{vulns}| \times |\text{products}| \]

Hybrid Security Maintenance

Part III:

Practical Recommendations On Controlling Risk & Effort Of Using Third Party Components
Strategies For Controlling Risks (1/2)

Secure Software Development Life Cycle
- Maintain a detailed software inventory
  (Do not forget the dependencies)
- Actively monitor vulnerability databases
- Assess project specific risk of third-party components

Obtaining components (or sources)
- Download from trustworthy sources
  (https, check signatures/checksums)

Strategies For Controlling Risks (2/2)

Project Selection
- Prefer projects with private bug trackers
- Evidences of a healthy/working SDLC
  - Documented security fixes/patches
    (no “secret” security fixes)
  - Documented security guidelines
  - Use of security testing tools

Project selection
- Large user base
- Active development community
- Technologies you are familiar with
- Compatible maintenance strategy/life cycle
- Smaller (in terms of code size) and less complex might be better

Part IV: Conclusion
Implement a secure consumption strategy:
• Risk assessment of third party consumption (at least security & licenses)
• Plan for the efforts of secure consumption
• Plan the efforts/costs for response and maintenance

Conclusion

Do not waste time with unimportant questions!
(Is FLOSS more/less secure as proprietary software)

Final advice:
• Accept that you can be hit by a “black swan” (e.g., heartbleed)
• If it happens:
  • Concentrate on understanding and fixing the issue
  • Understanding why you did not find the swan earlier should not be your first priority

Bibliography


