SESSION ID: PDAC-R11

How Understanding Risk Is Changing for Open Source Components

Chris Wysopal
CTO & Co-founder
Veracode
@weldpond
Built Mostly from Components

80% to 95% of modern apps consist of assembled components.
Open Source Repo Stats

Module Counts

<table>
<thead>
<tr>
<th>Language</th>
<th>Packages</th>
<th>Avg. Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl</td>
<td>39,416</td>
<td>2/day</td>
</tr>
<tr>
<td>Java</td>
<td>265,303</td>
<td>131/day</td>
</tr>
<tr>
<td>Node.js</td>
<td>762,073</td>
<td>507/day</td>
</tr>
<tr>
<td>.NET</td>
<td>140,541</td>
<td>67/day</td>
</tr>
<tr>
<td>PHP</td>
<td>210,210</td>
<td>134/day</td>
</tr>
<tr>
<td>Python</td>
<td>165,737</td>
<td>114/day</td>
</tr>
<tr>
<td>Ruby</td>
<td>149,579</td>
<td>27/day</td>
</tr>
</tbody>
</table>

http://www.modulecounts.com/ Data from January, 2019
Pulls from Docker Hub

DockerCon 2017
Open Source – More or Less Secure?

- Defect rate in open source is no better or worse than first party code
- The difference is that developers never revisit
- Integrated and abandoned
- It’s not a problem until a vulnerability is discovered
Sizing The Problem

96% of applications contain open source components.
*Source: Black Duck*

46 Applications have an average of 46 components.
*Source: Veracode*

67% of applications had vulnerabilities in those components.
*Source: Black Duck*

4 On average, vulnerabilities identified have been publicly known for 4 years.
*Source: Black Duck*
Integrated and Abandoned Explicitly - Struts

Struts

- Known Vulnerable
- Not Known Vulnerable
Integrated and Abandoned Implicitly – Apache Commons Collections

Commons Collections

- Known Vulnerable
- Not Known Vulnerable
Where do you find open source vulnerability data?
Not all public vulnerabilities are in the National Vulnerability Database (NVD)

- Public information about open source vulnerabilities is available directly from open source projects
- Security bulletins, release notes, commit comments, and source code comments contain vulnerability information
- This information is readily available to attackers and defenders
- Automated services can crawl this information daily. Security analysts can perform quality review, and compile augmented DB.
## Percentage of vulnerabilities not in the NVD – 31%

<table>
<thead>
<tr>
<th>Language</th>
<th>CVE</th>
<th>Reserved CVE</th>
<th>SVE</th>
<th>% SVE Low</th>
<th>% SVE High</th>
</tr>
</thead>
<tbody>
<tr>
<td>JS</td>
<td>604</td>
<td>47</td>
<td>490</td>
<td>42.94%</td>
<td>44.79%</td>
</tr>
<tr>
<td>PHP</td>
<td>522</td>
<td>14</td>
<td>128</td>
<td>19.28%</td>
<td>19.69%</td>
</tr>
<tr>
<td>DOTNET</td>
<td>58</td>
<td>0</td>
<td>1</td>
<td>1.69%</td>
<td>1.69%</td>
</tr>
<tr>
<td>JAVA</td>
<td>749</td>
<td>60</td>
<td>335</td>
<td>29.28%</td>
<td>30.90%</td>
</tr>
<tr>
<td>RUBY</td>
<td>284</td>
<td>43</td>
<td>268</td>
<td>45.04%</td>
<td>48.55%</td>
</tr>
<tr>
<td>PYTHON</td>
<td>389</td>
<td>59</td>
<td>228</td>
<td>33.73%</td>
<td>36.95%</td>
</tr>
<tr>
<td>GO</td>
<td>90</td>
<td>5</td>
<td>218</td>
<td>69.65%</td>
<td>70.78%</td>
</tr>
<tr>
<td>CPP</td>
<td>193</td>
<td>8</td>
<td>12</td>
<td>5.63%</td>
<td>5.85%</td>
</tr>
<tr>
<td>OBJECTIVEC</td>
<td>631</td>
<td>14</td>
<td>9</td>
<td>1.38%</td>
<td>1.41%</td>
</tr>
<tr>
<td>CSHARP</td>
<td>33</td>
<td>3</td>
<td>0</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>3553</td>
<td>253</td>
<td>1689</td>
<td><strong>30.74%</strong></td>
<td><strong>32.22%</strong></td>
</tr>
</tbody>
</table>

% SVE Low assumes reserved CVEs overlap with SVEs

% SVE High assumes reserved CVEs do not overlap with SVEs
Do I need to remediate every vulnerable component?
Component Vulnerability Exploitability

- A product is vulnerable when it contains a vulnerable component and the product uses the library in such a way that the vulnerable code can be exercised.

- Control flow analysis was used determine if vulnerable code is reachable from the product code.

- Analysis was not performed to determine if vulnerable code can be called directly by attacker or called when attacker has exploited another vulnerability.
For Java, Ruby and Python, less than 5% of products that contain a library with a vulnerability are vulnerable

<table>
<thead>
<tr>
<th>repos analyzed</th>
<th>% component vulnerabilities that make the products vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruby</td>
<td>624</td>
</tr>
<tr>
<td>Java</td>
<td>5897</td>
</tr>
<tr>
<td>Python</td>
<td>624</td>
</tr>
</tbody>
</table>

JavaScript study found 26.7% made products vulnerable

Towards Smoother Library Migrations: A Look at Vulnerable Dependency Migrations at Function Level for npm JavaScript Packages

What does a good process look like?
Building a solution into your development lifecycle.

- Integrate into your CI/CD pipeline so you have a record of what goes into production.
- Scanning your repos is OK but pipeline could apt-get update to a vulnerable version.
- Open tickets in your ticketing system for each component that should be updated.
- Create a policy of grace period by severity and vulnerability.
How much risk are organizations finding from open source components

Average vulns per app

<table>
<thead>
<tr>
<th>Language</th>
<th>go</th>
<th>objective-c</th>
<th>python</th>
<th>java</th>
<th>php</th>
<th>ruby</th>
<th>c#</th>
<th>javascript</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How much risk are organizations remediating?

Fix percentages by number of scans

- LOW
- MEDIUM
- HIGH

Scans:
- 2-10
- 11-30
- 31-80
- 81-130
- 130+
What if we know the component with a vulnerability is making my app vulnerable right now?

Avg # of scans to fix a vulnerable component: **22**

Avg # of scans to fix a vulnerable component with a vulnerable method: **17**

Proportion of vulnerable components fixed: **32%**

Proportion of vulnerable components with vulnerable methods fixed: **41%**

Developers **fix more and fix faster** when they have vulnerable method information available.
Use fewer, better suppliers

- Toyota’s lean manufacturing model uses fewer, better suppliers
- Use containers, libraries and frameworks that are proven to work and vetted by your security team and in your repository
- Keep track of what you have!
Apply What You Have Learned Today

• Next week you should:
  – Understand the process for managing open source code within your development organization

• In the first three months following this presentation you should:
  – Create an inventory of open source code
  – Remediate where outdated and vulnerable open source is used in critical applications

• Within six months you should:
  – Integrate a process into your development lifecycle to monitor what open source is going into production
Questions?

Thank You!

Chris Wysopal

@weldpond