Less is More
Quantifying the Security Benefits of Debloating Web Applications

Babak Amin Azad    Pierre Laperdrix    Nick Nikiforakis
Stony Brook University

PragSec Lab
What is software debloating?

“Reducing the **attack surface** by removing pieces of code that are not required by users.”
You’re vulnerable, but do you have to be?

Web Cache Poisoning vulnerability on Drupal

https://portswigger.net/blog/practical-web-cache-poisoning

GET /education?x=y HTTP/1.1
Host: store.unity.com
X-Original-URL: /gambling?x=y

Unused and keyed

Used and keyed

Unused and unkeyed

X-Original-URL
X-Rewrite-URL
Debloating Pipeline

1. Vulnerability to Code Mapping
2. Application Profiling By Usage Simulation
3. Record Code Coverage
4. Analyze Unused Files / Functions
5. Debloating Logic
6. Rerun Tests to Verify Correctness
7. Test Against Known Exploits
8. Analyze The Results
Identifying important functionalities of an application

- Find tutorials for these applications
- Automate them using Selenium

**Example of tasks covered by tutorials**

1. Login
2. Create a database
3. Create tables
4. Run queries
5. Drop database
6. ...

**What’s not covered by tutorials**

1. Some pages on the front of the application
2. Error handlers
Expanding the breadth of coverage

Monkey Testing

Spider

Vulnerability Scanner
Files covered by each testing tool

(a) phpMyAdmin 4.7.0
(b) MediaWiki 1.28.0
(c) Magento 2.0.5
(d) WordPress 4.7.1
File & Function level debloating

- Remove the contents of unused files/functions
- Use place holders
  - Log information about execution of removed code
  - Stop the execution flow to prevent entering an unknown state
Results #1: Reduction of LLOC after debloating

File Debloating
- Average 33% reduction
- WordPress: 9%
- Magento: 65% (400 KLLOC)

Function Debloating
- Average 47% reduction (+14%)
- WordPress: 31% (+22%)
- Magento 71% (+6%)
Results #2: Reduction of Cyclomatic Complexity

File Debloating
- Average of 32.5% reduction
- WordPress: 6%
- Magento: 74.3%

Function Debloating
- Average 50.3% reduction (+18%)
- WordPress: 24% (+18%)
- Magento 80.2% (+6%)
<table>
<thead>
<tr>
<th>ID</th>
<th>CVE</th>
<th>Software</th>
<th>Version</th>
<th>File Name</th>
<th>Triggered</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>CVE-2014-8959</td>
<td>phpMyAdmin</td>
<td>4.0.0</td>
<td>libraries/gis/pma_gis_factory.php</td>
<td>✗</td>
</tr>
<tr>
<td>63</td>
<td>CVE-2013-3240</td>
<td>phpMyAdmin</td>
<td>4.0.0</td>
<td>libraries/plugin_interface.lib.php</td>
<td>✓</td>
</tr>
<tr>
<td>24</td>
<td>CVE-2016-6619</td>
<td>phpMyAdmin</td>
<td>4.0.0</td>
<td>libraries/Table.class.php</td>
<td>✓</td>
</tr>
<tr>
<td>22</td>
<td>CVE-2016-6609</td>
<td>phpMyAdmin</td>
<td>4.0.0</td>
<td>libraries/plugins/export/ExportPhparray.class.php</td>
<td>✓</td>
</tr>
<tr>
<td>21</td>
<td>CVE-2016-9866</td>
<td>phpMyAdmin</td>
<td>4.0.0</td>
<td>prefs_manage.php</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Results #3: Reduction of CVEs

<table>
<thead>
<tr>
<th>Application</th>
<th>Strategy</th>
<th>Total Removed CVEs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>File Debloating</td>
<td>4/20 20 %</td>
</tr>
<tr>
<td>phpMyAdmin</td>
<td>Function Debloating</td>
<td>12/20 60 %</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>File Debloating</td>
<td>8/21 38 %</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>10/21 47.6 %</td>
</tr>
<tr>
<td>WordPress</td>
<td>File Debloating</td>
<td>0/20 0 %</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>2/20 10 %</td>
</tr>
<tr>
<td>Magento</td>
<td>File Debloating</td>
<td>1/8 12.5 %</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>3/8 37.5 %</td>
</tr>
</tbody>
</table>
Types of vulnerabilities removed by debloating

- Crypto and cookie related vulnerabilities usually can’t be removed by debloating.
- CSRF vulnerabilities are only removed when the underlying feature is removed.
- Code execution vulnerabilities can either be removed or broken by removing the POI gadgets.
## Effect of external dependencies on software bloat

<table>
<thead>
<tr>
<th>Application</th>
<th>Before debloating</th>
<th>After function-level debloating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LLOC in main App</td>
<td>LLOC in packages</td>
</tr>
<tr>
<td>phpMyAdmin 4.7.0</td>
<td>36k</td>
<td>82k</td>
</tr>
<tr>
<td>MediaWiki 1.28.0</td>
<td>133k</td>
<td>51k</td>
</tr>
<tr>
<td>Magento 2.0.5</td>
<td>396k</td>
<td>213k</td>
</tr>
</tbody>
</table>
Statistics about removed external packages

<table>
<thead>
<tr>
<th>Application</th>
<th># Packages</th>
<th># packages completely removed</th>
<th># packages with &lt; 30 % of lines removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpMyAdmin 4.7.0</td>
<td>45</td>
<td>38 (84 %)</td>
<td>4</td>
</tr>
<tr>
<td>MediaWiki 1.28.0</td>
<td>40</td>
<td>24 (60 %)</td>
<td>12</td>
</tr>
<tr>
<td>Magento 2.0.5</td>
<td>71</td>
<td>58 (82 %)</td>
<td>2</td>
</tr>
</tbody>
</table>

But if a package is never used, does it contribute to the attack surface?
PHP Object Injection (POI) attacks

- Unsafe object deserialization vulnerability is the target of this exploit.
  (Also known as Property Oriented Programming, POP)
- Attacker can control value of properties on injected objects.
- But the attacker cannot control execution of functions.
- The chain is made based on magic functions.
- The chain usually ends with a write to file system or a database transaction.

Magic functions:
- __construct()
- __toString()
- __destruct()
- __wakeup()
- ...
## Results #4: Reduction of object injection gadgets

<table>
<thead>
<tr>
<th>Application</th>
<th>Package</th>
<th>Removed by Debloating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>File</td>
</tr>
<tr>
<td><strong>phpMyAdmin 4.7.0</strong></td>
<td>Doctrine</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Guzzle</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>MediaWiki 1.28.0</strong></td>
<td>Monolog</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Magento 2.0.5</strong></td>
<td>Doctrine</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Monolog</td>
<td>❌</td>
</tr>
<tr>
<td></td>
<td>Zendframework</td>
<td>❌</td>
</tr>
</tbody>
</table>
Source code and the artifacts are publicly available

- Debloating pipeline to evaluate and debloat custom applications
- Debloated web applications
- Source code coverage information
- CVE to source code mappings & Exploits

https://debloating.com
Conclusion

- Debloating can reduce web applications attack surface significantly
  - Up to 71% reduction in LLOC
  - Up to 60% reduction in CVEs
  - Up to 100% removal of POI Gadgets

- Web vulnerabilities & their exploitation is different, as a result web debloating is different (Targeting actual vulnerabilities rather than dead code)

- We also need to focus on usability and performance of debloating schemes

- Artifacts and debloated applications are available at: https://debloating.com