

Fuzzing Your Favorite Interpreter

EMMANUEL LAW



Background

- Principal Security Consultant @ Aura Info Sec
- Pentesting for living
- @libnex
- Found some PHP bugs...



Minimum Demonstrate the presence of a security bug with probable remote exploitation potential. \$500

The project maintainers have final decision on which issues constitute security vulnerabilities. Only issues that are tagged as **Type: Security** by a project maintainer will be considered for bounty eligibility. The Panel will respect their decision, and we ask that you do as well.

It's important to keep in mind that not all submissions will qualify for a bounty, and that the decision to award a bounty is entirely at the discretion of the Panel.

Submission Process

- Disclose a previously unknown security vulnerability directly to the project maintainers.
- Follow the disclosure process established by the project maintainers.
- Clearly demonstrate the security vulnerability. Respect the time of the project volunteers as they cannot invest significant effort into incomplete reports. Low-quality reports may be disqualified.
- Once a public security advisory has been issued, please submit a report here. You must not send us the details of the vulnerability until it has been validated, accepted, and publicly disclosed by the project maintainers.



Hackers thanked (32)



ryat

Reputation: 346



fms

Reputation: 208



l4w

Reputation: 156



libnex

Reputation: 141



haquaman

Reputation: 119

[All Hackers](#)



Fuzzing Interpreters



Build From Scratch



Off-The-Shelf



Writing a Custom Fuzzer from Scratch



Pros

- Custom Strategies
- Find Uniq Bugs

Cons

- Time + Effort
- Portability to other languages



Off The Shelf

Pros

- Speed
- Power of the Open Source Community

Cons

- Less customization
- Competitionlots of them



Fuzzing Interpreters



Build From Scratch

VS



Off-The-Shelf



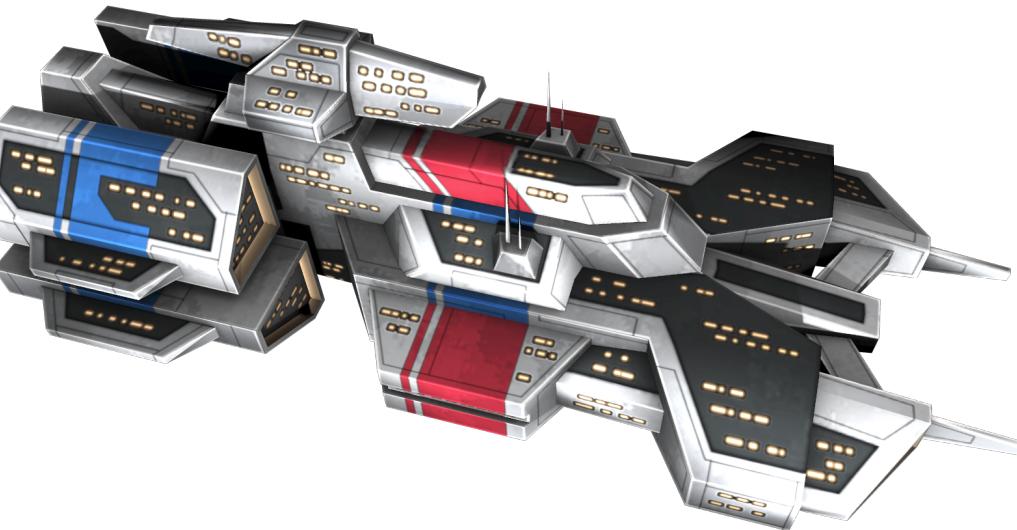
Attack Plan

Fuzzing

Triage

RCA





Battle Plan

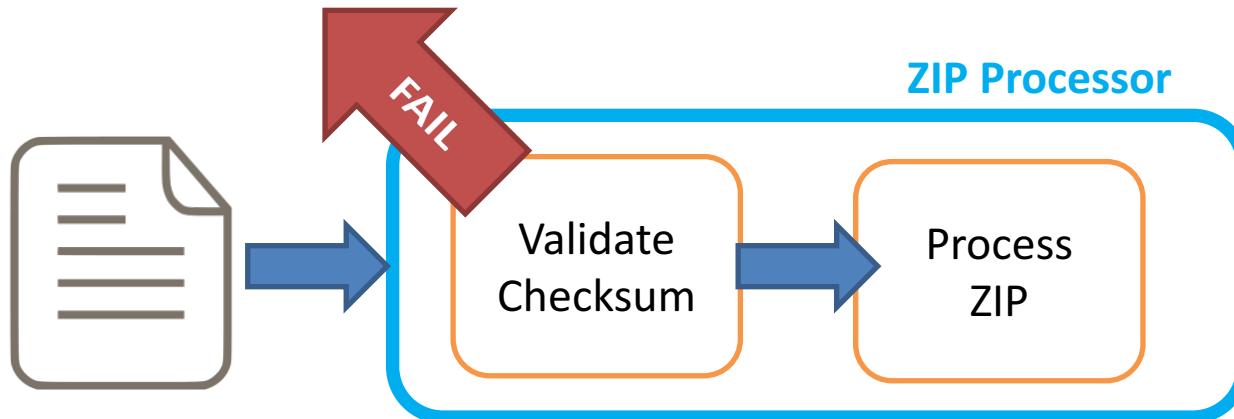
What are we fuzzing?

- Attack Surface Area



BattlePlan : Attacking Files Parsers

- Examples: Zip, Images, Phar, PYZ
- Take the road less travelled
- Patch-out Checksum verification



BattlePlan: Fuzzing Corpus



BattlePlan: Fuzzing Corpus

- More Unique => Better chance of finding a crash
- Exercises as many code path as possible
- Harness Regression Test cases:
 - Test edge cases
 - Don't forget test cases from sister projects



Fuzzing

Choosing a Fuzzer



Choosing a Fuzzer

- 101 Fuzzers out there
- Things to consider:
 - Speed
 - Popularity
 - Easy of use
 - Constraints: Source code?
 - Buzz words: Evolutionary Fuzzing, In-memory fuzzing



Fuzzing: American Fuzzy Lop (AFL)

```
american fuzzy lop 1.74b (readelf)

process timing
  run time : 0 days, 0 hrs, 8 min, 24 sec
  last new path : 0 days, 0 hrs, 1 min, 59 sec
  last uniq crash : 0 days, 0 hrs, 3 min, 17 sec
  last uniq hang : 0 days, 0 hrs, 3 min, 23 sec
overall results
  cycles done : 0
  total paths : 812
  uniq crashes : 8
  uniq hangs : 10

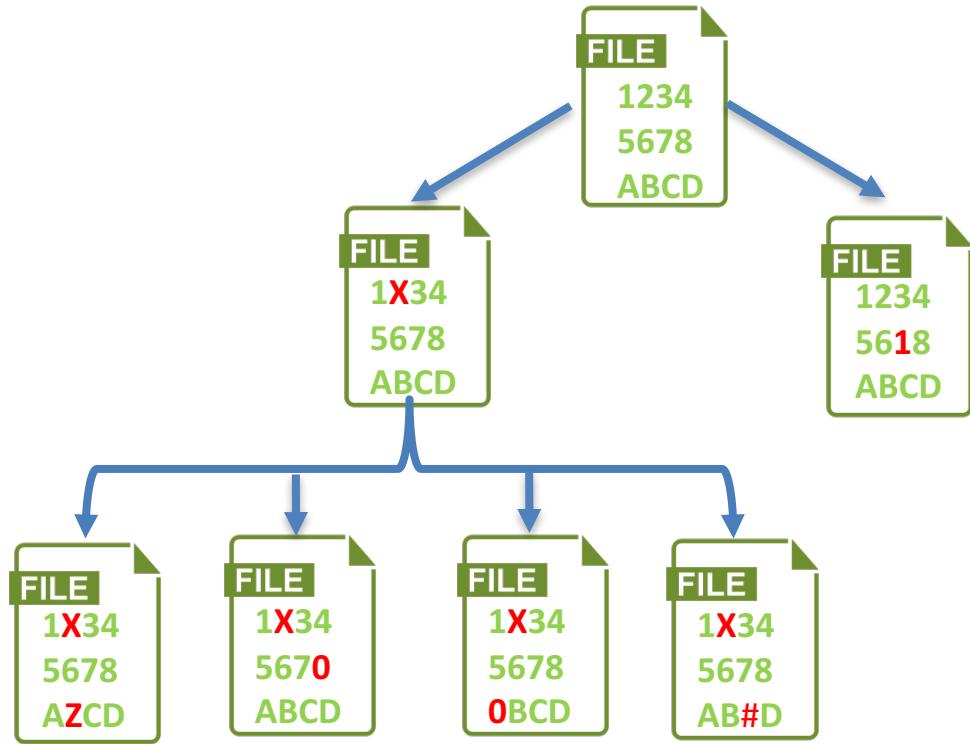
cycle progress
  now processing : 0 (0.00%)
  paths timed out : 0 (0.00%)
map coverage
  map density : 3158 (4.82%)
  count coverage : 2.56 bits/tuple
stage progress
  now trying : arith 8/8
  stage execs : 295k/326k (90.31%)
  total execs : 552k
  exec speed : 1114/sec
findings in depth
  favored paths : 1 (0.12%)
  new edges on : 318 (39.16%)
  total crashes : 63 (8 unique)
  total hangs : 191 (10 unique)
fuzzing strategy yields
  bit flips : 447/75.5k, 59/75.5k, 59/75.5k
  byte flips : 7/9436, 0/5858, 6/5950
  arithmetics : 0/0, 0/0, 0/0
  known ints : 0/0, 0/0, 0/0
  dictionary : 0/0, 0/0, 0/0
  havoc : 0/0, 0/0
  trim : 0.00%/1166, 38.39%
path geometry
  levels : 2
  pending : 812
  pend fav : 1
  own finds : 811
  imported : n/a
  variable : 0

[cpu: 15%]
```

- Gold Standard
- EVERYONE is using this 😊
- Feedback driven



Feedback Driven/Evolutionary/Genetic Fuzzing



Radamsa

- General Purpose Fuzzer
- Language/Data agnostic
- Semi-Smart
- Extremely easy to use



Other Fuzzers

- honggfuzz
- Choronzon
- zzuf
- So many many more..

Different Fuzzers will find different bugs



Fuzzing: Getting better Mileage

- Address Sanitizer (aka ASAN):
 - Compile into your interpreter
 - Memory error detector
 - Minimal overhead



So you have found some crashes.....



Triage

- Purpose
 - Grouping of similar crashes
 - Prioritize your crashes



Triage

- Comes free with Address Sanitizer

```
=5268==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffffff8f10 at pc 0x7ffff551d1eb bp 0x7fffffff7ec0
WRITE of size 4096 at 0x7fffffff8f10 thread T0
#0 0x7ffff551d1ea (/usr/lib/x86_64-linux-gnu/libasan.so.1+0x2e1ea)
#1 0x9353ca in phar_set_inode /home/elaw/php-5.6.7/ext/phar/phar_internal.h:540
#2 0x941015 in phar_parse_zipfile /home/elaw/php-5.6.7/ext/phar/zip.c:638
#3 0x974a85 in phar_open_from_fp /home/elaw/php-5.6.7/ext/phar/phar.c:1703
#4 0x9727fa in phar_create_or_parse_filename /home/elaw/php-5.6.7/ext/phar/phar.c:1346
#5 0x9724da in phar_open_or_create_filename /home/elaw/php-5.6.7/ext/phar/phar.c:1315
#6 0x98c857 in zim_PharDownConstruct /home/elaw/php-5.6.7/ext/phar/phar_object.c:1189

Address 0x7fffffff8f10 is located in stack of thread T0 at offset 4128 in frame
#0 0x934f85 in phar_set_inode /home/elaw/php-5.6.7/ext/phar/phar_internal.h:534

This frame has 1 object(s):
[32, 4128) 'tmp' ==> Memory access at offset 4128 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
(longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow ???:0 ??
Shadow bytes around the buggy address:
0x10007fff5190: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10007fff51a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10007fff51b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10007fff51c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x10007fff51d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x10007fff51e0: 00 00 [f3] f3 f3 00 00 00 00 00 00 00 00 00 00 00
```

Stack Trace

Visual Mem-map



Triage: Exploitability

- !exploitable

```
gdb-peda$ exploitable
Description: Access violation near NULL on source operand
Short description: SourceAvNearNull (16/22)
Hash: d5dfd9cdde872c76db6b0d537c7e6f2f.132b523e45ed0a73c793e758614f357e
Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
Explanation: The target crashed on an access violation at an address matching the source operand of the current instruction. This likely indicates that the application crashed on a simple NULL dereference to data structure that has no immediate effect on control of the processor.
Other tags: AccessViolation (21/22)
```



Triage: Test case minimization

- Fuzzdiff, Afl-min etc
- Find the minimal changes that causes the crash



Root Cause Analysis



Root Cause Analysis

- Trying to find the answers:
 - What is causing the Crash
 - Is it exploitable
- Very tedious and time consuming
- Remember you are competing on speed..



Root Cause Analysis

- I spend a lot of time in GDB
- PEDA* is your friend

```
Breakpoint 1, 0x0000000000454810 in main ()
(gdb) █
```

*Python Exploit Development Assistance



```
RSP: 0x7fffffff1e130 --> 0x7fffffff348 --> 0x7fffffff61d ("~/home/elaw/php-7.0.0/sapi/cli/php_pure_00")
RIP: 0xbb4d3c (<main+24>:          mov    DWORD PTR [rbp-0x4],0x0)
R8 : 0x14f6d40 --> 0x7ffff0f63c60 --> 0x0
R9 : 0x7ffff7deae20 (<_dl_fini>:      push   rbp)
R10: 0x7fffffff0f0 --> 0x0
R11: 0x7ffff0be0a50 (<__libc_start_main>: push   r14)
R12: 0x4456f0 (<_start>:           xor    ebp,ebp)
R13: 0x7fffffff340 --> 0x2
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-----] code
0xbb4d28 <main+4>: sub    rsp,0x130
0xbb4d2f <main+11>: mov    DWORD PTR [rbp-0x124],edi
0xbb4d35 <main+17>: mov    QWORD PTR [rbp-0x130],rsi
=> 0xbb4d3c <main+24>: mov    DWORD PTR [rbp-0x4],0x0
0xbb4d43 <main+31>:  mov    DWORD PTR [rbp-0x8],0x0
0xbb4d4a <main+38>:  mov    DWORD PTR [rbp-0xc],0x0
0xbb4d51 <main+45>:  mov    QWORD PTR [rbp-0x50],0x0
0xbb4d59 <main+53>:  mov    DWORD PTR [rbp-0x54],0x1
[-----] stack
0000| 0x7fffffff1e130 --> 0x7fffffff348 --> 0x7fffffff61d ("~/home/elaw/php-7.0.0/sapi/cli/php_pure_00")
0008| 0x7fffffff1e138 --> 0x2f7fb59c8
0016| 0x7fffffff1e140 --> 0x7fffffff290 --> 0x0
0024| 0x7fffffff1e148 --> 0x7ffff7ffe500 --> 0x7ffff7ffe460 --> 0x7ffff7fb5758 --> 0x7ffff7fb575f 0x0
0032| 0x7fffffff1e150 --> 0x7fffffff2b8 --> 0x0
0040| 0x7fffffff1e158 --> 0x7ffff7ffe1a8 --> 0x0
0048| 0x7fffffff1e160 --> 0x1
0056| 0x7fffffff1e168 --> 0x7ffff7de577d (<_dl_lookup_symbol_x+349>:      cmp    ebx,0x0)
[-----]
Legend: code, data, rodata, value
```

AURA INFORMATION
Breakpoint 1, main (argc=0x2, argv=0x7fffffff348) at /home/elaw/php-7.0.0/sapi/cli/php_cli.c:1173
1173 int exit_status = SUCCESS;
gdb-peda\$



Root Cause Analysis

- Really? GDB?? pfft.. *scorn*

```
[code]
0x1000000ebd: jmp 0x100000ea8
0x1000000ec2: push 0xe
0x1000000ec7: jmp 0x100000ea8
0x1000000ec: push 0xb
0x1000000ed1: jmp 0x100000ea8
0x1000000ed6: sub ch,BYTE PTR [rdx]
0x1000000ed8: sub ah,BYTE PTR [rax]
0x1000000eda: je 0x100000f41
0x1000000edc: jae 0x100000f52
0x1000000ede: pop rdi
0x1000000edf: data16 jne 0x100000f50
0x1000000ee2: movsxwd esi,WORD PTR [rcx+rbp*2+0x6f]
0x1000000ee6: outs dx,BYTE PTR ds:[rsi]
0x1000000ee7: sub BYTE PTR [rcx],ch
0x1000000ee9: or al,BYTE PTR [rax]
0x1000000ebe: jae 0x100000f59
0x1000000eed: gs gs jo 0x100000ef1
0x1000000ef1: sub ch,BYTE PTR [rdx]
0x1000000ef3: sub ah,BYTE PTR [rax]
0x1000000ef5: push rbx
0x1000000ef6: ins BYTE PTR es:[rdi],dx
0x1000000ef7: gs gs jo 0x100000f64

[gdb]
gdb$ 0x0000000100000e23 in main ()
gdb$ 0x0000000100000e23 in main ()
gdb$ 0x0000000100000e6 in main ()
gdb$ 0x0000000100000e75 in main ()
gdb$ 0x0000000100000e77 in main ()
gdb$ 0x0000000100000e96 in ?? ()
gdb$ 0x0000000100000eb8 in ?? ()
gdb$ 0x0000000100000ebd in ?? ()
gdb$ 

[breakpoints]
#1 0x0000000100000CF0 h:1 main
#0 0x0000000100000ebd in ?? ()
#1 0x0000000000000000 in ?? ()

[regs:general]
[ o d I T s Z a P c ]
RIP: 0000000100000EBB
RAX: 0000000000000000
RBX: 0000000000000000
RBP: 00007FFF5FBFF940
RSP: 00007FFF5FBFF8E8
RDI: 00000000100000F63
RSI: 00007FFF5FBFF960
RDX: 00007FFF5FBFF970
RCX: 00007FFF5FBFFA70
R8: 0000000000000000
R9: 00007FFF5FBFEA08
R10: 00000000000000320
R11: 00000000000000240
R12: 00000000000000000
R13: 00000000000000000
R14: 00000000000000000
R15: 00000000000000000
CS: 0028 DS: N/A
ES: N/A FS: 0000
GS: 0000 SS: N/A

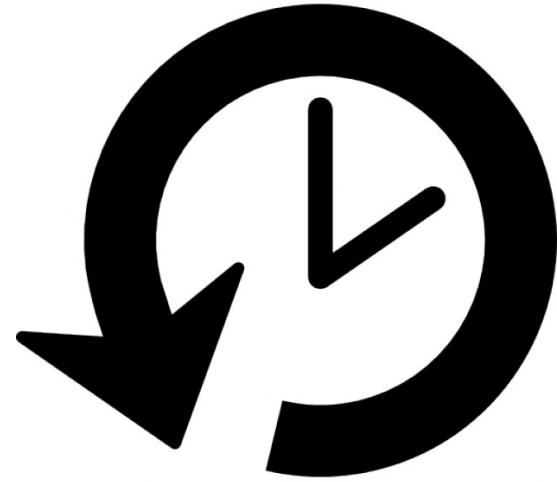
[backtrace]
```

Voltron



The art of

Reverse Debugging



Root Cause Analysis: Reverse Debugging

- Debugging tends to be very linear

```
PHPAPI char *php_url_scanner_adapt_single_url(const char *name, size_t len, const char *url, size_t urllen, int urlencode)
{
    char *result;
    smart_str surl = {0};
    smart_str buf = {0};
    smart_str url_app = {0};
    zend_string *encoded;

    smart_str_appendl(&surl, url, urllen);

    if (urlencode) {
        encoded = php_raw_url_encode(name, str);
        smart_str_appendl(&url_app, ZSTR_VAL(encoded));
        zend_string_free(encoded);
    } else {
        smart_str_appendl(surl, url_name);
    }
}
```

```
PHPAPI zend_string *php_raw_url_encode(char const *s, size_t len)
{
    register int x, y;
    zend_string *str;

    str = zend_string_alloc(len, 0);
    for (x = 0, y = 0; len-- ; x++, y++) {
        ZSTR_VAL(str)[y] = (unsigned char) s[x];
    }
}
```

```
static zend_always_inline zend_string *zend_string_alloc(size_t len)
{
    zend_string *ret = (zend_string *)pemalloc(ZEND_MM_ALLOCATION_ALIGNMENT, len);
    GC_REFCOUNT(ret) = 1;
}
```



Root Cause Analysis: Reverse Debugging

- Record command in GDB
- Provides:
 - Reverse Step
 - Reverse Next
 - Reverse Continue
- Revert to deterministic Memory State



Lets Make Fuzzing Great Again



@libnex

