Fuzzing Your Favorite Interpreter

EMMANUEL LAW

aura INFORMATION SECURITY | POWERED BY KORDIA

Background

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


hackerone
About
Product
Resources
Contact
Directory
Blog
Try HackerOne
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 consider 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 maintainer
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

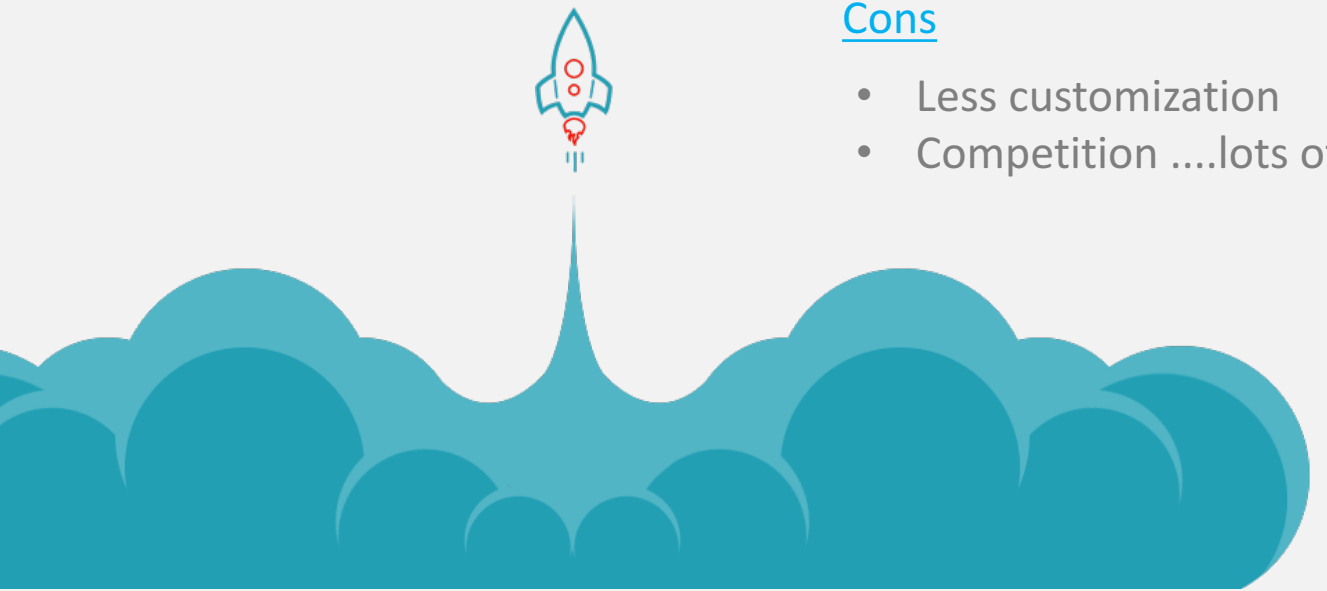
Pros

- 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
- Competition ....lots of them

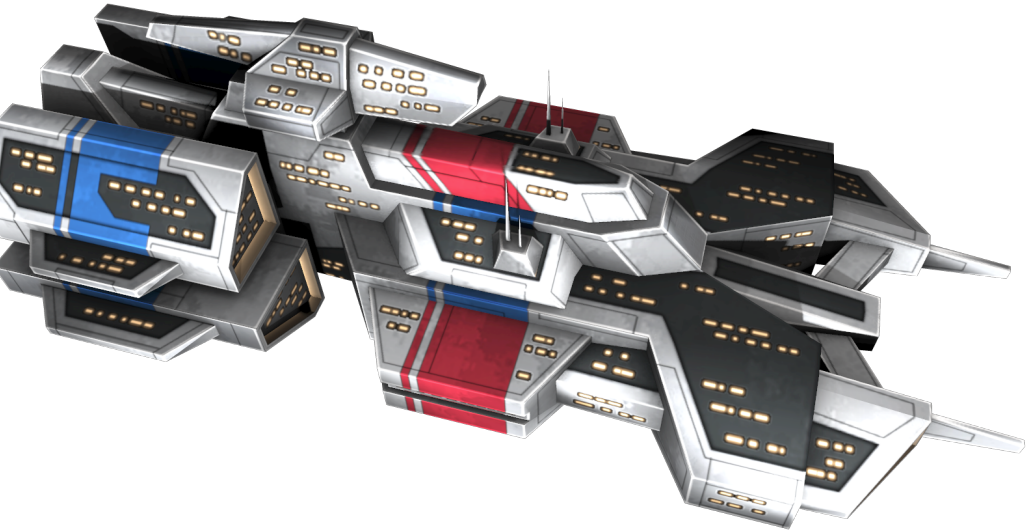
Fuzzing Interpreters

VS

Build From Scratch

Off-The-Shelf

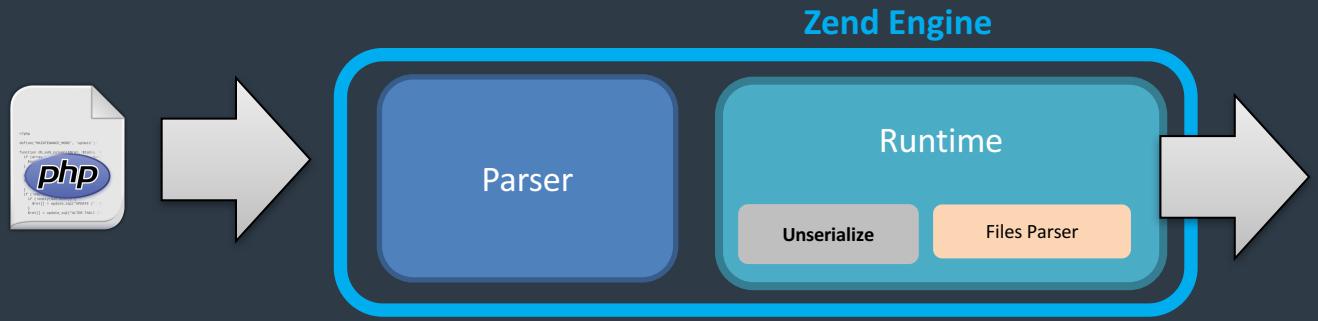

Attack Plan
Fuzzing
Triage
RCA



Battle Plan

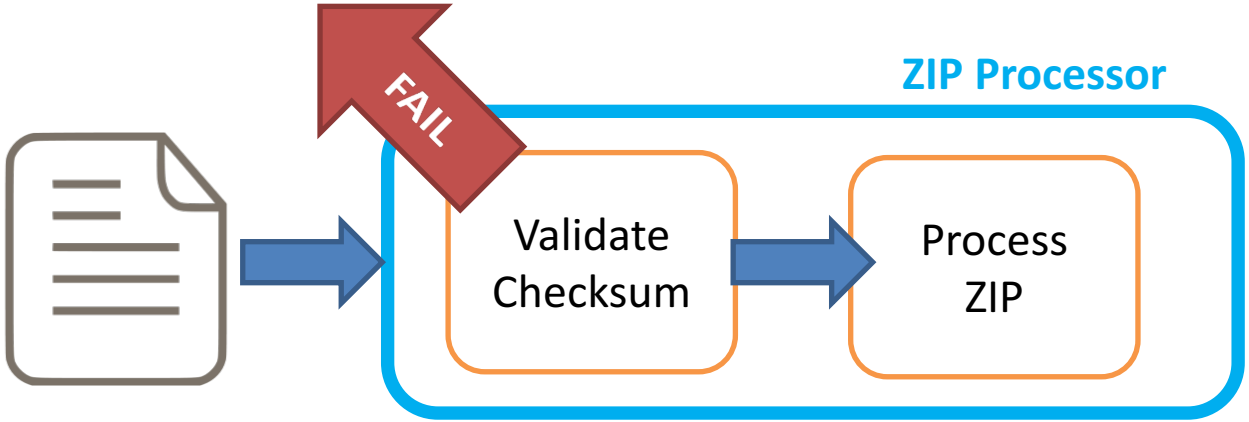
What are we fuzzing?

- Attack Surface Area

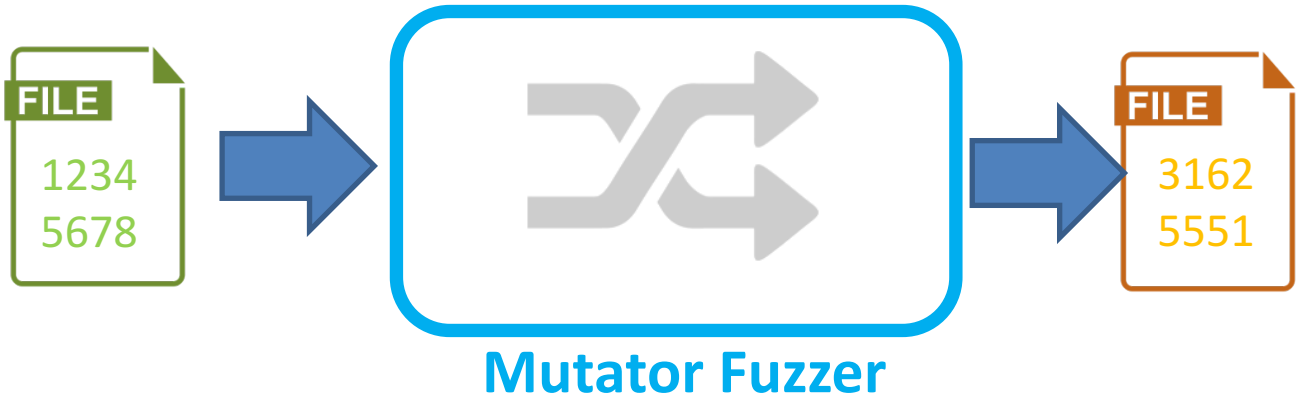


BattlePlan : Attacking Files Parsers

- 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

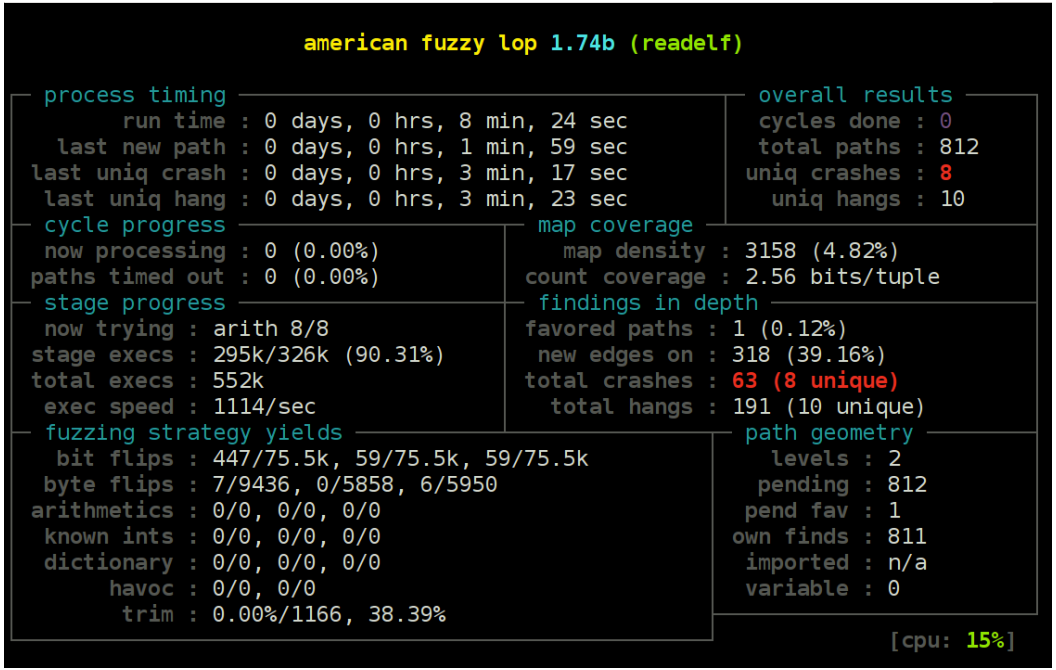
Fuzzing

Choosing a Fuzzer

Choosing a Fuzzer

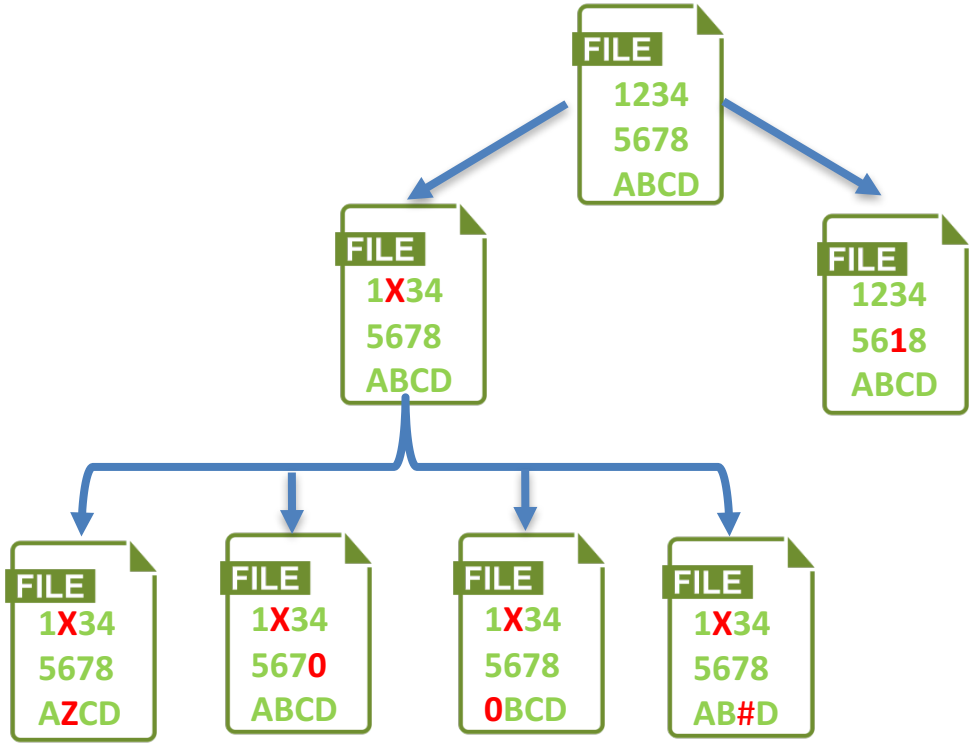
- 101 Fuzzers out there
- Things to consider:
  - Speed
  - Popularity
  - Easy of use
  - Constrains: Source code?
  - Buzz words: Evolutionary Fuzzing, In-memory fuzzing

Fuzzing: American Fuzzy Lop (AFL)



- 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

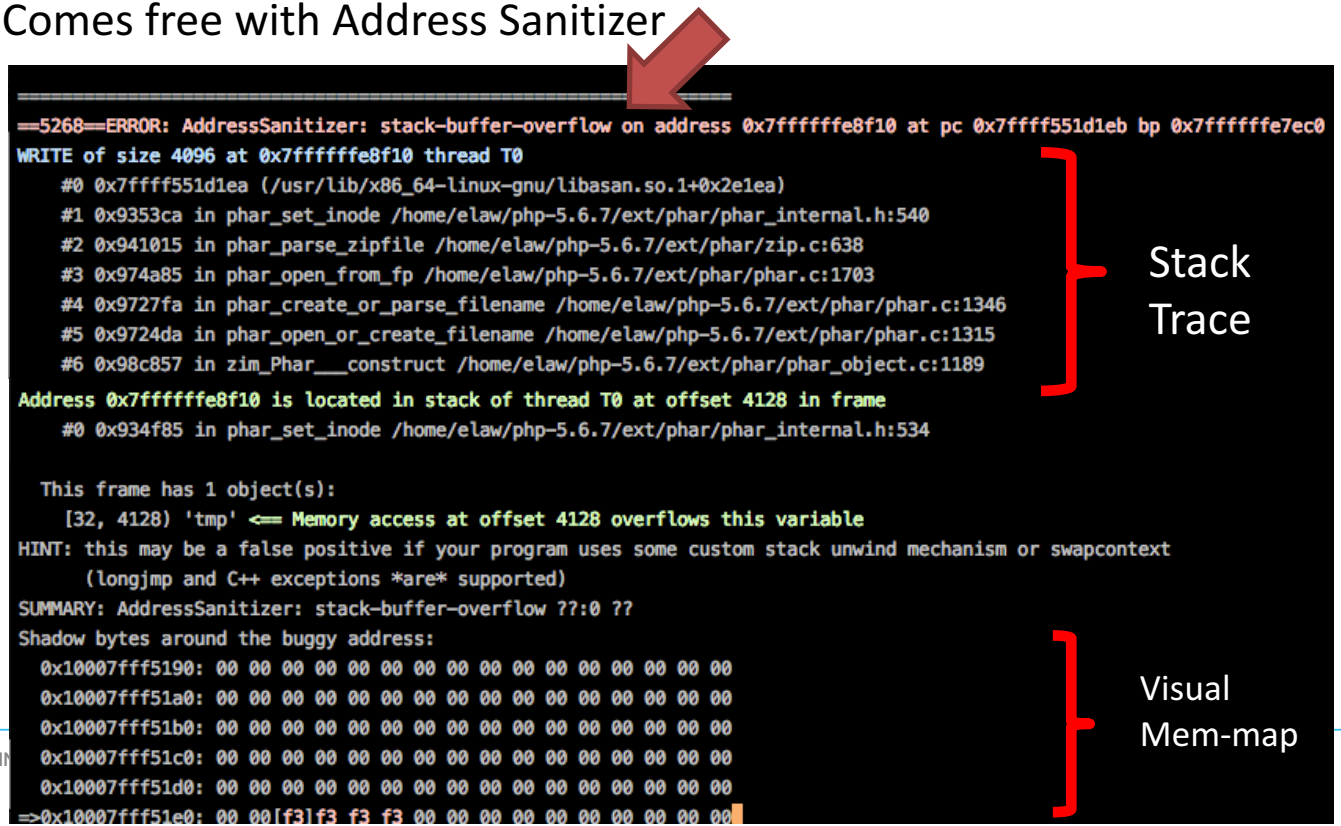
So you have found some crashes…..

Triage

- Purpose
  - Grouping of similar crashes
  - Prioritize your crashes

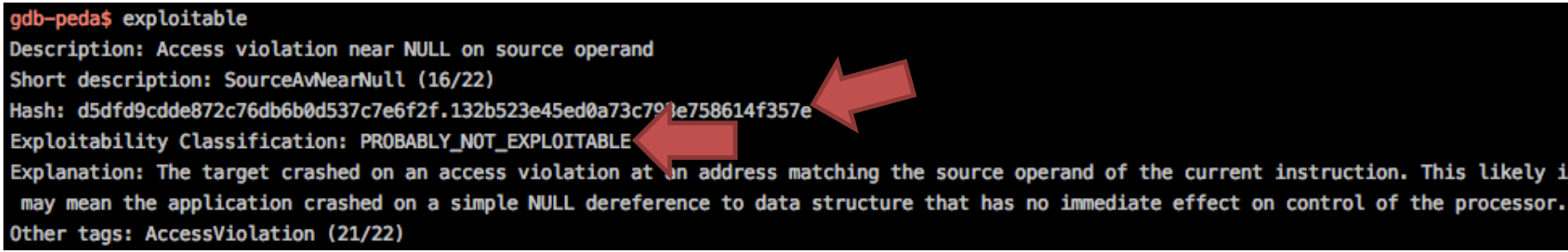
Triage

- Comes free with Address Sanitizer



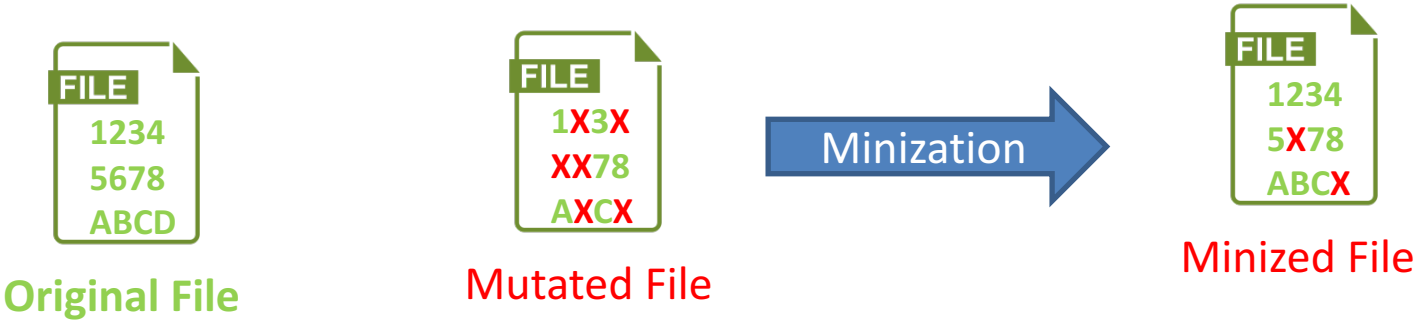
Triage: Exploitability

- !exploitable



Triage: Test case minization

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



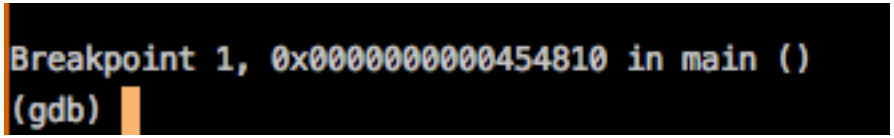
Root Cause Analysis

Root Cause Analysis

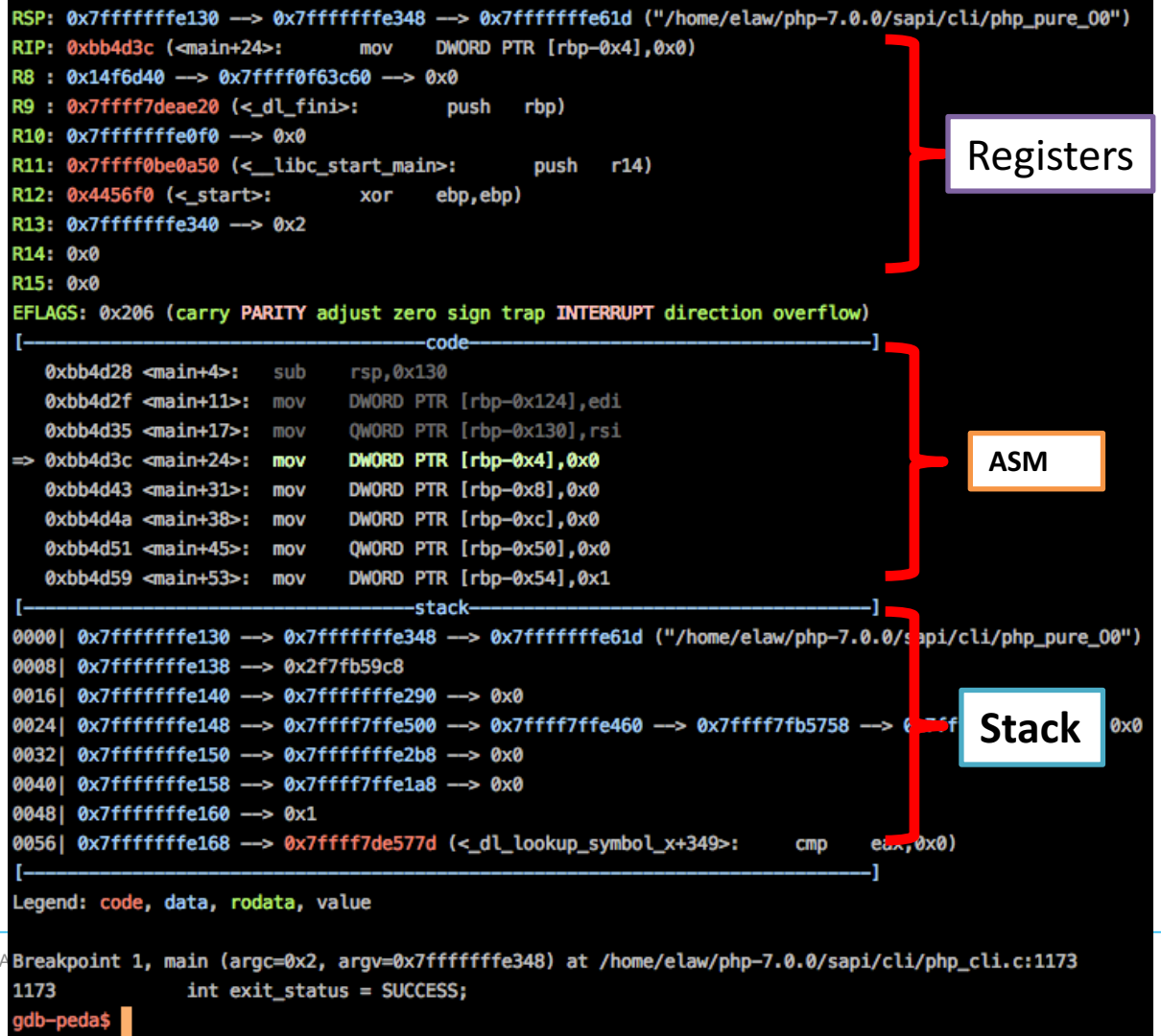
- Trying the find the answers:
  - What is causing the Crash
  - Is it exploitable
- Very tedious and time consuming
- Remember you are competing on speed..

Root Cause Analysis

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

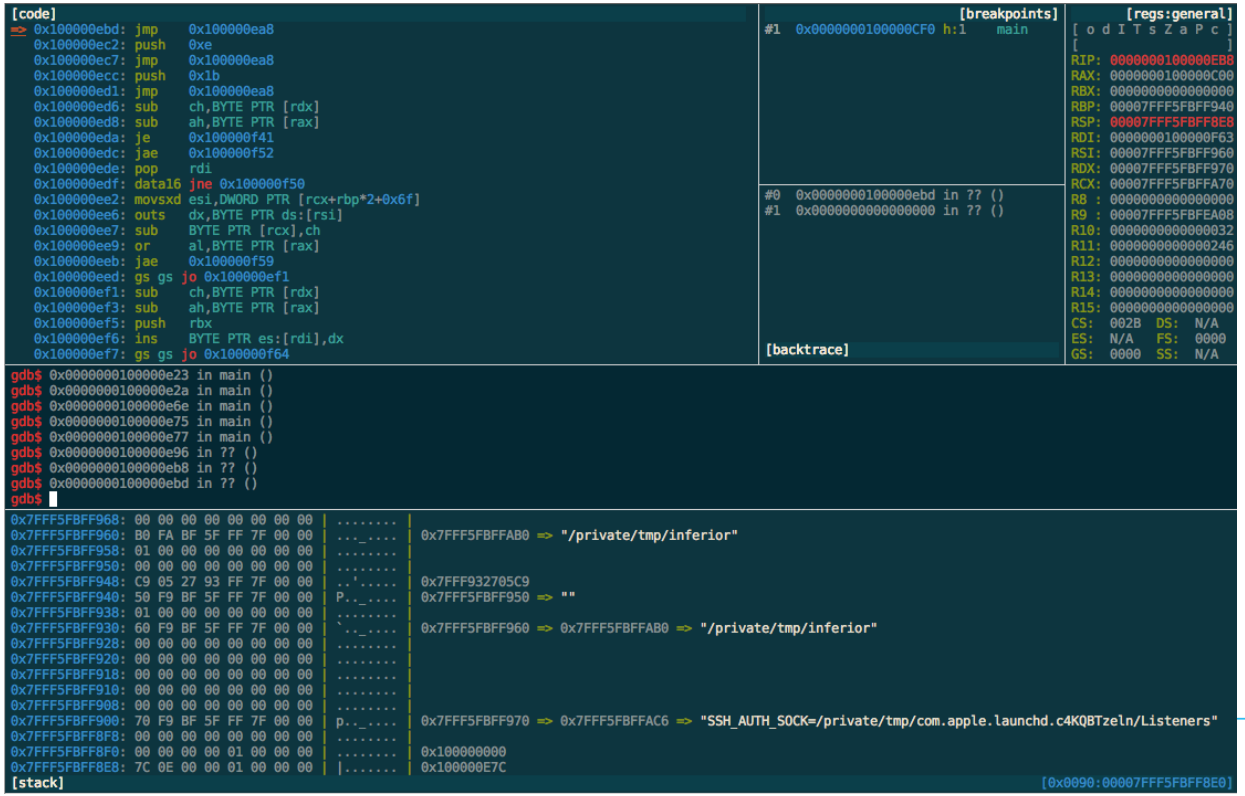


*Python Exploit Development Assistance


RSP: 0x7fffffffe130 --> 0x7fffffffe348 --> 0x7fffffffe61d ("/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: 0x7fffffffe0f0 --> 0x0
R11: 0x7ffff0be0a50 (<__libc_start_main>: push r14)
R12: 0x4456f0 (<_start>: xor ebp,ebp)
R13: 0x7fffffffe340 --> 0x2
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
Registers
[----------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
ASM
[----------stack----------]
0000| 0x7fffffffe130 --> 0x7fffffffe348 --> 0x7fffffffe61d ("/home/elaw/php-7.0.0/sapi/cli/php_pure_00")
0008| 0x7fffffffe138 --> 0x2f7fb59c8
0016| 0x7fffffffe140 --> 0x7fffffffe290 --> 0x0
0024| 0x7fffffffe148 --> 0x7ffff7ffe500 --> 0x7ffff7ffe460 --> 0x7ffff7fb5758 --> 0x0
0032| 0x7fffffffe150 --> 0x7fffffffe2b8 --> 0x0
0040| 0x7fffffffe158 --> 0x7ffff7ffe1a8 --> 0x0
0048| 0x7fffffffe160 --> 0x1
0056| 0x7fffffffe168 --> 0x7ffff7de577d (<_dl_lookup_symbol_x+349>: cmp eax,0x0)
Stack
[----------]
Legend: code, data, rodata, value
Breakpoint 1, main (argc=0x2, argv=0x7fffffffe348) 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?? pffft.. *scorn*



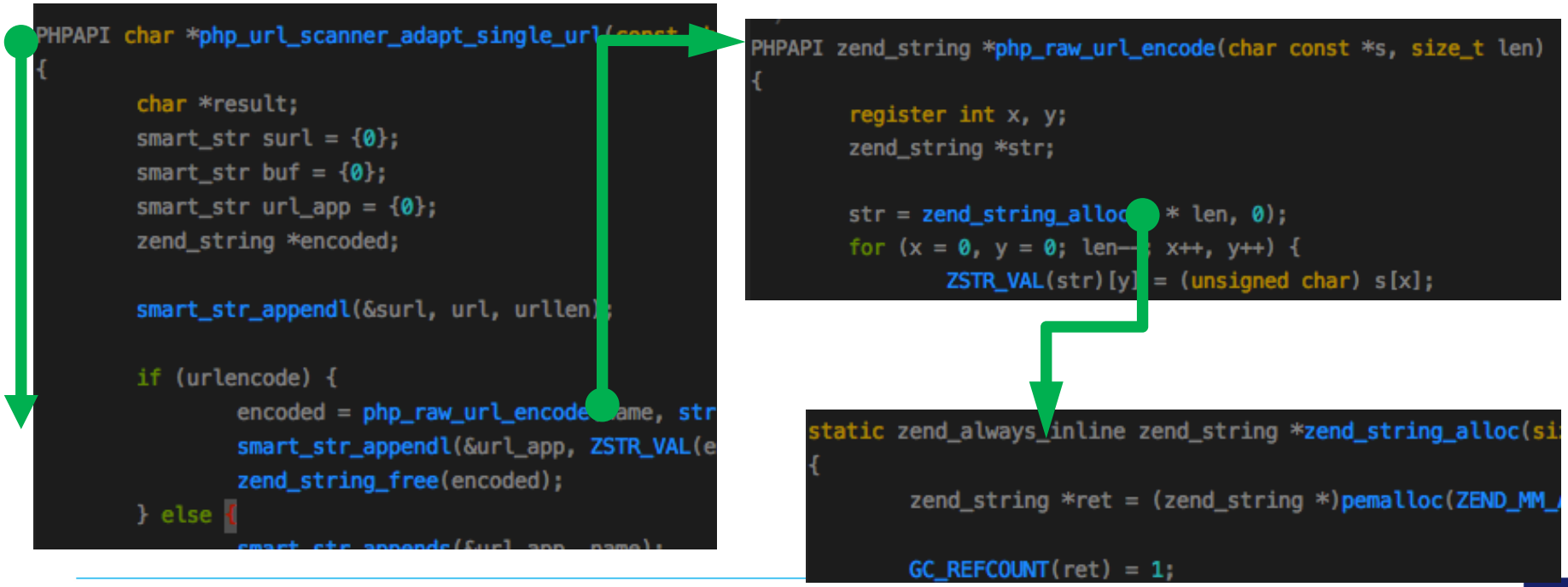
Voltron

The art of

Reverse Debugging

Root Cause Analysis: Reverse Debugging

- Debugging tends to be very linear



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