SP2023 Week 06 • 2023-03-05

Fuzzing

Richard & Juniper

Announcements

- Social this Thursday (March 9th)
- Have a relaxing spring break!

ctf.sigpwny.com

sigpwny{main(rand())}

Table of contents

What is fuzzing?

Fuzzing basics

- Automated testing by sending random inputs
- Goal is to induce crashes or otherwise invalid program behavior
- Crashes indicate a potential vulnerability
- Usually used by companies to test their own software

Terminology

- Coverage
  - Amount of code reached for a given input
- Corpus
  - Collection of "interesting" inputs (high coverage)
- CFG
  - Control flow graph: each node is a "basic block"

Techniques

Snapshots

- Program startup and shutdown can be slow
- Save the state of the program after it starts for faster reloading

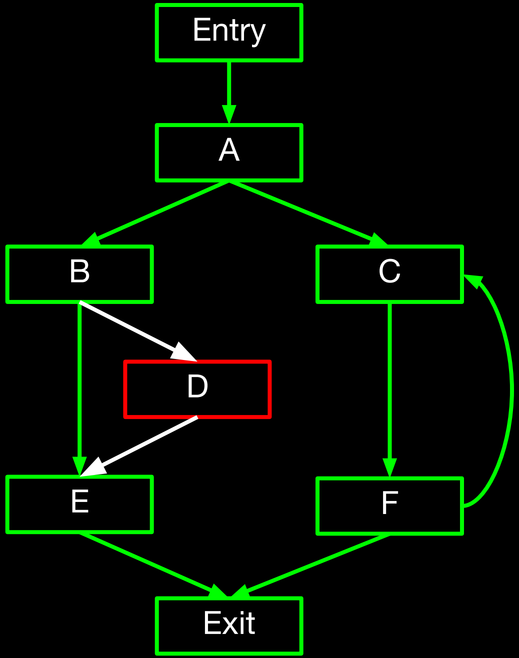
Mutation-based

- Testing variants of valid inputs
- e.g. modifying png files to produce semi-valid inputs to libpng

Techniques

Structure-aware

- Aware of code paths in a program
- More code coverage → more bugs found
- Can combine with symbolic execution (white-box) or mutation-based (gray-box) to increase code coverage



Advanced Techniques

- differential fuzzing
  - test different implementations of the same spec
- concolic fuzzing
  - use symbolic execution to find interesting paths
- coverage-guided tracing
  - on-demand instrumentation for binary targets

Active research

- HALucinator
  - rehost firmware for fuzzing on another machine
- Jetset
  - using symbolic execution to find firmware initialization constraints

Tools

`libfuzzer`

- Integrated with LLVM (Clang)
- Source based
- Compile with flag `-fsanitize=fuzzer`
    - Runs libfuzzer's own main
    - Run with `./binary_name <corpus_directory>`
- Clang adds coverage instrumentation
- Implement function

```
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  RunMyProgramWithInput(Data, Size);
  return 0;
}
```

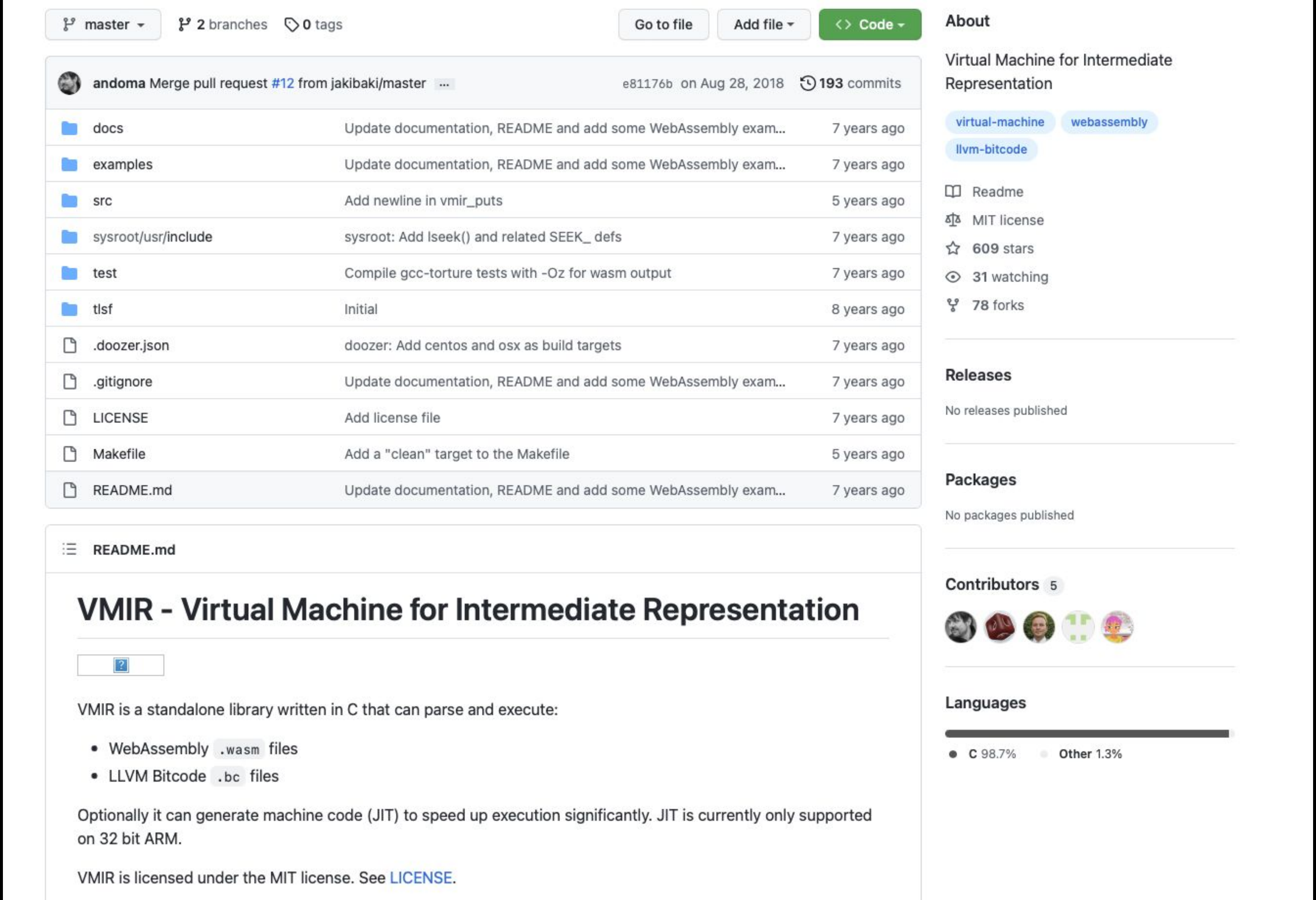
libfuzzer (cont.)

- helpful options
  - -fork=N run multiple fuzzers in parallel
  - -timeout change timeout
  - -rss_limit_mb change memory limit
  - -malloc_limit_mb change single malloc limit

AFL++

- Supports binary-only fuzzing
    - QEMU, Unicorn, WINE runners
- Binary level coverage
    - Integration with DynamoRIO and Pintool

Example


master · 2 branches · 0 tags · Go to file · Add file · Code

andoma Merge pull request #12 from jakibaki/master … e81176b on Aug 28, 2018 · 193 commits

docs	Update documentation, README and add some WebAssembly exam…	7 years ago
examples	Update documentation, README and add some WebAssembly exam…	7 years ago
src	Add newline in vmir_puts	5 years ago
sysroot/usr/include	sysroot: Add lseek() and related SEEK_ defs	7 years ago
test	Compile gcc-torture tests with -Oz for wasm output	7 years ago
tlsf	Initial	8 years ago
.doozer.json	doozer: Add centos and osx as build targets	7 years ago
.gitignore	Update documentation, README and add some WebAssembly exam…	7 years ago
LICENSE	Add license file	7 years ago
Makefile	Add a "clean" target to the Makefile	5 years ago
README.md	Update documentation, README and add some WebAssembly exam…	7 years ago

README.md

VMIR - Virtual Machine for Intermediate Representation

VMIR is a standalone library written in C that can parse and execute:

- WebAssembly `.wasm` files
- LLVM Bitcode `.bc` files

Optionally it can generate machine code (JIT) to speed up execution significantly. JIT is currently only supported on 32 bit ARM.

VMIR is licensed under the MIT license. See LICENSE.

About

Virtual Machine for Intermediate Representation

virtual-machine · webassembly · llvm-bitcode

Readme · MIT license · 609 stars · 31 watching · 78 forks

Releases

No releases published

Packages

No packages published

Contributors 5

Languages

● C 98.7% · Other 1.3%

VMIR

- WebAssembly and LLVM bitcode runtime
- **JIT** (Just in time) compilation
- **Written in C**
- Written in 2016 and not widely used
  - Should be full of bugs!
- Example will use libfuzzer from LLVM

First steps

- Clone repository
- Follow build instructions
- Check that it runs

```
$ ./vmir examples/prebuilt/sha1sum.wasm
Declared table size:0
Declared memory size:2
hello
f572d396fae9206628714fb2ce00f72e94f2258f  -
```

Harness

- Create a new file with `LLVMFuzzerTestOneInput` (fuzzer entry point)
- Copy & paste from existing code in src/main.c

```
#include <stdint.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "vmir.h"

int LLVMFuzzerTestOneInput(const uint8_t *Data,
                           size_t Size) {
  uint8_t *buf = malloc(Size);
  memcpy(buf, Data, Size);

#define MB(x) ((x) * 1024 * 1024)

  void *mem = calloc(1, MB(64));

  ir_unit_t *iu = vmir_create(mem, MB(64), MB(1),
                              MB(1), NULL);
  if(vmir_load(iu, buf, Size)) {
    free(mem);
    free(buf);
    vmir_destroy(iu);
    return -1;
  }
  free(buf);

  int rval;
  vmir_run(iu, &rval, 0, NULL);

  vmir_destroy(iu);

  free(mem);
  return 0;
}
```

beware: input must stay constant!

don't leak memory!

Building

- Use clang (for libfuzzer)
- Set -fsanitize=fuzzer flag when compiling and linking

Makefile

```
CC=clang

fuzz: ${DEPS}
	$(CC) -O2 ${CFLAGS} -fsanitize=fuzzer -g $(filter-out src/main.c,$(SRCS)) 
src/fuzz.c -lm -o $@
```

exclude src/main.c, which has its own main function

Running

- Run binary from build process (./fuzz)
- ./fuzz corpus
  - pass in directory to corpus
- ./fuzz -fork=8 corpus
  - fuzz in parallel (faster)

Example output

ignore program's own
output (pipe to /dev/null)

```
$ ./fuzz corpus > /dev/null
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 892864804
INFO: Loaded 1 modules   (7457 inline 8-bit counters): 7457 [0x559f727a35f0, 0x559f727a5311),
INFO: Loaded 1 PC tables (7457 PCs): 7457 [0x559f727a5318,0x559f727c2528),
INFO:      323 files found in corpus
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: seed corpus: files: 323 min: 1b max: 37b total: 5245b rss: 27Mb
#324    INITED cov: 343 ft: 839 corp: 262/4139b exec/s: 0 rss: 37Mb
#509    REDUCE cov: 343 ft: 839 corp: 262/4134b lim: 42 exec/s: 0 rss: 37Mb L: 18/37 MS: 5
InsertByte-ShuffleBytes-ChangeBinInt-ShuffleBytes-EraseBytes-
#566    REDUCE cov: 343 ft: 839 corp: 262/4133b lim: 42 exec/s: 0 rss: 37Mb L: 27/37 MS: 2 InsertRepeatedBytes-EraseBytes-
#657    NEW    cov: 343 ft: 840 corp: 263/4153b lim: 42 exec/s: 0 rss: 45Mb L: 20/37 MS: 1 CMP- DE: "\377'"-
#674    NEW    cov: 343 ft: 841 corp: 264/4172b lim: 42 exec/s: 0 rss: 45Mb L: 19/37 MS: 2 InsertRepeatedBytes-ShuffleBytes-
#691    NEW    cov: 344 ft: 842 corp: 265/4210b lim: 42 exec/s: 0 rss: 45Mb L: 38/38 MS: 2 PersAutoDict-CopyPart- DE: "\377'"-
```

how much
memory it's using

Crash!

type of error

where the bug is

```
UndefinedBehaviorSanitizer:DEADLYSIGNAL
==245980==ERROR: UndefinedBehaviorSanitizer: SEGV on unknown address 0x000000000000 (pc 0x559f72737685 bp 0x7fff4ee86e10 sp 0x7fff4ee86c50 T245980)
==245980==The signal is caused by a READ memory access.
==245980==Hint: address points to the zero page.
    #0 0x559f72737685 in export_function /home/richyliu/ctf/vmir/src/vmir_wasm_parser.c:400:22
    #1 0x559f72737685 in wasm_parse_section_exports /home/richyliu/ctf/vmir/src/vmir_wasm_parser.c:417:7
    #2 0x559f72737685 in wasm_parse_module /home/richyliu/ctf/vmir/src/vmir_wasm_parser.c:1430:7
    #3 0x559f72737685 in vmir_load /home/richyliu/ctf/vmir/src/vmir.c:920:5
    #4 0x559f72772d63 in LLVMFuzzerTestOneInput /home/richyliu/ctf/vmir/src/fuzz.c:21:6
    #5 0x559f726de893 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/richyliu/ctf/vmir/fuzz+0x53893)
    #6 0x559f726ddfe9 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*, bool, bool*)
(/home/richyliu/ctf/vmir/fuzz+0x52fe9)
    #7 0x559f726df7d9 in fuzzer::Fuzzer::MutateAndTestOne() (/home/richyliu/ctf/vmir/fuzz+0x547d9)
    #8 0x559f726e0355 in fuzzer::Fuzzer::Loop(std::vector<fuzzer::SizedFile, std::allocator<fuzzer::SizedFile> >&)
(/home/richyliu/ctf/vmir/fuzz+0x55355)
    #9 0x559f726ce492 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/home/richyliu/ctf/vmir/fuzz+0x43492)
    #10 0x559f726f8182 in main (/home/richyliu/ctf/vmir/fuzz+0x6d182)
    #11 0x7f17c3234d8f in __libc_start_call_main csu/../sysdeps/nptl/libc_start_call_main.h:58:16
    #12 0x7f17c3234e3f in __libc_start_main csu/../csu/libc-start.c:392:3
    #13 0x559f726c2ed4 in _start (/home/richyliu/ctf/vmir/fuzz+0x37ed4)

UndefinedBehaviorSanitizer can not provide additional info.
SUMMARY: UndefinedBehaviorSanitizer: SEGV /home/richyliu/ctf/vmir/src/vmir_wasm_parser.c:400:22 in export_function
==245980==ABORTING
MS: 1 ShuffleBytes-; base unit: 64cc66dd1bb340222ef736b19d3e15432a5c8dc0
0x0,0x61,0x73,0x6d,0xff,0x4,0x1,0xff,0x7,0x0,0xff,0x0,0x0,0x0,0x0,0x0,0xff,0xb,0x25,0x44,
\000asm\377\004\001\377\007\000\377\000\000\000\000\000\377\013%D
artifact_prefix='./'; Test unit written to ./crash-3cabc02dffa28898e72d42442be236d6b1b5858b
Base64: AGFzbf8EAf8HAP8AAAAAAP8LJUQ=
```

input saved to file

Now what?

- Try to reproduce bug
- Look at the code
- Find exploit
  - Not all bugs are exploitable
  - Patch smaller bugs and keep fuzzing
- Report responsibly and get $$$
  - Not applicable here, since this project has been dead for 5+ years

Practical tips

- Ideal fuzz target:
    - Medium size programs
    - Open source
    - No GUI
    - Low level interaction
    - Unsafe languages: C > C++ > Rust
- Optimizing fuzzing
    - Multiple threads
    - Minimize fuzzed section (reduce startup/teardown code)

Fuzzing Team

- What
  - look for vulnerabilities
  - fuzz test software
- Why
  - report them for money for the club
  - gain valuable experience
  - get clout for hacking real software
- When
  - gathering interest this semester
  - plan on starting next semester
- How
  - tell an admin if you're interested

Next Meetings

2023-03-09 - Next Thursday

- Social!
- Chill with us as spring break nears

`sigpwny{main(rand())}`

SIGPwny