Lec06: Tutorial on Fuzzing and Sanitizers

Taesoo Kim

Goals and Lessons

- Understand the benefits and limiations of fuzzers
- Hands-on experiences on two popuar fuzzers:
    - American Fuzzy Lop (AFL)
    - LibFuzzer
- Finding Heartbleed by using LibFuzzer!

Fuzzing is Amazingly Effective in Practice

- OSS-Fuzz reports that (after 5-month testing in 2017)
  - 10 trillion test inputs per day!
  - 47 open source projects (e.g., FFmpeg, FreeType2, PCRE2)
  - Over 1000 bugs!

Fuzzing is Amazingly Effective in Practice

- In 2018, Google reports that:
  - 5000+ CPU cores doing fuzz testing 24/7
  - 10x more bugs found compared to unit tests!
  - 5000+ bugs in Chromium, 1200+ bugs in ffmpeg!

Ref. Sanitize, Fuzz, and Harden Your C++ Code

What is Fuzzing? (aka Fuzz Testing)?

- aka., “random” input testing
- In fact, modern fuzzers, such as libFuzzer and AFL are quite smart!
    - Evolutionary: mutate inputs based on feedbacks
    - Coverage-based: edge coverage (e.g., A → B)

How well fuzzing can explore all paths?

```
int foo(int i1, int i2) {
  int x = i1;
  int y = i2;

  if (x > 80) {
    x = y * 2;
    y = 0;
    if (x == 256) {
*     __builtin_trap();
      return 1;
    }
  } else {
    x = 0; y = 0;
  }
  return 0;
}
```

DEMO: LibFuzzer

```
// $ clang -fsanitize=fuzzer ex.cc
// $ ./a.out
extern "C" int
LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
  if (size < 8)
    return 0;

  int i1, i2;
  i1 = *(int *)(&data[0]);
  i2 = *(int *)(&data[4]);
  foo(i1, i2);

  return 0;
}
```

Importance of High-quality Corpus

- In fact, fuzzing is not too good at exploring paths
    - e.g., if (a == 0xdeadbeef)
- So, paths should be (or mostly) given by corpus (sample inputs)
    - e.g., pdf files utilizing full features
    - but, not too many! (do not compromise your performance)
- A fuzzer will trigger the exploitable state
    - e.g., len in malloc()

AFL (American Fuzzy Lop)

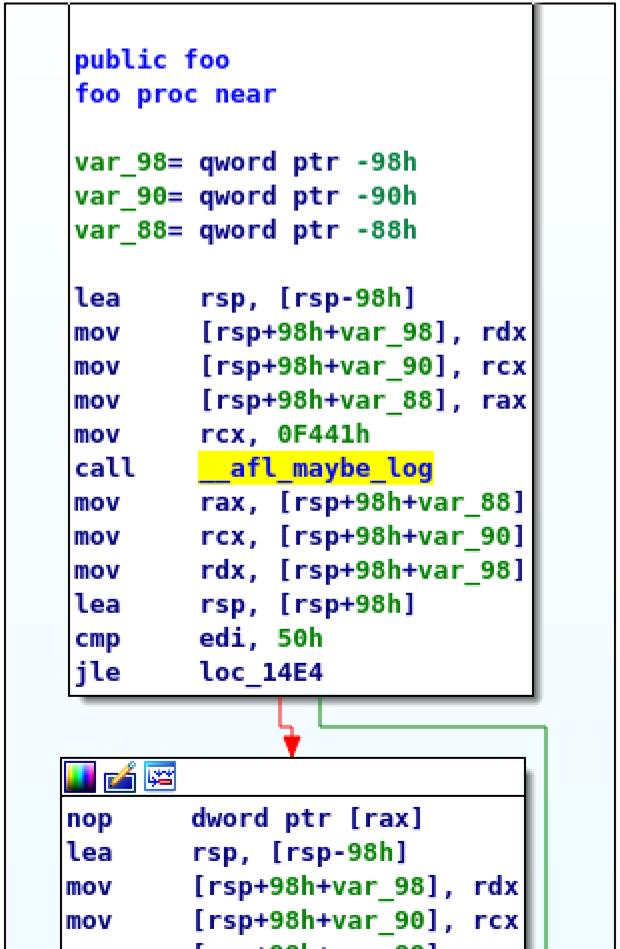
- VERY well-engineered fuzzer w/ lots of heuristics!

How to Create Mapping?

- Instrumentation
    - Source code → compiler (e.g., gcc, clang)
    - Binary → QEMU

```
if (block_address > elf_text_start
    && block_address < elf_text_end) {
  cur_location = (block_address >> 4) ^ (block_address << 8)
  shared_mem[cur_location ^ prev_location] ++;
  prev_location = cur_location >> 1;
}
```

Source Code Instrumentation



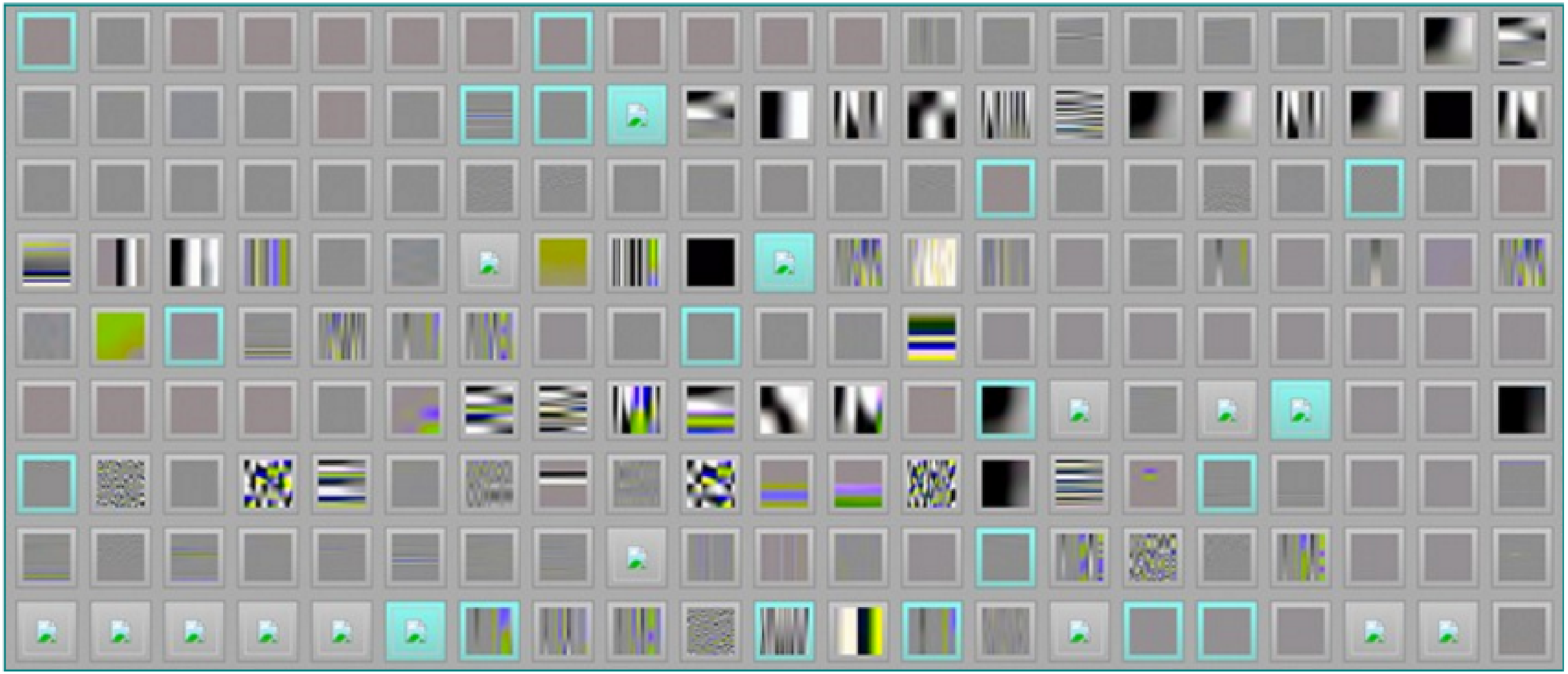
```
public foo
foo proc near

var_98= qword ptr -98h
var_90= qword ptr -90h
var_88= qword ptr -88h

lea     rsp, [rsp-98h]
mov     [rsp+98h+var_98], rdx
mov     [rsp+98h+var_90], rcx
mov     [rsp+98h+var_88], rax
mov     rcx, 0F441h
call    __afl_maybe_log
mov     rax, [rsp+98h+var_88]
mov     rcx, [rsp+98h+var_90]
mov     rdx, [rsp+98h+var_98]
lea     rsp, [rsp+98h]
cmp     edi, 50h
jle     loc_14E4
```

```
nop     dword ptr [rax]
lea     rsp, [rsp-98h]
mov     [rsp+98h+var_98], rdx
mov     [rsp+98h+var_90], rcx
```

AFL Arts



Ref. http://lcamtuf.coredump.cx/afl/

Other Types of Fuzzer

- Radamsa: syntax-aware fuzzer
- Cross-fuzz: function syntax for Javascript
- langfuzz: fuzzing program languages
- Driller/QSYM: fuzzing + symbolic execution

Today’s Tutorial

- In the tutorial:
  - Tut1: Fuzzing with AFL
  - Tut2: Fuzzing with LibFuzzer

```
# or use: https://tc.gts3.org/public/tmp/1180f-nutanix.tar.xz
$ wget https://www.dropbox.com/s/7nlsvkg68l70ez8/nutanix.tar.xz
$ unxz fuzzing.tar.xz
$ docker load -i fuzzing.tar
$ docker run --privileged -it fuzzing /bin/bash

# in docker
$ git pull
$ cat README
```

References

- libFuzzer in Chromium
- Sanitize, Fuzz, and Harden Your C++ Code