FuzzGen: Automatic Fuzzer Generation

29th USENIX Security Symposium
14th August, 2020
Motivation

- Fuzzing libraries is hard
  - Cannot run as standalone programs
  - No dependency information across API

- Goal: Invoke API in the *right order* with the *right arguments*
  - Build complex, shared state to pass between calls
  - Reduce false positives (e.g. don’t fuzz buffer lengths)

- Current approaches: AFL, libFuzzer
  - Low code coverage, manual, not scalable
Intuition Behind FuzzGen

- Library code alone is insufficient
- Leverage a whole system analysis to synthesize fuzzers
- Utilize “library consumers” to:
  - Infer library’s API
  - Expose API interactions
- Abstract API Dependence Graph
  - Translate into (lib)Fuzzer stub
Design

How it’s made
Constructing A²DG

Inferring Argument Values

Inferring API

Synthesizing fuzzer stubs
Inferring API

Synthesizing fuzzer stubs

Constructing A²DG

Inferring Argument Values
Inferring API

- $\mathcal{F}_{\text{lib}}$: All declared functions in the library
- $\mathcal{F}_{\text{incl}}$: All declared functions in all consumer header files

The final library’s API will be:

$$\mathcal{F}_{\text{API}} \leftarrow \mathcal{F}_{\text{lib}} \cap \mathcal{F}_{\text{incl}}$$
Synthesizing fuzzer stubs

Inferring API

Constructing A²DG

Inferring Argument Values

Synthesizing fuzzing stubs
Abstract API Dependence Graph (A²DG)

- Abstract layout of a single library consumer
- Exposes complicated API interactions & dependencies
- Encapsulates both **control & data** dependencies
- Directed graph of API calls, generated from CFG
  - Node: An API call
  - Edge: The control flow between 2 API calls

```
resched:
  xor  edi, edi
resched_internal_loop:
  inc  edi
  cmp  edi, NMAXTHREADS
  jno resched_dont_clear_nxtthrd:
  mov  [nxtthrd], edi
  mov  edx, [loctrl]
  mov  ax, word ptr[edx+edi*2 + THRDSINFO]
  jmp resched_internal_loop
resched_dont_clear_nxtthrd:
  mov  [nxtthrd], edi
  mov  ebx, edx, [loctrl]
  cmp  ebx, NMAXTHREADS
  jno resched_sem_lock:
  push  0xffffffff
  lea  edx, [sem]
  push call resched
  mov  edx, [loctrl]
  mov  ax, word ptr[edx+ebx*2 + THRDSINFO]
  cmp  ax, THREAD_RUNNING
  je  resched_dont_clear
  push  edi
  push edi
  lea  ebx, [sem]
  mov  ebx, [edi+ecx*2]
  call [The reference code]
  xor  edi, edi
  jmp resched_internal_loop
resched_done:
  mov  eax, SUCCESS
ret
```
A²DG Construction Example

A²DG

CFG

opus_packet_get_bandwidth

opus_packet_get_nb_channels

opus_decoder_create

opus_decoder_ctl

opus_decode

opus_decode

opus_decoder_destroy

ParseToc

opus_packet_get_bandwidth

opus_packet_get_nb_channels

LLVMFuzzerTestOneInput

opus_decoder_create

opus_decoder_ctl

opus_decode

opus_decode

opus_decoder_destroy
A²DG Coalescing

- Each consumer has its own A²DG
- Coalesce A²DGs into a single one
- At least one “common node” is required
  - Common Node: Same API call & same argument type
- Coalesce A²DGs by merging common nodes
A²DG Coalescing Example
A²DG Coalescing Example
Inferring API

Constructing $A^2$DG

Synthesizing fuzzer stubs

Inferring Argument Values
Inferring Argument Values

● Not all arguments should be fuzzed:
  ○ `void *memcpy(void *dest, const void *src, size_t n);`
  ○ `if (argc > 3) { ... }

● Decide **what** to fuzz and **how** to fuzz it
  ○ Infer **argument space** (Dataflow analysis + Backward slice)
  ○ Find dataflow dependencies across arguments

● Give **attributes** to each argument
Inferring API

Constructing A²DG

Inferring Argument Values

Synthesizing fuzzer stubs
Synthesizing Fuzzer Stubs

- **Goal:** Lift $A^2DG$ into C++ statements
- **Leverage fuzzer entropy to traverse $A^2DG$ at runtime**
  - Fuzzer explores the “good” paths
- **Fuzzers should be fast to maximize random input tests**
  - Encoding every $A^2DG$ edge reduces performance
- “Flatten” $A^2DG$
A²DG Flattening

- **Goal:** Preserve the order of every API call
- **Invoke every function exactly once**

**Flattening algorithm:**
- Drop backward edges from A²DG to make it acyclic
- Topologically sort to group nodes

**Results in a sequence of groups**
- Permute functions within group at runtime
A²DG Flattening Example

Group #1: opus_packet_get_bandwidth & opus_get_version_string
Group #2: opus_packet_get_nb_channels & opus_get_version_string
Group #3: opus_decoder_create
Group #4: opus_decoder_ctl & opus_decoder_decode
Group #5: opus_decoder_decode
Group #6: opus_decoder_decode
Group #7: opus_decoder_destroy
Group #8: opus_get_version_string
Evaluation

Proof of Work
Evaluation

- Evaluate on Debian & Android
  - 7 codec libraries
  - libfuzzer + ASAN
  - 24 hr experiments * 5 times each

- 17 Bugs Found, 6 got a CVE:
  - CVE-2019-2176
  - CVE-2019-2108
  - CVE-2019-2107
  - CVE-2019-2106
  - CVE-2017-13187
  - CVE-2017-0858 (duplicate)
Evaluation - Metrics

- Comparing against manually written fuzzers
  - If no fuzzer found online, we created one

- Average Edge Coverage
  - FuzzGen fuzzers: 54.94% vs 48.00% of manual fuzzers
  - FuzzGen explores more aspects of the library

- Measuring bugs found
  - FuzzGen fuzzers: 17 vs 29 of manual fuzzers
  - Manual fuzzers test more thoroughly “buggy” parts
Evaluation - Edge Coverage for libavc
Conclusion

- **Whole** system analysis infers API interactions.

- Automatically synthesize high entropy (lib)Fuzzer stubs
  - Construct complex program state
  - Achieve high code coverage

- Evaluation found 6 CVEs and 17 previously unknown bugs

- Source code: [https://github.com/HexHive/FuzzGen](https://github.com/HexHive/FuzzGen)
  - (~20,000 LoC in C++ using LLVM)