



FuzzGen: Automatic Fuzzer Generation

*29th USENIX Security Symposium
14th August, 2020*

Kyriakos Ispoglou
Google Inc.

Daniel Austin
Atlassian

Vishwath Mohan
Google Inc.

Mathias Payer
EPFL

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**

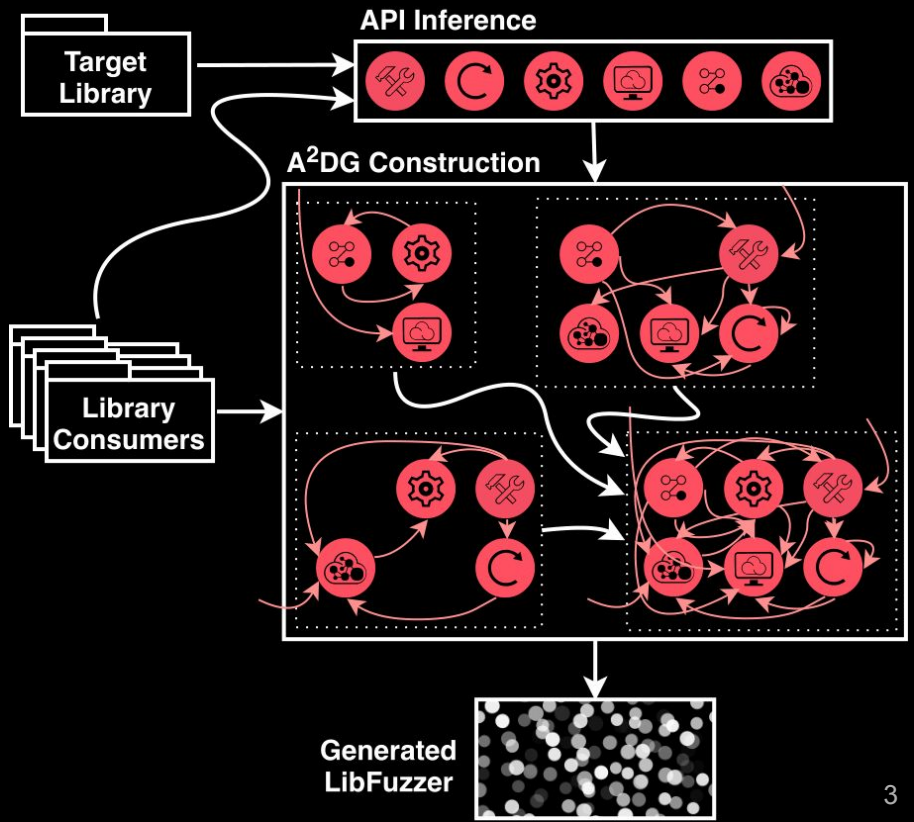
- Low code coverage, manual, not scalable

```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
```

```
resched:  
xor edi, edi  
resched_internal_loop:  
inc edi  
mov ebx, [nxtthrd]  
inc ebx  
cmp ebx, NMAXTHREADS  
jle resched_internal_loop  
xor ebx, ebx  
resched_dont_clear_nxtthrd:  
mov [nxtthrd], ebx  
mov ax, word ptr [eax+ebx*2+1KB*310FF]  
cmp [eax+ebx*2+1KB*310FF], ax  
jne resched_internal_loop  
resched_sem_lock:  
push 0xffffffff  
lea [eax], [eax+ebx*2+1KB*310FF]  
push [eax]  
call [ReleaseSemaphore]  
mov edi, edi  
mov ax, word ptr [eax+ebx*2+1KB*310FF]  
cmp [eax+ebx*2+1KB*310FF], ax  
je resched_done  
push 0  
push 1  
lea [eax], [eax+ebx*2+1KB*310FF]  
mov [eax], [eax+ebx*2+1KB*310FF]  
push [eax]  
call [ReleaseSemaphore]  
xor edi, edi  
jmp resched_internal_loop  
resched_done:  
mov eax, SUCCESS  
retn
```

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



```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
```

```
resched:  
    xor     edi, edi  
resched_internal_loop:  
    inc     edi  
    cmp     edi, NMAXTHREADS  
    jg      resched_error  
    mov     ebx, [nxtthrd]  
    inc     ebx  
    cmp     ebx, NMAXTHREADS  
    jl      resched_dont_clear_nxtthrd  
    xor     ebx, ebx  
resched_dont_clear_nxtthrd:  
    mov     [nxtthrd], ebx  
    mov     edx, [loctrl]  
    mov     ax, word ptr[edx+ebx*4 + THRDSTOFF]  
    cmp     ax, THREAD_RUNNING  
    jne     resched_internal_loop  
resched_sem_lock:  
    push   0xffffffff  
    lea    edx, [sem]  
    push   [edx + ebx*4]  
    call   [WaitForSingleObject]  
    mov    edx, [loctrl]  
    mov    ax, word ptr[edx+ebx*2 + THRDSTOFF]  
    cmp    ax, THREAD_RUNNING  
    je     resched_done  
    push  0  
    push  1  
    lea   ebx, [sem]  
    mov   ecx, [nxtthrd]  
    push [ebx + ecx*4]  
    call [ReleaseSemaphore]  
    xor   edi, edi  
    jmp  resched_internal_loop  
resched_done:  
    mov   eax, SUCCESS  
    retn
```

Design

How it's made

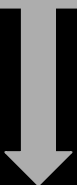
```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor    edi, edi
resched_internal_loop:
  inc    edi
  mov    esi, eax
  mov    resched_error
  mov    ebx, esi
  inc    ebx
  cmp    ebx, NMAXTHREADS
  jle    resched_dont_clear_nxtthrd
  mov    ebx, ebx
resched_dont_clear_nxtthrd:
  mov    [nxtthrd], ebx
  mov    edx, [loctrl]
  mov    ax, word ptr [edx+ebx*2 + THRDSTOFF]
  cmp    ax, THREAD_RUNNING
  jne    resched_internal_loop
resched_sem_lock:
  push  0xffffffff
  lea   edx, [sem]
  push  [edx + ebx*4]
  mov   [waitforsingleobj], edx, [loctrl]
  mov   ebx, [sem]
  cmp   eax, ebx
  je    resched_done
  push  0
  push  ebx
  lea   ecx, [nxtthrd]
  mov   [ebx + ecx*4]
  push [ebx + ecx*4]
  call [ReleaseSemaphore]
  xor   edi, edi
  jmp  resched_internal_loop
resched_done:
  mov  eax, SUCCESS
  ret
```

Inferring API

Constructing A²DG

Synthesizing fuzzer stubs

Inferring Argument Values



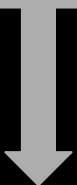
```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor    edi, edi
resched_internal_loop:
  inc    edi
  mov    eax, [resched_error]
  mov    ebx, [resched_error]
  nc     ebx, ebx
  mp    ebx, NMAXTHREADS
  l     resched_dont_clear_nxtthrd
  or     ebx, ebx
resched_dont_clear_nxtthrd:
  mov    [nxtthrd], ebx
  mov    edx, [loctrl]
  mov    ax, word ptr [edx+ebx*2 + THRDSTOFF]
  cmp    ax, THREAD_RUNNING
  jne    resched_internal_loop
resched_sem_lock:
  push  0xffffffff
  lea   edx, [sem]
  push  [edx + ebx*4]
  mov   [waitforsingleobj], edx, [loctrl]
  mov   ebx, [sem]
  mp   ebx, ebx
  e   resched_done
  push  0
  push  [edx + ebx*4]
  lea  ebx, [sem]
  mov  ecx, [nxtthrd]
  push [ebx + ecx*4]
  call [ReleaseSemaphore]
  xor  edi, edi
  jmp  resched_internal_loop
resched_done:
  mov  eax, SUCCESS
  ret
```

Inferring API

Constructing A²DG

Synthesizing fuzzer stubs

Inferring Argument Values



```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
```

```
resched:
  xor edi, edi
resched_internal_loop:
```

Inferring API

- \mathcal{F}_{lib} : All declared functions in the library
- \mathcal{F}_{incl} : All declared functions in all consumer header files
- The final library's API will be:

$$\mathcal{F}_{API} \leftarrow \mathcal{F}_{lib} \cap \mathcal{F}_{incl}$$

```

  inc edi
  mov ebx, [nxtthrd]
  inc ebx
  cmp ebx, NMYTHREADS
  xor ebx, ebx
resched_c:
  mov ecx, [eax - 4], ebx
  mov ecx, [loctrl]
  mov ecx, [rdx + ebx*2 + THRDSTOFF]
  cmp ecx, 0
  jne resched_internal_loop
resched_sem_lock:
  push 0xffffffff
  lea edx, [sem]
  push 0
  call [eax + ecx*4], ecx
  mov ecx, [loctrl]
  mov ax, word ptr [edx + ebx*2 + THRDSTOFF]
  cmp ax, THREAD_RUNNING
  je resched_done
  push 0
  push 1
  lea ebx, [sem]
  mov ecx, [nxtthrd]
  push [ebx + ecx*4]
  call [ReleaseSemaphore]
  xor edi, edi
  jmp resched_internal_loop
resched_done:
  mov eax, SUCCESS
  retn

```

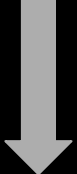
```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor     edi, edi
resched_internal_loop:
  inc     edi
  mov     esi, eax
  mov     resched_error
  mov     ebx, ebx
  inc     ebx
  cmp     ebx, NMAXTHREADS
  jle     resched_dont_clear_nxtthrd
  mov     ebx, ebx
resched_dont_clear_nxtthrd:
  mov     [nxtthrd], ebx
  mov     edx, [loctrl]
  mov     ax, word ptr [edx+ebx*2 + THRDSTOFF]
  cmp     ax, THREAD_RUNNING
  jne     resched_internal_loop
resched_sem_lock:
  push   0xffffffff
  lea   edx, [sem]
  push   [edx + ebx*4]
  mov   [waitforsingleobj], ecx
  mov   edx, [loctrl]
  mov   eax, [edx+ebx*2 + THRDSTOFF]
  cmp   eax, 0
  je    resched_done
  push  0
  push  0
  lea  ebx, [sem]
  mov  ecx, [nxtthrd]
  push [ebx + ecx*4]
  call [ReleaseSemaphore]
  xor  edi, edi
  jmp  resched_internal_loop
resched_done:
  mov  eax, SUCCESS
  ret
```

Inferring API

Constructing A²DG

Synthesizing fuzzer stubs

Inferring Argument Values



Abstract API Dependence Graph (A²DG)

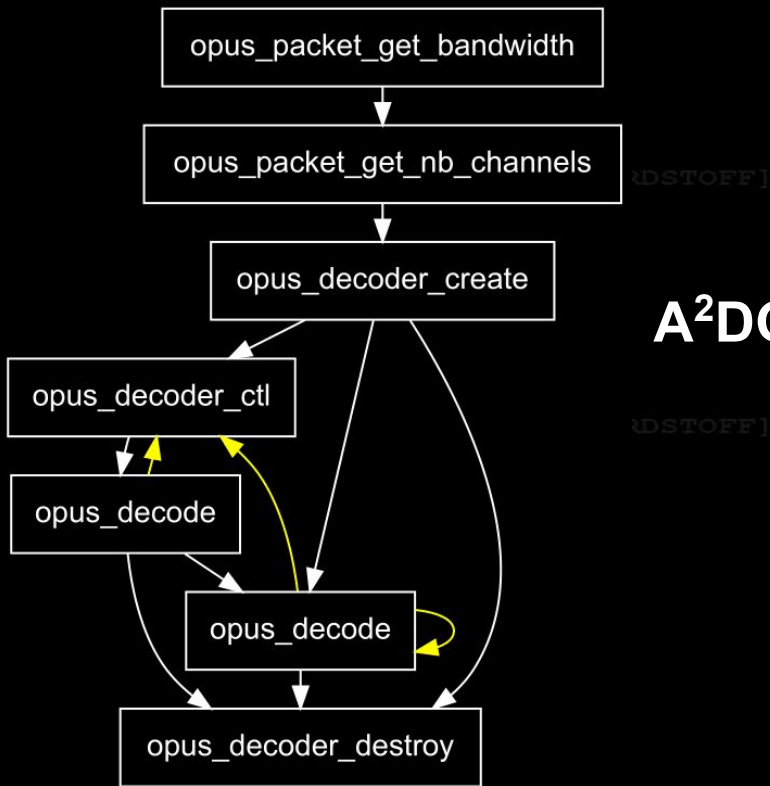
- Abstract layout of a single library consumer
- Exposes complicated API interactions & dependencies
- Encapsulates both control & data dependencies
- Directed graph of API calls, generated from CFG
 - Node: An API call
 - Edge: The control flow between 2 API calls

```

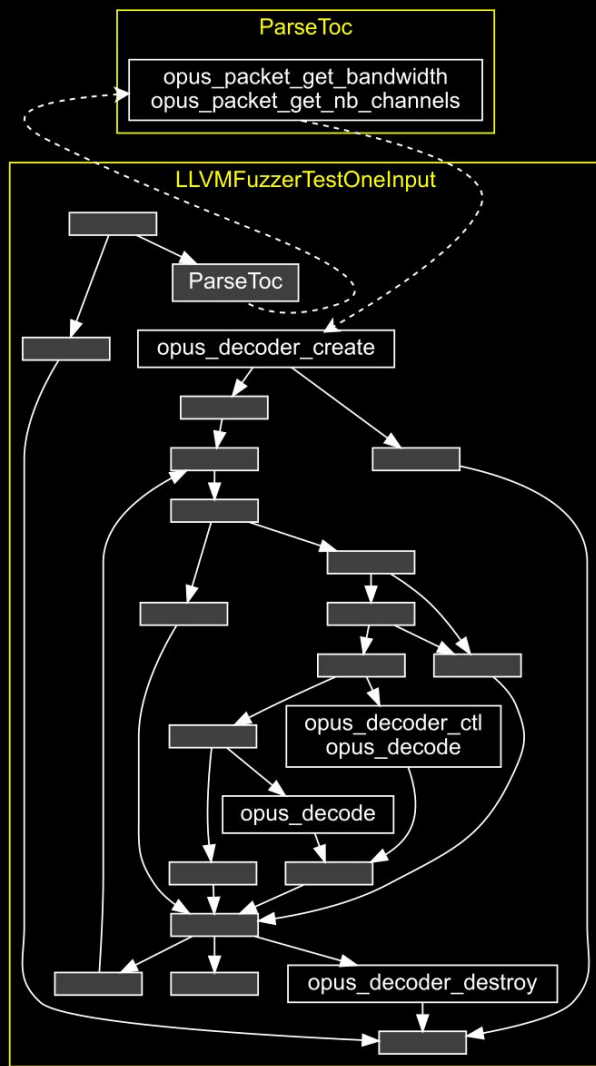
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor     edi, edi
resched_internal_loop:
  inc     edi
  mov     ebx, [nxtthrd]
  inc     ebx
  cmj     j1
  xor
resched:
  mov
  mov
  mov
  cmj
  jne
resched:
  pus
  les
  pus
  cal
  mov
  mov
  cmj
  je
  pus
  pus
  les
  mov
  pus
  cal
  xor
  jmp
resched:
  mov
  ret

```

A²DG Construction Example



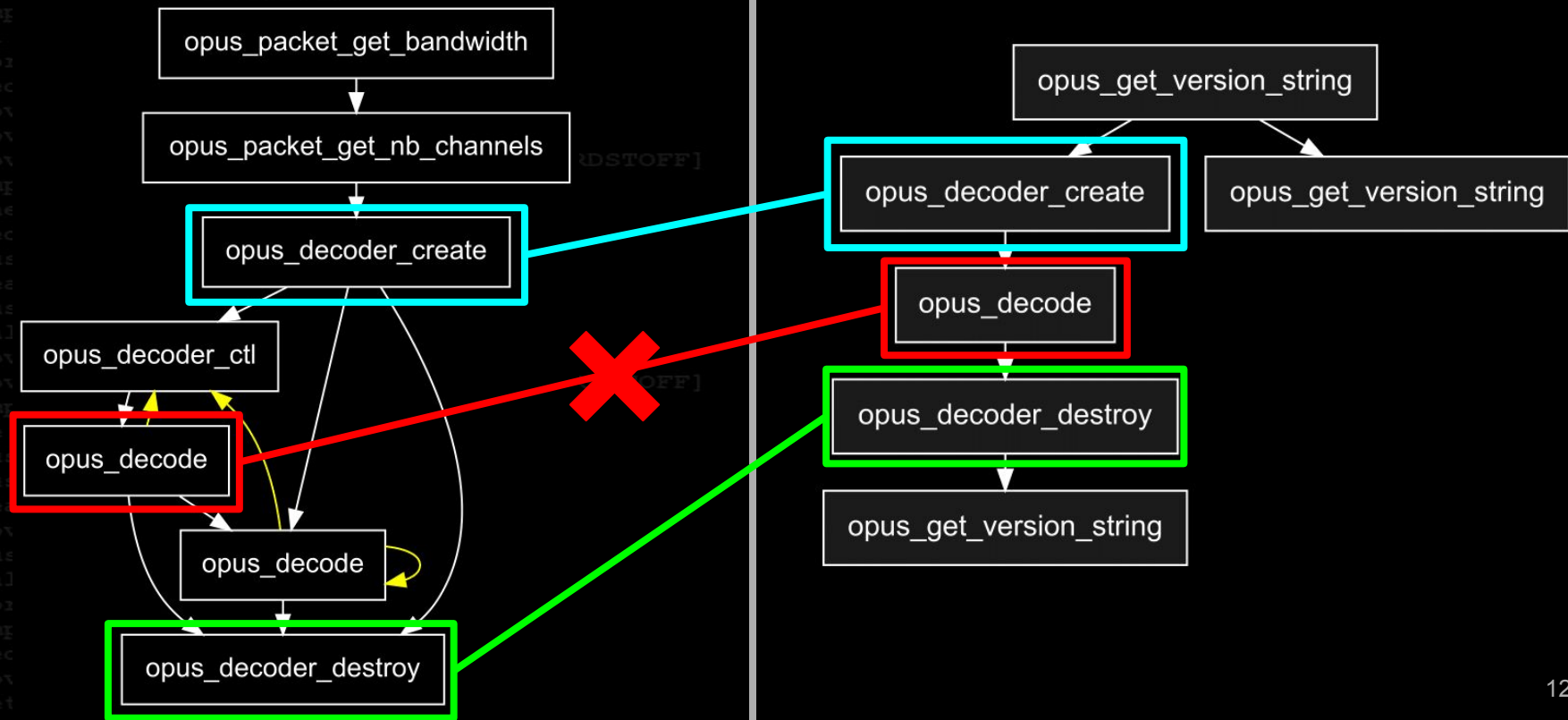
A²DG **CFG**



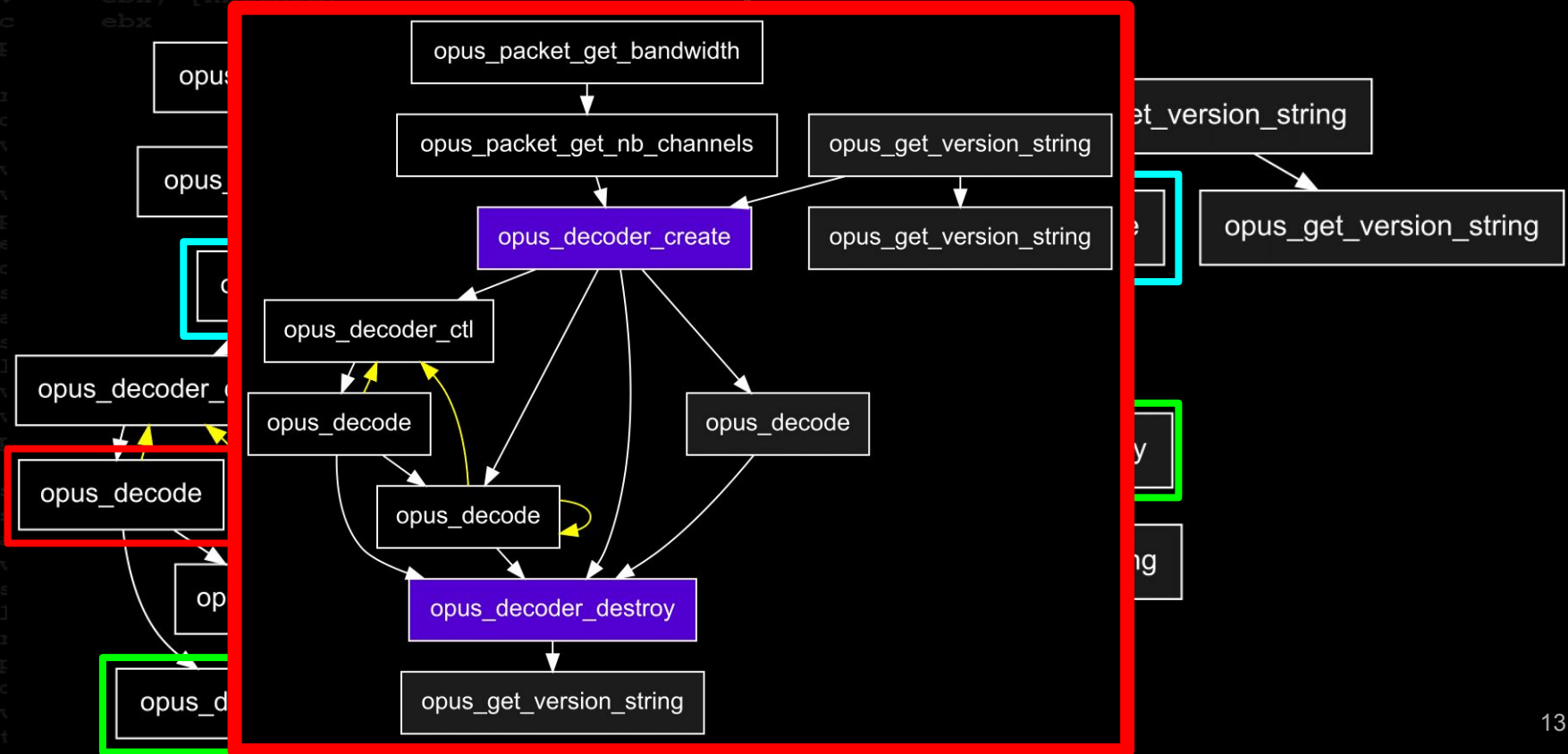
A²DG Coalescing

- 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



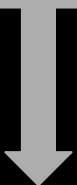
```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor     edi, edi
resched_internal_loop:
  inc     edi
  mov     esi, NMAXTHREADS
  mov     resched_error
  mov     ebx, resched_error
  inc     ebx
  cmp     ebx, NMAXTHREADS
  jle     resched_dont_clear_nxtthrd
  mov     ebx, ebx
resched_dont_clear_nxtthrd:
  mov     [nxtthrd], ebx
  mov     edx, [loctrl]
  mov     ax, word ptr [edx+ebx*2 + THRDSTOFF]
  cmp     ax, THREAD_RUNNING
  jne     resched_internal_loop
resched_sem_lock:
  push   0xffffffff
  lea    edx, [sem]
  push   [edx + ebx*4]
  mov     [wait_for_single_obj],
  mov     edx, [loctrl]
  mov     eax, [edx+ebx*2 + THRDSTOFF]
  cmp     eax, 0
  je     resched_done
  push   0
  push   0
  lea    ebx, [sem]
  mov     ecx, [nxtthrd]
  push   [ebx + ecx*4]
  call   [ReleaseSemaphore]
  xor     edi, edi
  jmp    resched_internal_loop
resched_done:
  mov     eax, SUCCESS
  retn
```

Inferring API

Constructing A²DG

Synthesizing fuzzer stubs

Inferring Argument Values



```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
```

```
resched:  
xor edi, edi  
resched_internal_loop:
```

Inferring Argument Values

- **Not all arguments should be fuzzed:**

- void *memcpy(void *dest, const void *src, size_t n);
- if (argc > 3) { ... }

```
resched_dont_clear_nxtthrd:
```

```
mov [nxtthrd], eax  
mov edx, [loctrl]  
mov ax, [eax * 2 + THRDSTOFF]  
cmp ax, THRDIDLE  
jne resched_internal_loop
```

```
resched_sem_lock:
```

```
push 0xffffffff  
lea edi, [sem]  
push edi  
call [WaitForSingleObject]
```

- **Decide what to fuzz and how to fuzz it**

- **Infer argument space (Dataflow analysis + Backward slice)**
- **Find dataflow dependencies across arguments**

```
mov edi, [eax * 4]  
cmp ax, THREAD_RUNNING  
je resched_internal_loop  
push 0  
push 1  
lea ebx, [sem]  
mov ecx, [nxtthrd]  
push [ebx + ecx * 4]
```

- **Give attributes to each argument**

```
call [WaitForSingleObject]  
xor edi, edi  
jmp resched_internal_loop
```

```
resched_done:  
mov eax, SUCCESS  
ret
```

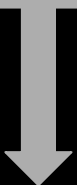
```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor     edi, edi
resched_internal_loop:
  inc     edi
  mov     esi, eax
  mov     resched_error
  mov     ebx, esi
  inc     ebx
  cmp     ebx, NMAXTHREADS
  jle     resched_dont_clear_nxtthrd
  mov     ebx, ebx
resched_dont_clear_nxtthrd:
  mov     [nxtthrd], ebx
  mov     edx, [loctrl]
  mov     ax, word ptr [edx+ebx*2 + THRDSTOFF]
  cmp     ax, THREAD_RUNNING
  jne     resched_internal_loop
resched_sem_lock:
  push   0xffffffff
  lea    edx, [sem]
  push   [edx + ebx*4]
  call   [waitforingleb]
  mov     edx, [loctrl]
  mov     eax, word ptr [edx+ebx*2 + THRDSTOFF]
  cmp     eax, 0
  je     resched_done
  push   0
  push   [sem]
  lea    ebx, [edx + ebx*4]
  mov     ecx, [nxtthrd]
  push   [ebx + ecx*4]
  call   [ReleaseSemaphore]
  xor     edi, edi
  jmp    resched_internal_loop
resched_done:
  mov     eax, SUCCESS
  retn
```

Inferring API

Constructing A²DG

Synthesizing fuzzer stubs

Inferring Argument Values



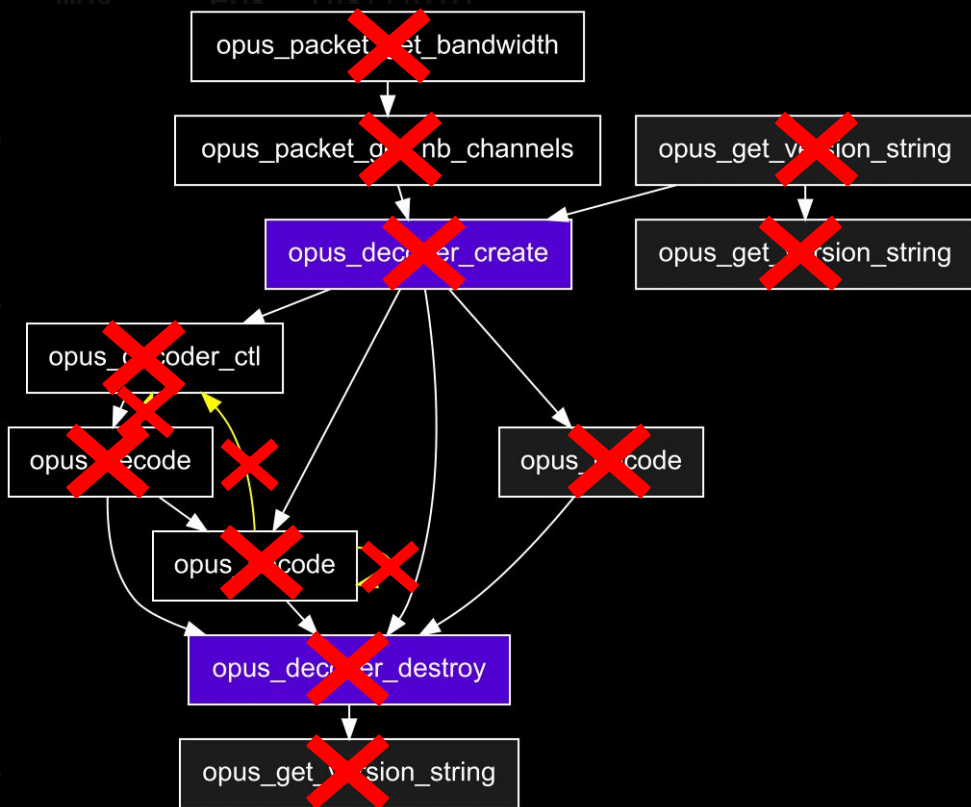
Synthesizing Fuzzer Stubs

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

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

```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
    xor     edi, edi
resched_internal_loop:
    inc     edi
    cmp     edi, NMAXTHREADS
    jg      resched_error
    mov     ebx, [nxtthrd]
    inc     ebx
    cmp     ebx, NMAXTHREADS
    jl      resched_dont_clear_nxtthrd
    xor     ebx, ebx
resched_dont_clear_nxtthrd:
    mov     [nxtthrd], ebx
    mov     ebx, [loctrl]
    mov     ptr [edx+ebx*2 + THRDSTOFF], ebx
    cmp     [edx+ebx*2 + THRDSTOFF], THREAD_RUNNING
    jne     resched_internal_loop
resched_semaphore:
    push   eax
    push   edx, [sem]
    push   [edx + ebx*4]
    call   [WaitForSingleObject]
    mov    edx, [loctrl]
    mov    ax, word ptr [edx+ebx*2 + THRDSTOFF]
    cmp    ax, THREAD_RUNNING
    je     resched_done
    push   0
    push   1
    lea   ebx, [sem]
    mov   ecx, [nxtthrd]
    push  [ebx + ecx*4]
    call [ReleaseSemaphore]
    xor   edi, edi
    jmp  resched_internal_loop
resched_done:
    mov   eax, SUCCESS
    retn
```

```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
resched:
  xor     edi, edi
resched_internal_loop:
  inc     edi
  mov     ebx, [nxtthrd]
  inc     ebx
  cmp     ebx, NMAXTHREADS
  xor     ebx, ebx
resched_dont_clear_nxtthrd:
  mov     edx, [loctrl]
  mov     ax, [loctrl + THRDSTOFF]
  cmp     ax, [loctrl + RUNNING]
  jne     resched_dont_clear_nxtthrd
resched_sel_lo:
  push   0xffffffff
  lea    edx, [sem]
  push   [edx + ebx*4]
  call  [WaitForSingleObject]
  mov     ax, [ptr[edx+ebx*2 + THRDSTOFF]]
  cmp     ax, [loctrl + RUNNING]
  je     resched_done
  push   0
  push   1
  lea    ebx, [nxtthrd]
  mov     ecx, [nxtthrd]
  push   [ReleaseSemaphore]
  call  [ReleaseSemaphore]
  xor     edi, edi
  jmp    resched_internal_loop
resched_done:
  mov     eax, SUCCESS
  retn
```

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)



```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
```

```
resched:  
xor edi, edi  
resched_internal_loop:  
inc edi
```

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**

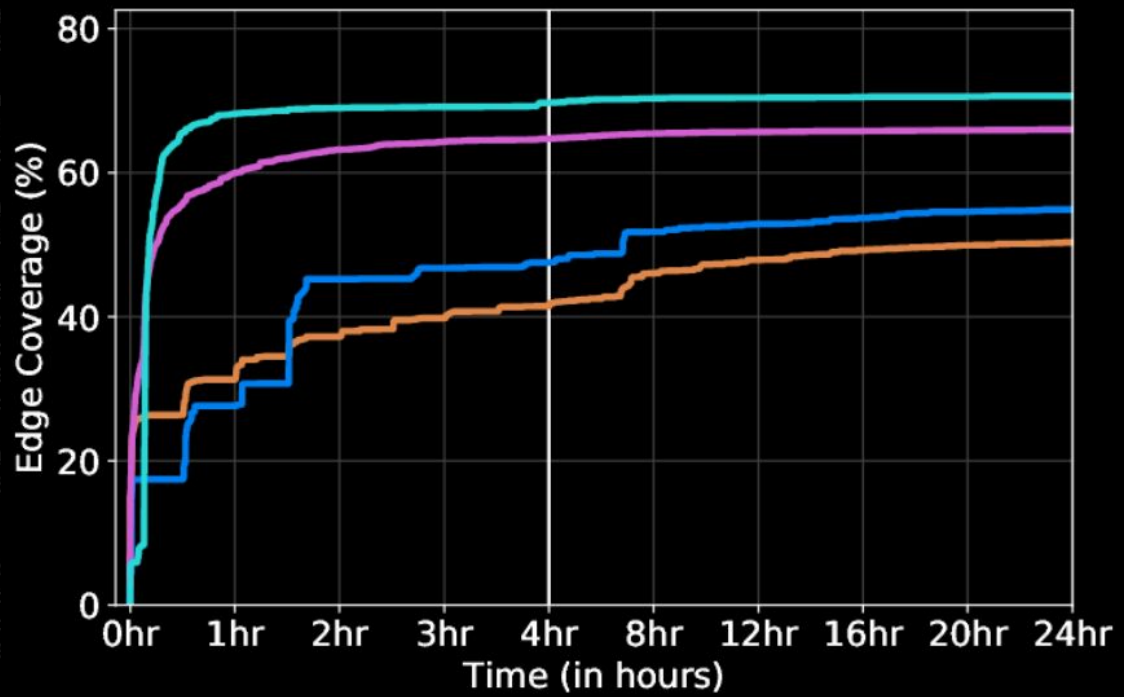
```
mov ebx, [nxtthrd]  
inc ebx  
cmp ebx, NMAXTHREADS  
xor ebx, ebx  
resched_done_clear_nxtthrd:  
mov [nxtthrd], 0  
mov edx, [loctrl]  
mov ax, word ptr [edx+ebx*2 + THRDSTOFF]  
cmp ax, THREAD_RUNNING  
jne resched_internal_loop  
resched_start_lock:  
push eax  
lea edx, [sem]  
push [edx]  
call [RtlFastExAcquireRelease]  
mov edx, [loctrl]  
mov ax, [edx+ebx*2 + THRDSTOFF]  
cmp ax, 0  
je resched_done  
push 0  
push 1  
lea ebx, [sem]  
mov [edx], ebx  
push [ebx+ecx*4]  
call [RtlFastExRelease]  
xor edi, edi  
jmp resched_internal_loop  
resched_done:  
mov eax, 0  
ret
```

```
B(0xe8) D(0x00, 0x00, 0x00, 0x00)
```

```
resched:  
xor edi, edi  
resched_internal_loop:  
inc edi
```

Evaluation - Edge Coverage for libavc

```
mov ebx, [nxtthrd]  
inc ebx  
cmp ebx, NMAX  
jl resched_c  
xor ebx, ebx  
resched_dont_clear_nx  
mov [nxtthrd]  
mov edx, [loc  
mov ax, word  
cmp ax, THREE  
jne resched_i  
resched_sem_lock:  
push 0xffffffff  
lea edx, [sem  
push [edx + e  
call [WaitForS  
mov edx, [loc  
mov ax, word  
cmp ax, THREE  
je resched_c  
push 0  
push 1  
lea ebx, [sem  
mov ecx, [nxt  
push [ebx + ec  
call [ReleaseS  
xor edi, edi  
jmp resched_i
```



```
resched done:  
n
```

Manual Fuzzer Average Manual Fuzzer Best Single Run FuzzGen Fuzzer Average FuzzGen Fuzzer Best Single Run

```
retn
```

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>
 - (~20.000 LoC in C++ using LLVM)