black hat USA 2018

AUGUST 4-9, 2018

MANDALAY BAY / LAS VEGAS

AFL's Blindspot and How to Resist AFL Fuzzing for Arbitrary ELF Binaries

Kang Li

360 Cyber Immunity Lab /
Team Disekt /
University of Georgia

Collaborators: Yue Yin, Guodong Zhu

#BHUSA / @BLACK HAT EVENTS

About Me

Professor of Computer Science at UGA

Founding Mentor of xCTF and Blue-Lotus

Founder of the Disekt, SecDawgs CTF Teams

2016 DARPA Cyber Grand Challenge Finalist

BULLDOG SECURITY SERVICES

disekt CTF Team

CGC CYBER GRAND_CHALLENGE

Cyber Immunity

Life as a Security Educator

- Write a simple buggy program
- Assign the binary (without symbols) and expect students to find bugs
- “Rest” until students finish (usually takes hours …)

THE TEACHER

IMG Src: https://cheezburger.com/7950357760/one-great-teacher

Peaceful Class Time


```
int cb(uchar *out) {
    int ret = 0;

    if (out[0] == 'M') {
        if (out[1] == 'A') {
            if (out[2] == 'G') {
                if (out[3] = 'I') {
                    if (out[4] == 'C') {
                        if (out[5] == '!') {
                            ret = 1;
                            /* printf("You Won!\n"); */
                            crash();
                        }
                    }
                }
            }
        }
    }
    /* printf("Please Try Again\n"); */
    return 0;
}
```

Peace Disrupted


```
                    american fuzzy lop 2.52b (cb)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 3 min, 51 sec      │  cycles done : 65     │
│   last new path : 0 days, 0 hrs, 1 min, 7 sec       │  total paths : 7      │
│ last uniq crash : 0 days, 0 hrs, 0 min, 6 sec       │ uniq crashes : 1      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 5 (71.43%)        │    map density : 0.05% / 0.08%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.00 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored paths : 7 (100.00%)            │
│ stage execs : 2028/2048 (99.02%)    │  new edges on : 7 (100.00%)            │
│ total execs : 336k                  │ total crashes : 1 (1 unique)           │
│  exec speed : 1322/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 0/320, 0/313, 0/299                   │    levels : 6          │
│  byte flips : 0/40, 0/33, 0/19                      │   pending : 0          │
│ arithmetics : 2/2237, 0/197, 0/70                   │  pend fav : 0          │
│  known ints : 0/244, 0/880, 0/824                   │ own finds : 6          │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 3/141k, 2/187k                        │ stability : 100.00%    │
│        trim : 0.00%/6, 0.00%                        ├────────────────────────┘
^C────────────────────────────────────────────────────┘          [cpu000: 53%]
```

total paths: 7
uniq crashes: 1

Success of AFL

- **Fast and Reliable Fuzzing**

  edge coverage stored in a compact bitmap (default 64KB)

  low test overhead, simple to use

- **Bugs Found in**

  Bind, PuTTY, tcpdump, ffmpeg, GnuTLS, libtiff, libpng, …

  more on the AFL sites (http://lcamtuf.coredump.cx/afl/)

- **Widely Used**

  by most of the 2016 CGC Finalist Teams

Why to Resist AFL fuzzing

- The *deafL* tool (this talk)
    - to force students to study binaries (instead of just running AFL)
- Other reasons:
    - to learn AFL's limitations and to develop better fuzzers

… …

The Fuzzing Process of AFL

1. Start with sample seed inputs
2. Mutate seed inputs to generate mutants
3. Collect code coverage (CFG edges) Information
4. Save as new seeds if coverage increases
5. Repeat from step 2

AFL Instrumentation

- **if with Source Code** (Compiler-aid Instrumentation, AFL-GCC)

1. *cur_location = <RANDOM#>;*
2. *shared_mem[cur_location ^ prev_location]++;*
3. *prev_location = cur_location >> 1;*

- **if with Binary Only** (AFL-QEMU)

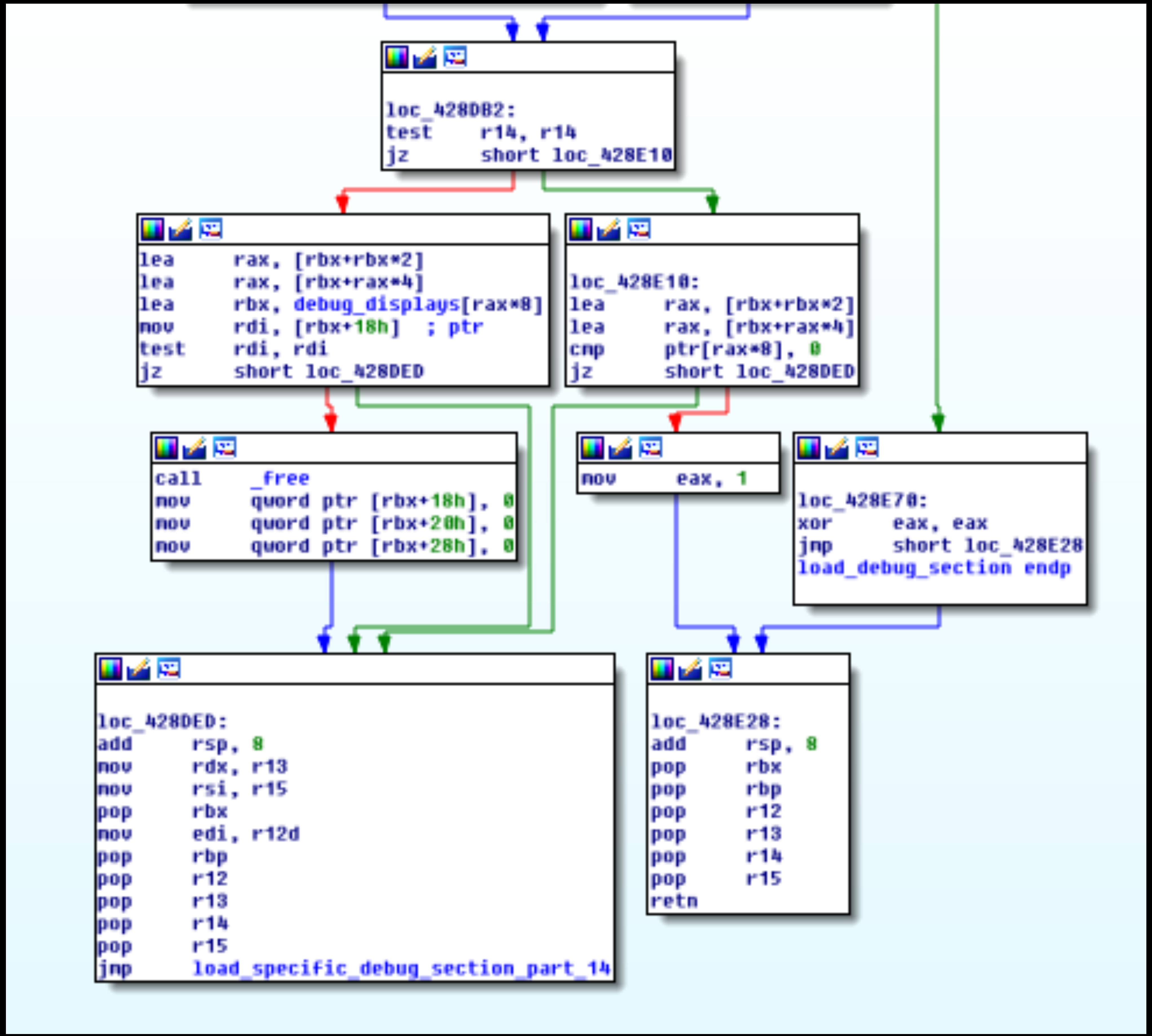
1. *cur_location = (block_address >> 4) ^ (block_address << 8);*
2. *shared_mem[cur_location ^ prev_location]++;*
3. *prev_location = cur_location >> 1;*

How Coverage Info is Collected in AFL

$ *readelf* *testcase_1*

Assuming the basic blocks being covered are:

…
0x428DB2
0x428E10
0x428DED
…



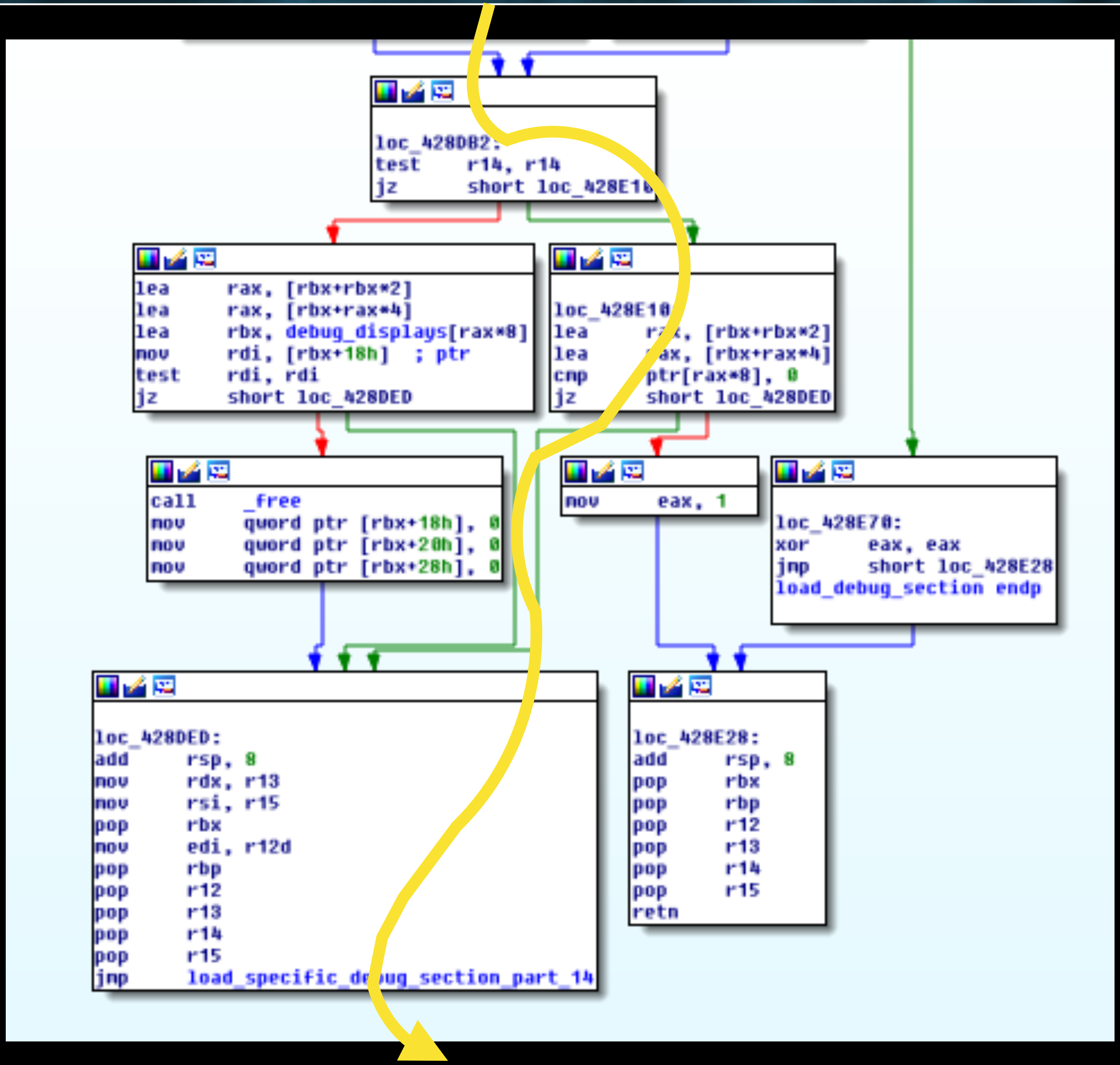
Program *readelf*'s Control Flow Graph (partial)

How Coverage Info is Collected in AFL

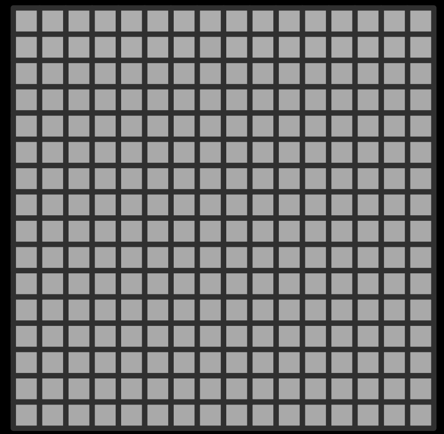
$ *readelf* *testcase_1*

Assuming the basic blocks being covered are:

…
0x428DB2
0x428E10
0x428DED
…



Program *readelf*'s Control Flow Graph (partial)



AFL's *shared_mem[ ]*

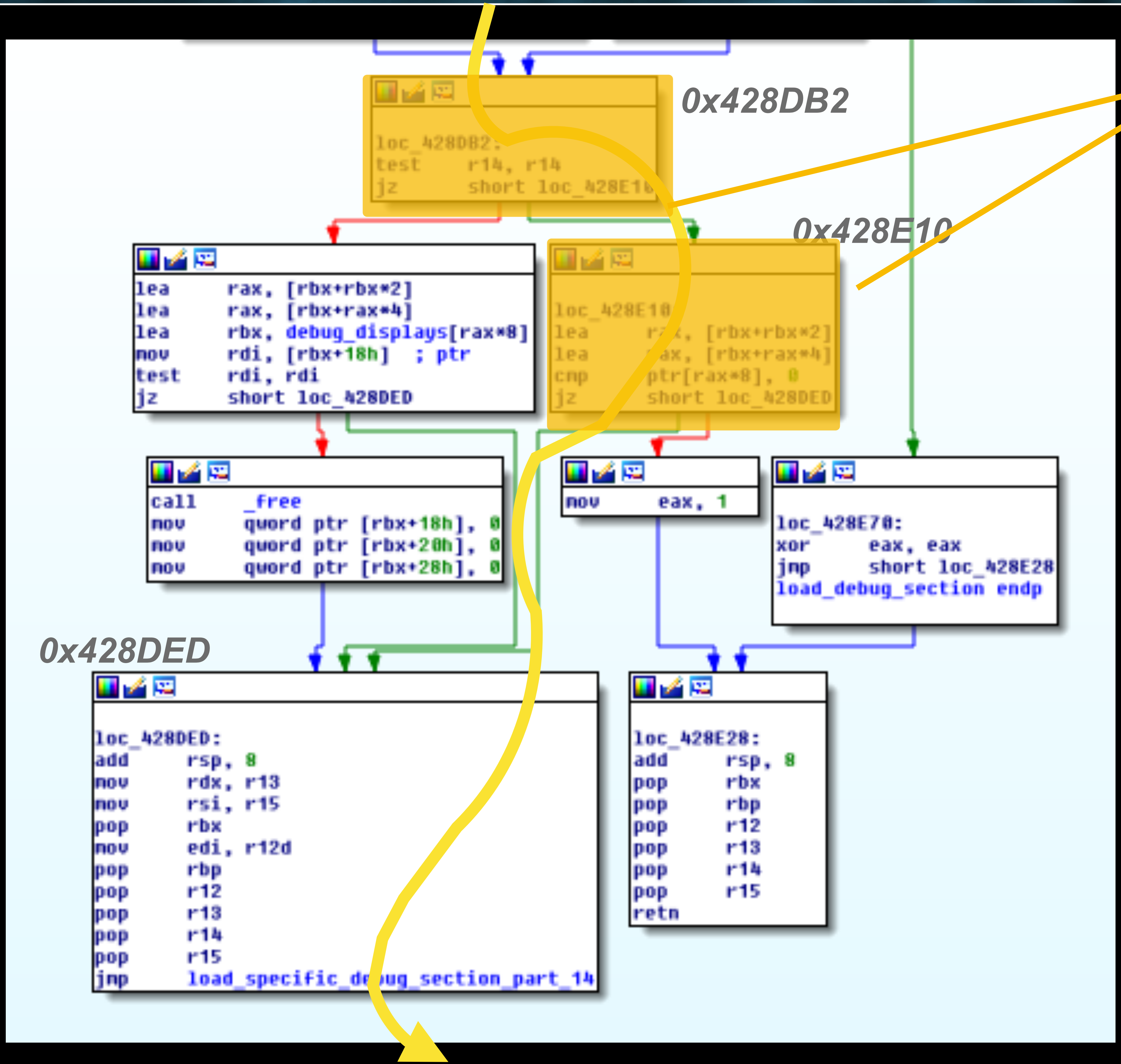
How Coverage Info is Collected in AFL

$ *readelf* *testcase_1*

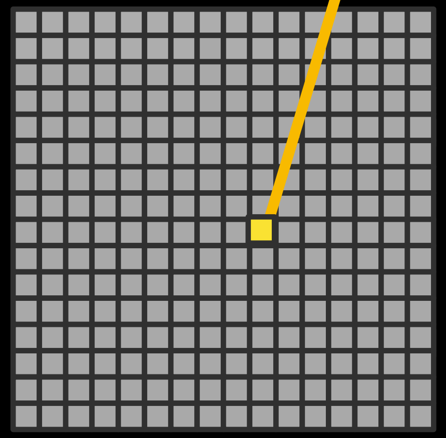
Assuming the basic blocks being covered are:

…
0x428DB2
0x428E10
0x428DED
…

hash(0x428DB2, 0x428E10)



Program *readelf*'s Control Flow Graph (partial)



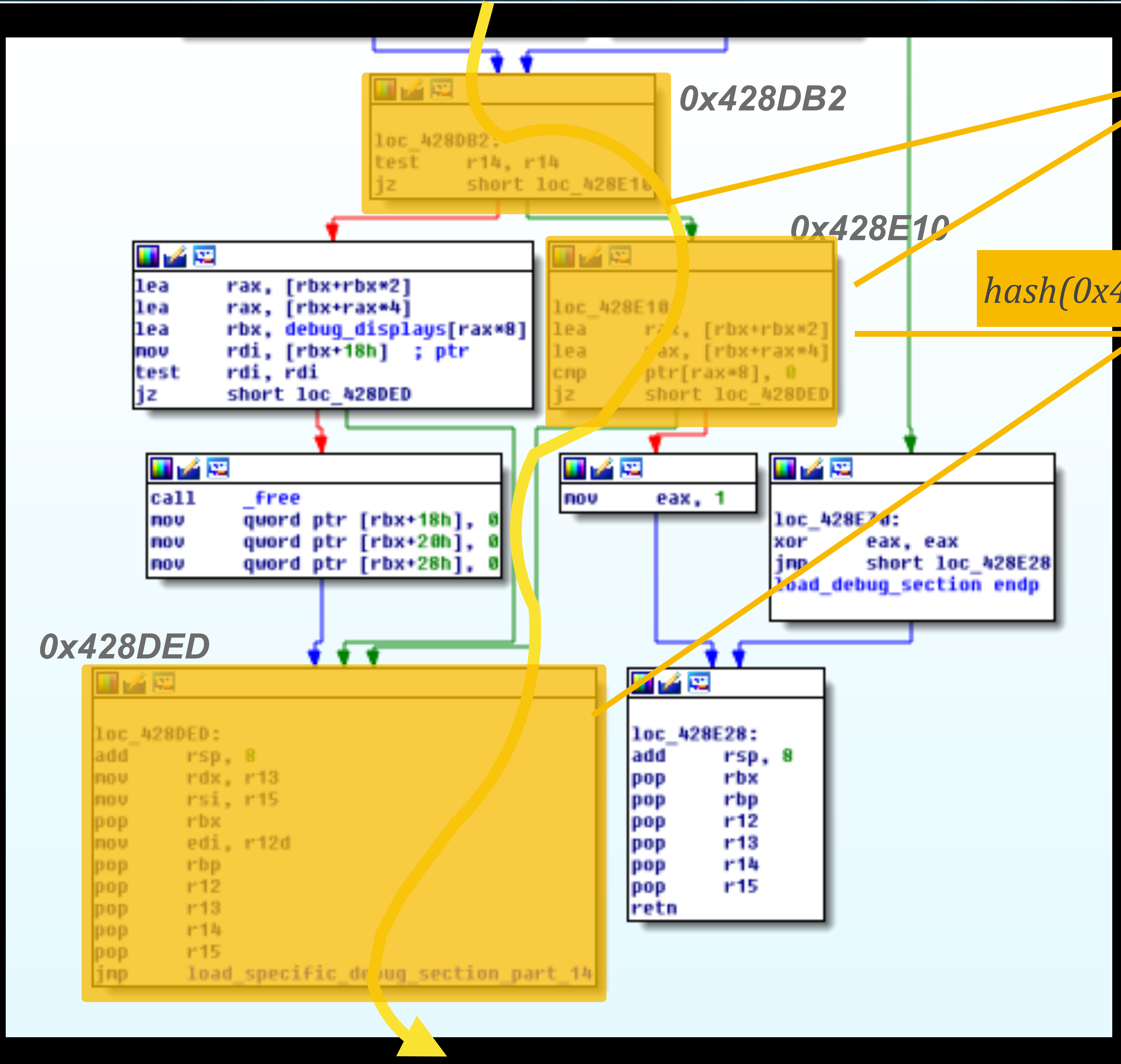
AFL's *shared_mem[ ]*

How Coverage Info is Collected in AFL

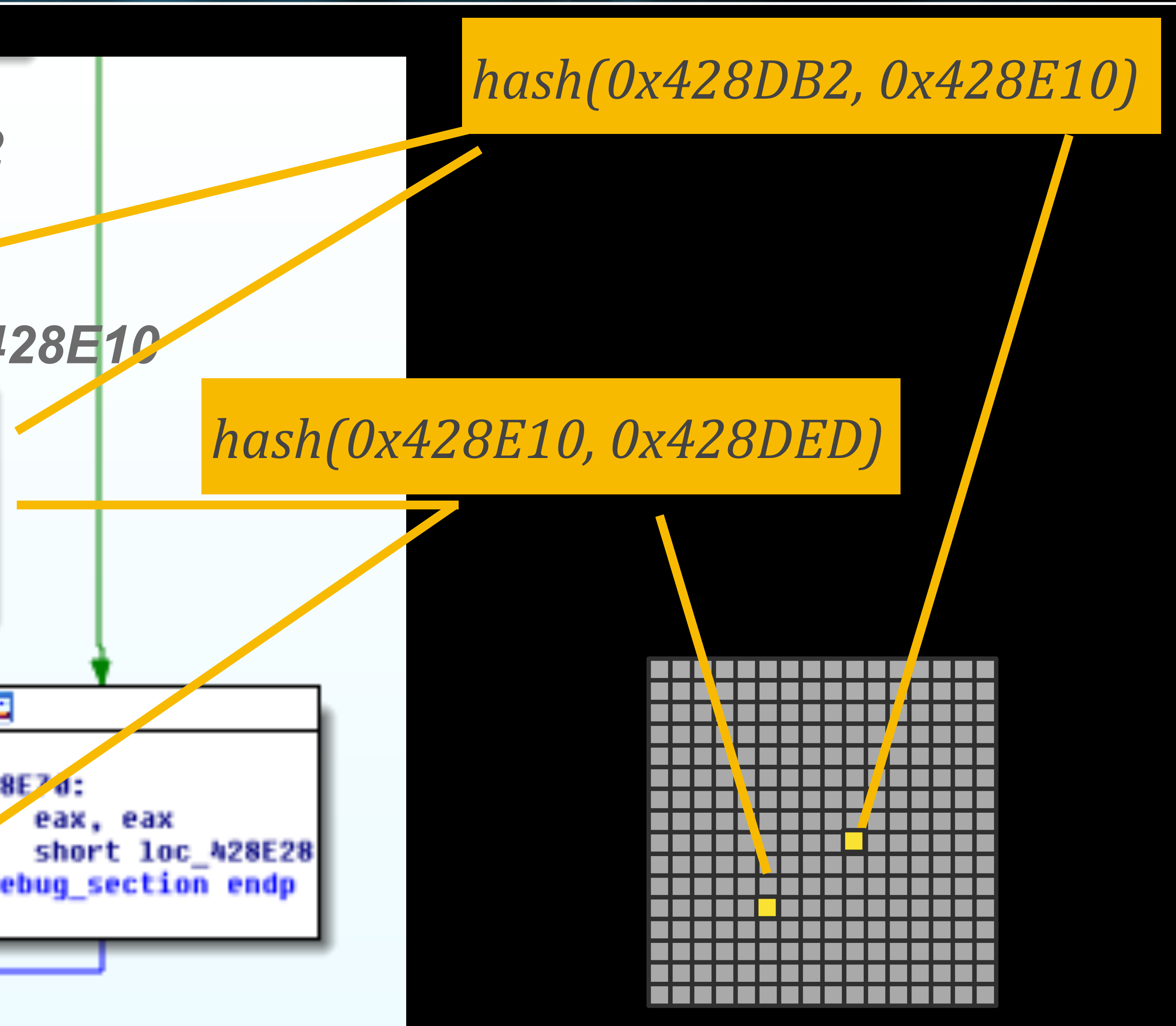
$ *readelf* *testcase_1*

Assuming the basic blocks being covered are:

…
0x428DB2
0x428E10
0x428DED
…



Program *readelf*'s Control Flow Graph (partial)



AFL's *shared_mem[ ]*

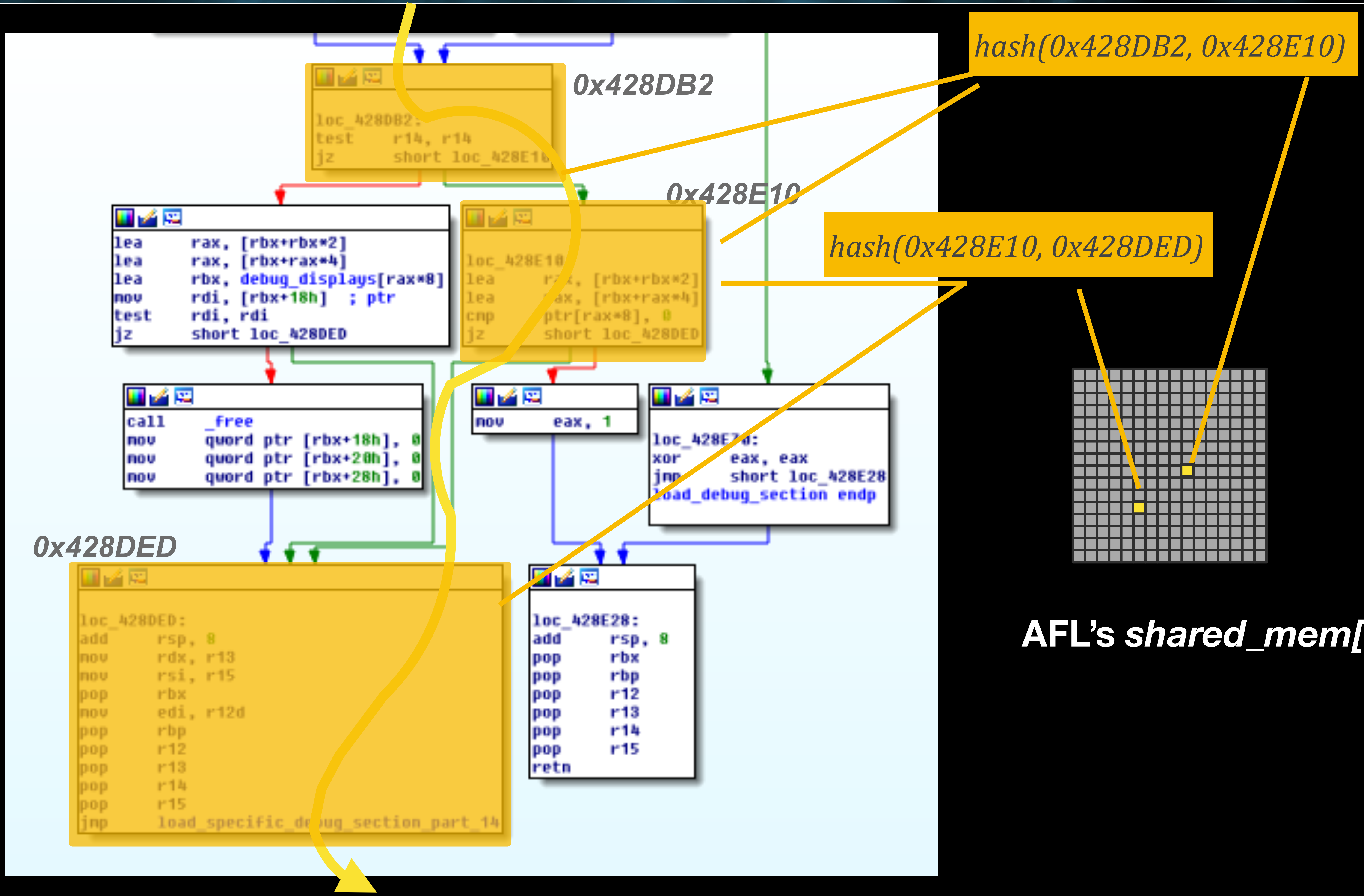
How Coverage Info is Collected in AFL

$ *readelf* *testcase_1*

Assuming the basic blocks being covered are:

…
0x428DB2
0x428E10
0x428DED
…

New Coverage Information!
testcase_1 saved in afl/queue



Program *readelf*'s Control Flow Graph (partial)

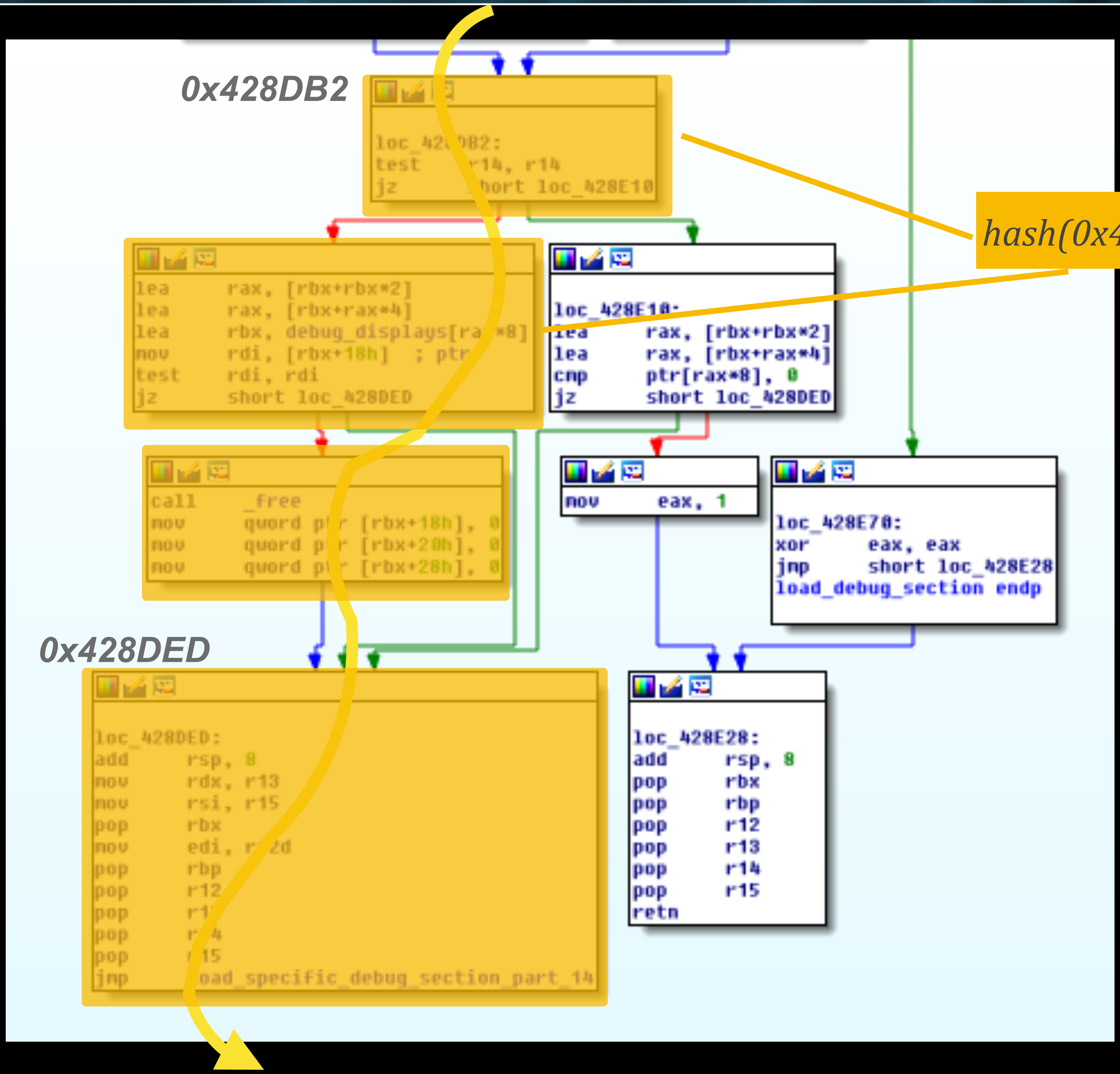
AFL's *shared_mem[ ]*

How Coverage Info is Collected in AFL

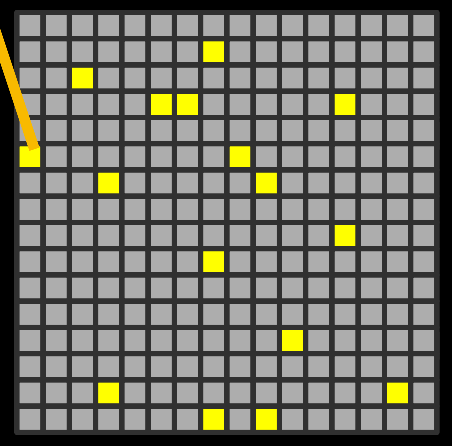
$ *readelf* *testcase_N*

if *shared_mem[ ]* is marked with new updates — find an input with a “new interest path”
…

New Coverage Information!
testcase_N saved in afl/queue



hash(0x428DB2, 0x428DBA)



AFL’s *shared_mem[ ]*

Program *readelf*’s Control Flow Graph (partial)

AFL's Blindspot

$ readelf testcase_X

Basic blocs being covered:

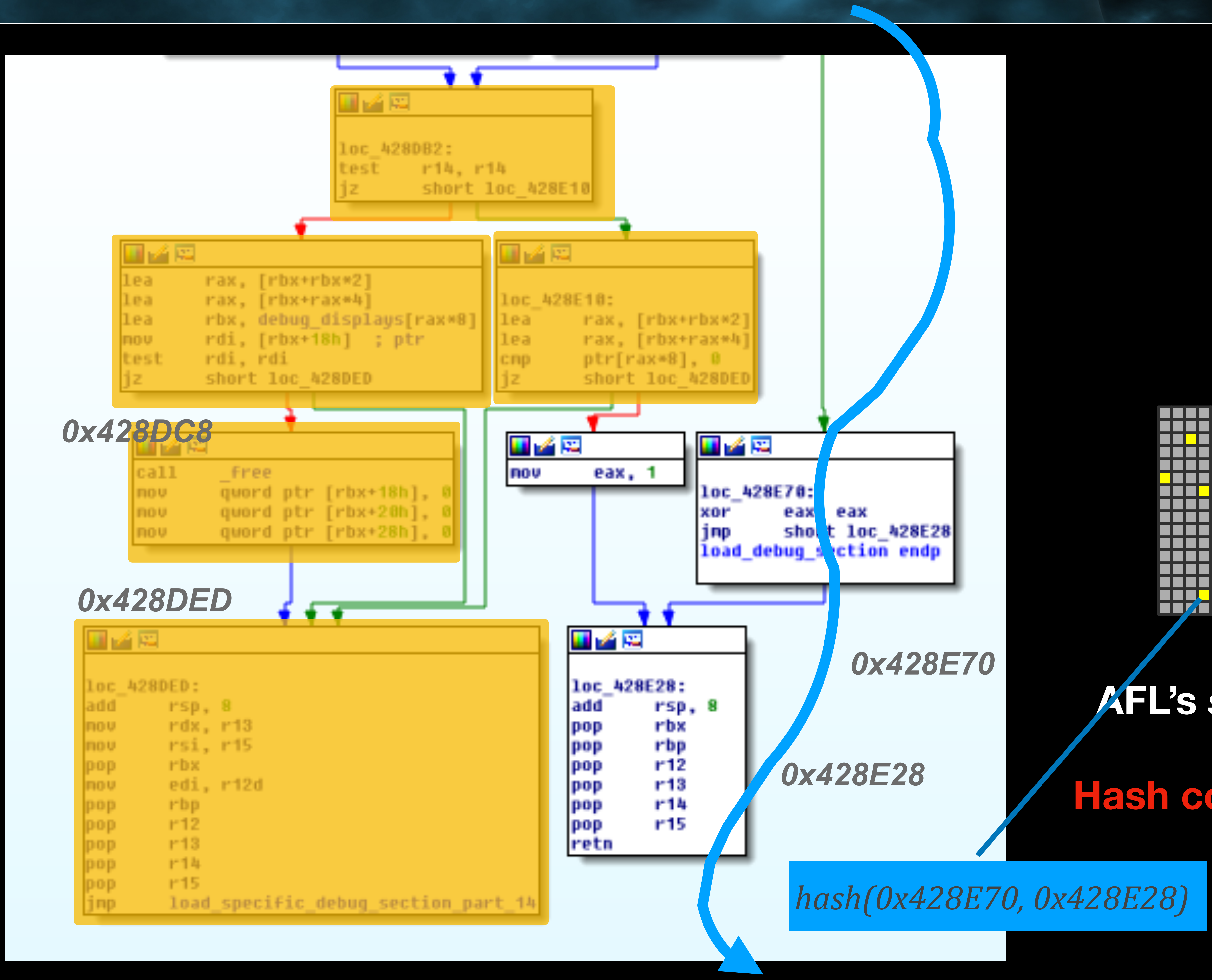
…

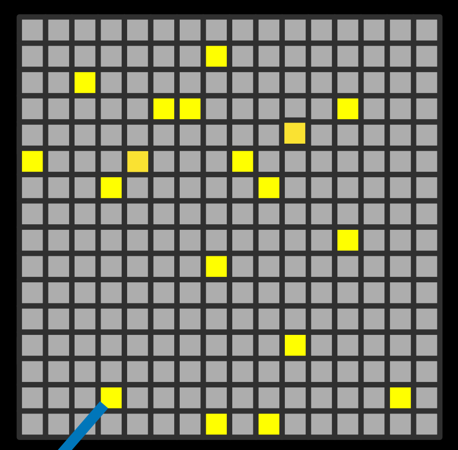
0x428E70

0x428E28

…

If no new updates in shared_mem[ ], AFL considers no new edges.





AFL's shared_mem[ ]

Hash conflict occur!

Program *readelf*'s Control Flow Graph (partial)

AFL's Blindspot

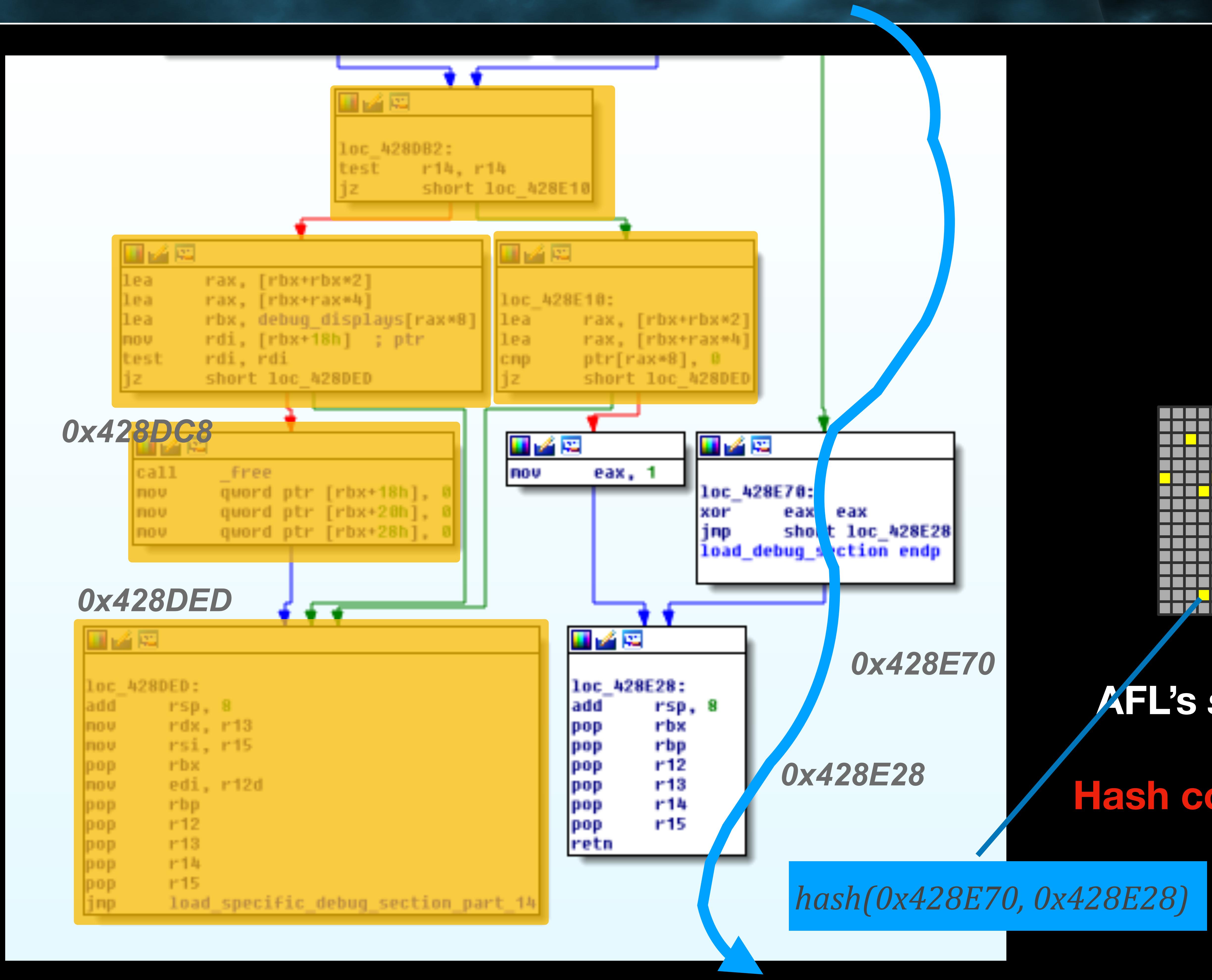
$ *readelf testcase_X*

Basic blocs being covered:

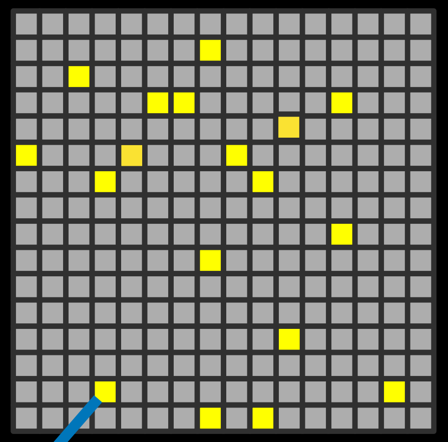
…
0x428E70
0x428E28
…

If no new updates in shared_mem[ ], AFL considers no new edges.

AFL fails to detect a new path,
testcase_X discarded!



Program *readelf*'s Control Flow Graph (partial)



AFL's *shared_mem[ ]*

Hash conflict occur!

Example of AFL's Blindspot (with *readelf*)

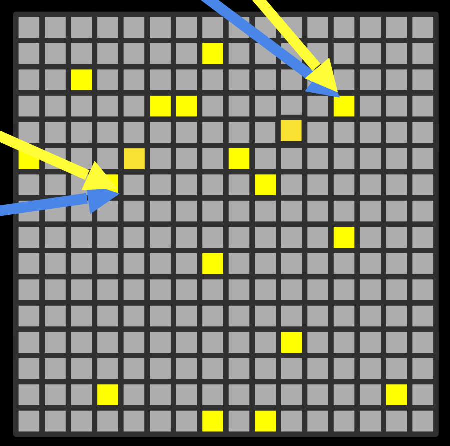
When we combine symbolic execution with AFL, we found AFL refuses to sync several inputs generated by our symbolic execution engine. Two pairs of conflict edges are shown below.

hash(0x41c9b0, 0x41c9d1) = 0x9bd0

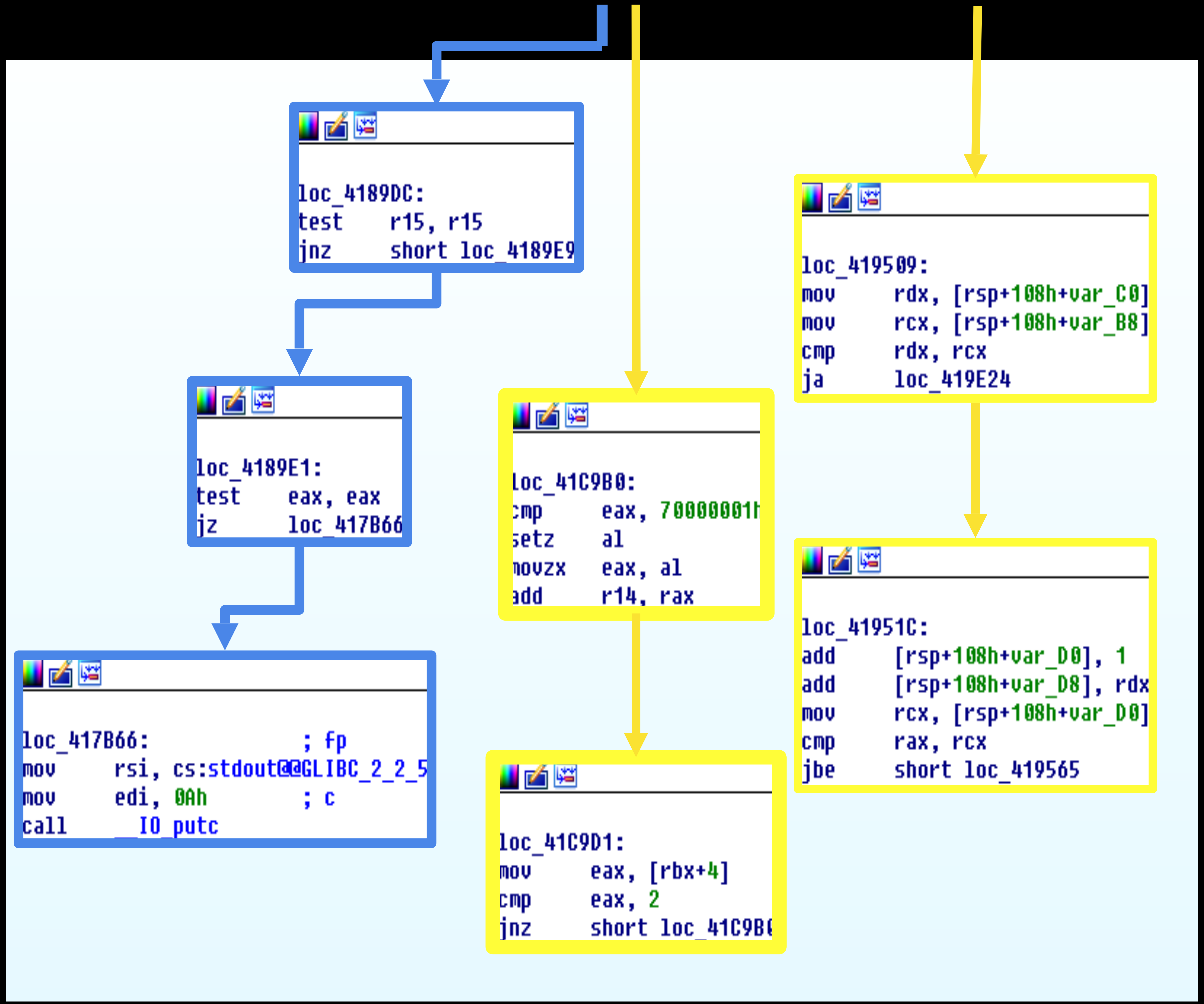
hash(0x4189dc, 0x4189e1) = 0x9bd0

hash(0x419509, 0x41951c) = 0xd79

hash(0x4189e1, 0x417b66) = 0xd79



AFL's *shared_mem[ ]*



readelf's Control Flow Graph (partial)

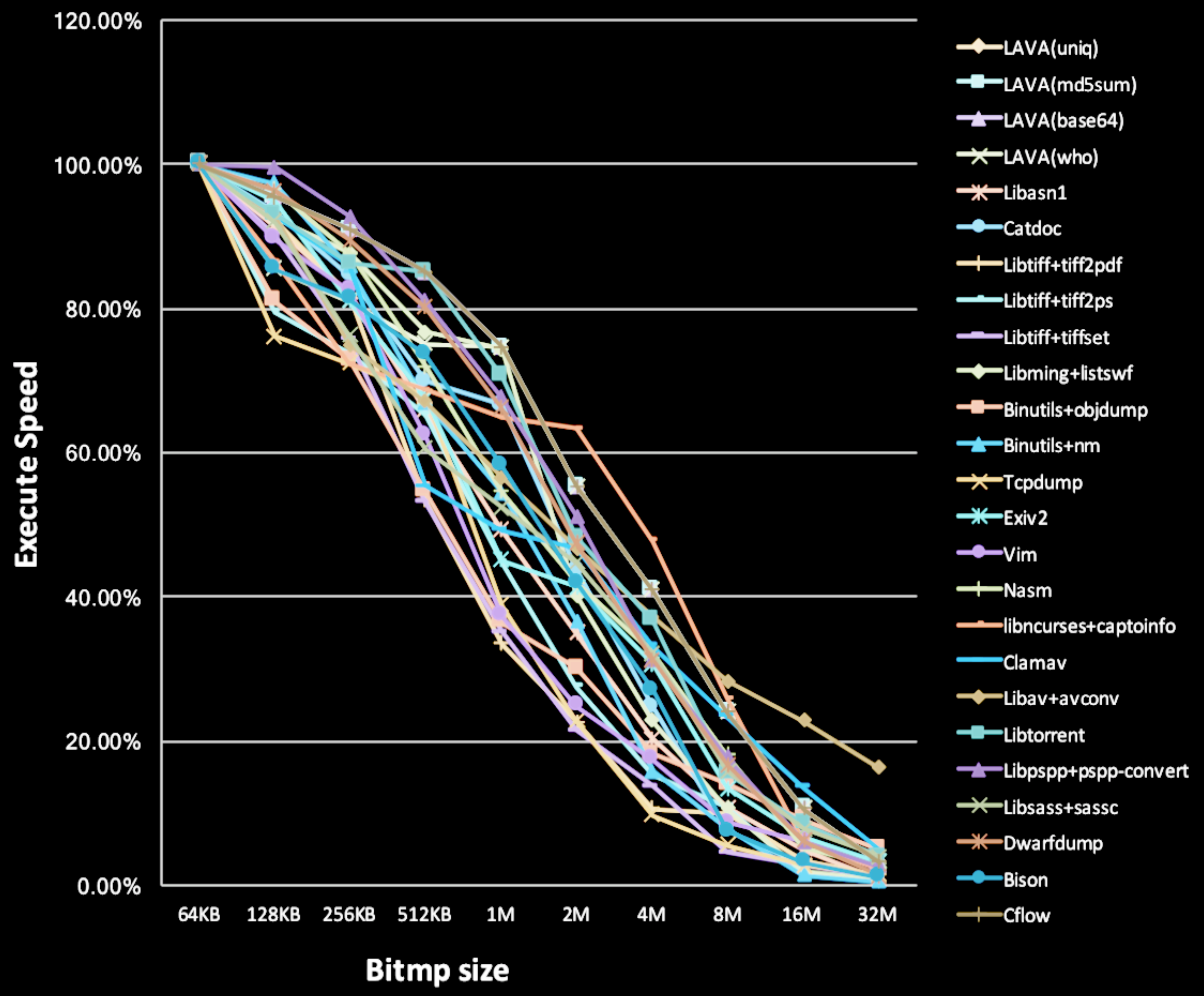
What about Using Large Bitmap Sizes?

Bitmap Sizes vs. AFL Speed

— "CollAFL: Path Sensitive Fuzzing", published in 2018 IEEE S&P

1. Large bitmap sizes reduce but do not eliminate hash conflicts.

2. Speed degrades significantly after bitmap size gets larger (than CPU mem cache size)



How to Resist AFL Fuzzing

- Add Complex Path Constraints

e.g. *if (input * input = long_int_value)*

- Add Delays for Known Invalid Inputs

e.g. insert sleep() call to slow down AFL execution

- Add Nondeterministic Events

e.g. dynamic code relocation

Usually Need Source Code

How to Resist AFL Fuzzing

- Add Complex Path Constraints

e.g. *if (input * input = long_int_value)*

- Add Delays for Known Invalid Inputs

e.g. insert sleep() call to slow down AFL execution

- Add Nondeterministic Events

e.g. dynamic code relocation

Usually Need Source Code

- **Disturb AFL's Seed Selection <— (this talk)**

Reducing AFL's ability to finding new paths by introducing fake edges to cause hash conflicts

Target at the AFL-QEMU mode

Resist through binary rewriting

Without Source Code

General Idea of *deafL*

- Suppress AFL's ability to mutate seeds and trigger crashes
- The *deafL* tool — Inject dummy code to a binary to create conflicting hash values to those edges leading toward crashes

General Idea of *deafL*

The *deafL* tool needs to provide answers to these 3 questions

- Which edges to target (to create hash conflicts)?
- How to create an edge that has a specific hash value?
- How to inject fake edges to a binary?

Which Edges to Target (for Hash Conflict)?

- A naïve solution:
  - Add fake edges to completely fill AFL's share_mem[ ]
  - Binaries become too fat and run very slow!

- Need to target only a small set of edges
  - Idea: find those edges that lead to the mutation of crash inputs

Which Edges to Target (for Hash Conflict)?

- Current Approach
  - Run AFL first to find crashes
  - Find those inputs that mutate to crashes (call them *targeted seed files*)
  - Find all edges that link between the initial seed inputs and the targeted seed files

Find Target Edges (example)

- Start from an AFL crash file

  crashes/id:000000,sig:11,src:000179+000048,op:splice,rep:2

- Find its parents (where it mutates from)

  queue/id:000179,src:000121+000178,op:splice,rep:4,+cov
  queue/id:000048,src:000000,op:havoc,rep:8

Find Target Edges (example)

- Start from an AFL crash file

  crashes/id:000000,sig:11,src:000179+000048,op:splice,rep:2

- Find its parents (where it mutates from)

  queue/id:000179,src:000121+000178,op:splice,rep:4,+cov

  queue/id:000048,src:000000,op:havoc,rep:8

- Find all code edges that covered by these parent inputs but not by the initial seed

  queue/id:0000:initial_seed_input

Sample output of finding target edges

```
[id:000179,src:000121+000178,op:splice,rep:
4,+cov]
introduced [9] new edges:
    [0x43f032, 0x43f06f] at index [0x5687]
    [0x43f06f, 0x43dc22] at index [0x37c1]
    [0x4a418e, 0x4a41c7] at index [0x7610]
    [0x4a41c7, 0x4a41ea] at index [0x7f90]
    [0x4a431f, 0x4a4331] at index [0xc8ab]
    [0x4a4331, 0x4a4386] at index [0x68a1]
    [0x4a7033, 0x4a7039] at index [0xd402]
    [0x4a7039, 0x4a7058] at index [0xb004]
    [0x4a7058, 0x4a7070] at index [0xa885]
    … …
```

Create Edges with Specific Hash Values

- Use a *cmp-jne* snippet to fake an edge
  - for a given "targeted edge":
    [*blk_A_addr, blk_B_addr*]
  - Assuming we have a starting address to insert code (known *prev_location*), calculating a target address so that

prev_location ^ cur_location = blk_A_addr ^ blk_B_addr

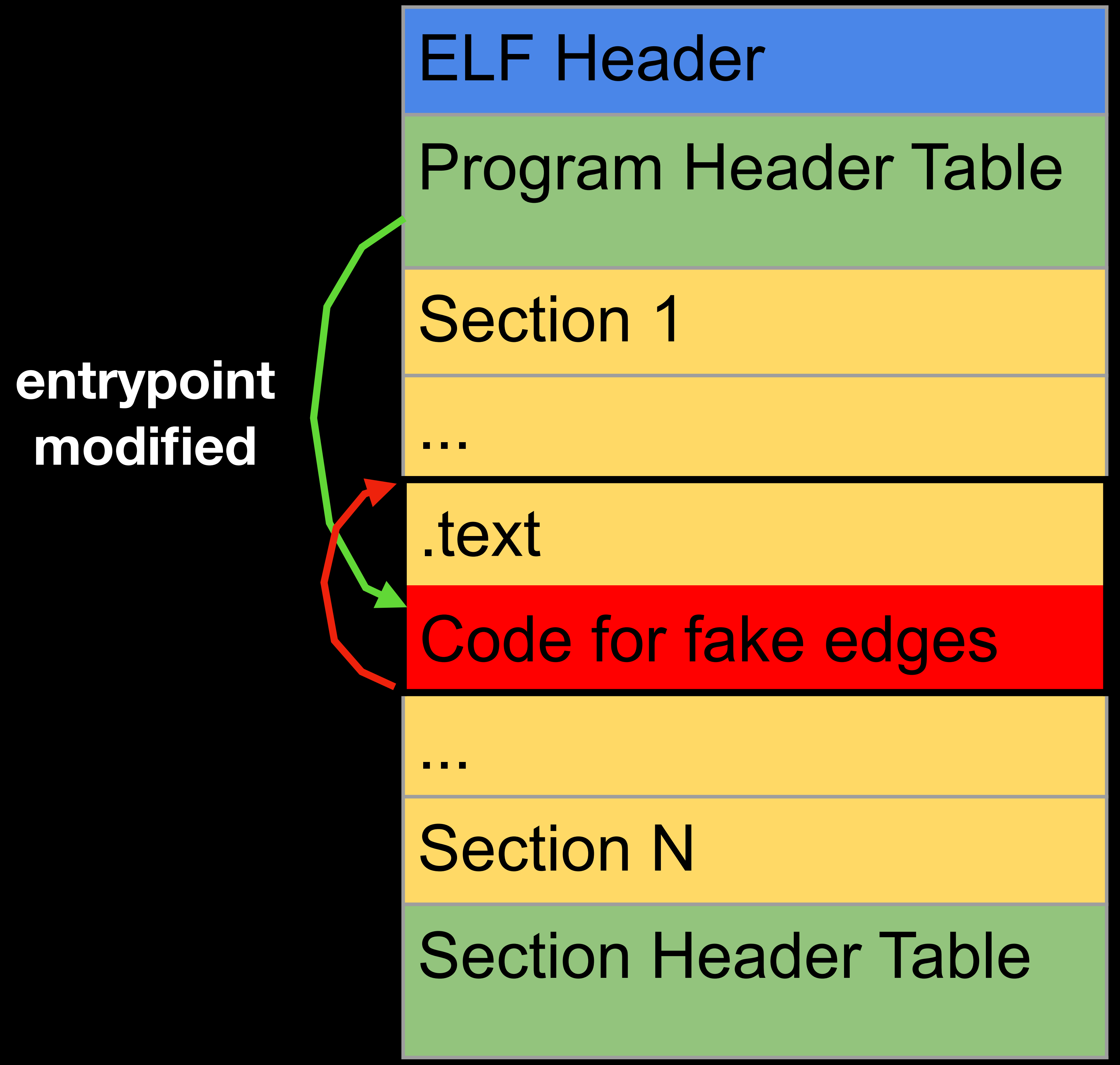
- Can generate a nested blob of *cmp-jne* snippets for a list of "targeted edges".

```
cur_location = (block_address >> 4) ^ (block_address << 8);
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

```
{prev_location'}:        cmp  %rsp  0x0
{prev_location'}+4:      jne  {cur_location'}
{prev_location'}+11:     nop
{prev_location'}+12:     nop
......
{cur_location'}:         nop
```

Injecting Edges with Hash Conflicts

- Code Injection Overview
  - Build on the python lief package
  - Modify entrypoint to points to inserted code (to fake edges)
  - Major changes to code and data
    - Update section table to extend .text size
    - Update all address/offset/data info after the inserted section:
      - ".dynamic", ".rela.dyn", ".rela.plt", ".symtab", ".dynsym"
      - Pointer references in ".text", ".data", ".rodata"



```
$ python Deafl.py examples/magic/cb
```

```c
int cb(uchar *out) {
    int ret = 0;

    if (out[0] == 'M') {
        if (out[1] == 'A') {
            if (out[2] == 'G') {
                if (out[3] = 'I') {
                    if (out[4] == 'C') {
                        if (out[5] == '!') {
                            ret = 1;
                            /* printf("You Won!\n"); */
                            crash();
                        }
                    }
                }
            }
        }
    }
    /* printf("Please Try Again\n"); */
    return 0;
}
```

Peace Disrupted

```
                  american fuzzy lop 2.52b (cb)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 3 min, 51 sec       │  cycles done : 65     │
│   last new path : 0 days, 0 hrs, 1 min, 7 sec        │  total paths : 7      │
│ last uniq crash : 0 days, 0 hrs, 0 min, 6 sec        │ uniq crashes : 1      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 5 (71.43%)        │    map density : 0.05% / 0.08%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.00 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored paths : 7 (100.00%)            │
│ stage execs : 2028/2048 (99.02%)    │  new edges on : 7 (100.00%)            │
│ total execs : 336k                  │ total crashes : 1 (1 unique)           │
│  exec speed : 1322/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 0/320, 0/313, 0/299                   │    levels : 6          │
│  byte flips : 0/40, 0/33, 0/19                      │   pending : 0          │
│ arithmetics : 2/2237, 0/197, 0/70                   │  pend fav : 0          │
│  known ints : 0/244, 0/880, 0/824                   │ own finds : 6          │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a        │
│       havoc : 3/141k, 2/187k                        │ stability : 100.00%    │
│        trim : 0.00%/6, 0.00%                        ├────────────────────────┘
^C──────────────────────────────────────────────────────┘          [cpu000: 53%]
```

Total Paths: 7
uniq Crashes: 1

Peace Restored

```
                american fuzzy lop 2.52b (new_cb)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 24 min, 37 sec       │  cycles done : 555    │
│   last new path : 0 days, 0 hrs, 23 min, 46 sec       │  total paths : 6      │
│ last uniq crash : none seen yet                       │ uniq crashes : 0      │
│  last uniq hang : none seen yet                       │   uniq hangs : 0      │
├─ cycle progress ───────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 4 (66.67%)        │    map density : 0.06% / 0.09%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.00 bits/tuple       │
├─ stage progress ───────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored paths : 6 (100.00%)            │
│ stage execs : 108/512 (21.09%)      │  new edges on : 6 (100.00%)            │
│ total execs : 2.59M                 │ total crashes : 0 (0 unique)           │
│  exec speed : 1820/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ──────────┴───────────┬─ path geometry ────────────┤
│   bit flips : 0/256, 0/250, 0/238               │    levels : 5              │
│  byte flips : 0/32, 0/26, 0/16                  │   pending : 0              │
│ arithmetics : 2/1790, 0/52, 0/0                 │  pend fav : 0              │
│  known ints : 0/190, 0/724, 0/704               │ own finds : 5              │
│  dictionary : 0/0, 0/0, 0/0                     │  imported : n/a            │
│       havoc : 2/1.11M, 1/1.47M                  │ stability : 100.00%        │
│        trim : 0.00%/5, 0.00%                    ├────────────────────────────┘
└─────────────────────────────────────────────────┘             [cpu001: 39%]
```

Previous Result:

total paths: 7

uniq crashes: 1

Apply *deafL* to other binaries

```
              american fuzzy lop 2.52b (tcpdump)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 5 min, 55 sec       │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec        │  total paths : 298    │
│ last uniq crash : 0 days, 0 hrs, 1 min, 21 sec       │ uniq crashes : 1      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 27 (9.06%)        │    map density : 0.46% / 4.74%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.15 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : arith 8/8             │ favored paths : 190 (63.76%)          │
│ stage execs : 5628/6629 (84.90%)    │  new edges on : 236 (79.19%)          │
│ total execs : 370k                  │ total crashes : 1 (1 unique)          │
│  exec speed : 1062/sec              │  total tmouts : 0 (0 unique)          │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 60/11.9k, 38/11.9k, 13/11.8k          │    levels : 3          │
│  byte flips : 0/1486, 1/1470, 1/1438                │   pending : 283        │
│ arithmetics : 63/76.6k, 1/54.4k, 0/30.7k            │  pend fav : 177        │
│  known ints : 6/5353, 4/24.5k, 10/43.9k             │ own finds : 297        │
│  dictionary : 0/0, 0/0, 0/1312                      │  imported : n/a        │
│       havoc : 93/85.2k, 0/0                         │ stability : 100.00%    │
│        trim : 19.94%/592, 0.00%                     ├────────────────────────┘
^C────────────────────────────────────────────────────┘          [cpu001: 37%]

+++ Testing aborted by user +++
[+] We're done here. Have a nice day!
```

CVE 2015-3138

```
$ python Deafl.py examples/tcpdump_cve2015-3138/tcpdump
```

```
    american fuzzy lop 2.52b (tcpdump_d52da20b46f343138c6e94464c04f269)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 4 days, 21 hrs, 8 min, 31 sec      │  cycles done : 22     │
│   last new path : 0 days, 0 hrs, 12 min, 58 sec      │  total paths : 6245   │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ───────────────────┬─ map coverage ──┴───────────────────────┤
│  now processing : 2449* (39.22%)   │    map density : 0.81% / 21.52%         │
│ paths timed out : 0 (0.00%)        │ count coverage : 2.80 bits/tuple        │
├─ stage progress ───────────────────┼─ findings in depth ─────────────────────┤
│  now trying : interest 16/8        │ favored paths : 1937 (31.02%)           │
│ stage execs : 1020/2833 (36.00%)   │  new edges on : 2528 (40.48%)           │
│ total execs : 201M                 │ total crashes : 0 (0 unique)            │
│  exec speed : 609.2/sec            │  total tmouts : 0 (0 unique)            │
├─ fuzzing strategy yields ──────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 1312/8.91M, 499/8.91M, 262/8.90M     │    levels : 28         │
│  byte flips : 40/1.11M, 25/617k, 14/637k           │   pending : 2259       │
│ arithmetics : 1029/34.0M, 47/29.7M, 11/21.8M       │  pend fav : 0          │
│  known ints : 181/2.09M, 308/9.33M, 174/17.5M      │ own finds : 6244       │
│  dictionary : 0/0, 0/0, 354/26.9M                  │  imported : n/a        │
│       havoc : 1988/30.5M, 0/0                      │ stability : 100.00%    │
│        trim : 21.65%/372k, 44.85%                  ├────────────────────────┘
└────────────────────────────────────────────────────┘          [cpu001: 51%]
```

No crash found after more than 4 days

CVE 2015-3138

```
$ python Deafl.py examples/objcopy_cve2018-10534/objcopy
```

```
                american fuzzy lop 2.52b (objcopy)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 1 min, 11 sec        │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec         │  total paths : 164    │
│ last uniq crash : 0 days, 0 hrs, 0 min, 46 sec        │ uniq crashes : 1      │
│  last uniq hang : none seen yet                       │   uniq hangs : 0      │
├─ cycle progress ───────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 32 (19.51%)       │    map density : 2.72% / 4.75%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.74 bits/tuple       │
├─ stage progress ───────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored paths : 64 (39.02%)            │
│ stage execs : 2680/6144 (43.62%)    │  new edges on : 93 (56.71%)            │
│ total execs : 20.6k                 │ total crashes : 1 (1 unique)           │
│  exec speed : 539.6/sec             │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────┬─ path geometry ────────────┤
│   bit flips : n/a, n/a, n/a                      │    levels : 3              │
│  byte flips : n/a, n/a, n/a                      │   pending : 156            │
│ arithmetics : n/a, n/a, n/a                      │  pend fav : 58             │
│  known ints : n/a, n/a, n/a                      │ own finds : 163            │
│  dictionary : n/a, n/a, n/a                      │  imported : n/a            │
│       havoc : 138/11.1k, 20/4712                 │ stability : 100.00%        │
│        trim : 46.79%/825, n/a                    ├────────────────────────────┘
^C───────────────────────────────────────────────────┘          [cpu000: 29%]
```

With a seed that is similar to the CVE crash input

CVE 2018-10534

```
     american fuzzy lop 2.52b (objcopy_9b64fd0ee800428fade689496d8914ba)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 9 hrs, 6 min, 55 sec       │  cycles done : 14     │
│   last new path : 0 days, 0 hrs, 0 min, 24 sec       │  total paths : 2998   │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : 0 days, 1 hrs, 33 min, 20 sec      │   uniq hangs : 5      │
├─ cycle progress ────────────────┬─ map coverage ─────┴───────────────────────┤
│  now processing : 19* (0.63%)   │    map density : 2.92% / 13.34%           │
│ paths timed out : 0 (0.00%)     │ count coverage : 3.64 bits/tuple          │
├─ stage progress ────────────────┼─ findings in depth ────────────────────────┤
│  now trying : splice 6          │ favored paths : 444 (14.81%)               │
│ stage execs : 88/96 (91.67%)    │  new edges on : 811 (27.05%)               │
│ total execs : 10.9M             │ total crashes : 0 (0 unique)               │
│  exec speed : 434.1/sec         │  total tmouts : 5 (5 unique)               │
├─ fuzzing strategy yields ───────┴───────────────┬─ path geometry ────────────┤
│   bit flips : n/a, n/a, n/a                     │    levels : 35             │
│  byte flips : n/a, n/a, n/a                     │   pending : 1666           │
│ arithmetics : n/a, n/a, n/a                     │  pend fav : 0              │
│  known ints : n/a, n/a, n/a                     │ own finds : 2997           │
│  dictionary : n/a, n/a, n/a                     │  imported : n/a            │
│       havoc : 2002/4.73M, 995/5.94M             │ stability : 100.00%        │
│        trim : 40.61%/239k, n/a                  ├────────────────────────────┘
└─────────────────────────────────────────────────┘          [cpu001: 58%]
```

With a seed that is similar to the CVE crash input

CVE 2018-10534

Limitations of *deafL*

- Injected code can be easily identified
  - potentially can be muted by another round of binary rewriting
- Only resist AFL-QEMU
  - may not work with other instrumentation schemes (Intel-PT, PIN, DynamoRIO)
- Only reduce AFL's ability to explore new paths
  - does not eliminate AFL's chance to find specific paths
  - no guarantees due to random mutations

Other Misc Methods to Resist AFL

- Leverage the Limitation of AFL-QEMU
    - AFL-QEMU only tracks edges in an EFL binary's 1st code segment
    - Move code to a new code segment to avoid AFL tracking
- Inserting False Termination Signals
    - Abort at normal exit points to generate fake crashes

Summary

- AFL's high efficiency comes from its compact data structure for edge coverage (*shared_mem[ ]*)
- Hash conflict creates a blindspot for AFL — limits its ability to explore paths
- The *deafL* tool — binary rewriting to resist AFL fuzzing
- Intentionally create hash conflicts for edges that lead to the mutation of crash inputs

Q&A

kangli.ctf@gmail.com