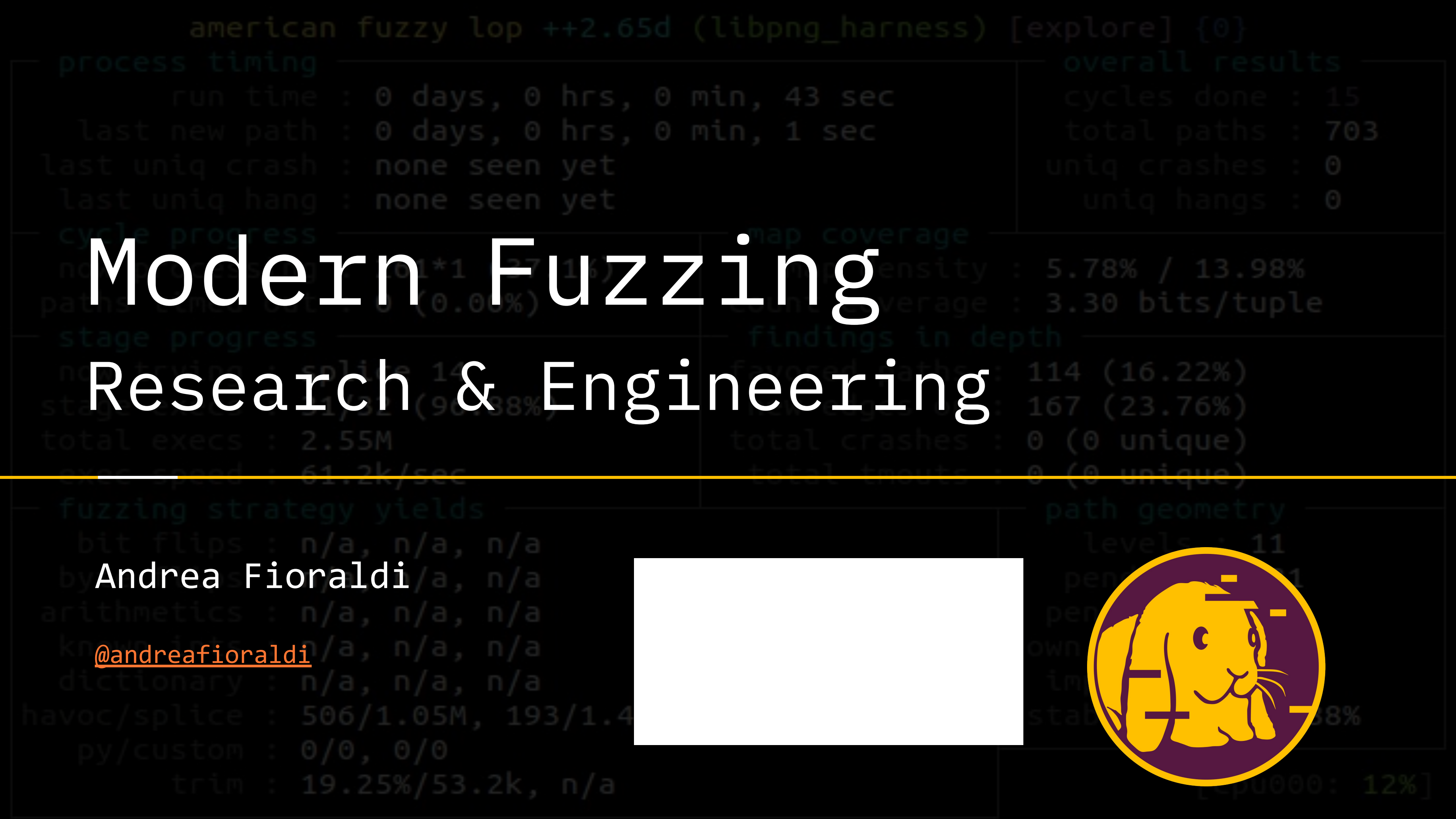
Modern Fuzzing

Research & Engineering

Andrea Fioraldi

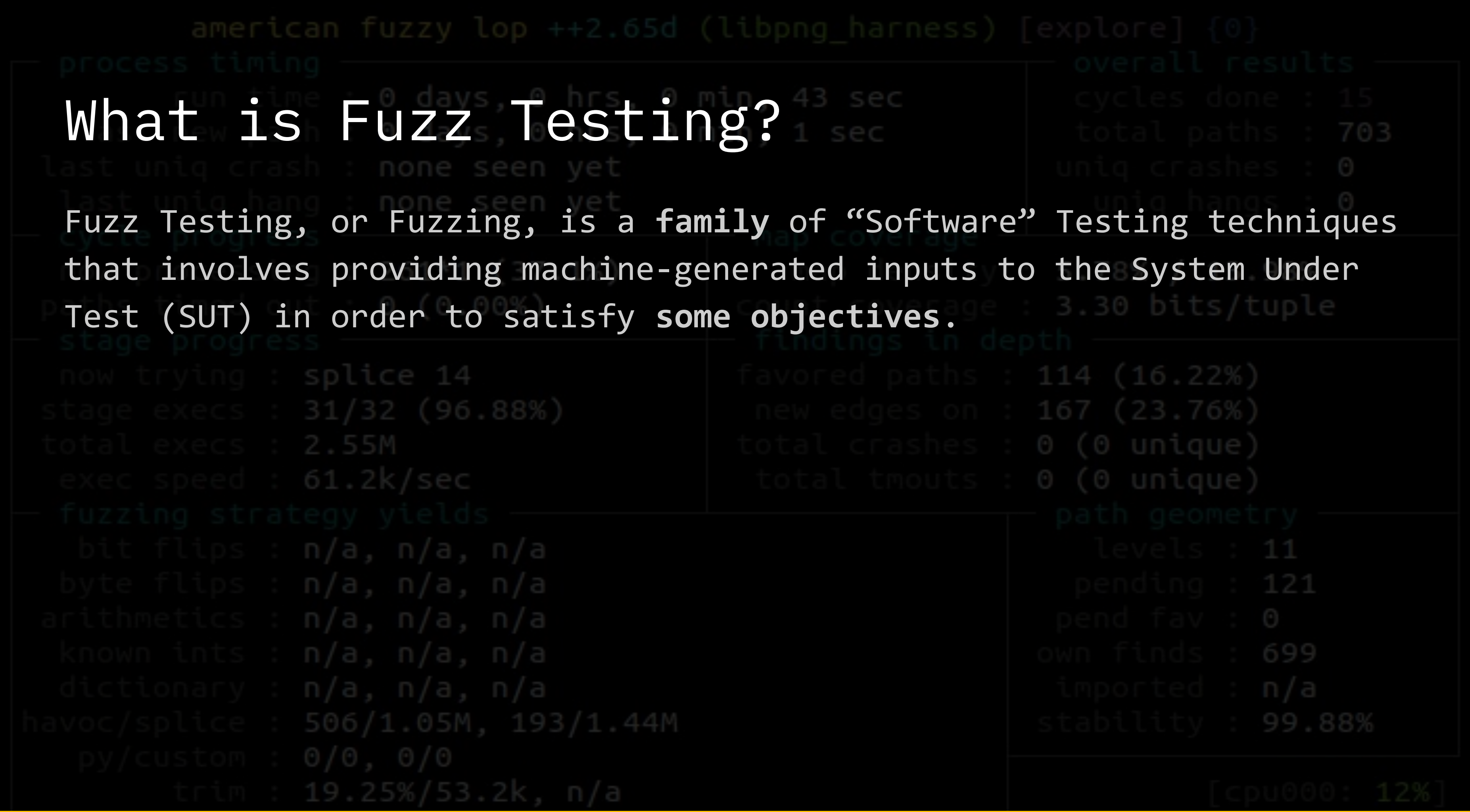
@andreafioraldi



```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
 process timing                                          overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 1 sec          total paths : 703
last uniq crash : none seen yet                       uniq crashes : 0
 last uniq hang : none seen yet                         uniq hangs : 0
 cycle progress                         map coverage
                                          map density : 5.78% / 13.98%
paths timed out : 0 (0.00%)            count coverage : 3.30 bits/tuple
 stage progress                         findings in depth
                                        favored paths : 114 (16.22%)
                                        new edges on : 167 (23.76%)
total execs : 2.55M                    total crashes : 0 (0 unique)
 exec speed : 61.2k/sec                 total tmouts : 0 (0 unique)
 fuzzing strategy yields                               path geometry
  bit flips : n/a, n/a, n/a                              levels : 11
                                                                 
arithmetics : n/a, n/a, n/a                                       
                                                                  
 dictionary : n/a, n/a, n/a                                       
havoc/splice : 506/1.05M, 193/1.4                                 
  py/custom : 0/0, 0/0
       trim : 19.25%/53.2k, n/a                        [cpu000: 12%]
```

What is Fuzz Testing?

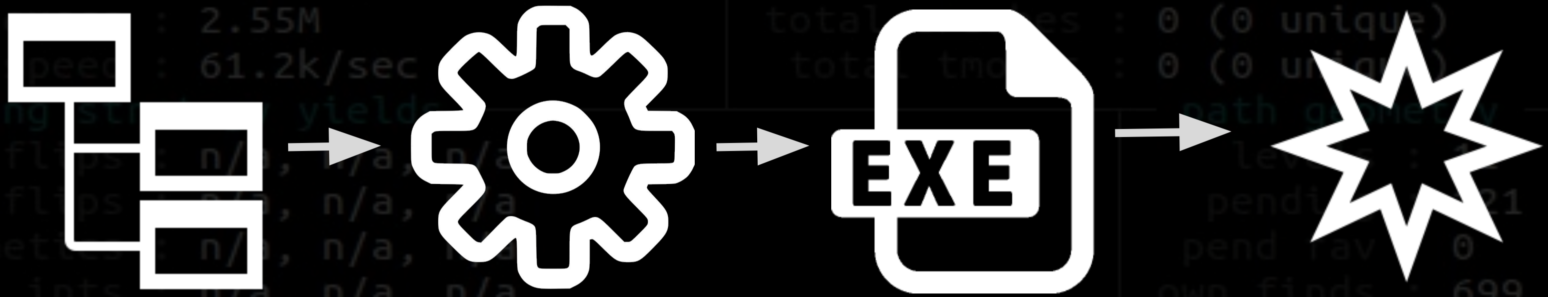
Fuzz Testing, or Fuzzing, is a **family** of “Software” Testing techniques that involves providing machine-generated inputs to the System Under Test (SUT) in order to satisfy **some objectives**.



```
            american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                          overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
   last new path : [illegible] days, 0 hrs, [illegible], 1 sec   total paths : 703
 last uniq crash : none seen yet                        uniq crashes : 0
  last uniq hang : none seen yet                          uniq hangs : 0
  cycle progress                        map coverage
  [illegible]                             [illegible]
 paths timed out : 0 (0.00%)            count coverage : 3.30 bits/tuple
  stage progress                        findings in depth
  now trying : splice 14                favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)            new edges on : 167 (23.76%)
 total execs : 2.55M                    total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                 total tmouts : 0 (0 unique)
  fuzzing strategy yields                               path geometry
   bit flips : n/a, n/a, n/a                              levels : 11
  byte flips : n/a, n/a, n/a                             pending : 121
 arithmetics : n/a, n/a, n/a                            pend fav : 0
  known ints : n/a, n/a, n/a                           own finds : 699
  dictionary : n/a, n/a, n/a                            imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                    stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                               [cpu000: 12%]
```

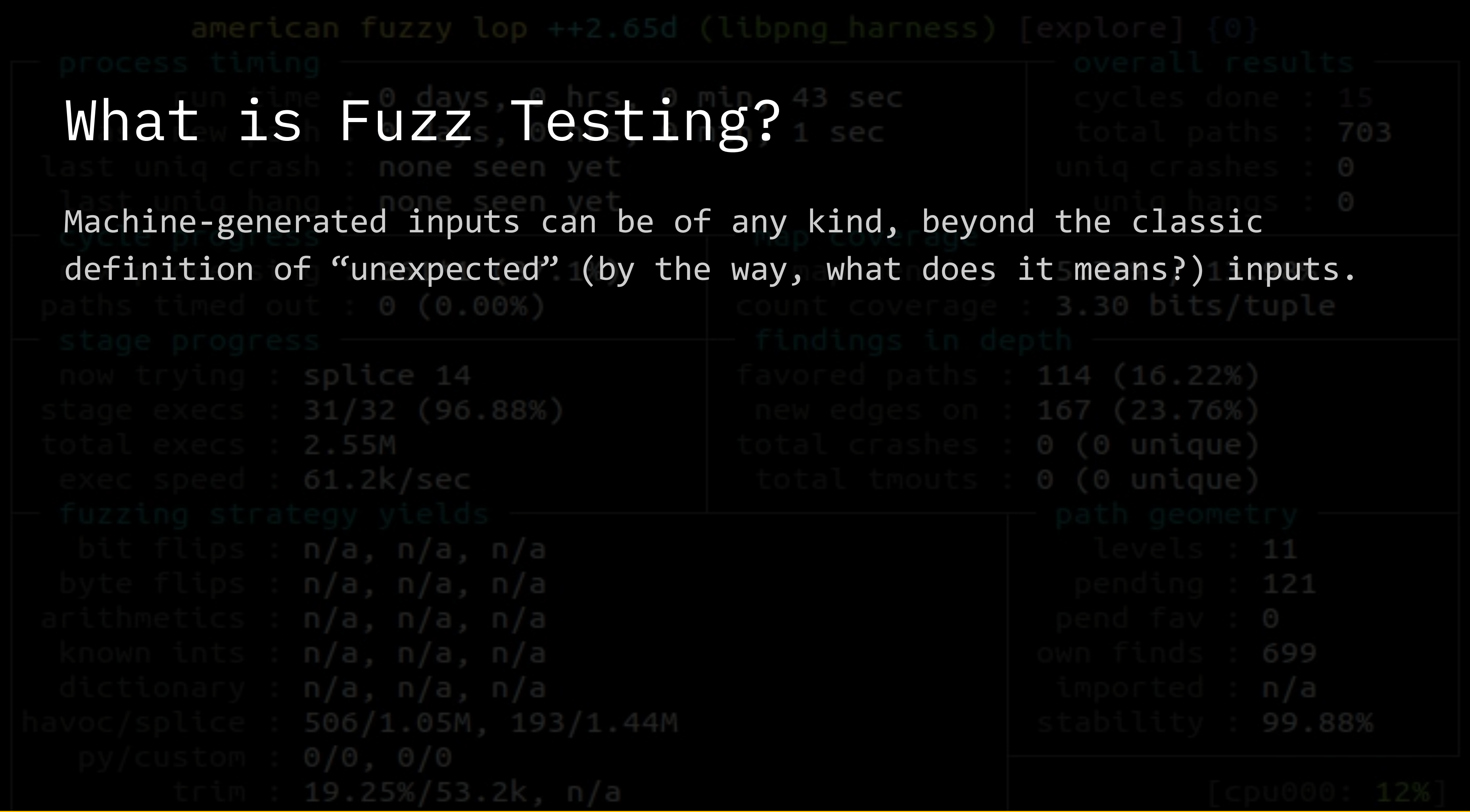
What is Fuzz Testing?

Fuzz Testing, or Fuzzing, is a **family** of “Software” Testing techniques that involves providing machine-generated inputs to the System Under Test (SUT) in order to satisfy **some objectives**.



What is Fuzz Testing?

Machine-generated inputs can be of any kind, beyond the classic definition of “unexpected” (by the way, what does it means?) inputs.

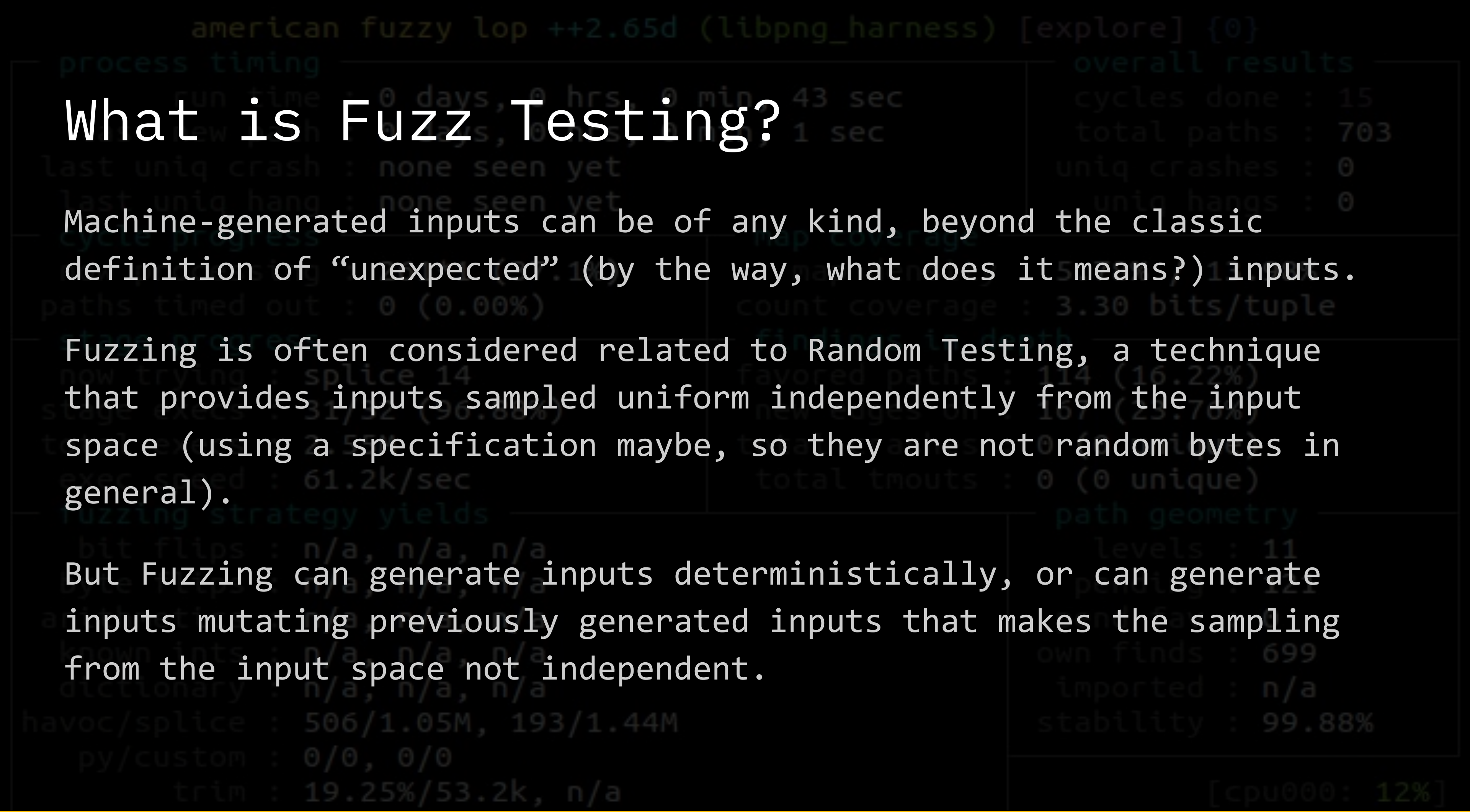


```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec        │  cycles done : 15     │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec         │  total paths : 703    │
│ last uniq crash : none seen yet                        │ uniq crashes : 0      │
│  last uniq hang : none seen yet                        │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : [illegible]       │    map density : [illegible]           │
│ paths timed out : 0 (0.00%)          │ count coverage : 3.30 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 14              │ favored paths : 114 (16.22%)           │
│ stage execs : 31/32 (96.88%)         │  new edges on : 167 (23.76%)           │
│ total execs : 2.55M                  │ total crashes : 0 (0 unique)           │
│  exec speed : 61.2k/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                          │    levels : 11          │
│  byte flips : n/a, n/a, n/a                          │   pending : 121         │
│ arithmetics : n/a, n/a, n/a                          │  pend fav : 0           │
│  known ints : n/a, n/a, n/a                          │ own finds : 699         │
│  dictionary : n/a, n/a, n/a                          │  imported : n/a         │
│havoc/splice : 506/1.05M, 193/1.44M                   │ stability : 99.88%      │
│   py/custom : 0/0, 0/0                               ├────────────────────────┘
│        trim : 19.25%/53.2k, n/a                      │          [cpu000: 12%]
```

What is Fuzz Testing?

Machine-generated inputs can be of any kind, beyond the classic definition of “unexpected” (by the way, what does it means?) inputs.

Fuzzing is often considered related to Random Testing, a technique that provides inputs sampled uniform independently from the input space (using a specification maybe, so they are not random bytes in general).



What is Fuzz Testing?

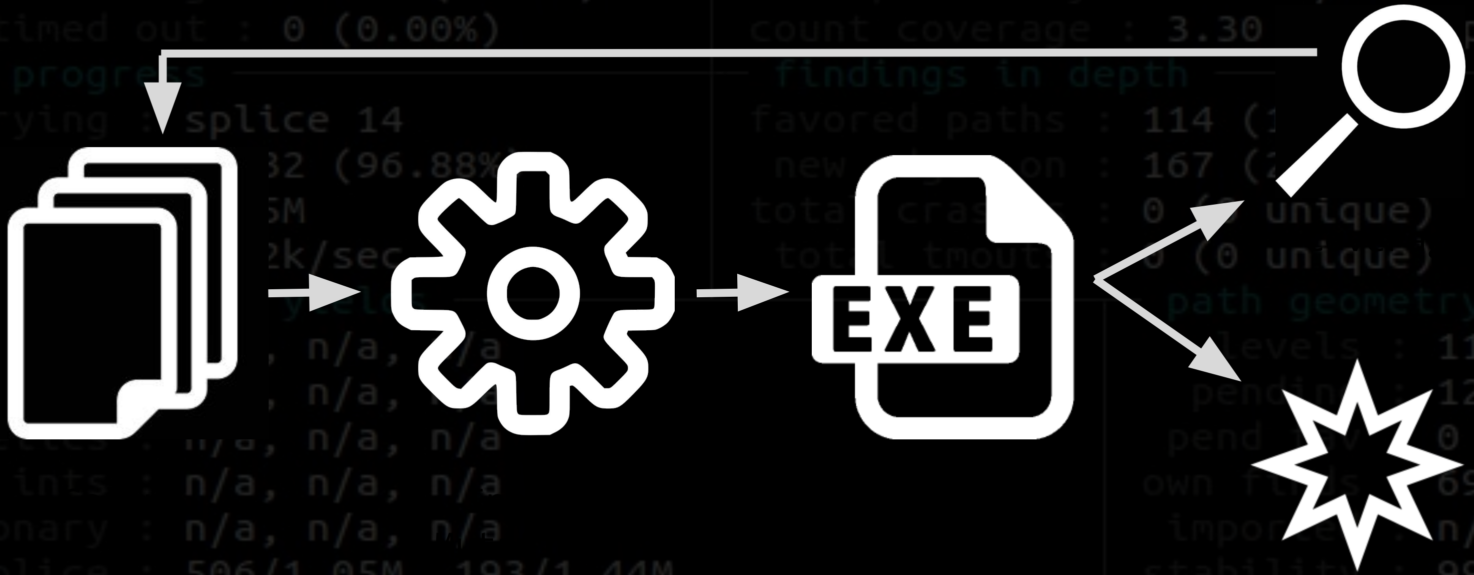
Machine-generated inputs can be of any kind, beyond the classic definition of “unexpected” (by the way, what does it means?) inputs.

Fuzzing is often considered related to Random Testing, a technique that provides inputs sampled uniform independently from the input space (using a specification maybe, so they are not random bytes in general).

But Fuzzing can generate inputs deterministically, or can generate inputs mutating previously generated inputs that makes the sampling from the input space not independent.

Widely discussed SOTA Fuzzing in 2022

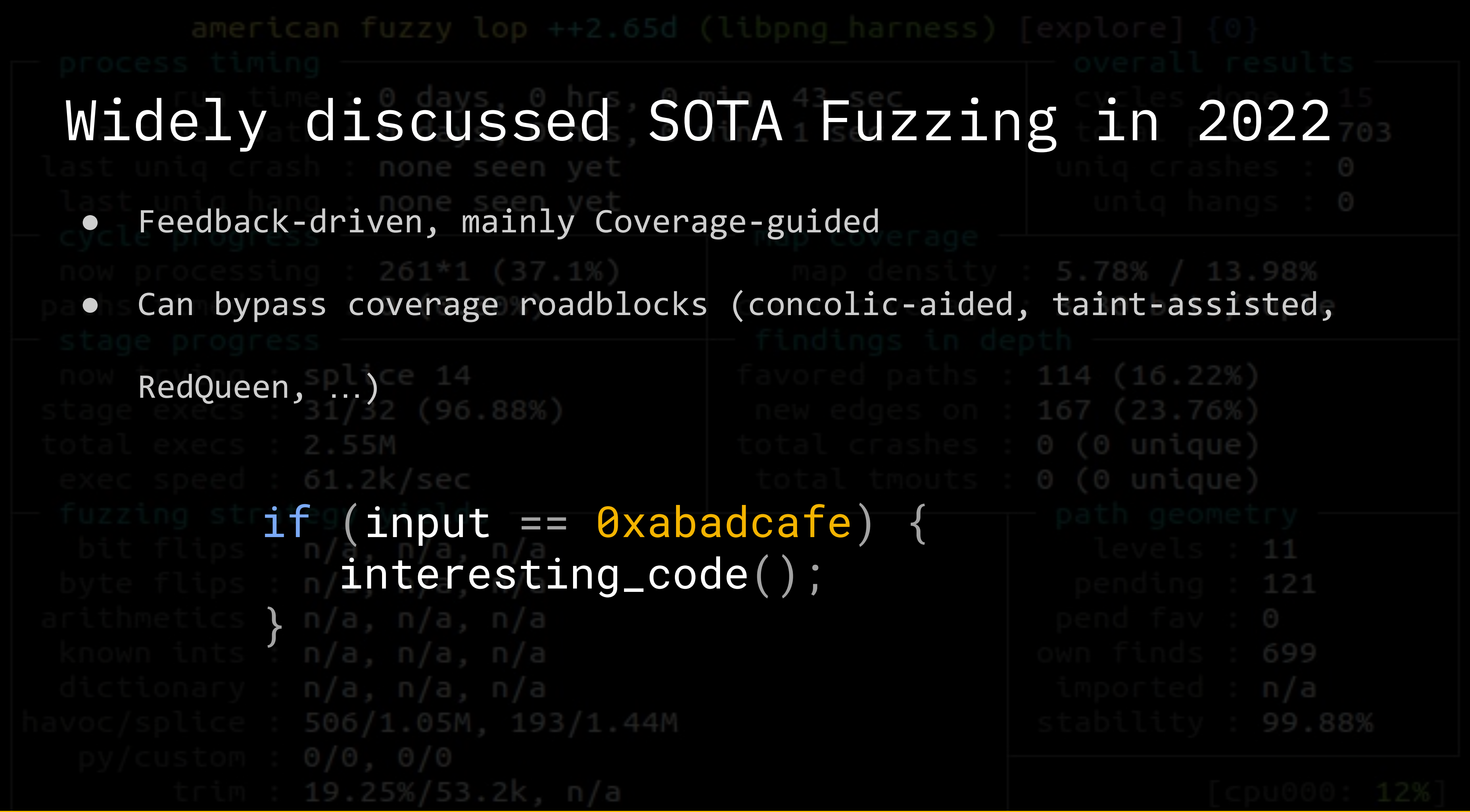
- Feedback-driven, mainly Coverage-guided



Widely discussed SOTA Fuzzing in 2022

- Feedback-driven, mainly Coverage-guided
- Can bypass coverage roadblocks (concolic-aided, taint-assisted, RedQueen, …)

```
if (input == 0xabadcafe) {
    interesting_code();
}
```



```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                      overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec        cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 1 sec         total paths : 703
last uniq crash : none seen yet                      uniq crashes : 0
 last uniq hang : none seen yet                        uniq hangs : 0
  cycle progress                      map coverage
  now processing : 261*1 (37.1%)        map density : 5.78% / 13.98%
  stage progress                      findings in depth
  now trying : splice 14              favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)          new edges on : 167 (23.76%)
 total execs : 2.55M                  total crashes : 0 (0 unique)
  exec speed : 61.2k/sec               total tmouts : 0 (0 unique)
  fuzzing strategy yields                            path geometry
   bit flips : n/a, n/a, n/a                           levels : 11
  byte flips : n/a, n/a, n/a                          pending : 121
 arithmetics : n/a, n/a, n/a                         pend fav : 0
  known ints : n/a, n/a, n/a                        own finds : 699
  dictionary : n/a, n/a, n/a                         imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                 stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                          [cpu000: 12%]
```

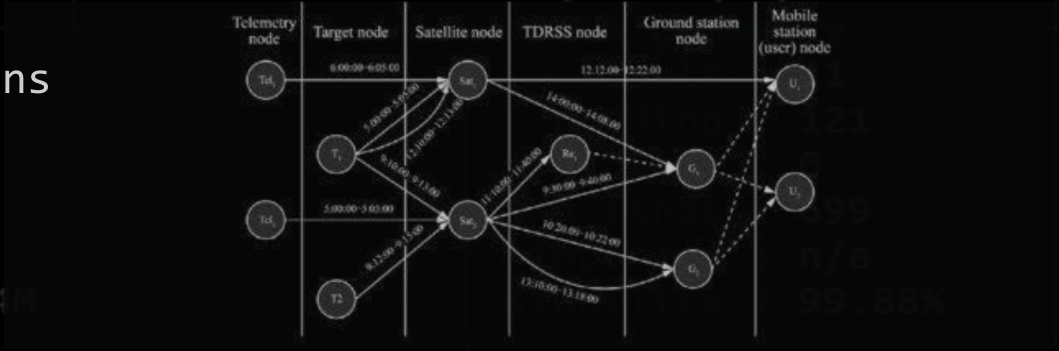
Widely discussed SOTA Fuzzing in 2022

- Feedback-driven, mainly Coverage-guided
- Can bypass coverage roadblocks (concolic-aided, taint-assisted, RedQueen, …)
- Input models help to fuzz deeper

```
<start>   ::= <expr>
<expr>    ::= <term> + <expr> | <term> - <expr> | <term>
<term>    ::= <term> * <factor> | <term> / <factor> | <factor>
<factor>  ::= +<factor> | -<factor> | (<expr>) | <integer> |
<integer>.<integer>
<integer> ::= <digit><integer> | <digit>
<digit>   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

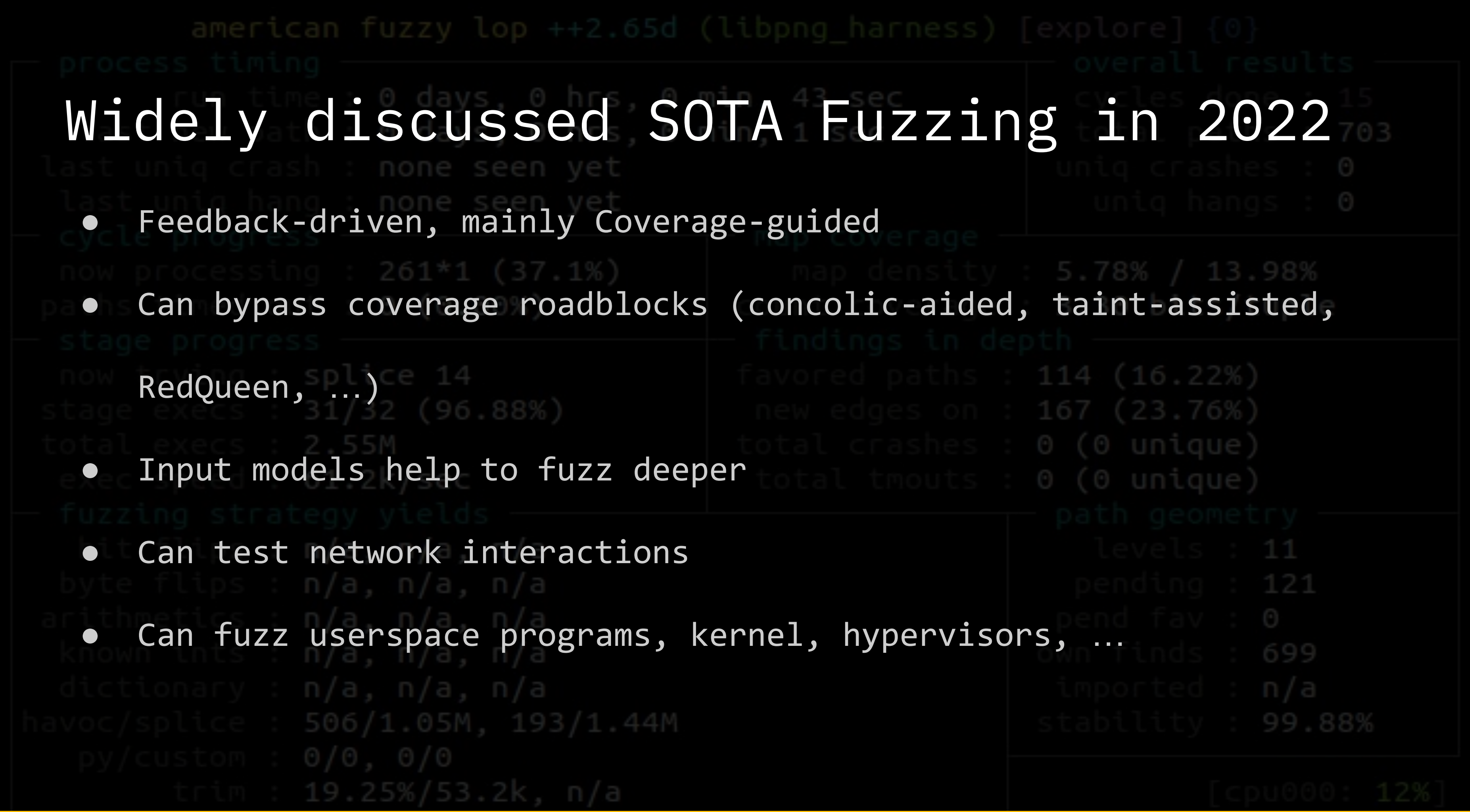
Widely discussed SOTA Fuzzing in 2022

- Feedback-driven, mainly Coverage-guided
- Can bypass coverage roadblocks (concolic-aided, taint-assisted, RedQueen, …)
- Input models help to fuzz deeper
- Can test network interactions



Widely discussed SOTA Fuzzing in 2022

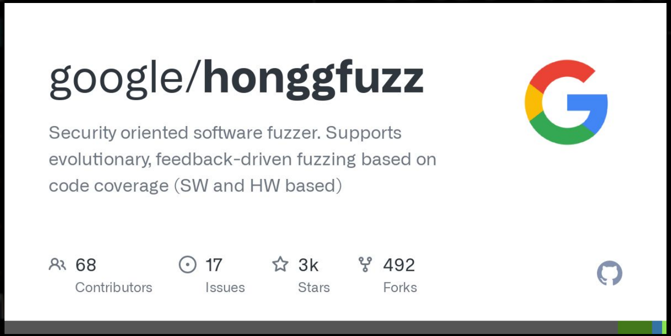
- Feedback-driven, mainly Coverage-guided
- Can bypass coverage roadblocks (concolic-aided, taint-assisted, RedQueen, …)
- Input models help to fuzz deeper
- Can test network interactions
- Can fuzz userspace programs, kernel, hypervisors, …



```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                          overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec            cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 1 sec             total paths : 703
last uniq crash : none seen yet                          uniq crashes : 0
 last uniq hang : none seen yet                            uniq hangs : 0
  cycle progress                          map coverage
  now processing : 261*1 (37.1%)            map density : 5.78% / 13.98%
  paths timed out : 0 (0.00%)
  stage progress                          findings in depth
  now trying : splice 14                 favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)             new edges on : 167 (23.76%)
 total execs : 2.55M                     total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                  total tmouts : 0 (0 unique)
  fuzzing strategy yields                                path geometry
   bit flips : n/a, n/a, n/a                               levels : 11
  byte flips : n/a, n/a, n/a                              pending : 121
 arithmetics : n/a, n/a, n/a                             pend fav : 0
  known ints : n/a, n/a, n/a                            own finds : 699
  dictionary : n/a, n/a, n/a                             imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                     stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                           [cpu000: 12%]
```

Widely used tools in 2022

LLVM’s Libfuzzer



google/honggfuzz

Security oriented software fuzzer. Supports evolutionary, feedback-driven fuzzing based on code coverage (SW and HW based)

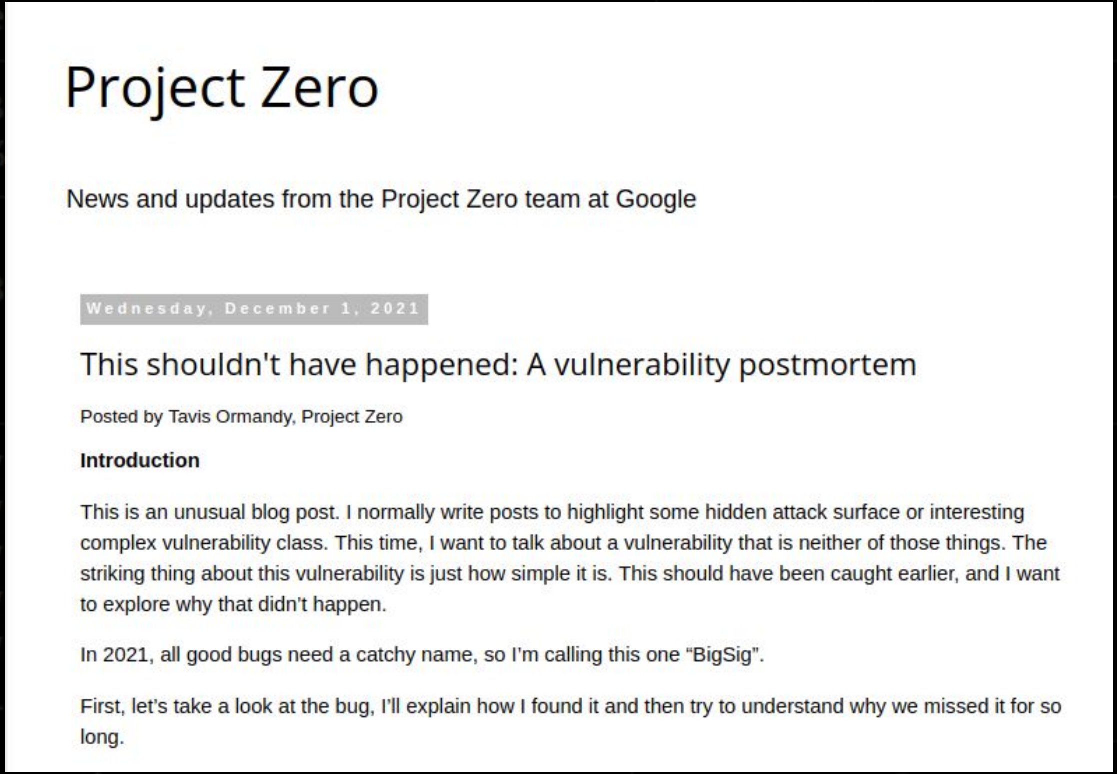
68 Contributors · 17 Issues · 3k Stars · 492 Forks

```
        american fuzzy lop ++2.65d (l
 process timing
       run time : 0 days, 0 hrs, 0 mi
  last new path : 0 days, 0 hrs, 0 mi
last uniq crash : none seen yet
 last uniq hang : none seen yet
 cycle progress
 now processing : 261*1 (37.1%)
paths timed out : 0 (0.00%)
 stage progress
 now trying : splice 14
stage execs : 31/32 (96.88%)
total execs : 2.55M
 exec speed : 61.2k/sec
 fuzzing strategy yields
  bit flips : n/a, n/a, n/a
 byte flips : n/a, n/a, n/a
arithmetics : n/a, n/a, n/a
 known ints : n/a, n/a, n/a
 dictionary : n/a, n/a, n/a
havoc/splice : 506/1.05M, 193/1.44M
  py/custom : 0/0, 0/0
       trim : 19.25%/53.2k, n/a
```

We still miss bugs

Yes, even in heavily-fuzzed projects in OSS-Fuzz

Still finding these bugs by hand…



Project Zero

News and updates from the Project Zero team at Google

Wednesday, December 1, 2021

This shouldn't have happened: A vulnerability postmortem

Posted by Tavis Ormandy, Project Zero

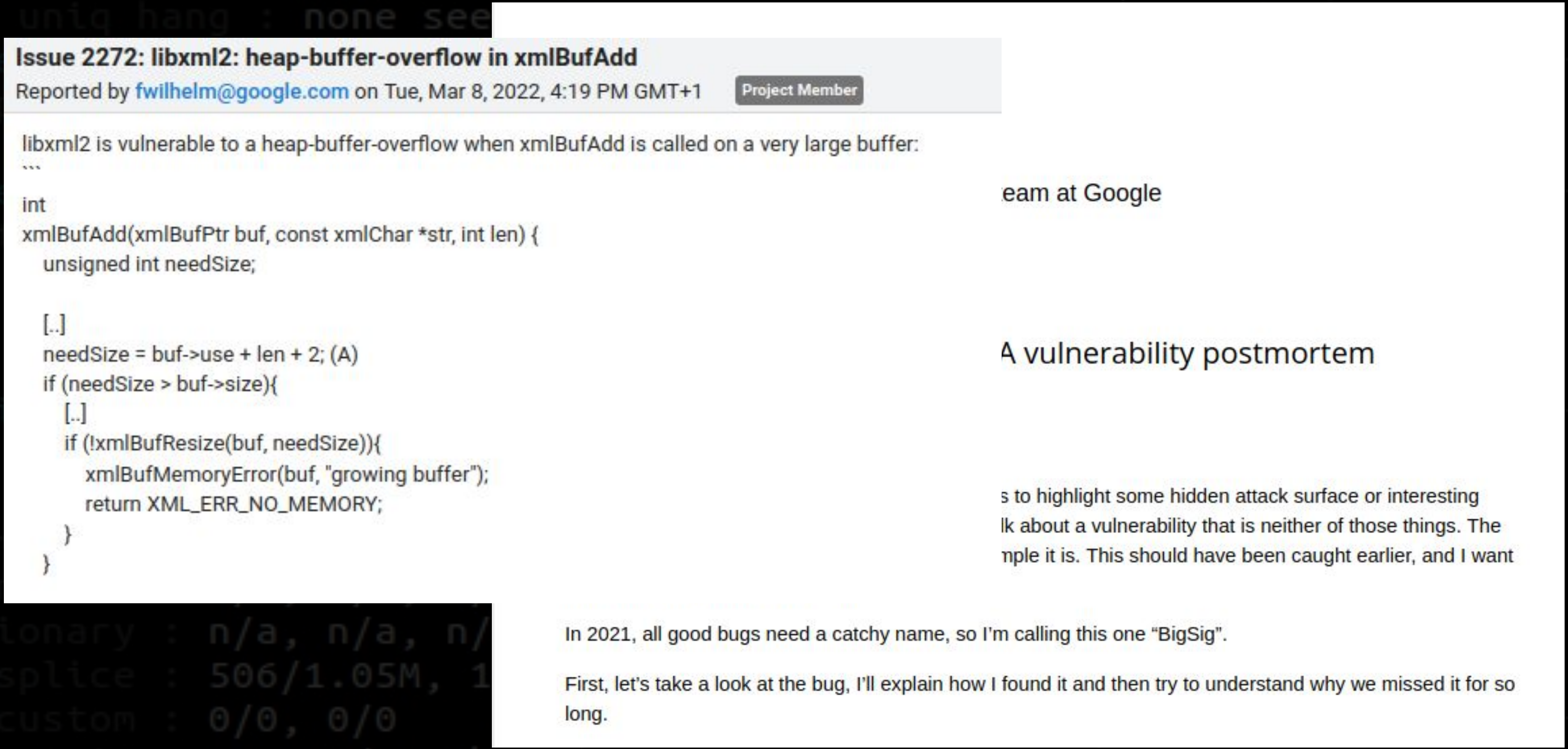
Introduction

This is an unusual blog post. I normally write posts to highlight some hidden attack surface or interesting complex vulnerability class. This time, I want to talk about a vulnerability that is neither of those things. The striking thing about this vulnerability is just how simple it is. This should have been caught earlier, and I want to explore why that didn't happen.

In 2021, all good bugs need a catchy name, so I'm calling this one "BigSig".

First, let's take a look at the bug, I'll explain how I found it and then try to understand why we missed it for so long.

Still finding these bugs by hand…



Issue 2272: libxml2: heap-buffer-overflow in xmlBufAdd

Reported by fwilhelm@google.com on Tue, Mar 8, 2022, 4:19 PM GMT+1

libxml2 is vulnerable to a heap-buffer-overflow when xmlBufAdd is called on a very large buffer:

```
int
xmlBufAdd(xmlBufPtr buf, const xmlChar *str, int len) {
  unsigned int needSize;

  [..]
  needSize = buf->use + len + 2; (A)
  if (needSize > buf->size){
    [..]
    if (!xmlBufResize(buf, needSize)){
      xmlBufMemoryError(buf, "growing buffer");
      return XML_ERR_NO_MEMORY;
    }
  }
```

eam at Google

A vulnerability postmortem

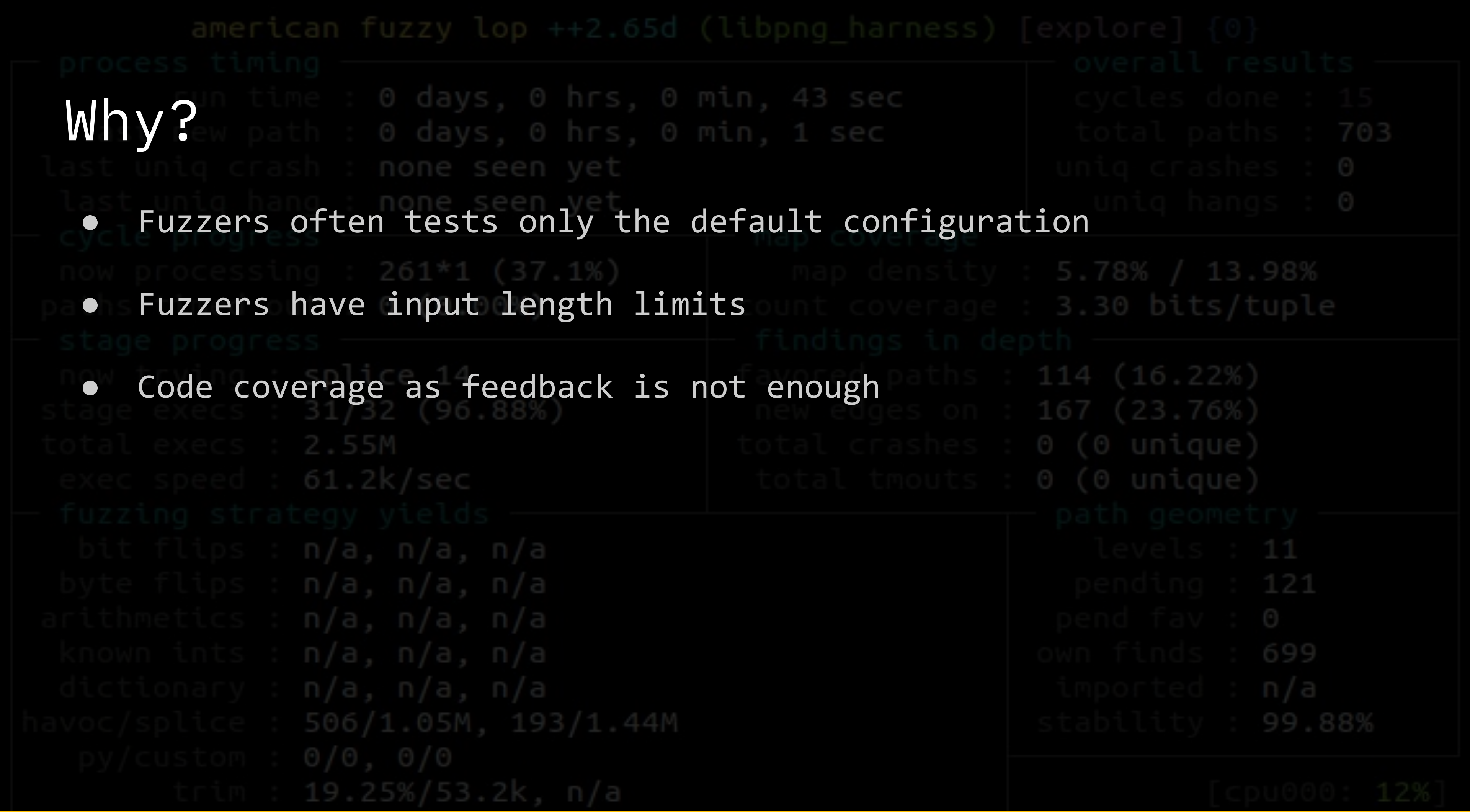
s to highlight some hidden attack surface or interesting
lk about a vulnerability that is neither of those things. The
nple it is. This should have been caught earlier, and I want

In 2021, all good bugs need a catchy name, so I'm calling this one "BigSig".

First, let's take a look at the bug, I'll explain how I found it and then try to understand why we missed it for so long.

Why?

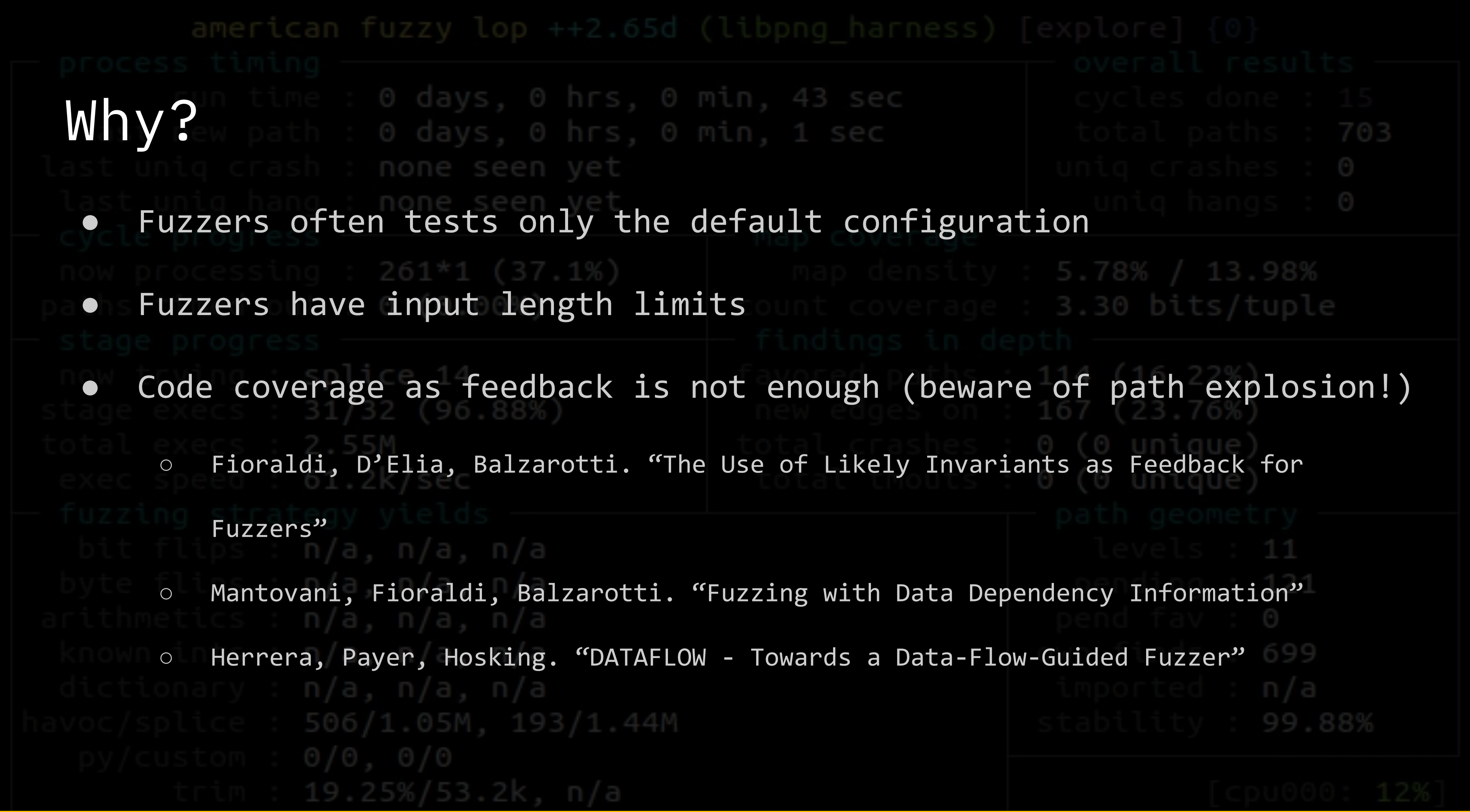
- Fuzzers often tests only the default configuration
- Fuzzers have input length limits
- Code coverage as feedback is not enough



```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
   process timing                                        overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
   last new path : 0 days, 0 hrs, 0 min, 1 sec           total paths : 703
 last uniq crash : none seen yet                        uniq crashes : 0
  last uniq hang : none seen yet                          uniq hangs : 0
   cycle progress                       map coverage
  now processing : 261*1 (37.1%)          map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)           count coverage : 3.30 bits/tuple
   stage progress                       findings in depth
  now trying : splice 14              favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)          new edges on : 167 (23.76%)
 total execs : 2.55M                  total crashes : 0 (0 unique)
  exec speed : 61.2k/sec               total tmouts : 0 (0 unique)
   fuzzing strategy yields                               path geometry
   bit flips : n/a, n/a, n/a                               levels : 11
  byte flips : n/a, n/a, n/a                              pending : 121
 arithmetics : n/a, n/a, n/a                             pend fav : 0
  known ints : n/a, n/a, n/a                            own finds : 699
  dictionary : n/a, n/a, n/a                             imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                     stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                                [cpu000: 12%]
```

Why?

- Fuzzers often tests only the default configuration
- Fuzzers have input length limits
- Code coverage as feedback is not enough (beware of path explosion!)
  - ○ Fioraldi, D’Elia, Balzarotti. “The Use of Likely Invariants as Feedback for Fuzzers”
  - ○ Mantovani, Fioraldi, Balzarotti. “Fuzzing with Data Dependency Information”
  - ○ Herrera, Payer, Hosking. “DATAFLOW - Towards a Data-Flow-Guided Fuzzer”



```
      american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                   overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec     cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 1 sec      total paths : 703
last uniq crash : none seen yet                   uniq crashes : 0
 last uniq hang : none seen yet                     uniq hangs : 0
  cycle progress                     map coverage
  now processing : 261*1 (37.1%)       map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)        count coverage : 3.30 bits/tuple
  stage progress                     findings in depth
  now trying : splice 14            favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)        new edges on : 167 (23.76%)
 total execs : 2.55M                total crashes : 0 (0 unique)
  exec speed : 61.2k/sec             total tmouts : 0 (0 unique)
  fuzzing strategy yields                           path geometry
   bit flips : n/a, n/a, n/a                         levels : 11
  byte flips : n/a, n/a, n/a                        pending : 121
 arithmetics : n/a, n/a, n/a                       pend fav : 0
  known ints : n/a, n/a, n/a                      own finds : 699
  dictionary : n/a, n/a, n/a                       imported : n/a
havoc/splice : 506/1.05M, 193/1.44M               stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                          [cpu000: 12%]
```
]

An Example

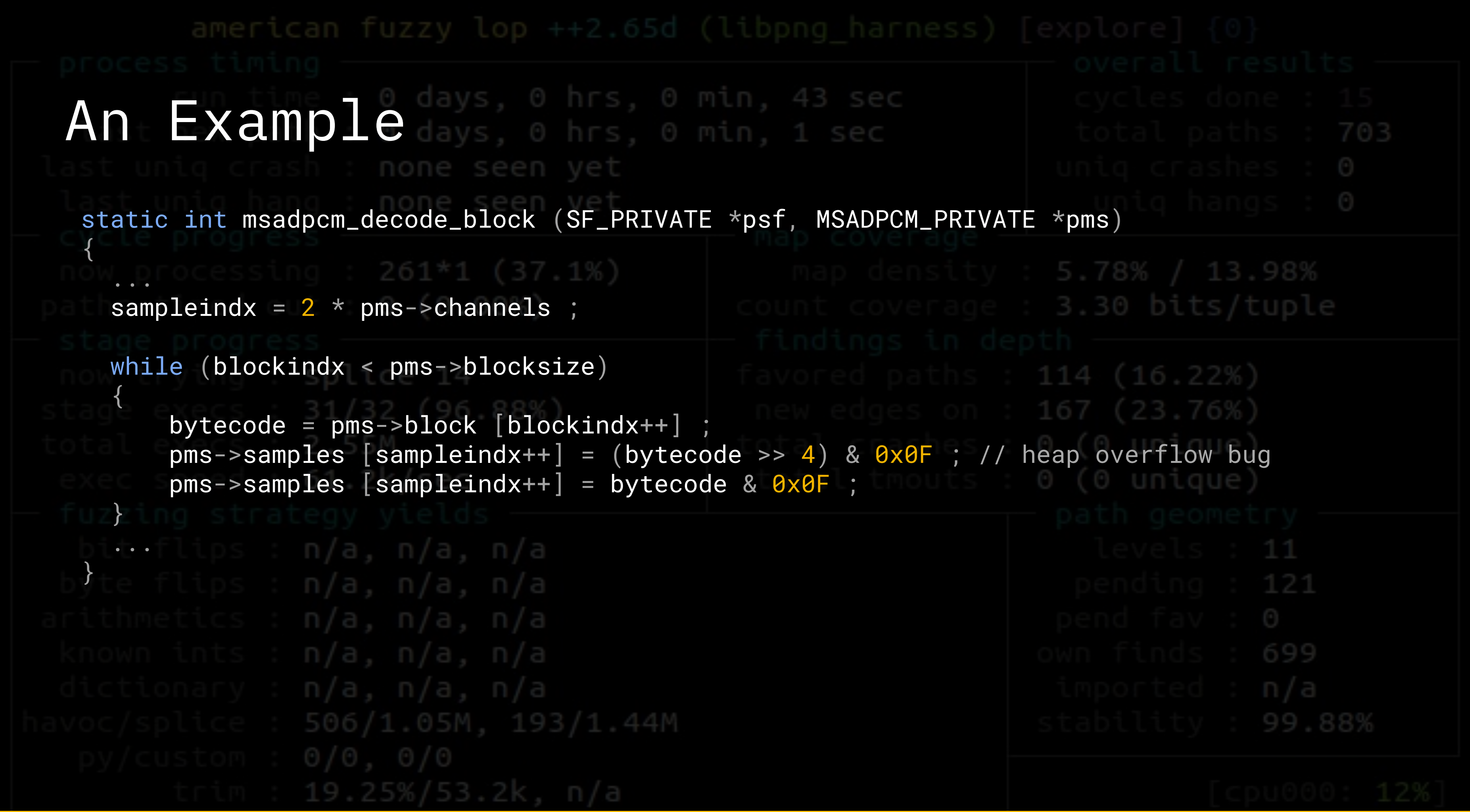
```
int wavlike_msadpcm_init (SF_PRIVATE *psf, int blockalign, int samplesperblock)
{
  MSADPCM_PRIVATE  *pms ;
  unsigned int pmssize ;
  ...
  pmssize = sizeof (MSADPCM_PRIVATE) + blockalign + 3 * psf->sf.channels * samplesperblock
;
  ...
  pms->samples   = pms->dummydata ; // array in pms
  pms->block     = (unsigned char*) (pms->dummydata + psf->sf.channels * samplesperblock) ;
  pms->channels   = psf->sf.channels ;
  pms->blocksize  = blockalign ;
  ...
}
```

An Example

```
int wavlike_msadpcm_init (SF_PRIVATE *psf, int blockalign, int samplesperblock)
{
   MSADPCM_PRIVATE  *pms ;
   unsigned int pmssize ;
   ...
   pmssize = sizeof (MSADPCM_PRIVATE) + blockalign + 3 * psf->sf.channels * samplesperblock
;
   ...
   pms->samples   = pms->dummydata ; // array in pms
   pms->block     = (unsigned char*) (pms->dummydata + psf->sf.channels * samplesperblock) ;
   pms->channels   = psf->sf.channels ;
   pms->blocksize  = blockalign ;
   ...
}
```

An Example

```
static int msadpcm_decode_block (SF_PRIVATE *psf, MSADPCM_PRIVATE *pms)
{
  ...
  sampleindx = 2 * pms->channels ;

  while (blockindx < pms->blocksize)
  {
      bytecode = pms->block [blockindx++] ;
      pms->samples [sampleindx++] = (bytecode >> 4) & 0x0F ; // heap overflow bug
      pms->samples [sampleindx++] = bytecode & 0x0F ;
  }
  ...
}
```



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing                                      overall results
      run time : 0 days, 0 hrs, 0 min, 43 sec       cycles done : 15
               : 0 days, 0 hrs, 0 min, 1 sec        total paths : 703
last uniq crash : none seen yet                    uniq crashes : 0
last uniq hang : none seen yet                       uniq hangs : 0
cycle progress                          map coverage
now processing : 261*1 (37.1%)             map density : 5.78% / 13.98%
                                        count coverage : 3.30 bits/tuple
stage progress                          findings in depth
                                        favored paths : 114 (16.22%)
stage execs : 31/32 (96.88%)            new edges on : 167 (23.76%)
                                                       0 (0 unique)
                                                       0 (0 unique)
fuzzing strategy yields                                path geometry
  bit flips : n/a, n/a, n/a                              levels : 11
 byte flips : n/a, n/a, n/a                             pending : 121
arithmetics : n/a, n/a, n/a                            pend fav : 0
 known ints : n/a, n/a, n/a                           own finds : 699
 dictionary : n/a, n/a, n/a                            imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                   stability : 99.88%
  py/custom : 0/0, 0/0
       trim : 19.25%/53.2k, n/a                           [cpu000: 12%]
```

An Example

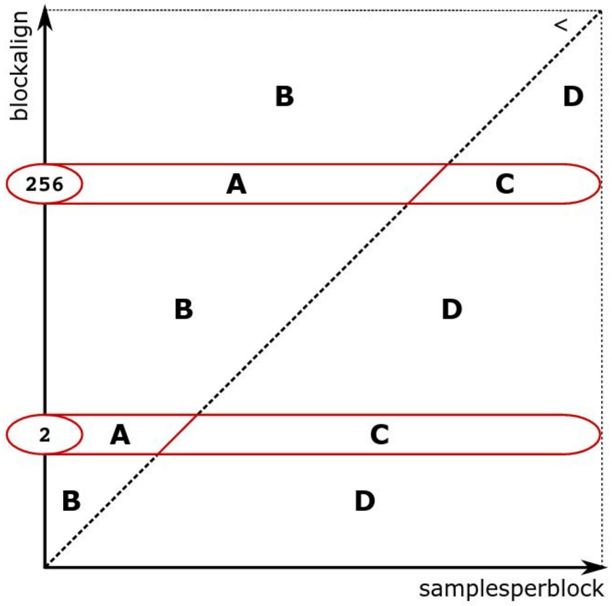
```
static int msadpcm_decode_block (SF_PRIVATE *psf, MSADPCM_PRIVATE *pms)
{
  ...
  sampleindx = 2 * pms->channels ;

  while (blockindx < pms->blocksize)
  {
      bytecode = pms->block [blockindx++] ;
      pms->samples [sampleindx++] = (bytecode >> 4) & 0x0F ; // heap overflow bug
      pms->samples [sampleindx++] = bytecode & 0x0F ;
  }
  ...
}
```

This only happens when the program is in a specific state, characterized by a small allocation size for the pms buffer and a pms->blocksize value sufficiently high to force the loop to write out of the bounds of the array.

However, none of these requirements can be extracted from code coverage, as there are no branches in the program that involve these thresholds

An Example

```
pmssize = sizeof (MSADPCM_PRIVATE) + blockalign + 3 * psf->sf.channels * samplesperblock
```



Invariant	Condition
LI_1	blockalign $\in \{0,2,256\}$
LI_2	blockalign $<$ samplesperblock

Invariant	A	B	C	D
LI_1	✓	✗	✓	✗
LI_2	✗	✗	✓	✓

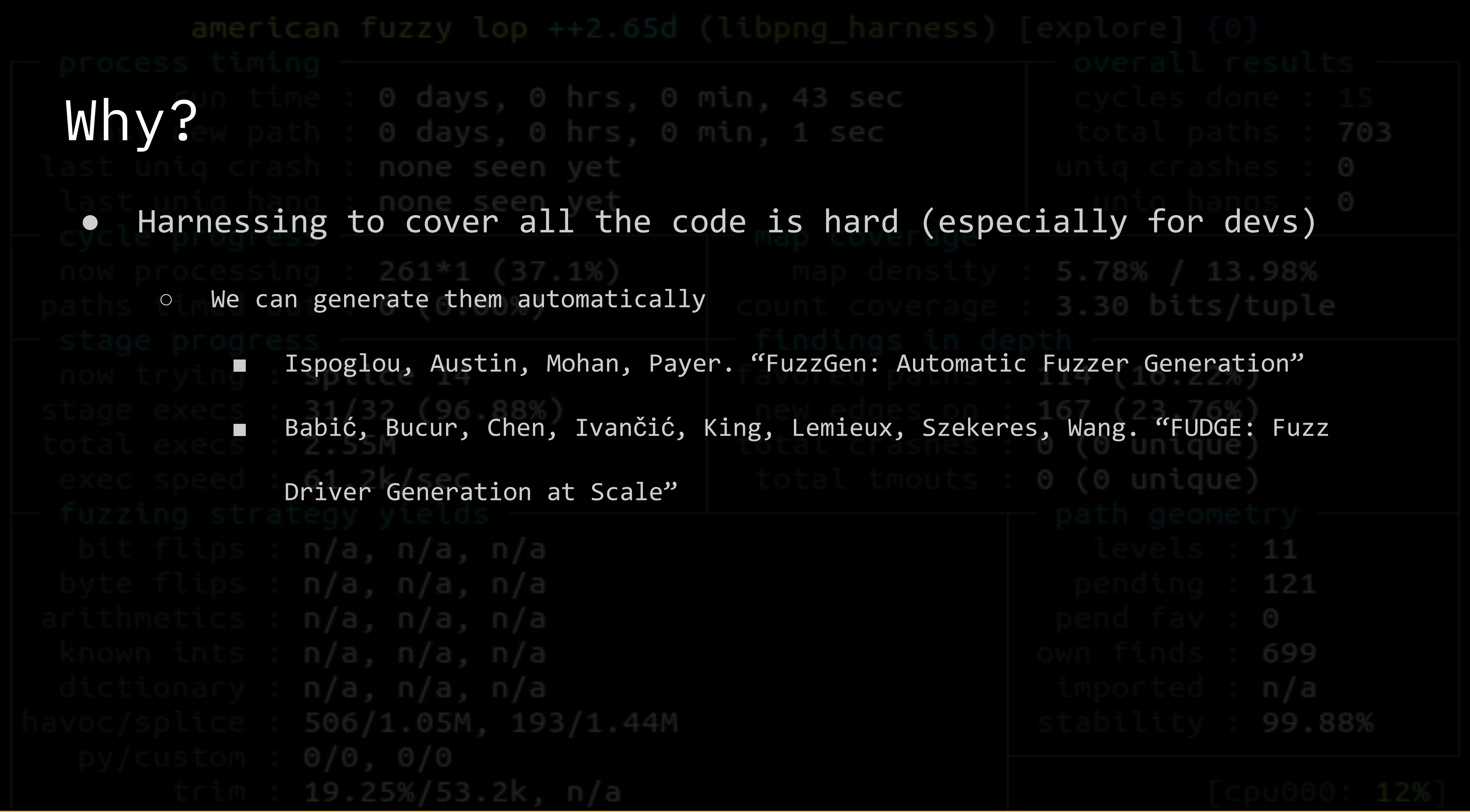
Why?

- Fuzzers often tests only the default configuration
- Fuzzers have input length limits
- Code coverage as feedback is not enough
- Harnessing to cover all the code is hard (especially for devs)

```
         american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                         overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 1 sec           total paths : 703
 last uniq crash : none seen yet                       uniq crashes : 0
  last uniq hang : none seen yet                         uniq hangs : 0
  cycle progress                        map coverage
  now processing : 261*1 (37.1%)           map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)            count coverage : 3.30 bits/tuple
  stage progress                        findings in depth
  now trying : splice 14                favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)           new edges on : 167 (23.76%)
 total execs : 2.55M                   total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                total tmouts : 0 (0 unique)
  fuzzing strategy yields                              path geometry
   bit flips : n/a, n/a, n/a                             levels : 11
  byte flips : n/a, n/a, n/a                            pending : 121
 arithmetics : n/a, n/a, n/a                           pend fav : 0
  known ints : n/a, n/a, n/a                          own finds : 699
  dictionary : n/a, n/a, n/a                           imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                   stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                         [cpu000: 12%]
```

Why?

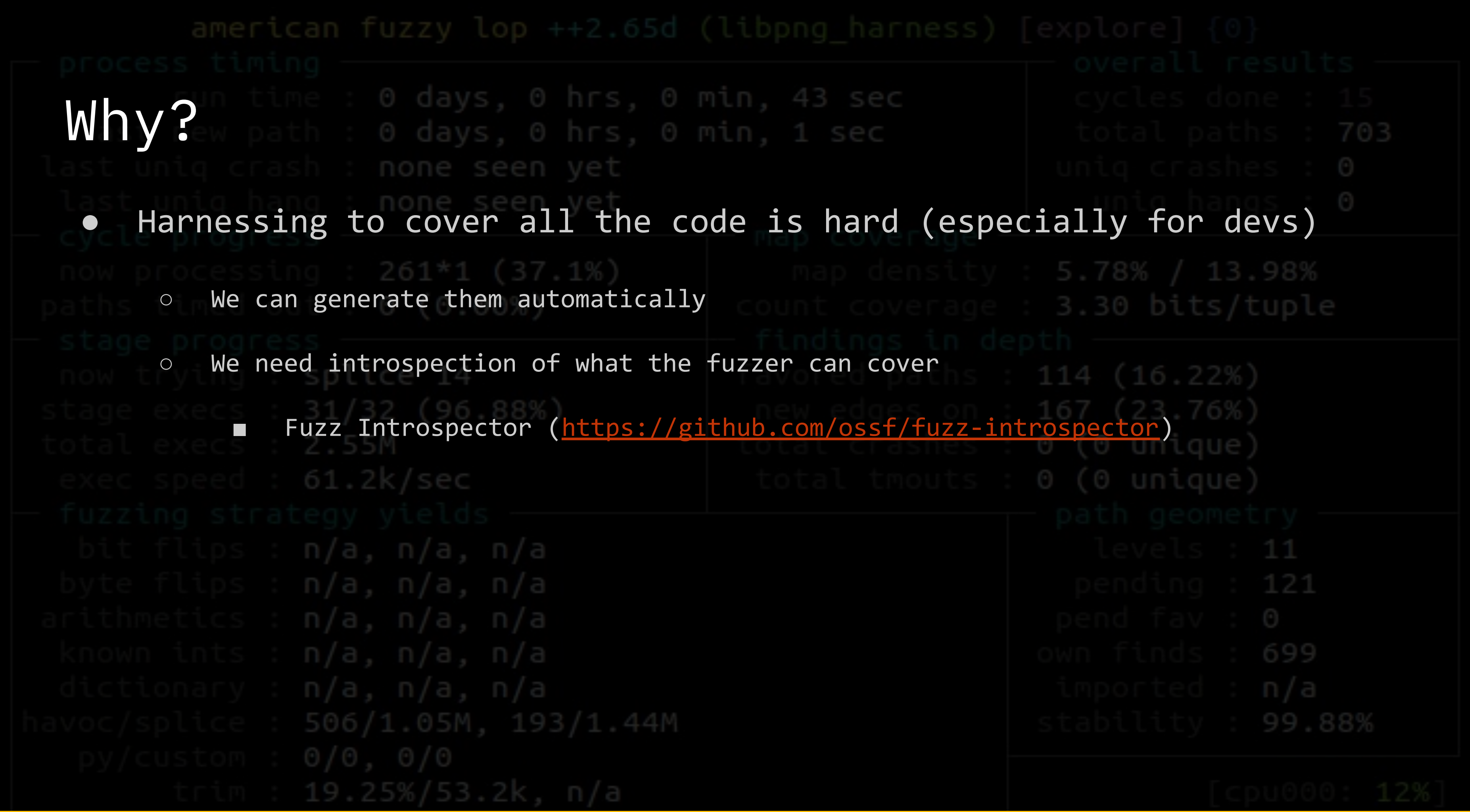
- Harnessing to cover all the code is hard (especially for devs)
  - We can generate them automatically
    - Ispoglou, Austin, Mohan, Payer. “FuzzGen: Automatic Fuzzer Generation”
    - Babić, Bucur, Chen, Ivančić, King, Lemieux, Szekeres, Wang. “FUDGE: Fuzz Driver Generation at Scale”



```
         american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                        overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec         cycles done : 15
   last new path : 0 days, 0 hrs, 0 min, 1 sec          total paths : 703
 last uniq crash : none seen yet                       uniq crashes : 0
  last uniq hang : none seen yet                         uniq hangs : 0
  cycle progress                       map coverage
  now processing : 261*1 (37.1%)          map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)           count coverage : 3.30 bits/tuple
  stage progress                       findings in depth
  now trying : splice 14               favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)           new edges on : 167 (23.76%)
 total execs : 2.55M                   total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                total tmouts : 0 (0 unique)
  fuzzing strategy yields                               path geometry
   bit flips : n/a, n/a, n/a                              levels : 11
  byte flips : n/a, n/a, n/a                             pending : 121
 arithmetics : n/a, n/a, n/a                            pend fav : 0
  known ints : n/a, n/a, n/a                           own finds : 699
  dictionary : n/a, n/a, n/a                            imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                    stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                              [cpu000: 12%]
```

Why?

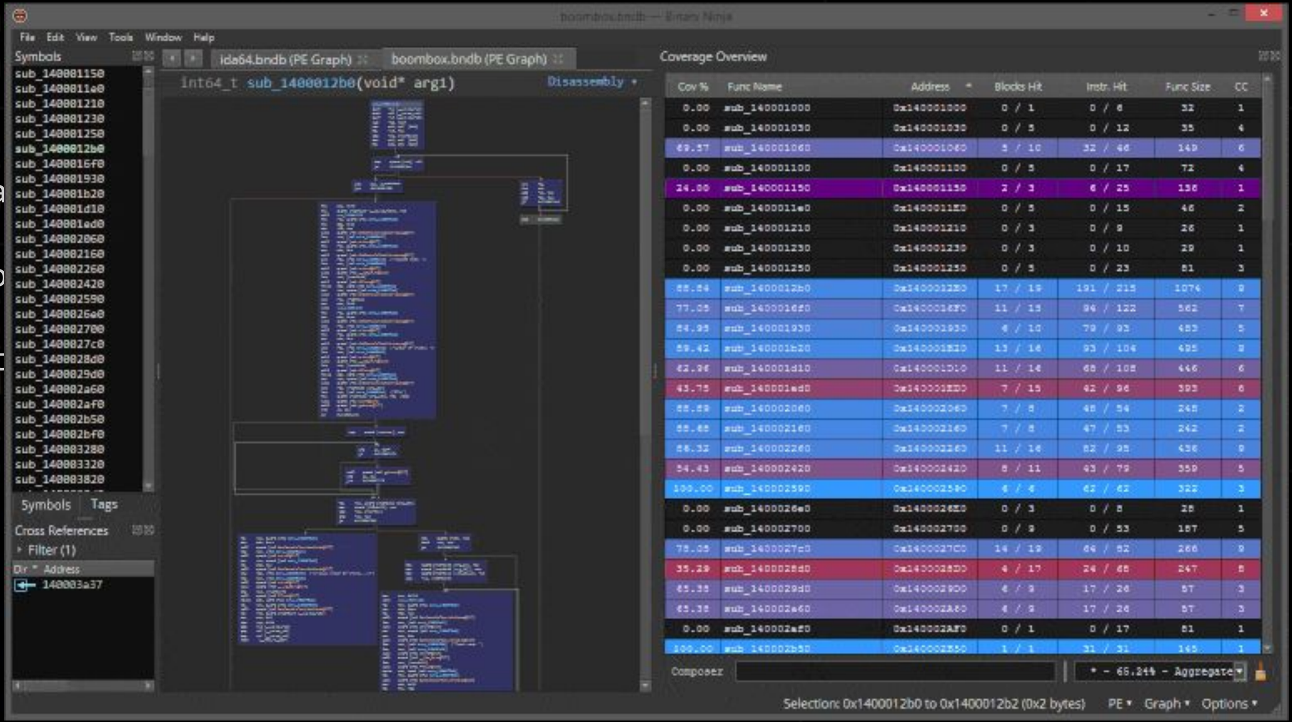
- Harnessing to cover all the code is hard (especially for devs)
  - We can generate them automatically
  - We need introspection of what the fuzzer can cover
    - Fuzz Introspector (https://github.com/ossf/fuzz-introspector)



```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                         overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
   last new path : 0 days, 0 hrs, 0 min, 1 sec          total paths : 703
 last uniq crash : none seen yet                       uniq crashes : 0
  last uniq hang : none seen yet                         uniq hangs : 0
  cycle progress                        map coverage
  now processing : 261*1 (37.1%)          map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)           count coverage : 3.30 bits/tuple
  stage progress                        findings in depth
  now trying : splice 14               favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)           new edges on : 167 (23.76%)
 total execs : 2.55M                   total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                total tmouts : 0 (0 unique)
  fuzzing strategy yields                                path geometry
   bit flips : n/a, n/a, n/a                              levels : 11
  byte flips : n/a, n/a, n/a                             pending : 121
 arithmetics : n/a, n/a, n/a                            pend fav : 0
  known ints : n/a, n/a, n/a                           own finds : 699
  dictionary : n/a, n/a, n/a                            imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                    stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                         [cpu000: 12%]
```

Why?

- Harnessing to
    - We can genera
    - We need intro
        - Fuzz Int



```
int64_t sub_1400012b0(void* arg1)
```

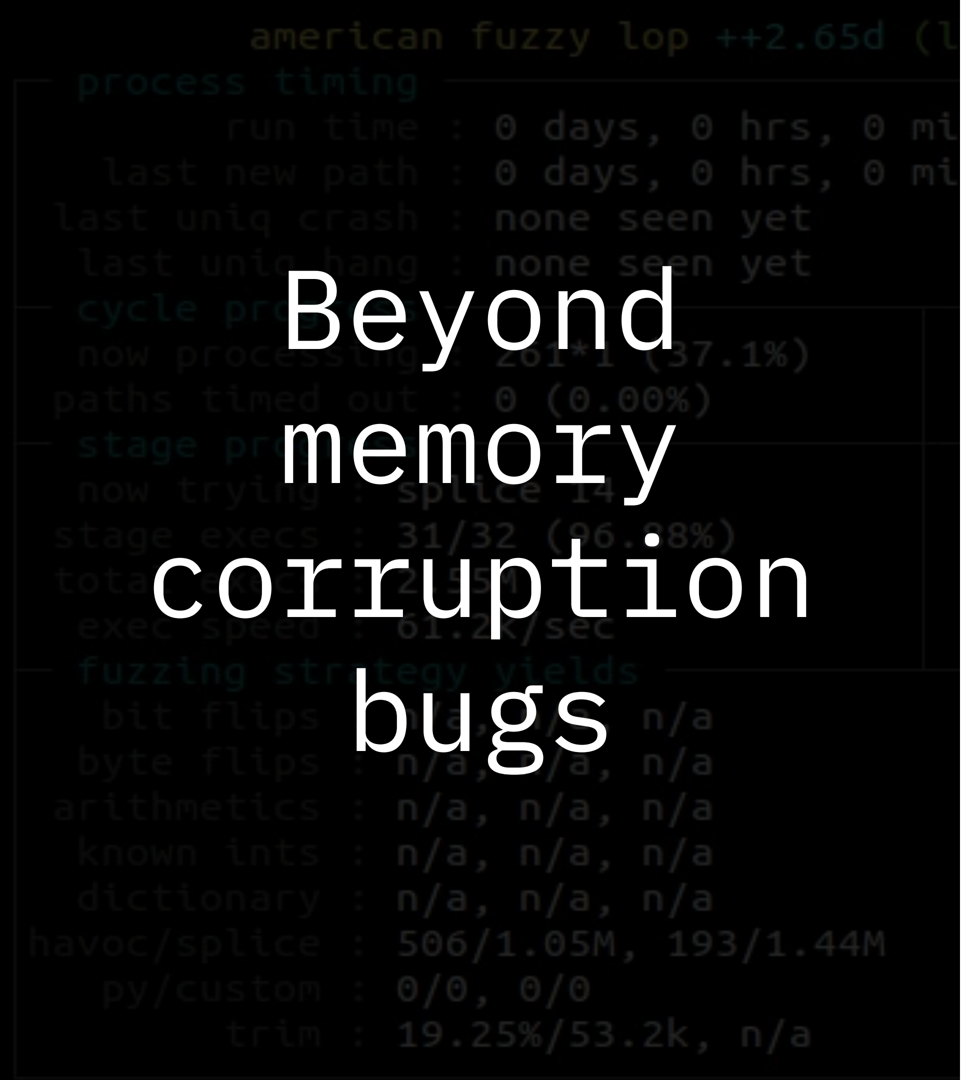
Coverage Overview

Cov %	Func Name	Address	Blocks Hit	Instr. Hit	Func Size	CC
0.00	sub_140001000	0x140001000	0 / 1	0 / 6	32	1
0.00	sub_140001030	0x140001030	0 / 5	0 / 12	35	4
69.57	sub_140001060	0x140001060	5 / 10	32 / 46	149	6
0.00	sub_140001100	0x140001100	0 / 5	0 / 17	72	4
24.00	sub_140001150	0x140001150	2 / 3	6 / 25	136	1
0.00	sub_1400011e0	0x1400011E0	0 / 5	0 / 15	46	2
0.00	sub_140001210	0x140001210	0 / 3	0 / 9	26	1
0.00	sub_140001230	0x140001230	0 / 3	0 / 10	29	1
0.00	sub_140001250	0x140001250	0 / 5	0 / 23	81	3
88.84	sub_1400012b0	0x1400012B0	17 / 19	191 / 215	1076	9
77.05	sub_1400016f0	0x1400016F0	11 / 15	94 / 122	562	7
84.95	sub_140001930	0x140001930	6 / 10	79 / 93	483	5
89.42	sub_140001b20	0x140001B20	13 / 16	93 / 104	495	8
62.96	sub_140001d10	0x140001D10	11 / 16	68 / 108	446	6
43.75	sub_140001ed0	0x140001ED0	7 / 15	42 / 96	393	6
88.89	sub_140002060	0x140002060	7 / 8	48 / 54	248	2
88.68	sub_140002160	0x140002160	7 / 8	47 / 53	242	2
86.32	sub_140002260	0x140002260	11 / 16	82 / 95	436	9
54.43	sub_140002420	0x140002420	8 / 11	43 / 79	359	5
100.00	sub_140002590	0x140002590	6 / 6	62 / 62	322	3
0.00	sub_1400026e0	0x1400026E0	0 / 3	0 / 8	28	1
0.00	sub_140002700	0x140002700	0 / 9	0 / 53	187	5
78.05	sub_1400027c0	0x1400027C0	14 / 19	64 / 82	266	9
35.29	sub_1400028d0	0x1400028D0	4 / 17	24 / 68	247	8
65.38	sub_1400029d0	0x1400029D0	6 / 9	17 / 26	87	3
65.38	sub_140002a60	0x140002A60	6 / 9	17 / 26	87	3
0.00	sub_140002af0	0x140002AF0	0 / 1	0 / 17	81	1
100.00	sub_140002b50	0x140002B50	1 / 1	31 / 31	165	1

* - 65.24% - Aggregate

Selection: 0x1400012b0 to 0x1400012b2 (0x2 bytes)

Beyond memory corruption bugs



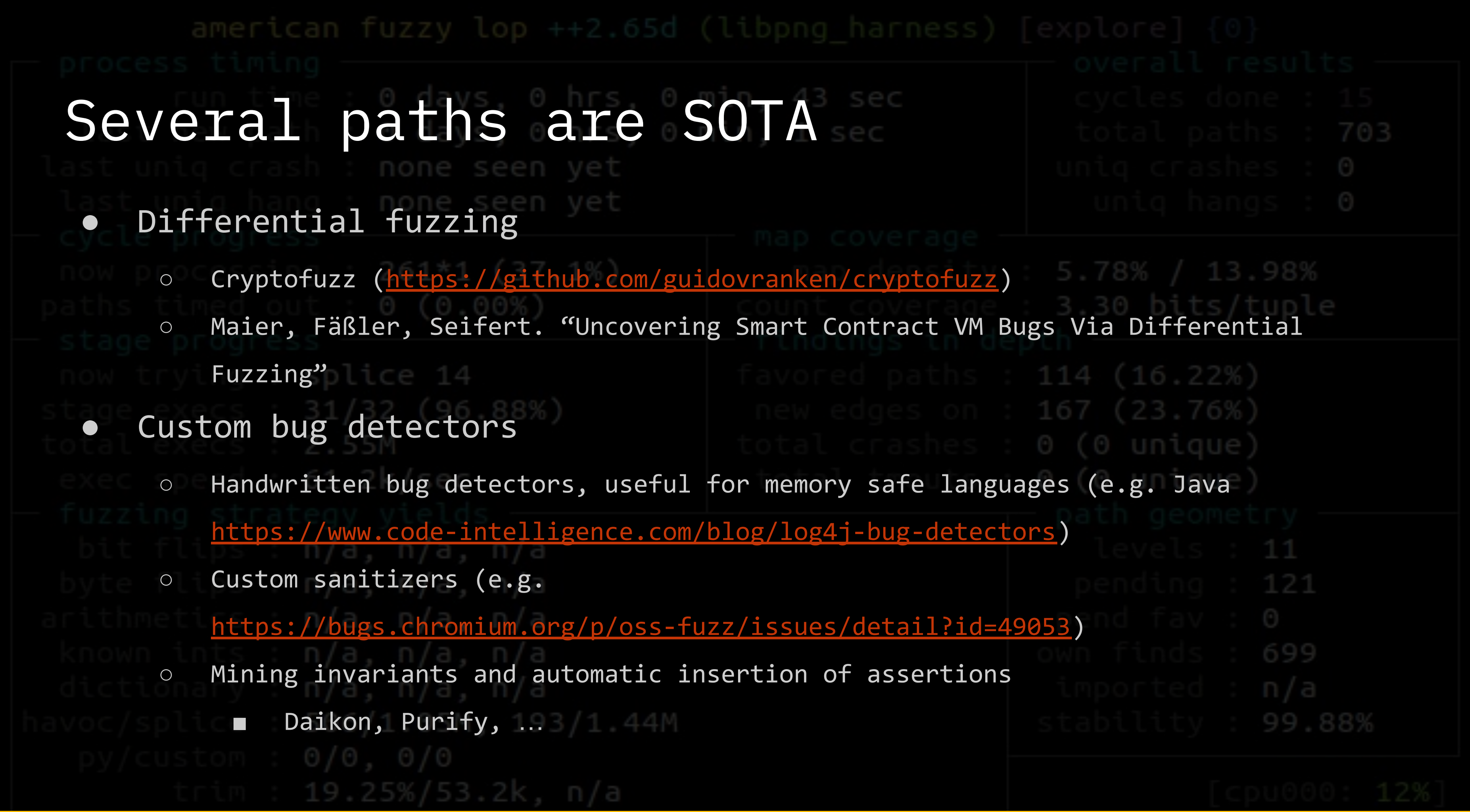
```
        american fuzzy lop ++2.65d (l
  process timing
        run time : 0 days, 0 hrs, 0 mi
   last new path : 0 days, 0 hrs, 0 mi
 last uniq crash : none seen yet
  last uniq hang : none seen yet
  cycle progress
  now processing : 261*1 (37.1%)
 paths timed out : 0 (0.00%)
  stage progress
  now trying : splice 14
 stage execs : 31/32 (96.88%)
 total execs : 2.55M
  exec speed : 61.2k/sec
  fuzzing strategy yields
   bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
 arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
havoc/splice : 506/1.05M, 193/1.44M
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a
```

A SQL injection is not causing a segfault in your application

Several paths are SOTA

- Differential fuzzing
  - Cryptofuzz (https://github.com/guidovranken/cryptofuzz)
  - Maier, Fäßler, Seifert. “Uncovering Smart Contract VM Bugs Via Differential Fuzzing”

```
      american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                   overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec        cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 0 sec        total paths : 703
 last uniq crash : none seen yet                     uniq crashes : 0
  last uniq hang : none seen yet                       uniq hangs : 0
  cycle progress                        map coverage
  now processing : 261*1 (37.1%)          map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)           count coverage : 3.30 bits/tuple
  stage progress                        findings in depth
  now trying : splice 14               favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)           new edges on : 167 (23.76%)
 total execs : 2.55M                   total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                total tmouts : 0 (0 unique)
  fuzzing strategy yields                              path geometry
   bit flips : n/a, n/a, n/a                            levels : 11
  byte flips : n/a, n/a, n/a                           pending : 121
 arithmetics : n/a, n/a, n/a                          pend fav : 0
  known ints : n/a, n/a, n/a                         own finds : 699
  dictionary : n/a, n/a, n/a                          imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                  stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                              [cpu000: 12%]
```

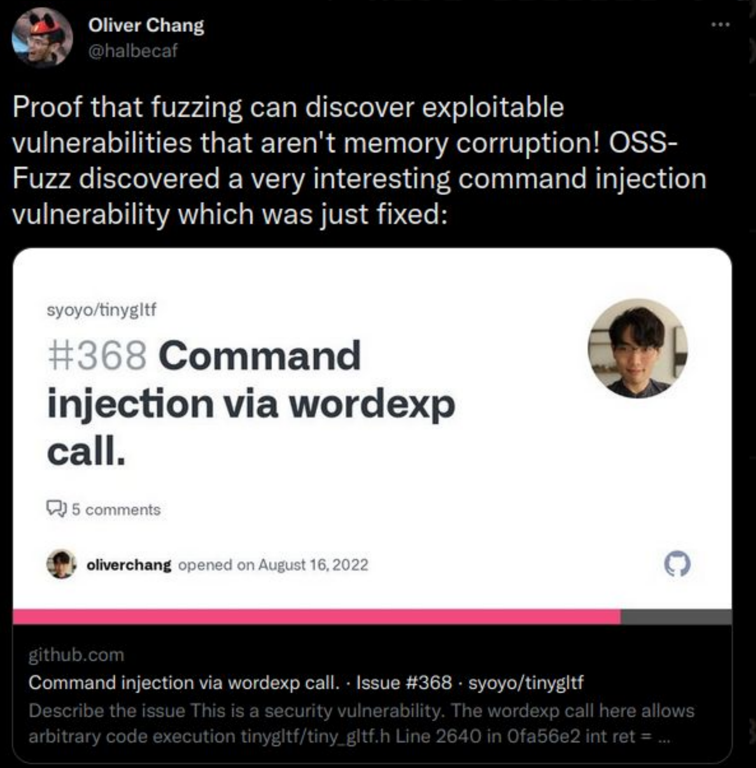


Several paths are SOTA

- Differential fuzzing
  - Cryptofuzz (https://github.com/guidovranken/cryptofuzz)
  - Maier, Fäßler, Seifert. “Uncovering Smart Contract VM Bugs Via Differential Fuzzing”
- Custom bug detectors
  - Handwritten bug detectors, useful for memory safe languages (e.g. Java https://www.code-intelligence.com/blog/log4j-bug-detectors)
  - Custom sanitizers (e.g. https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=49053)
  - Mining invariants and automatic insertion of assertions
    - Daikon, Purify, …

Several paths are SOTA

- Differential fuzzing
  - Cryptofuzz (https://github.com/guidovr
  - Maier, Fäßler, Seifert. “Uncovering Sma Fuzzing”
- Custom bug detectors
  - Handwritten bug detectors, useful for r https://www.code-intelligence.com/blog/
  - Custom sanitizers (e.g. https://bugs.chromium.org/p/oss-fuzz/is
  - Mining invariants and automatic insert
    - Daikon, Purify, …



Oliver Chang
@halbecaf

Proof that fuzzing can discover exploitable vulnerabilities that aren't memory corruption! OSS-Fuzz discovered a very interesting command injection vulnerability which was just fixed:

syoyo/tinygltf

#368 Command injection via wordexp call.

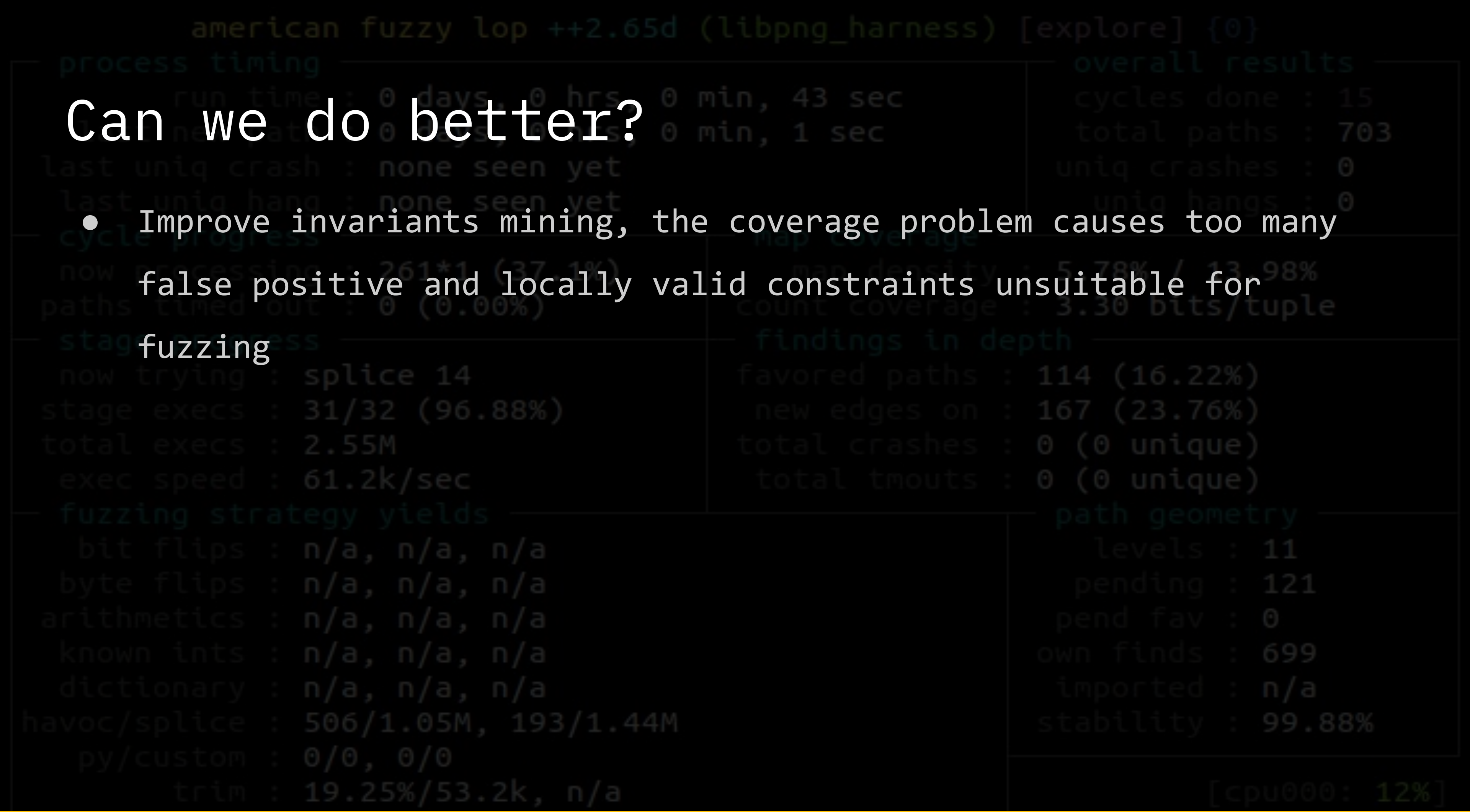
5 comments

oliverchang opened on August 16, 2022

github.com
Command injection via wordexp call. · Issue #368 · syoyo/tinygltf
Describe the issue This is a security vulnerability. The wordexp call here allows arbitrary code execution tinygltf/tiny_gltf.h Line 2640 in 0fa56e2 int ret = ...

Can we do better?

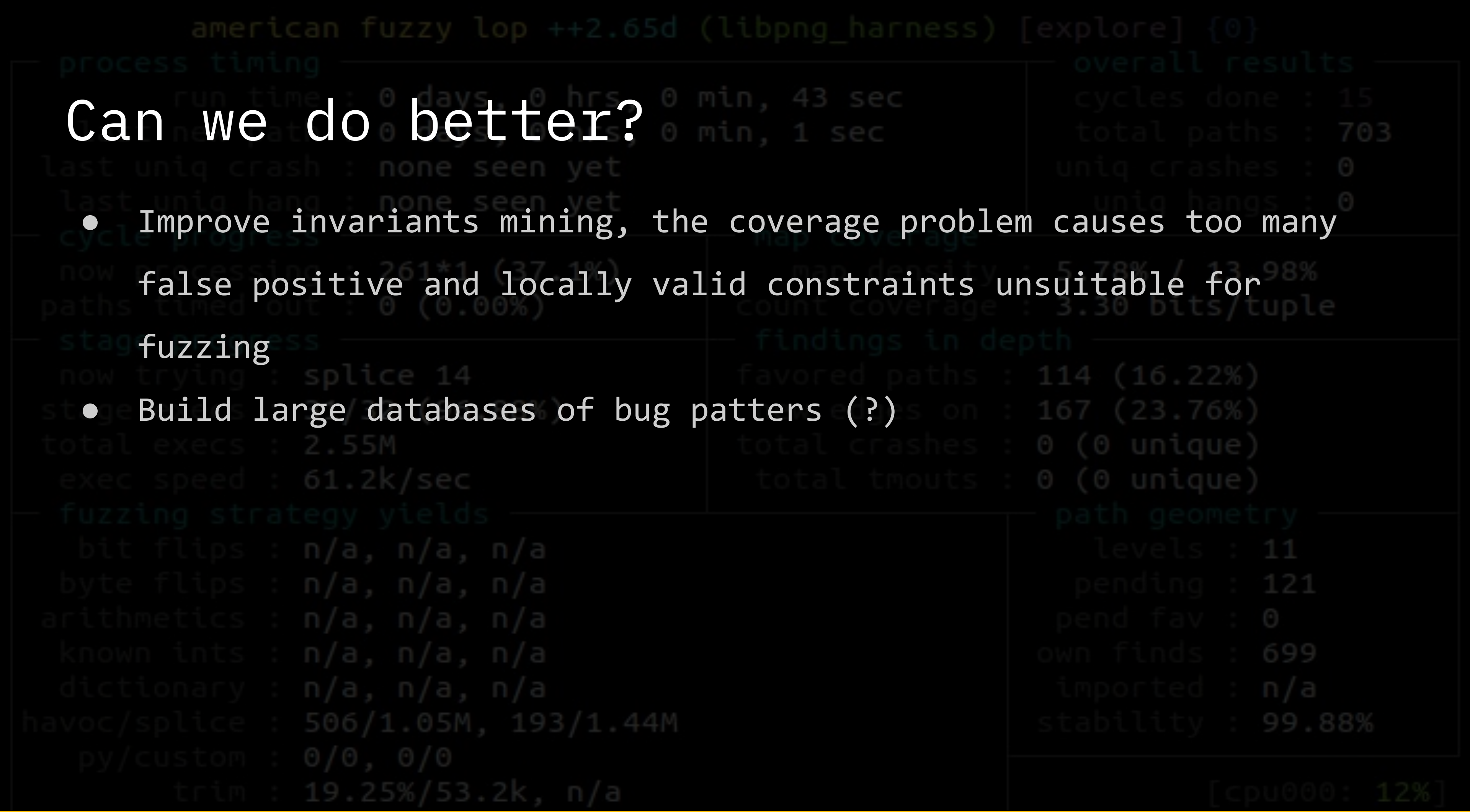
- Improve invariants mining, the coverage problem causes too many false positive and locally valid constraints unsuitable for fuzzing



```
       american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ──────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec        │  cycles done : 15    │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec         │  total paths : 703   │
│ last uniq crash : none seen yet                       │ uniq crashes : 0     │
│  last uniq hang : none seen yet                       │   uniq hangs : 0     │
├─ cycle progress ───────────────────┬─ map coverage ───┴──────────────────────┤
│  now processing : 261*1 (37.1%)    │    map density : 5.78% / 13.98%        │
│ paths timed out : 0 (0.00%)        │ count coverage : 3.30 bits/tuple       │
├─ stage progress ───────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 14            │ favored paths : 114 (16.22%)           │
│ stage execs : 31/32 (96.88%)       │  new edges on : 167 (23.76%)           │
│ total execs : 2.55M                │ total crashes : 0 (0 unique)           │
│  exec speed : 61.2k/sec            │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ──────────┴────────────────┬─ path geometry ───────┤
│   bit flips : n/a, n/a, n/a                         │    levels : 11        │
│  byte flips : n/a, n/a, n/a                         │   pending : 121       │
│ arithmetics : n/a, n/a, n/a                         │  pend fav : 0         │
│  known ints : n/a, n/a, n/a                         │ own finds : 699       │
│  dictionary : n/a, n/a, n/a                         │  imported : n/a       │
│havoc/splice : 506/1.05M, 193/1.44M                  │ stability : 99.88%    │
│   py/custom : 0/0, 0/0                              ├───────────────────────┘
│        trim : 19.25%/53.2k, n/a                     │          [cpu000: 12%]
```

Can we do better?

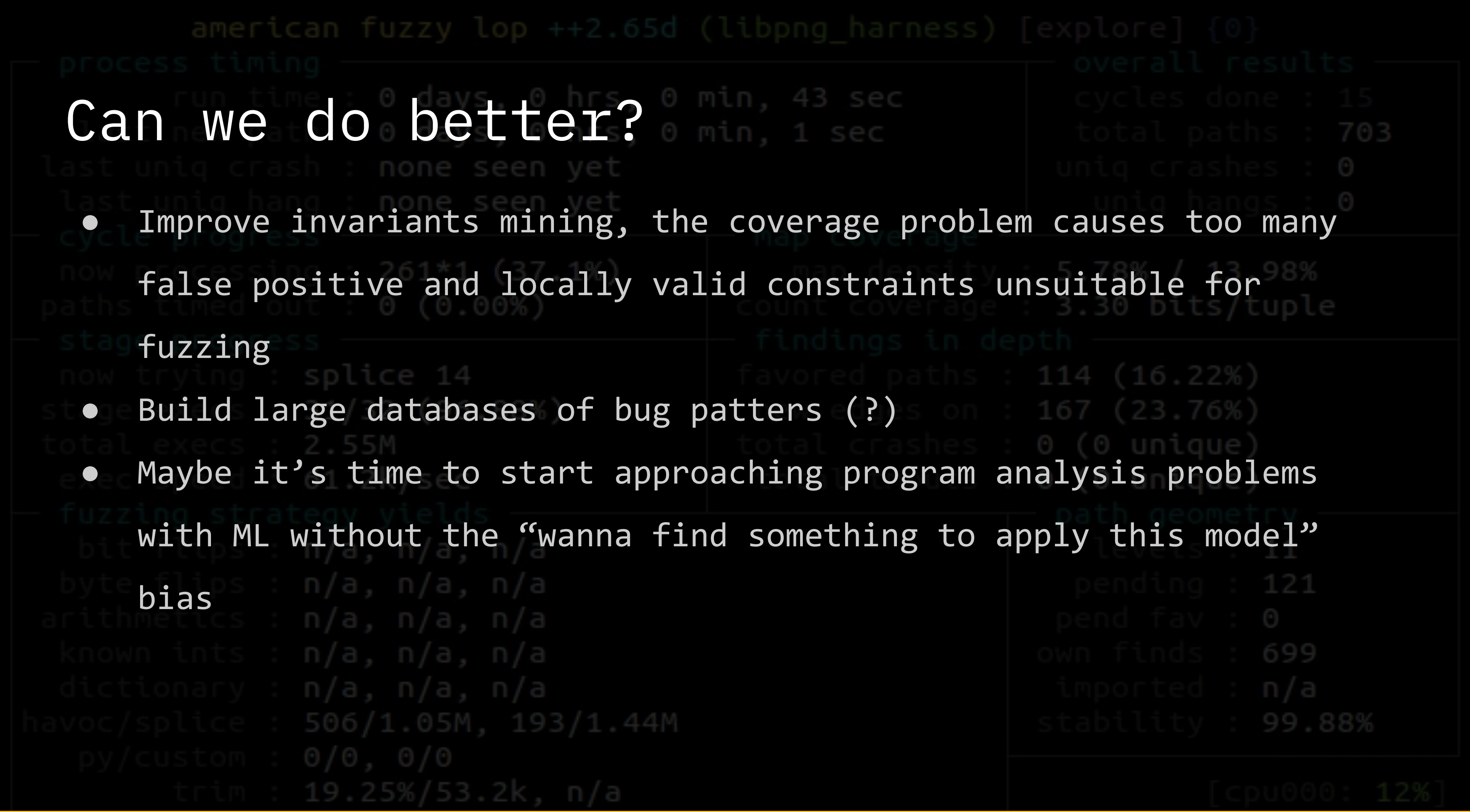
- Improve invariants mining, the coverage problem causes too many false positive and locally valid constraints unsuitable for fuzzing
- Build large databases of bug patters (?)



```
          american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ───────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec         │  cycles done : 15    │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec          │  total paths : 703   │
│ last uniq crash : none seen yet                        │ uniq crashes : 0     │
│  last uniq hang : none seen yet                        │   uniq hangs : 0     │
├─ cycle progress ─────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 261*1 (37.1%)      │    map density : 5.78% / 13.98%       │
│ paths timed out : 0 (0.00%)          │ count coverage : 3.30 bits/tuple      │
├─ stage progress ─────────────────────┼─ findings in depth ───────────────────┤
│  now trying : splice 14              │ favored paths : 114 (16.22%)          │
│ stage execs : [illegible]            │  new edges on : 167 (23.76%)          │
│ total execs : 2.55M                  │ total crashes : 0 (0 unique)          │
│  exec speed : 61.2k/sec              │  total tmouts : 0 (0 unique)          │
├─ fuzzing strategy yields ────────────┴─────────────┬─ path geometry ─────────┤
│   bit flips : n/a, n/a, n/a                        │    levels : 11          │
│  byte flips : n/a, n/a, n/a                        │   pending : 121         │
│ arithmetics : n/a, n/a, n/a                        │  pend fav : 0           │
│  known ints : n/a, n/a, n/a                        │ own finds : 699         │
│  dictionary : n/a, n/a, n/a                        │  imported : n/a         │
│havoc/splice : 506/1.05M, 193/1.44M                 │ stability : 99.88%      │
│   py/custom : 0/0, 0/0                             ├─────────────────────────┘
│        trim : 19.25%/53.2k, n/a                    │          [cpu000: 12%]
```

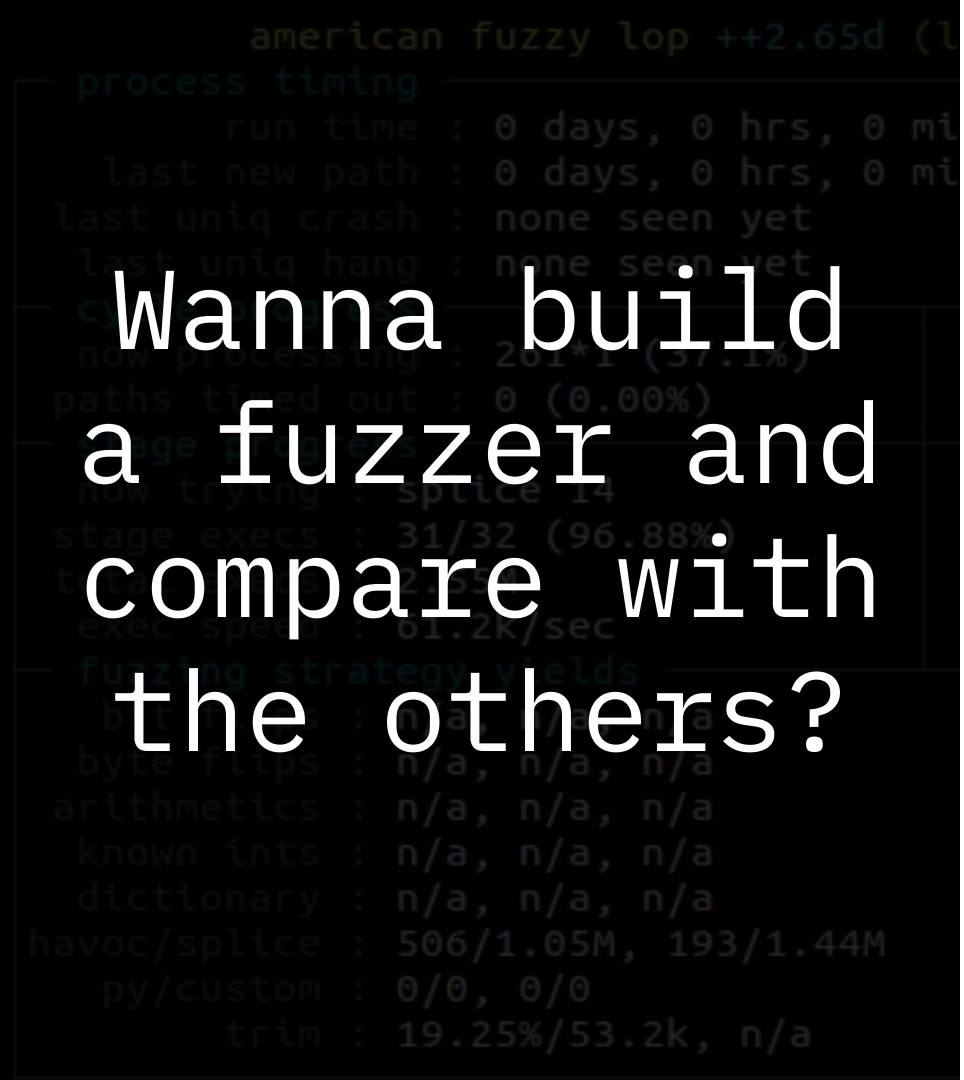
Can we do better?

- Improve invariants mining, the coverage problem causes too many false positive and locally valid constraints unsuitable for fuzzing
- Build large databases of bug patters (?)
- Maybe it’s time to start approaching program analysis problems with ML without the “wanna find something to apply this model” bias



```
           american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                          overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec            cycles done : 15
  last new path : 0 days, 0 hrs, 0 min, 1 sec            total paths : 703
 last uniq crash : none seen yet                        uniq crashes : 0
  last uniq hang : none seen yet                          uniq hangs : 0
  cycle progress                        map coverage
  now processing : 261*1 (37.1%)          map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)           count coverage : 3.30 bits/tuple
  stage progress                        findings in depth
  now trying : splice 14               favored paths : 114 (16.22%)
 stage execs : [illegible]             new edges on : 167 (23.76%)
 total execs : 2.55M                   total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                total tmouts : [illegible]
  fuzzing strategy yields                                 path geometry
   bit flips : n/a, n/a, n/a                               levels : 11
  byte flips : n/a, n/a, n/a                              pending : 121
 arithmetics : n/a, n/a, n/a                             pend fav : 0
  known ints : n/a, n/a, n/a                            own finds : 699
  dictionary : n/a, n/a, n/a                             imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                     stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                              [cpu000: 12%]
```

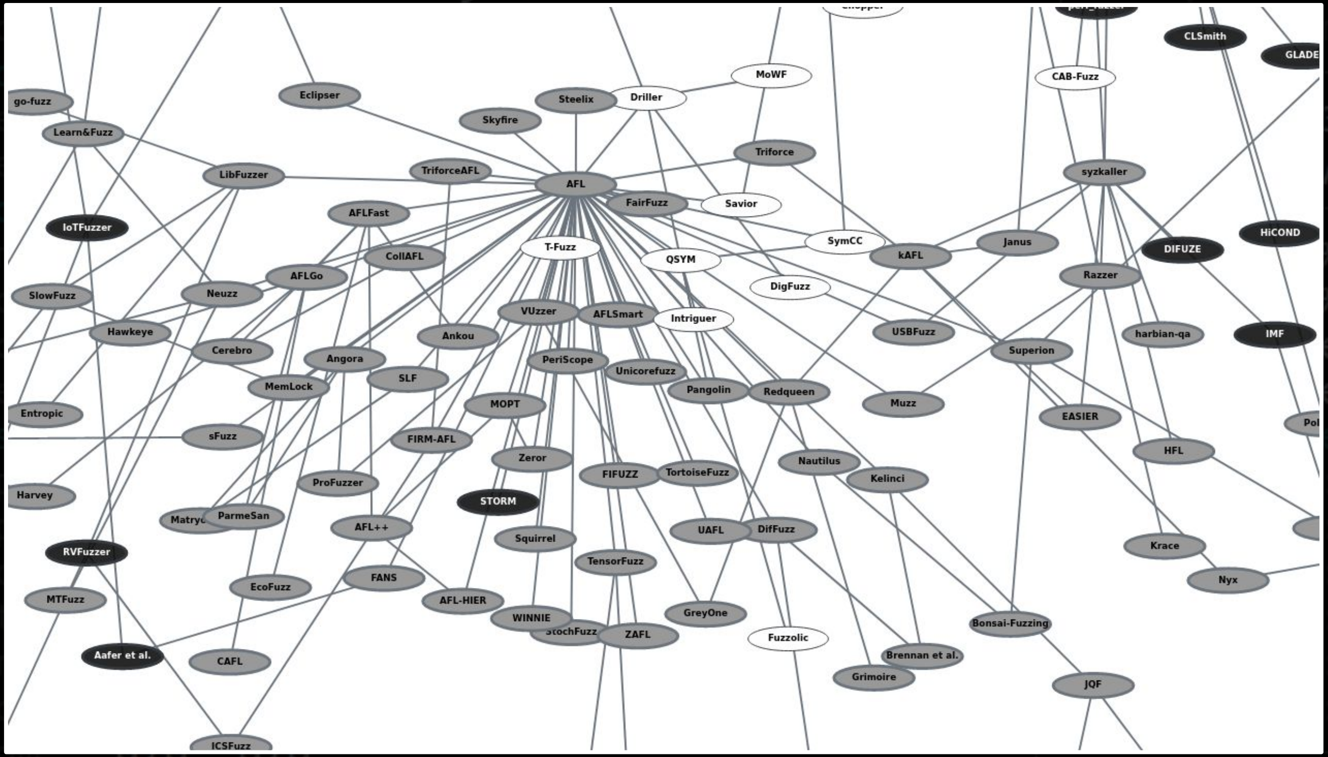
Wanna build a fuzzer and compare with the others?



```
        american fuzzy lop ++2.65d (l
 process timing
        run time : 0 days, 0 hrs, 0 mi
   last new path : 0 days, 0 hrs, 0 mi
 last uniq crash : none seen yet
  last uniq hang : none seen yet
 cycle progress
  now processing : 261*1 (37.1%)
 paths timed out : 0 (0.00%)
 stage progress
  now trying : splice 14
 stage execs : 31/32 (96.88%)
 total execs : 2.65M
  exec speed : 61.2k/sec
 fuzzing strategy yields
   bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
 arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
havoc/splice : 506/1.05M, 193/1.44M
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a
```

Good luck.

Problem: Fuzzers Fragmentation



From https://fuzzing-survey.org/

Cause: Monolithic Codebases

Fuzzers are

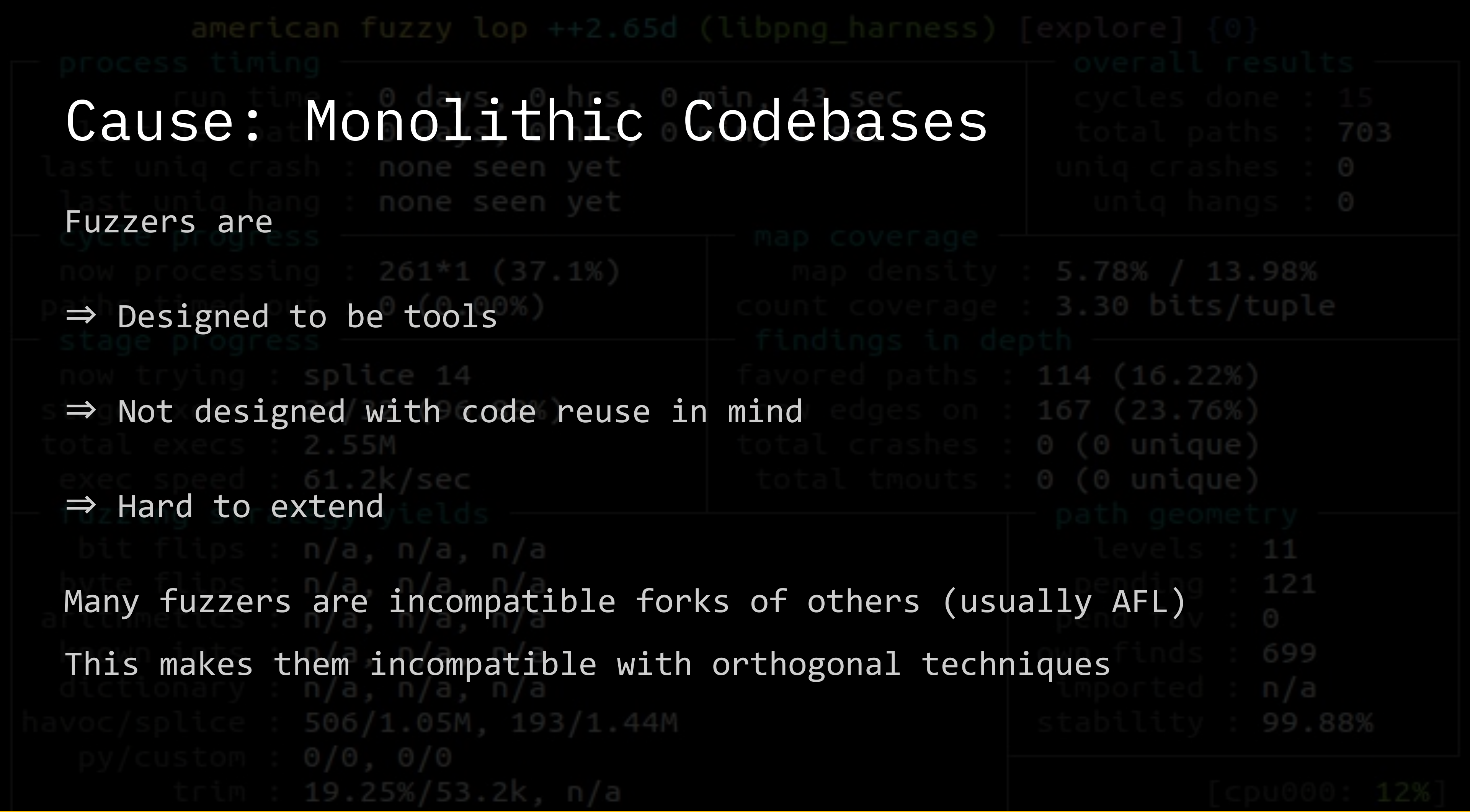
⇒ Designed to be tools

⇒ Not designed with code reuse in mind

⇒ Hard to extend

Many fuzzers are incompatible forks of others (usually AFL)

This makes them incompatible with orthogonal techniques

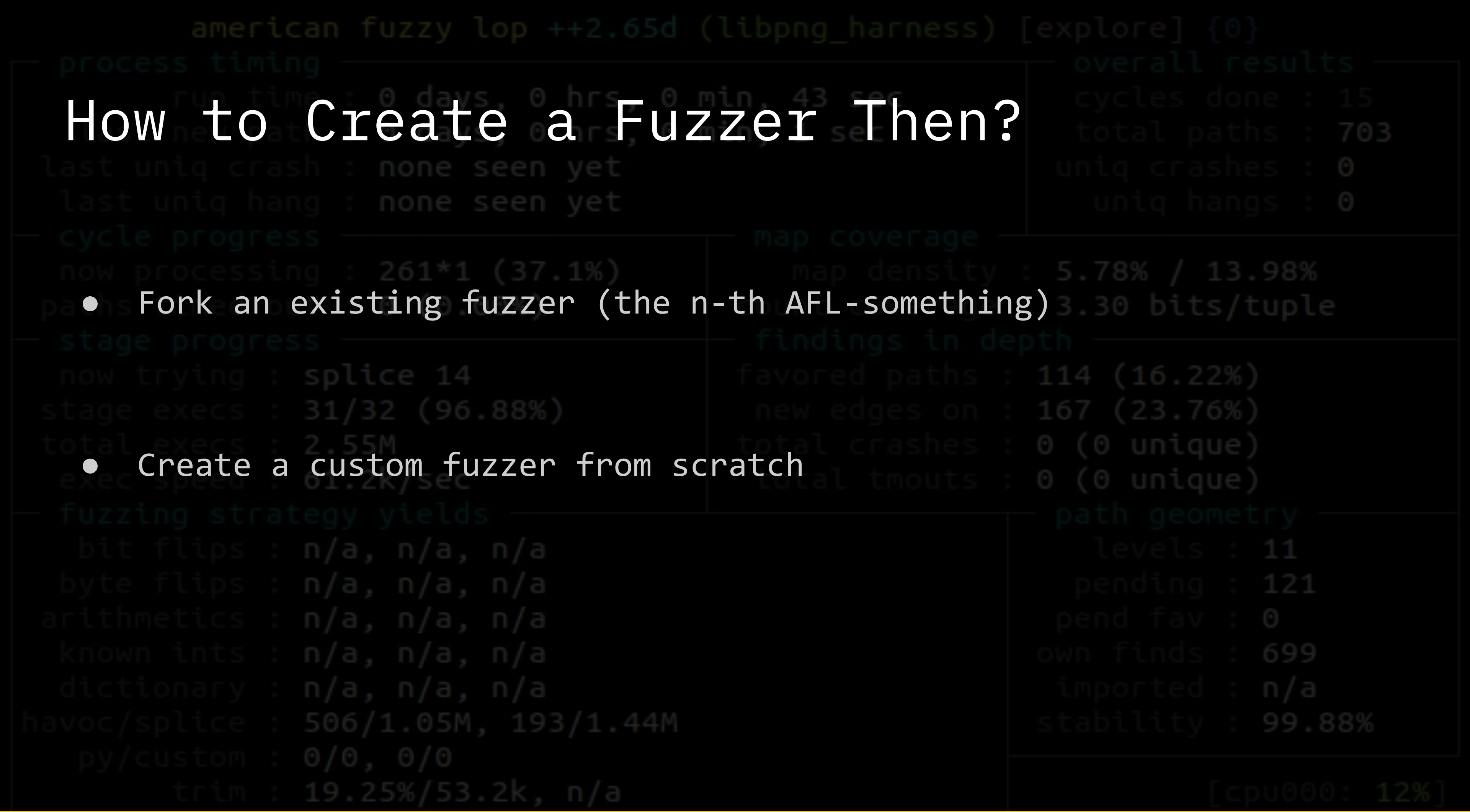


```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                        overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
  [illegible]                                           total paths : 703
 last uniq crash : none seen yet                       uniq crashes : 0
  last uniq hang : none seen yet                         uniq hangs : 0
  cycle progress                        map coverage
  now processing : 261*1 (37.1%)           map density : 5.78% / 13.98%
  paths timed out : 0 (0.00%)           count coverage : 3.30 bits/tuple
  stage progress                        findings in depth
  now trying : splice 14                favored paths : 114 (16.22%)
 stage execs : [illegible]               new edges on : 167 (23.76%)
 total execs : 2.55M                   total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                 total tmouts : 0 (0 unique)
  fuzzing strategy yields                              path geometry
   bit flips : n/a, n/a, n/a                            levels : 11
  byte flips : n/a, n/a, n/a                           pending : 121
 arithmetics : n/a, n/a, n/a                          pend fav : 0
  known ints : n/a, n/a, n/a                         own finds : 699
  dictionary : n/a, n/a, n/a                          imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                  stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                          [cpu000: 12%]
```

How to Create a Fuzzer Then?

- Fork an existing fuzzer (the n-th AFL-something)

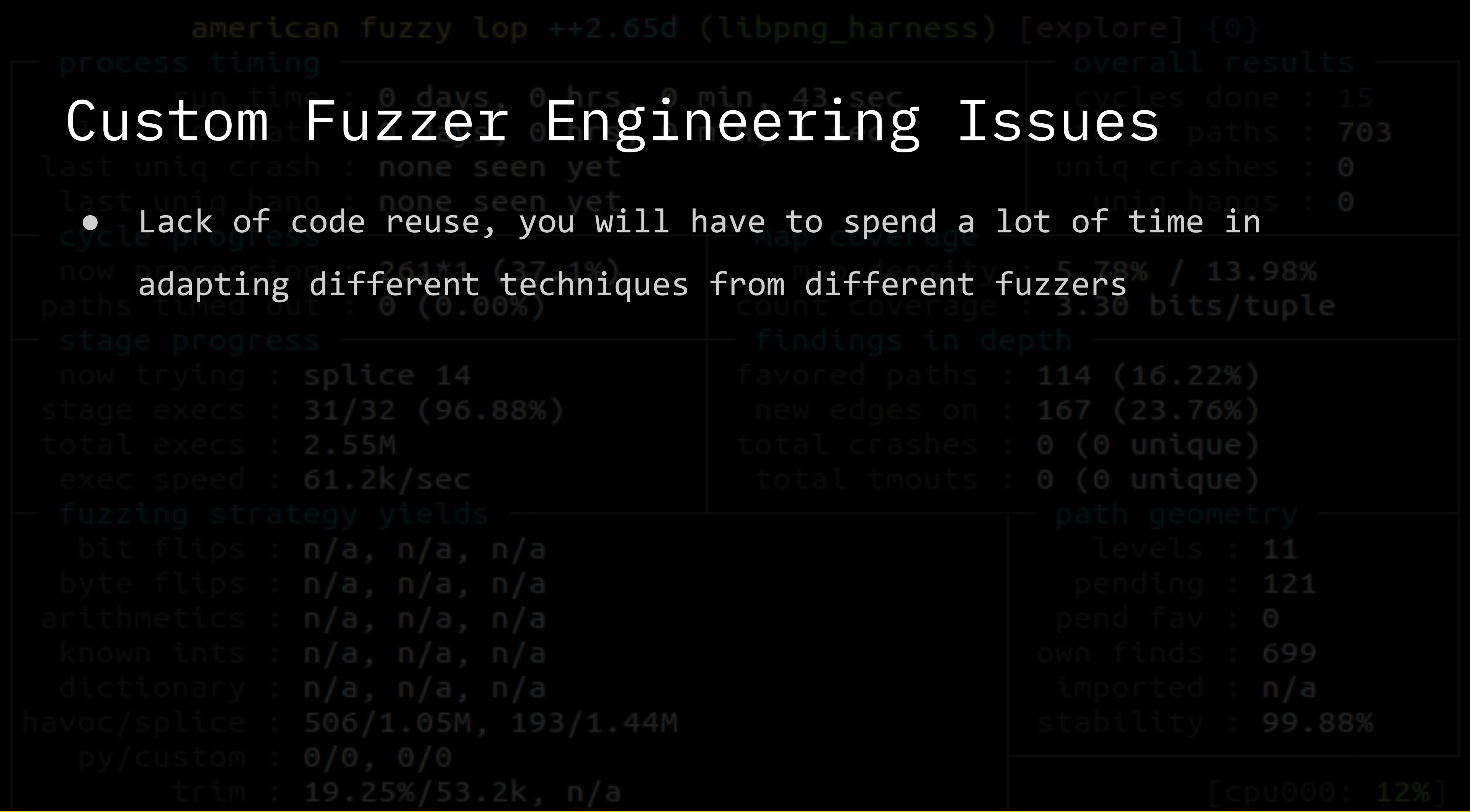
- Create a custom fuzzer from scratch



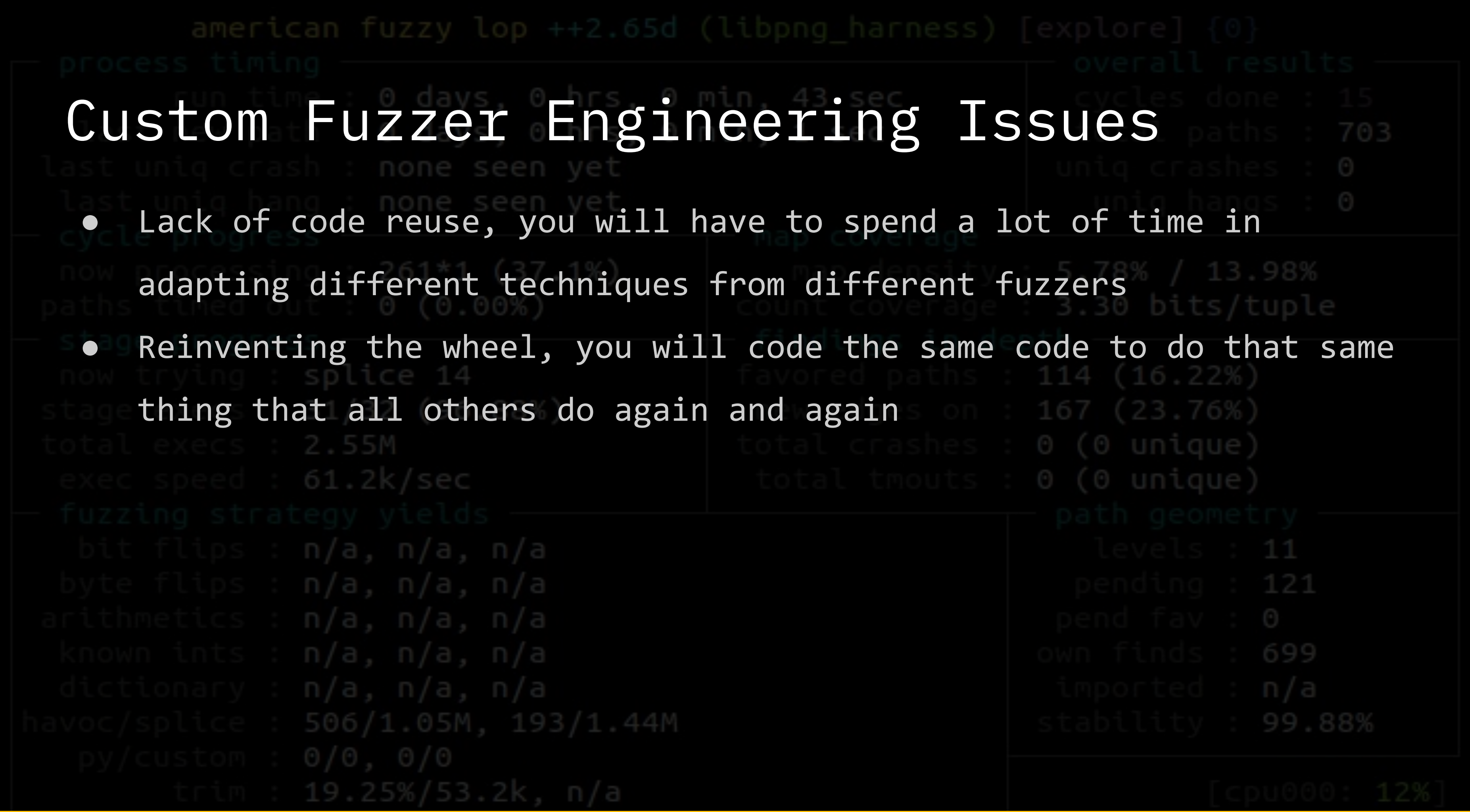
```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec       │  cycles done : 15    │
│   last new path : [illegible]                        │  total paths : 703   │
│ last uniq crash : none seen yet                      │ uniq crashes : 0     │
│  last uniq hang : none seen yet                      │   uniq hangs : 0     │
├─ cycle progress ───────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 261*1 (37.1%)    │    map density : 5.78% / 13.98%        │
│ paths timed out : [illegible]      │ count coverage : 3.30 bits/tuple       │
├─ stage progress ───────────────────┼─ findings in depth ────────────────────┤
│  now trying : splice 14            │ favored paths : 114 (16.22%)           │
│ stage execs : 31/32 (96.88%)       │  new edges on : 167 (23.76%)           │
│ total execs : 2.55M                │ total crashes : 0 (0 unique)           │
│  exec speed : 61.2k/sec            │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ──────────┴───────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                        │    levels : 11         │
│  byte flips : n/a, n/a, n/a                        │   pending : 121        │
│ arithmetics : n/a, n/a, n/a                        │  pend fav : 0          │
│  known ints : n/a, n/a, n/a                        │ own finds : 699        │
│  dictionary : n/a, n/a, n/a                        │  imported : n/a        │
│havoc/splice : 506/1.05M, 193/1.44M                 │ stability : 99.88%     │
│  py/custom : 0/0, 0/0                              ├────────────────────────┘
│        trim : 19.25%/53.2k, n/a                    │          [cpu000: 12%]
```

Custom Fuzzer Engineering Issues

- Lack of code reuse, you will have to spend a lot of time in adapting different techniques from different fuzzers

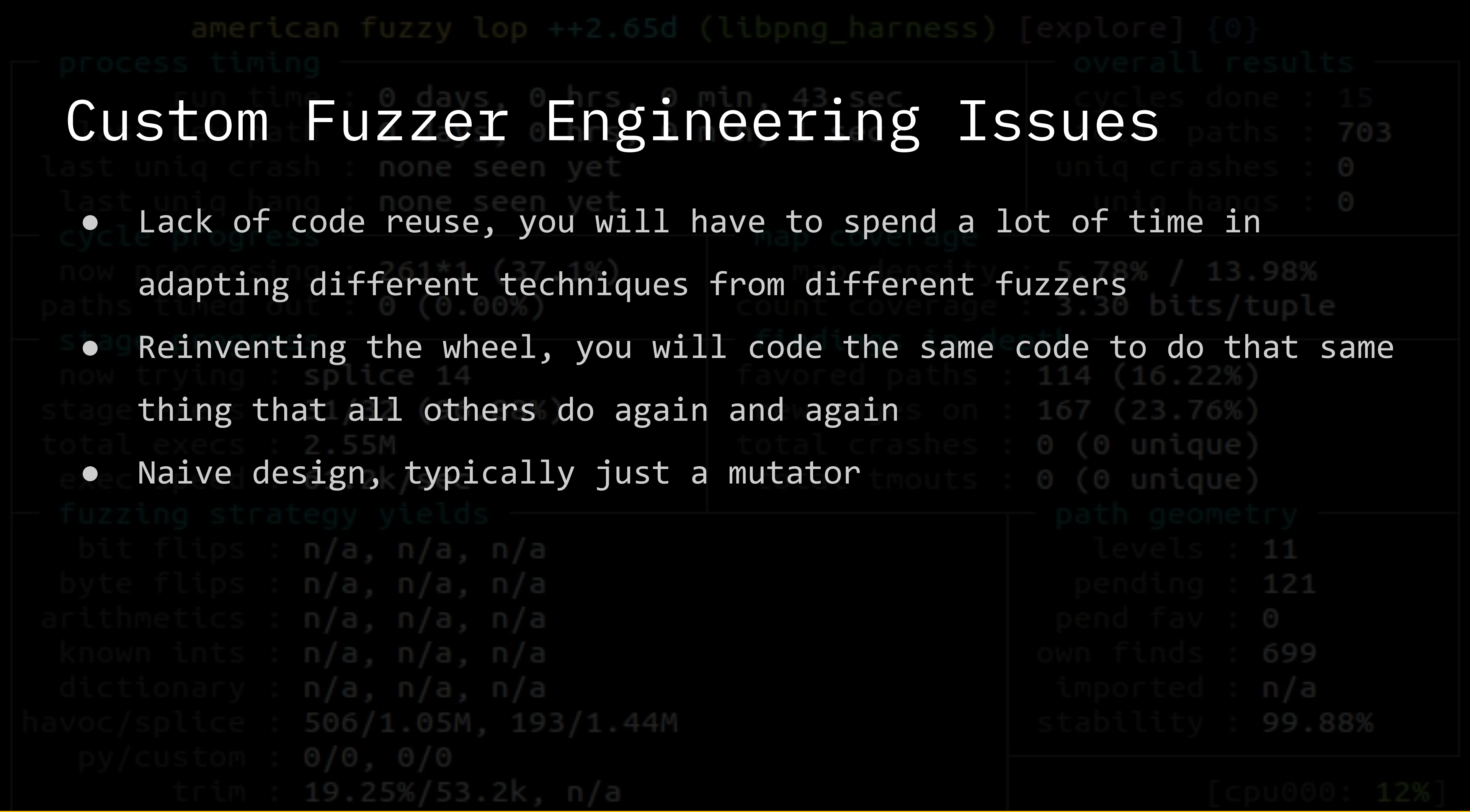


```
         american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ─────────────────────────────────────┬─ overall results ───┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec        │  cycles done : 15   │
│   last new path : [illegible] days, 0 hrs, [illegible] │  total paths : 703  │
│ last uniq crash : none seen yet                        │ uniq crashes : 0    │
│  last uniq hang : none seen yet                        │   uniq hangs : 0    │
├─ cycle progress ──────────────────┬─ map coverage ─────┴────────────────────┤
│  now processing : 261*1 (37.1%)   │    map density : 5.78% / 13.98%         │
│ paths timed out : 0 (0.00%)       │ count coverage : 3.30 bits/tuple        │
├─ stage progress ──────────────────┼─ findings in depth ─────────────────────┤
│  now trying : splice 14           │ favored paths : 114 (16.22%)            │
│ stage execs : 31/32 (96.88%)      │  new edges on : 167 (23.76%)            │
│ total execs : 2.55M               │ total crashes : 0 (0 unique)            │
│  exec speed : 61.2k/sec           │  total tmouts : 0 (0 unique)            │
├─ fuzzing strategy yields ─────────┴─────────────────┬─ path geometry ───────┤
│   bit flips : n/a, n/a, n/a                          │    levels : 11        │
│  byte flips : n/a, n/a, n/a                          │   pending : 121       │
│ arithmetics : n/a, n/a, n/a                          │  pend fav : 0         │
│  known ints : n/a, n/a, n/a                          │ own finds : 699       │
│  dictionary : n/a, n/a, n/a                          │  imported : n/a       │
│havoc/splice : 506/1.05M, 193/1.44M                   │ stability : 99.88%    │
│   py/custom : 0/0, 0/0                               ├───────────────────────┘
│        trim : 19.25%/53.2k, n/a                      │          [cpu000: 12%]
```



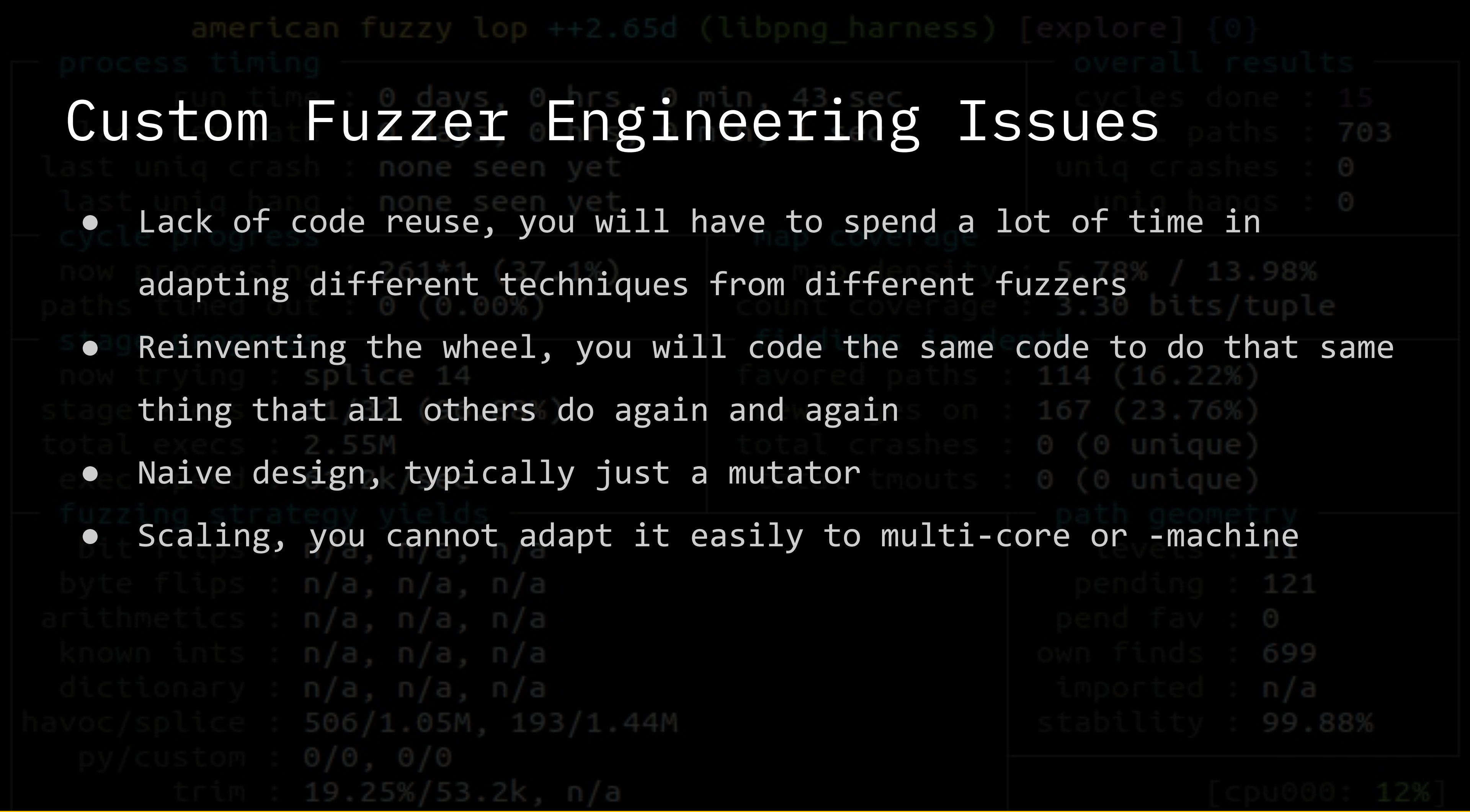
Custom Fuzzer Engineering Issues

- Lack of code reuse, you will have to spend a lot of time in adapting different techniques from different fuzzers
- Reinventing the wheel, you will code the same code to do that same thing that all others do again and again



Custom Fuzzer Engineering Issues

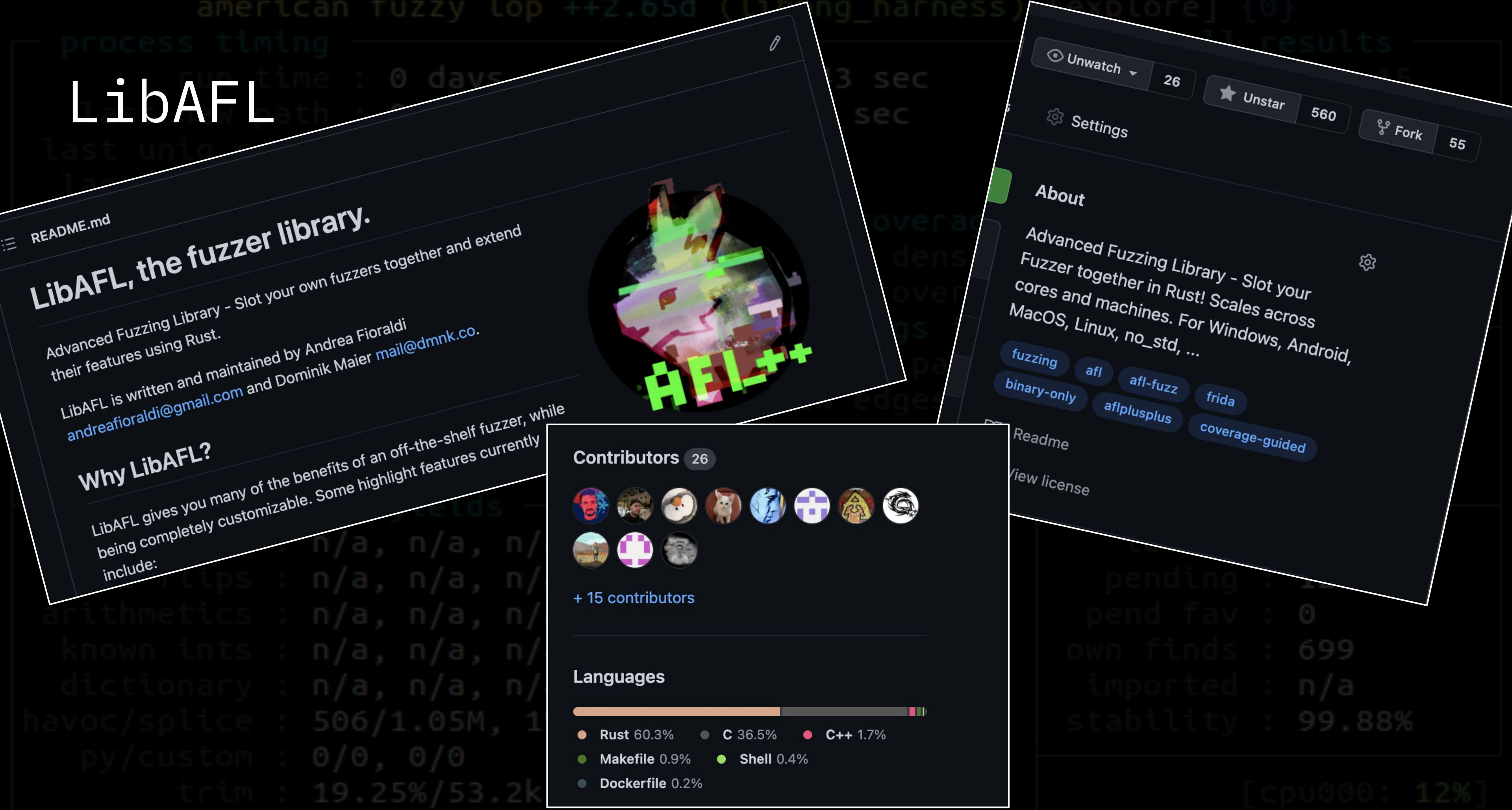
- Lack of code reuse, you will have to spend a lot of time in adapting different techniques from different fuzzers
- Reinventing the wheel, you will code the same code to do that same thing that all others do again and again
- Naive design, typically just a mutator



Custom Fuzzer Engineering Issues

- Lack of code reuse, you will have to spend a lot of time in adapting different techniques from different fuzzers
- Reinventing the wheel, you will code the same code to do that same thing that all others do again and again
- Naive design, typically just a mutator
- Scaling, you cannot adapt it easily to multi-core or -machine

LibAFL



README.md

LibAFL, the fuzzer library.

Advanced Fuzzing Library - Slot your own fuzzers together and extend their features using Rust.

LibAFL is written and maintained by Andrea Fioraldi andreafioraldi@gmail.com and Dominik Maier mail@dmnk.co.

Why LibAFL?

LibAFL gives you many of the benefits of an off-the-shelf fuzzer, while being completely customizable. Some highlight features currently include:

About

Advanced Fuzzing Library - Slot your Fuzzer together in Rust! Scales across cores and machines. For Windows, Android, MacOS, Linux, no_std, ...

Contributors 26

+ 15 contributors

Languages

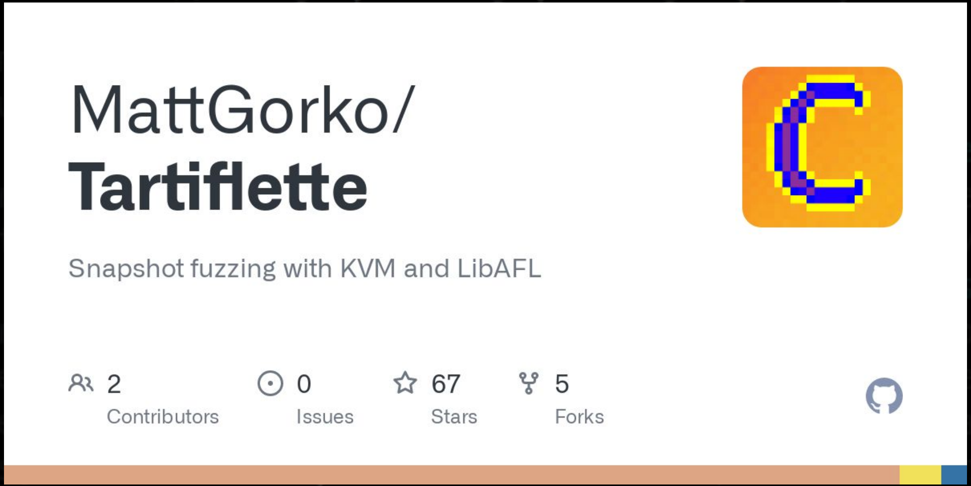
- Rust 60.3%
- C 36.5%
- C++ 1.7%
- Makefile 0.9%
- Shell 0.4%
- Dockerfile 0.2%

What?

LibAFL is a library for fuzzers that are

- Fast (low IPC, runtime overhead)
- Scalable (almost linearly to 200+ cores)
- Portable (Android, Windows, MacOS, Linux, Kernels, …)
- State-of-the-Art (Hybrid-, Grammar-, Token-, Feedback-Fuzzing)
- Multi-instrumentation (binary-only Frida & Qemu, Clang, Python,…)

And, most importantly, very extendable with your own components.



MattGorko/
Tartiflette

Snapshot fuzzing with KVM and LibAFL

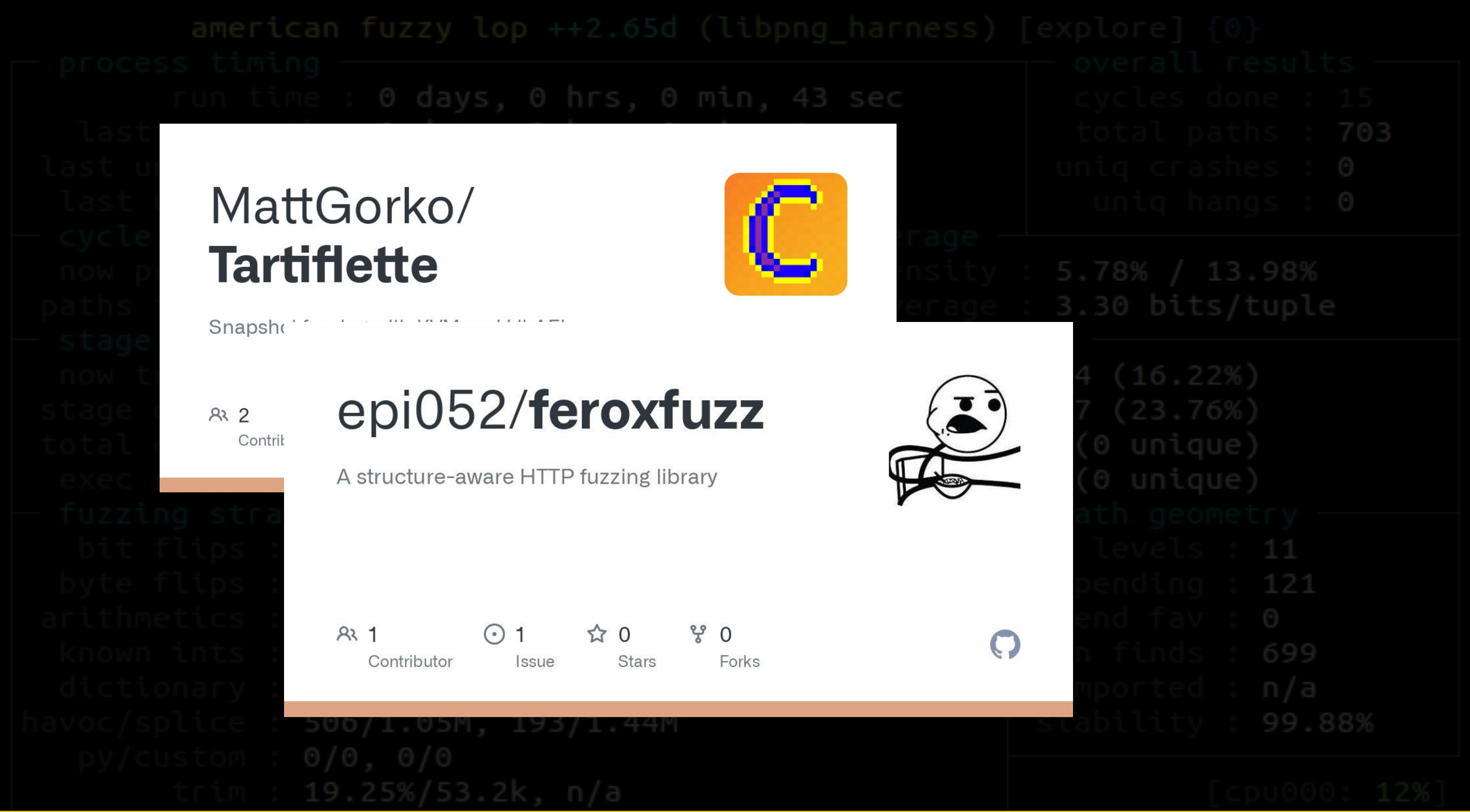
2 Contributors

0 Issues

67 Stars

5 Forks

```
         american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                          overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec            cycles done : 15
    last                                                   total paths : 703
 last u                                                   uniq crashes : 0
   last                                                     uniq hangs : 0
  cycle                                     erage
   now p                                     nsity : 5.78% / 13.98%
 paths                                       erage : 3.30 bits/tuple
  stage                                    s in depth
   now tr                                   paths : 114 (16.22%)
 stage                                      es on : 167 (23.76%)
 total                                      ashes : 0 (0 unique)
   exec                                      outs : 0 (0 unique)
  fuzzing strategy yields                           path geometry
    bit flips : n/a, n/a, n/a                          levels : 11
   byte flips : n/a, n/a, n/a                         pending : 121
 arithmetics : n/a, n/a, n/a                         pend fav : 0
   known ints : n/a, n/a, n/a                       own finds : 699
   dictionary : n/a, n/a, n/a                        imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                 stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                              [cpu000: 12%]
```



```
      american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                       overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
   last                                                  total paths : 703
 last u                                                 uniq crashes : 0
  last                                                    uniq hangs : 0
  cycle                                        erage
  now p                                        nsity : 5.78% / 13.98%
 paths                                         erage : 3.30 bits/tuple
  stage
  now tr                                       4 (16.22%)
 stage                                         7 (23.76%)
 total                                          (0 unique)
  exec                                          (0 unique)
  fuzzing stra                                 ath geometry
   bit flips :                                   levels : 11
  byte flips :                                  pending : 121
 arithmetics :                                  end fav : 0
  known ints :                                  n finds : 699
  dictionary :                                  mported : n/a
havoc/splice : 506/1.05M, 193/1.44M           stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                        [cpu000: 12%]
```

MattGorko/
Tartiflette

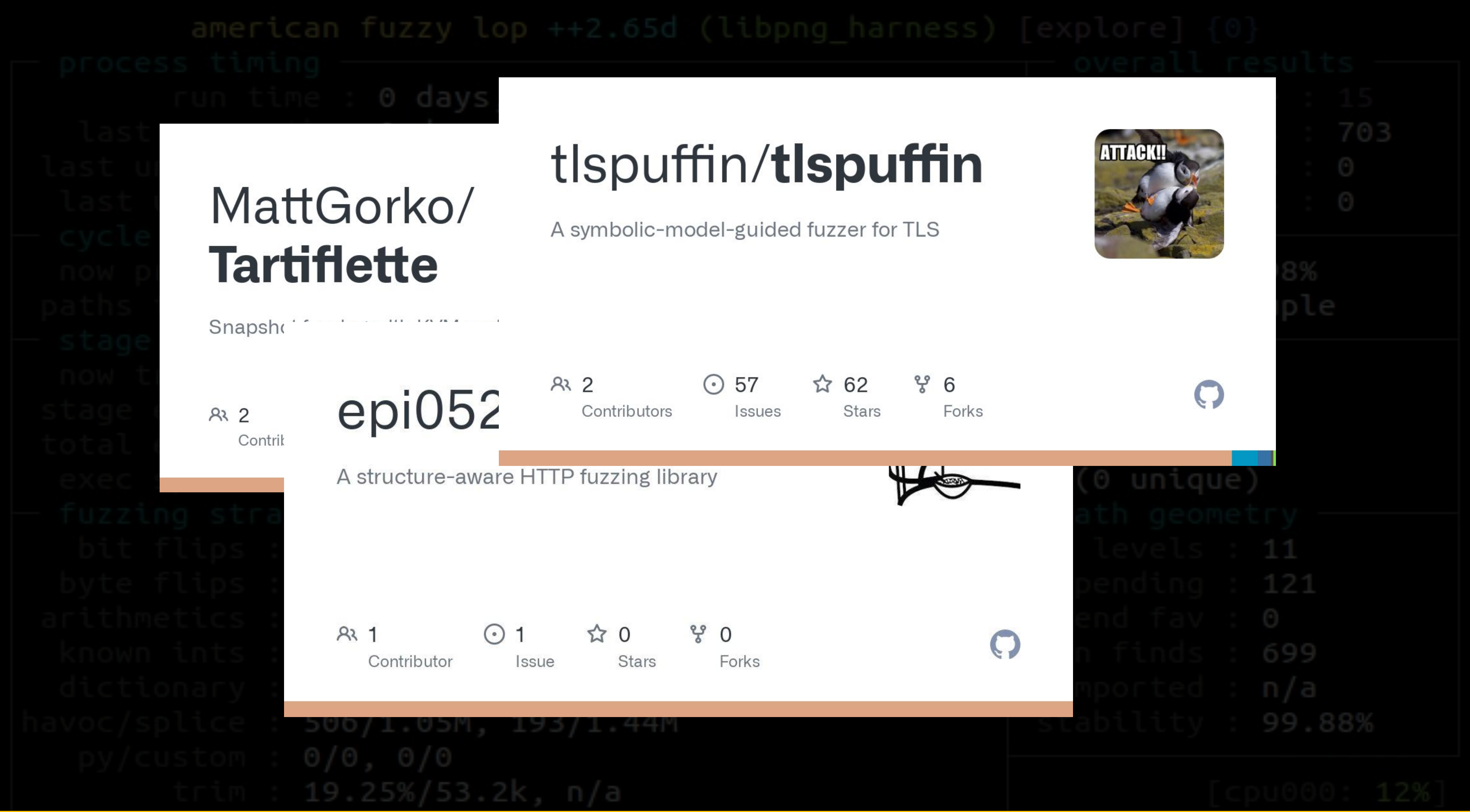
Snapsh

2
Contrib

epi052/**feroxfuzz**

A structure-aware HTTP fuzzing library

1 Contributor
1 Issue
0 Stars
0 Forks



```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
 process timing                                            overall results
      run time : 0 days                                          : 15
  last                                                           : 703
 last u                                                          : 0
  last                                                           : 0
  cycle
  now p                                                          8%
 paths                                                           ple
  stage
  now t
 stage
 total
  exec                                                      (0 unique)
  fuzzing stra                                          ath geometry
   bit flips :                                          levels : 11
  byte flips :                                         pending : 121
 arithmetics :                                         end fav : 0
  known ints :                                         n finds : 699
  dictionary :                                         mported : n/a
havoc/splice : 506/1.05M, 193/1.44M                  stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                               [cpu000: 12%]
```

MattGorko/
Tartiflette

Snapsho

2
Contri

epi052

A structure-aware HTTP fuzzing library

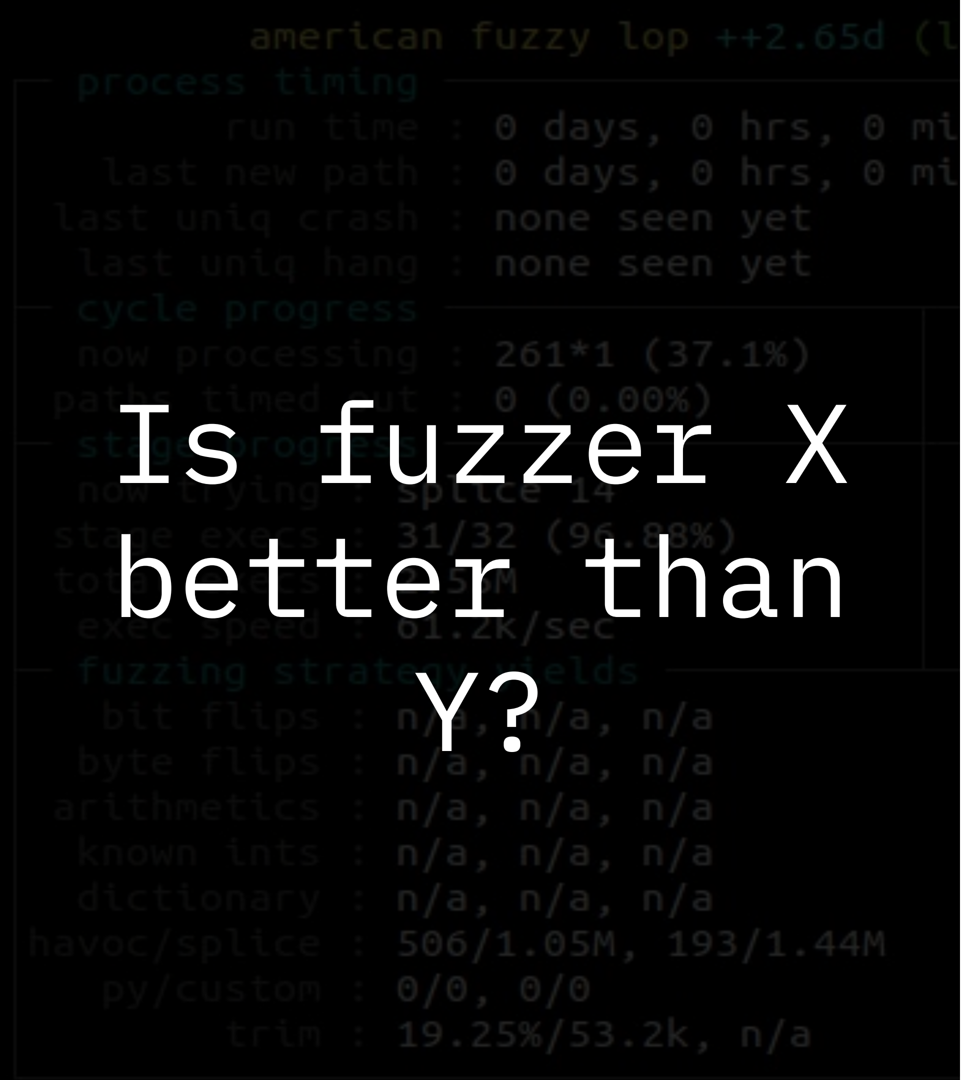
1 Contributor | 1 Issue | 0 Stars | 0 Forks

tlspuffin/**tlspuffin**

A symbolic-model-guided fuzzer for TLS

2 Contributors | 57 Issues | 62 Stars | 6 Forks

Is fuzzer X better than Y?

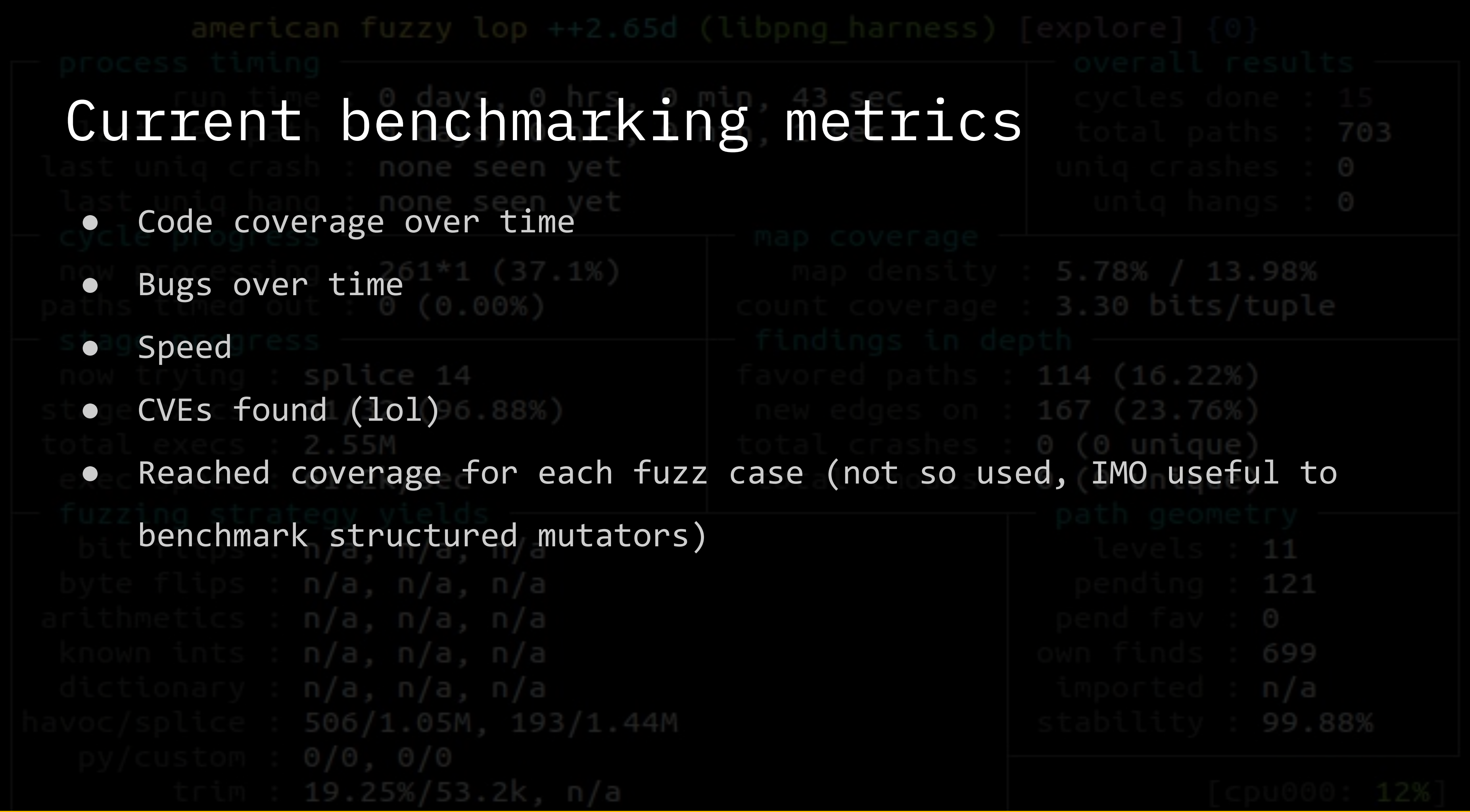


```
        american fuzzy lop ++2.65d (l
  process timing
        run time : 0 days, 0 hrs, 0 mi
   last new path : 0 days, 0 hrs, 0 mi
 last uniq crash : none seen yet
  last uniq hang : none seen yet
  cycle progress
  now processing : 261*1 (37.1%)
 paths timed out : 0 (0.00%)
  stage progress
  now trying : splice 14
 stage execs : 31/32 (96.88%)
 total execs : 2.[illegible]5M
  exec speed : 61.2k/sec
  fuzzing strategy yields
   bit flips : n/a, n/a, n/a
  byte flips : n/a, n/a, n/a
 arithmetics : n/a, n/a, n/a
  known ints : n/a, n/a, n/a
  dictionary : n/a, n/a, n/a
havoc/splice : 506/1.05M, 193/1.44M
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a
```

We don’t know. Really, we can only speculate about this.

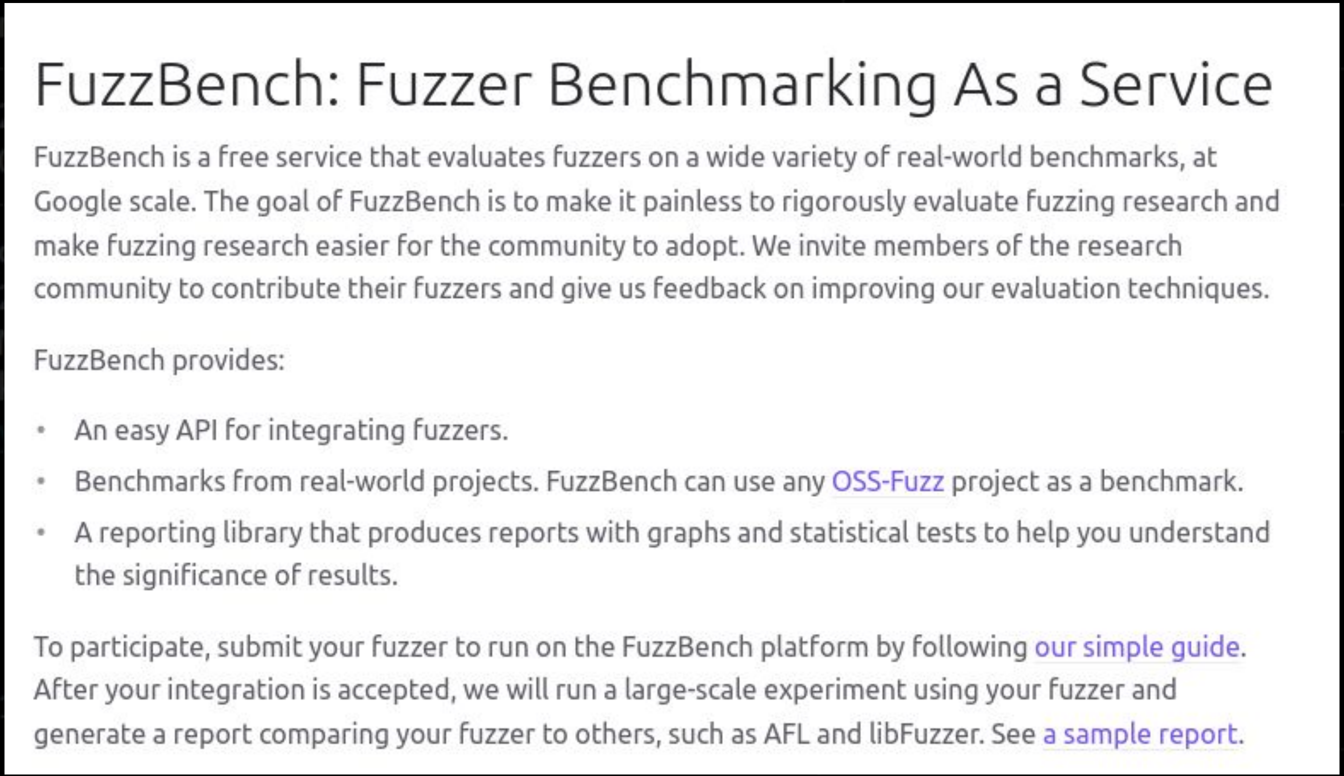
Current benchmarking metrics

- Code coverage over time
- Bugs over time
- Speed
- CVEs found (lol)
- Reached coverage for each fuzz case (not so used, IMO useful to benchmark structured mutators)



```
          american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                        overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec          cycles done : 15
  last new path : [illegible]                           total paths : 703
last uniq crash : none seen yet                        uniq crashes : 0
 last uniq hang : none seen yet                          uniq hangs : 0
  cycle progress                      map coverage
  now processing : 261*1 (37.1%)        map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)         count coverage : 3.30 bits/tuple
  stage progress                      findings in depth
  now trying : splice 14             favored paths : 114 (16.22%)
 stage execs : [illegible] (96.88%)   new edges on : 167 (23.76%)
 total execs : 2.55M                 total crashes : 0 (0 unique)
  exec speed : [illegible]            total tmouts : [illegible]
  fuzzing strategy yields                             path geometry
   bit flips : n/a, n/a, n/a                            levels : 11
  byte flips : n/a, n/a, n/a                           pending : 121
 arithmetics : n/a, n/a, n/a                          pend fav : 0
  known ints : n/a, n/a, n/a                         own finds : 699
  dictionary : n/a, n/a, n/a                          imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                  stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                             [cpu000: 12%]
```

Standard benchmarks ATM



FuzzBench: Fuzzer Benchmarking As a Service

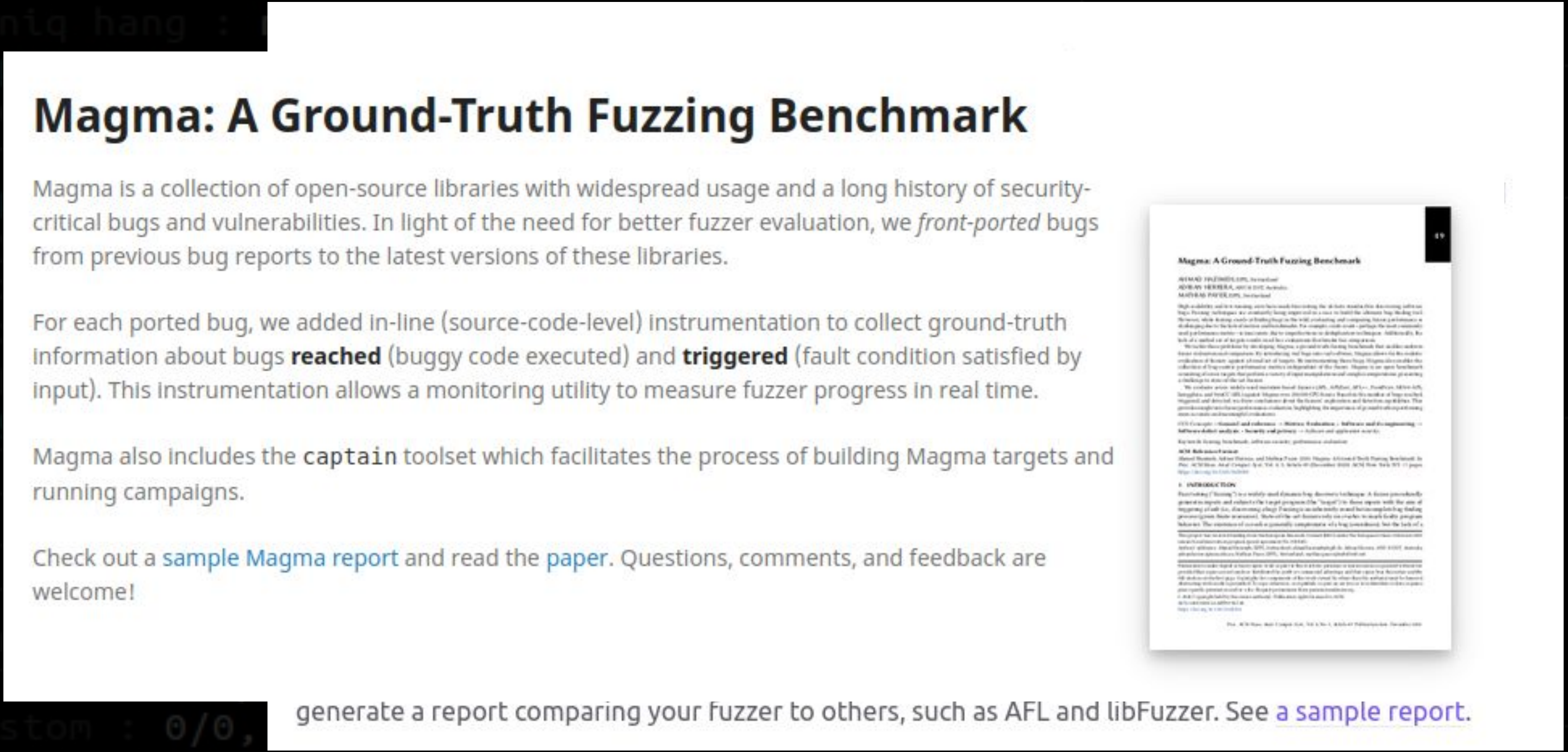
FuzzBench is a free service that evaluates fuzzers on a wide variety of real-world benchmarks, at Google scale. The goal of FuzzBench is to make it painless to rigorously evaluate fuzzing research and make fuzzing research easier for the community to adopt. We invite members of the research community to contribute their fuzzers and give us feedback on improving our evaluation techniques.

FuzzBench provides:

- An easy API for integrating fuzzers.
- Benchmarks from real-world projects. FuzzBench can use any OSS-Fuzz project as a benchmark.
- A reporting library that produces reports with graphs and statistical tests to help you understand the significance of results.

To participate, submit your fuzzer to run on the FuzzBench platform by following our simple guide. After your integration is accepted, we will run a large-scale experiment using your fuzzer and generate a report comparing your fuzzer to others, such as AFL and libFuzzer. See a sample report.

Standard benchmarks ATM



Magma: A Ground-Truth Fuzzing Benchmark

Magma is a collection of open-source libraries with widespread usage and a long history of security-critical bugs and vulnerabilities. In light of the need for better fuzzer evaluation, we *front-ported* bugs from previous bug reports to the latest versions of these libraries.

For each ported bug, we added in-line (source-code-level) instrumentation to collect ground-truth information about bugs **reached** (buggy code executed) and **triggered** (fault condition satisfied by input). This instrumentation allows a monitoring utility to measure fuzzer progress in real time.

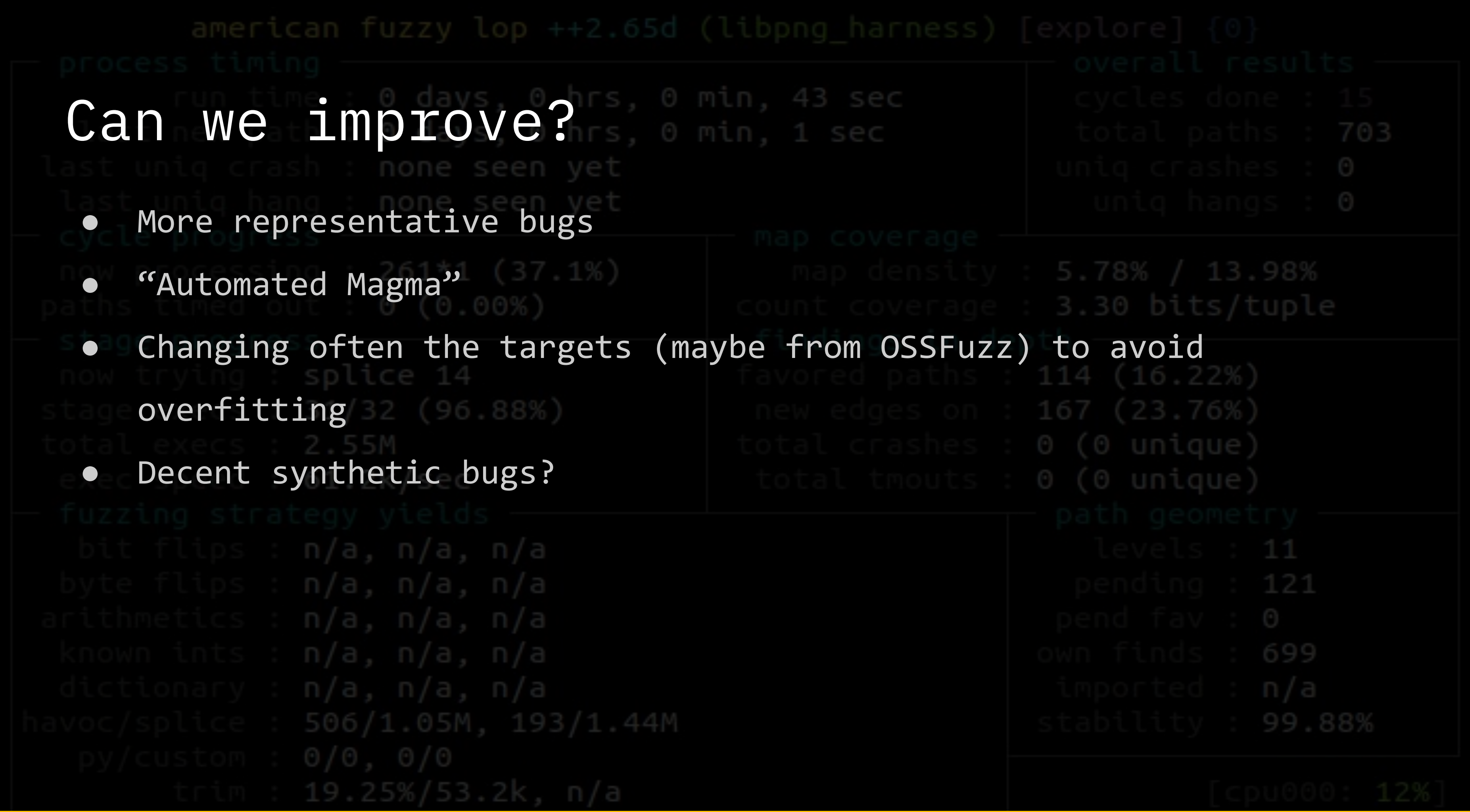
Magma also includes the `captain` toolset which facilitates the process of building Magma targets and running campaigns.

Check out a sample Magma report and read the paper. Questions, comments, and feedback are welcome!

generate a report comparing your fuzzer to others, such as AFL and libFuzzer. See a sample report.

Can we improve?

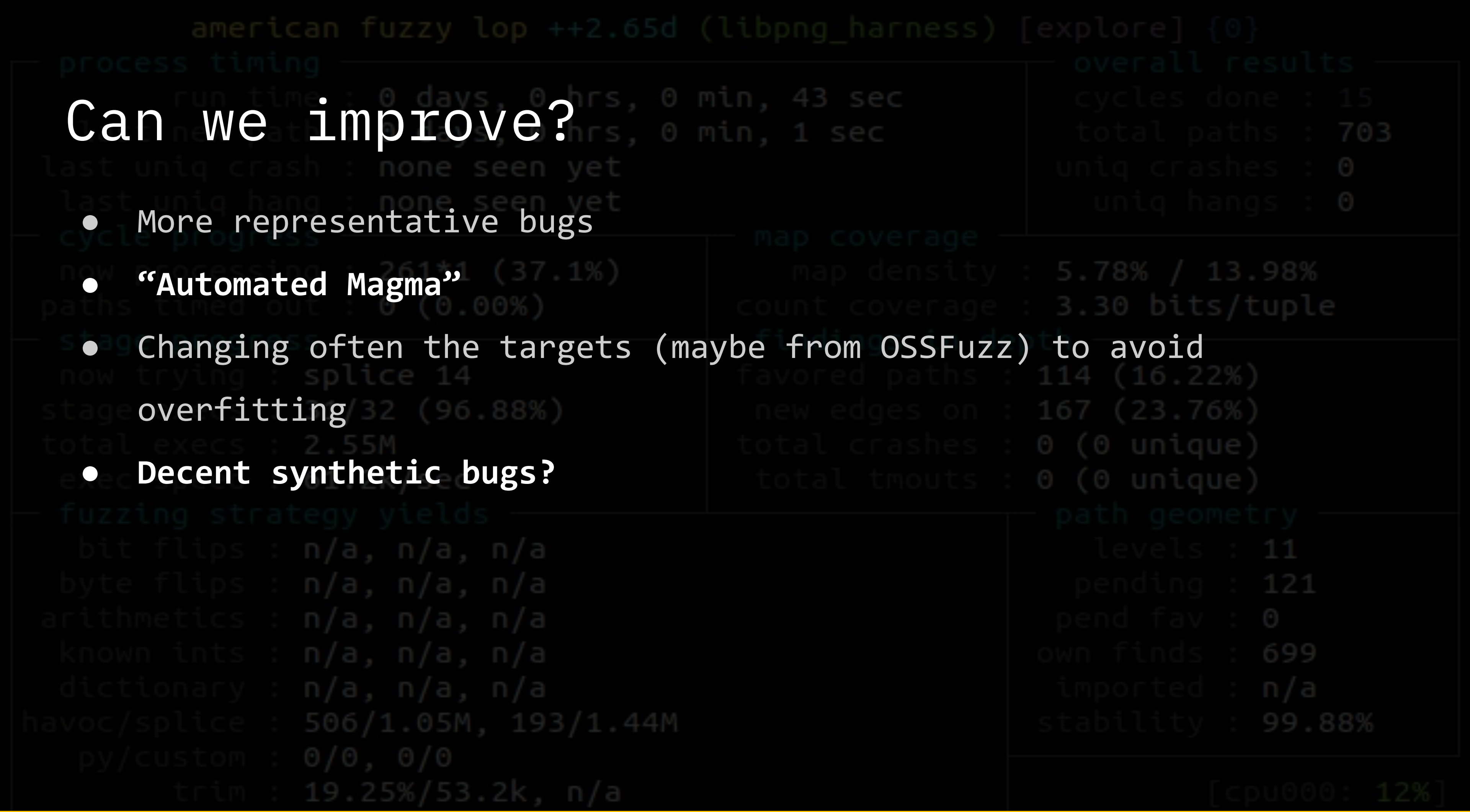
- More representative bugs
- “Automated Magma”
- Changing often the targets (maybe from OSSFuzz) to avoid overfitting
- Decent synthetic bugs?



```
         american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ────────────────────────────────────┬─ overall results ───┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec      │  cycles done : 15   │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec       │  total paths : 703  │
│ last uniq crash : none seen yet                     │ uniq crashes : 0    │
│  last uniq hang : none seen yet                     │   uniq hangs : 0    │
├─ cycle progress ───────────────────┬─ map coverage ─┴─────────────────────┤
│  now processing : 261*1 (37.1%)    │    map density : 5.78% / 13.98%      │
│ paths timed out : 0 (0.00%)        │ count coverage : 3.30 bits/tuple     │
├─ stage progress ───────────────────┼─ findings in depth ──────────────────┤
│  now trying : splice 14            │ favored paths : 114 (16.22%)         │
│ stage execs : 31/32 (96.88%)       │  new edges on : 167 (23.76%)         │
│ total execs : 2.55M                │ total crashes : 0 (0 unique)         │
│  exec speed : 61.2k/sec            │  total tmouts : 0 (0 unique)         │
├─ fuzzing strategy yields ──────────┴───────────────┬─ path geometry ──────┤
│   bit flips : n/a, n/a, n/a                         │    levels : 11       │
│  byte flips : n/a, n/a, n/a                         │   pending : 121      │
│ arithmetics : n/a, n/a, n/a                         │  pend fav : 0        │
│  known ints : n/a, n/a, n/a                         │ own finds : 699      │
│  dictionary : n/a, n/a, n/a                         │  imported : n/a      │
│havoc/splice : 506/1.05M, 193/1.44M                  │ stability : 99.88%   │
│   py/custom : 0/0, 0/0                              ├──────────────────────┘
│        trim : 19.25%/53.2k, n/a                     │          [cpu000: 12%]
```

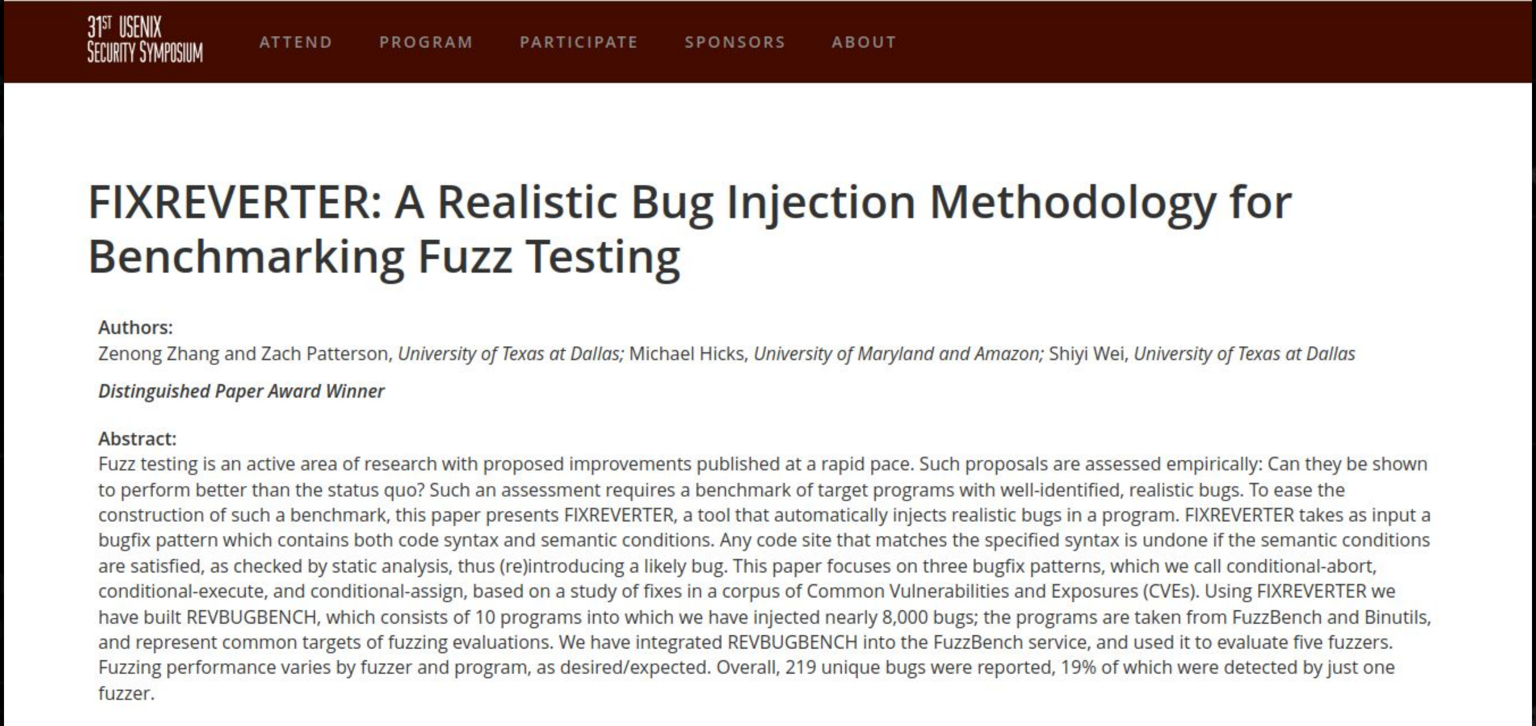
Can we improve?

- More representative bugs
- **“Automated Magma”**
- Changing often the targets (maybe from OSSFuzz) to avoid overfitting
- **Decent synthetic bugs?**



```
         american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
 process timing                                          overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec           cycles done : 15
   last new path : 0 days, 0 hrs, 0 min, 1 sec            total paths : 703
 last uniq crash : none seen yet                         uniq crashes : 0
  last uniq hang : none seen yet                           uniq hangs : 0
 cycle progress                       map coverage
  now processing : 261*1 (37.1%)         map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)          count coverage : 3.30 bits/tuple
 stage progress                       findings in depth
  now trying : splice 14             favored paths : 114 (16.22%)
 stage execs : [illegible]/32 (96.88%)      new edges on : 167 (23.76%)
 total execs : 2.55M                 total crashes : 0 (0 unique)
  exec speed : 61.2k/sec              total tmouts : 0 (0 unique)
 fuzzing strategy yields                              path geometry
   bit flips : n/a, n/a, n/a                            levels : 11
  byte flips : n/a, n/a, n/a                           pending : 121
 arithmetics : n/a, n/a, n/a                          pend fav : 0
  known ints : n/a, n/a, n/a                         own finds : 699
  dictionary : n/a, n/a, n/a                          imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                  stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                               [cpu000: 12%]
```

Can we improve?



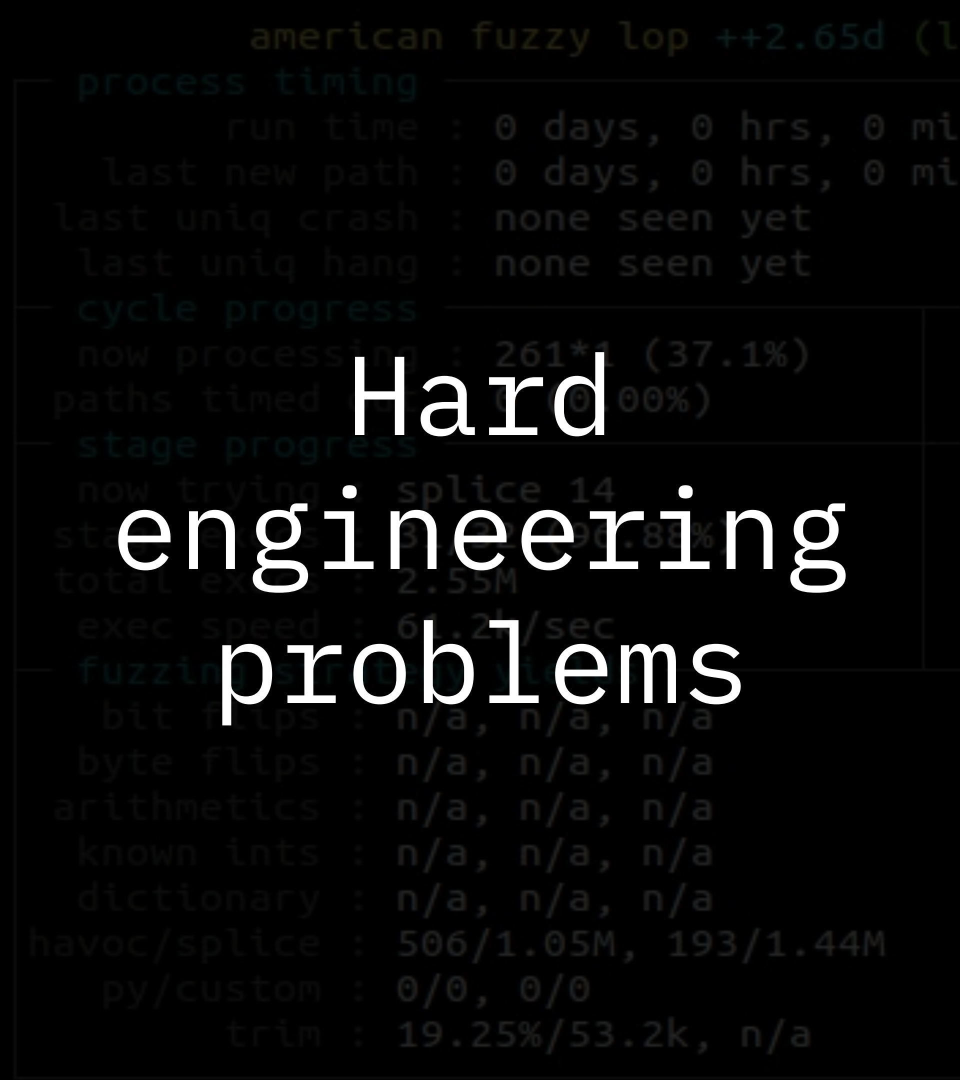
FIXREVERTER: A Realistic Bug Injection Methodology for Benchmarking Fuzz Testing

Authors:
Zenong Zhang and Zach Patterson, *University of Texas at Dallas;* Michael Hicks, *University of Maryland and Amazon;* Shiyi Wei, *University of Texas at Dallas*

Distinguished Paper Award Winner

Abstract:
Fuzz testing is an active area of research with proposed improvements published at a rapid pace. Such proposals are assessed empirically: Can they be shown to perform better than the status quo? Such an assessment requires a benchmark of target programs with well-identified, realistic bugs. To ease the construction of such a benchmark, this paper presents FIXREVERTER, a tool that automatically injects realistic bugs in a program. FIXREVERTER takes as input a bugfix pattern which contains both code syntax and semantic conditions. Any code site that matches the specified syntax is undone if the semantic conditions are satisfied, as checked by static analysis, thus (re)introducing a likely bug. This paper focuses on three bugfix patterns, which we call conditional-abort, conditional-execute, and conditional-assign, based on a study of fixes in a corpus of Common Vulnerabilities and Exposures (CVEs). Using FIXREVERTER we have built REVBUGBENCH, which consists of 10 programs into which we have injected nearly 8,000 bugs; the programs are taken from FuzzBench and Binutils, and represent common targets of fuzzing evaluations. We have integrated REVBUGBENCH into the FuzzBench service, and used it to evaluate five fuzzers. Fuzzing performance varies by fuzzer and program, as desired/expected. Overall, 219 unique bugs were reported, 19% of which were detected by just one fuzzer.

Hard engineering problems

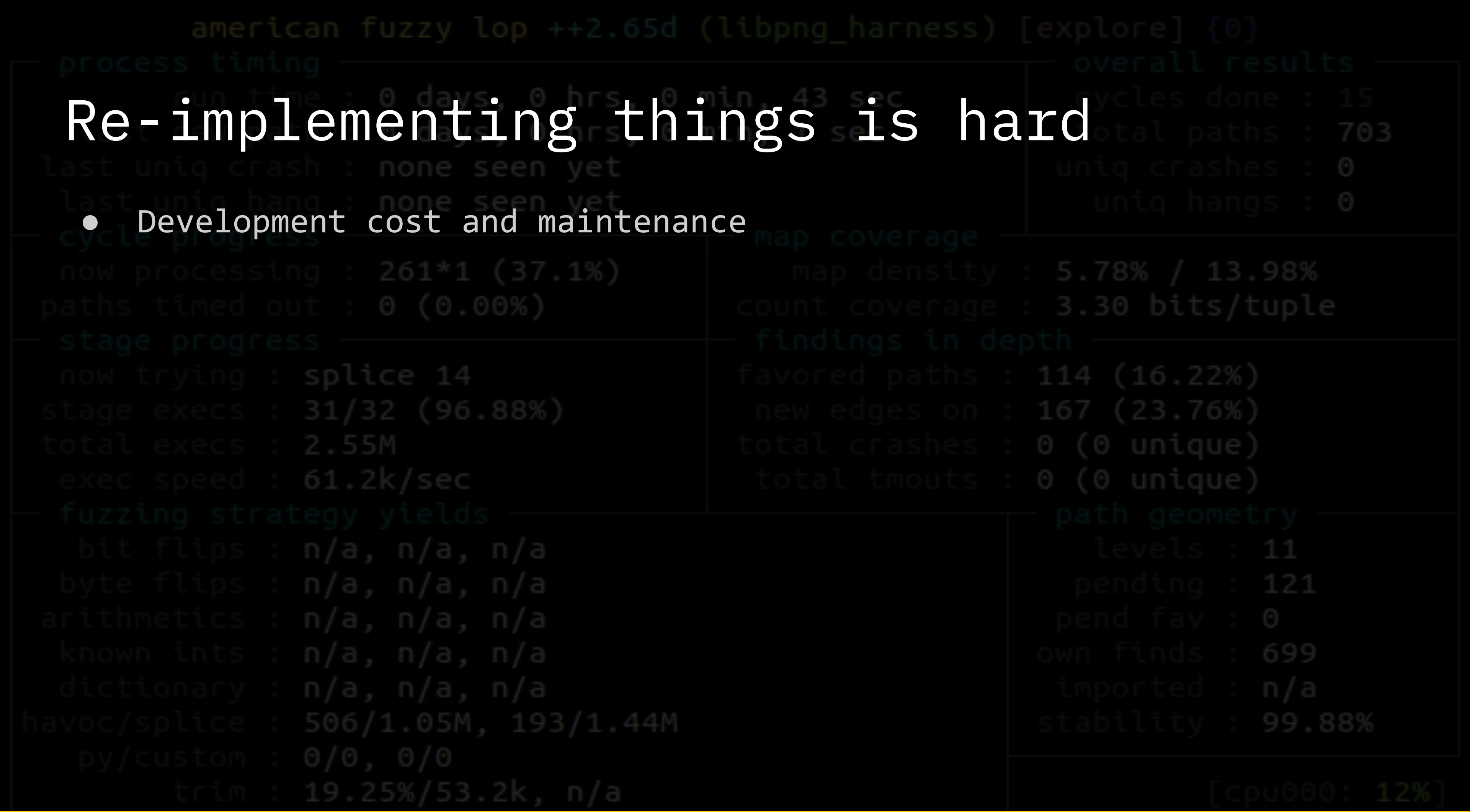


```
       american fuzzy lop ++2.65d (l
  process timing
         run time : 0 days, 0 hrs, 0 mi
    last new path : 0 days, 0 hrs, 0 mi
  last uniq crash : none seen yet
   last uniq hang : none seen yet
  cycle progress
   now processing : 261*1 (37.1%)
  paths timed out : 0 (0.00%)
  stage progress
   now trying : splice 14
  stage execs : 31/32 (96.88%)
  total execs : 2.55M
   exec speed : 61.2k/sec
  fuzzing strategy yields
    bit flips : n/a, n/a, n/a
   byte flips : n/a, n/a, n/a
  arithmetics : n/a, n/a, n/a
   known ints : n/a, n/a, n/a
   dictionary : n/a, n/a, n/a
 havoc/splice : 506/1.05M, 193/1.44M
    py/custom : 0/0, 0/0
         trim : 19.25%/53.2k, n/a
```

There’s a paper about it, problem solved.

Re-implementing things is hard

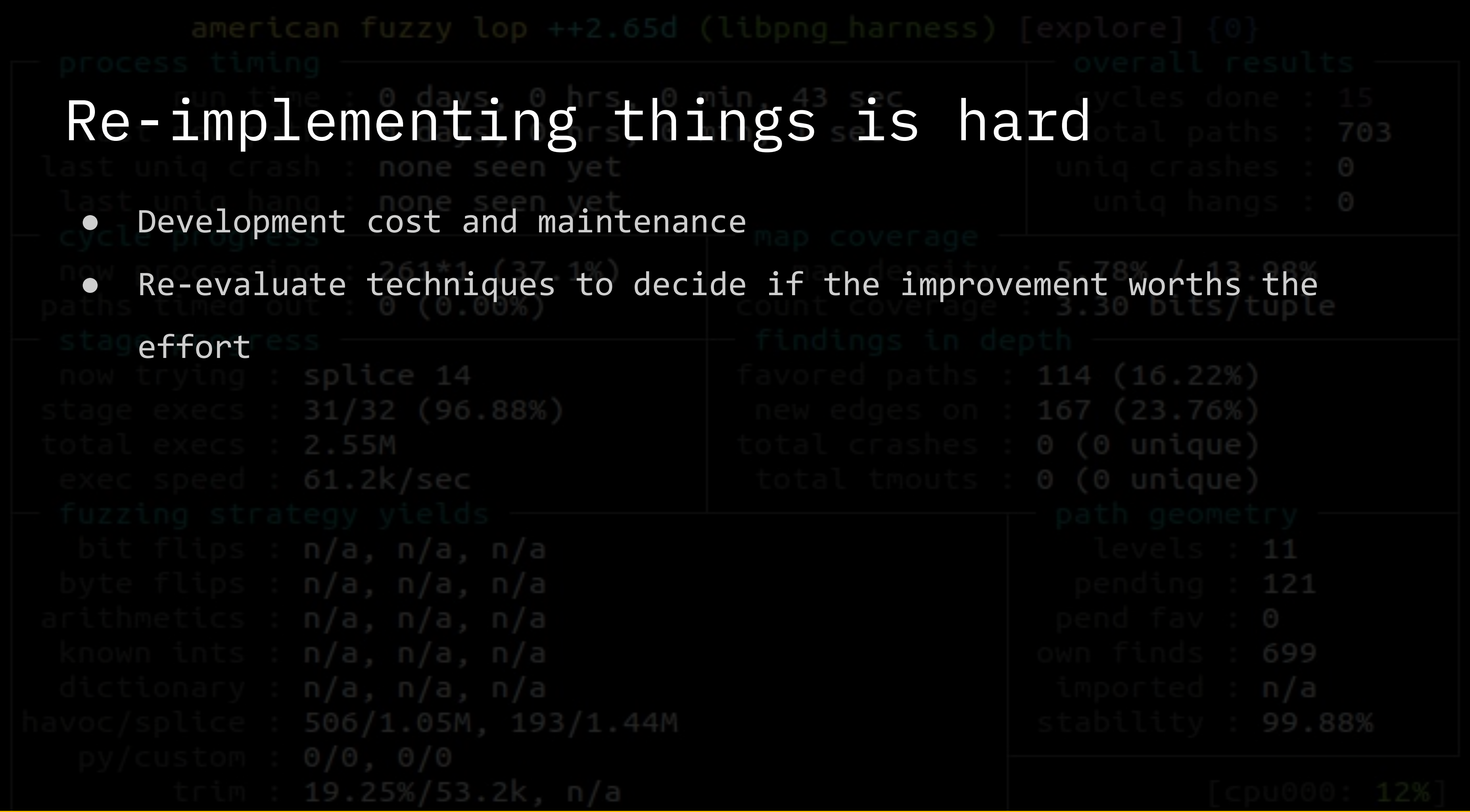
- Development cost and maintenance



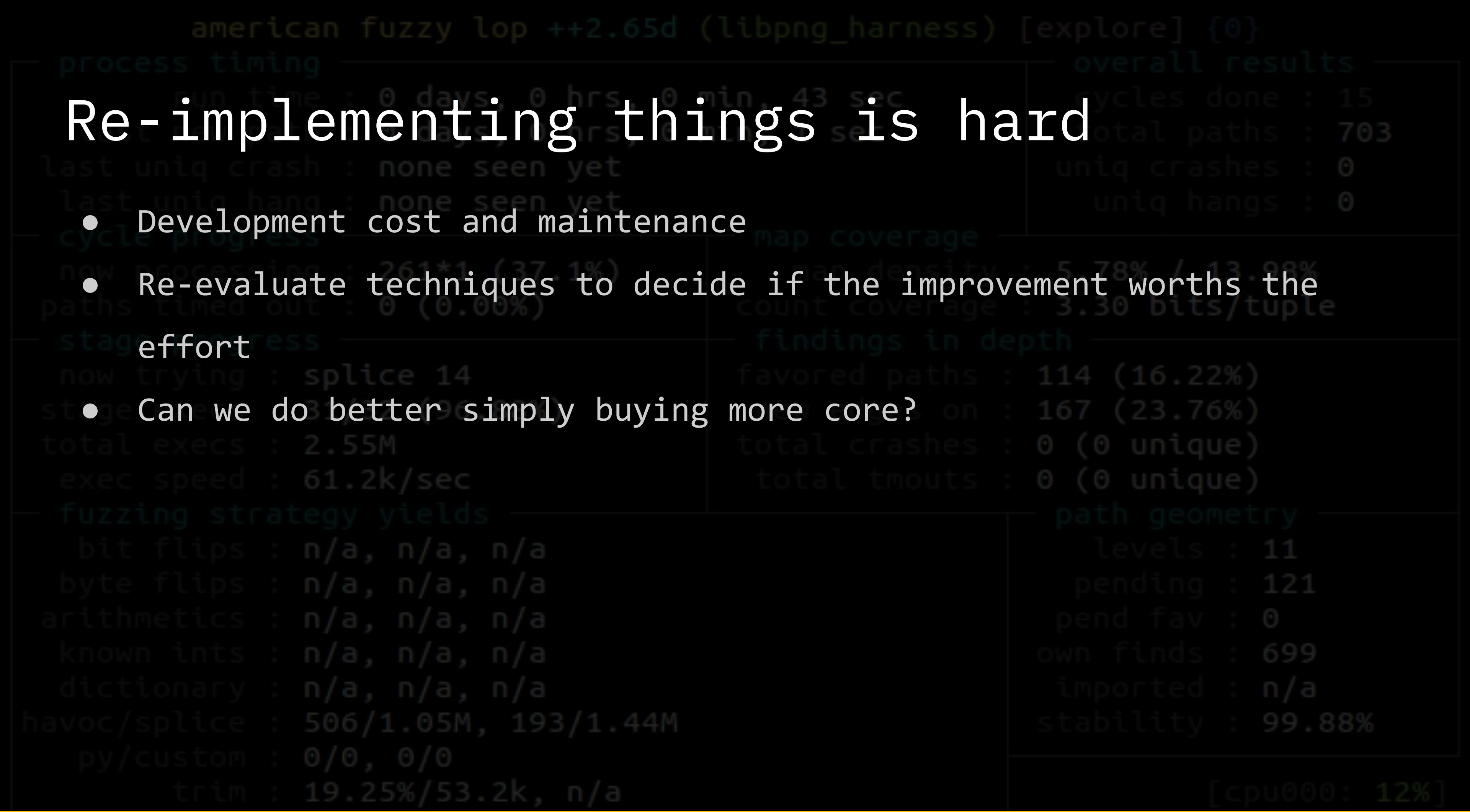
```
            american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                         overall results
         run time : 0 days, 0 hrs, 0 min, 43 sec        cycles done : 15
    [illegible] : [illegible] days, 0 hrs, 0 min, [illegible] sec       total paths : 703
  last uniq crash : none seen yet                      uniq crashes : 0
   last uniq hang : none seen yet                        uniq hangs : 0
  cycle progress                      map coverage
   now processing : 261*1 (37.1%)       map density : 5.78% / 13.98%
  paths timed out : 0 (0.00%)        count coverage : 3.30 bits/tuple
  stage progress                      findings in depth
   now trying : splice 14            favored paths : 114 (16.22%)
  stage execs : 31/32 (96.88%)        new edges on : 167 (23.76%)
  total execs : 2.55M                total crashes : 0 (0 unique)
   exec speed : 61.2k/sec             total tmouts : 0 (0 unique)
  fuzzing strategy yields                              path geometry
    bit flips : n/a, n/a, n/a                            levels : 11
   byte flips : n/a, n/a, n/a                           pending : 121
  arithmetics : n/a, n/a, n/a                          pend fav : 0
   known ints : n/a, n/a, n/a                         own finds : 699
   dictionary : n/a, n/a, n/a                          imported : n/a
 havoc/splice : 506/1.05M, 193/1.44M                  stability : 99.88%
    py/custom : 0/0, 0/0
         trim : 19.25%/53.2k, n/a                                 [cpu000: 12%]
```

Re-implementing things is hard

- Development cost and maintenance
- Re-evaluate techniques to decide if the improvement worths the effort



```
        american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                          overall results
        run time : 0 days, 0 hrs, 0 min, 43 sec           cycles done : 15
   last new path : 0 days, 0 hrs, 0 min, 0 sec           total paths : 703
 last uniq crash : none seen yet                         uniq crashes : 0
  last uniq hang : none seen yet                           uniq hangs : 0
  cycle progress                         map coverage
  now processing : 261*1 (37.1%)           map density : 5.78% / 13.98%
 paths timed out : 0 (0.00%)            count coverage : 3.30 bits/tuple
  stage progress                         findings in depth
  now trying : splice 14                 favored paths : 114 (16.22%)
 stage execs : 31/32 (96.88%)             new edges on : 167 (23.76%)
 total execs : 2.55M                     total crashes : 0 (0 unique)
  exec speed : 61.2k/sec                  total tmouts : 0 (0 unique)
  fuzzing strategy yields                                path geometry
   bit flips : n/a, n/a, n/a                               levels : 11
  byte flips : n/a, n/a, n/a                              pending : 121
 arithmetics : n/a, n/a, n/a                             pend fav : 0
  known ints : n/a, n/a, n/a                            own finds : 699
  dictionary : n/a, n/a, n/a                             imported : n/a
havoc/splice : 506/1.05M, 193/1.44M                     stability : 99.88%
   py/custom : 0/0, 0/0
        trim : 19.25%/53.2k, n/a                              [cpu000: 12%]
```

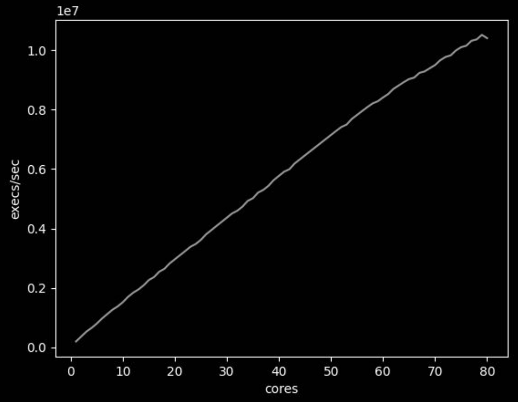


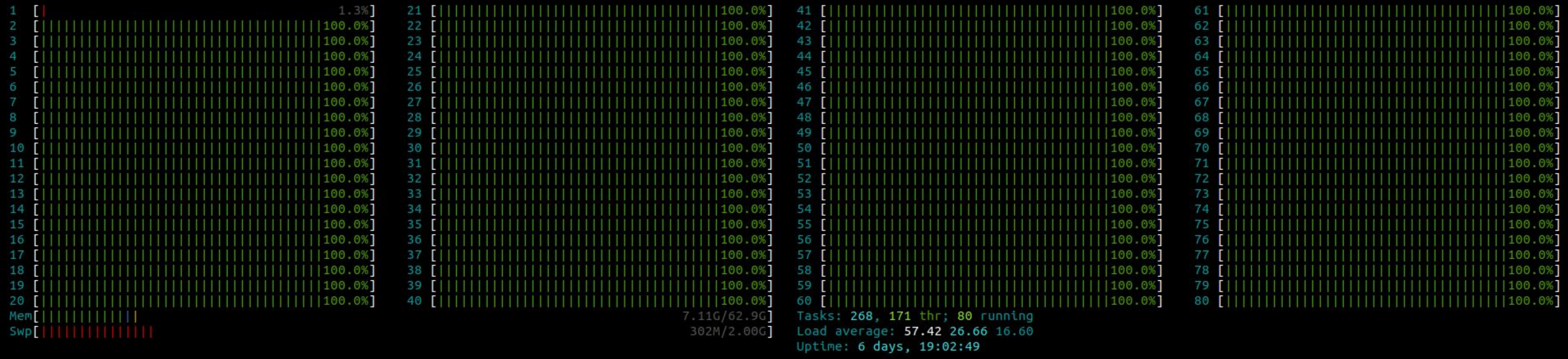
Re-implementing things is hard

- Development cost and maintenance
- Re-evaluate techniques to decide if the improvement worths the effort
- Can we do better simply buying more core?

Re-implementing things is

- Development cost and maintenance
- Re-evaluate techniques to decide if the impr effort
- Can we do better simply buying more core?



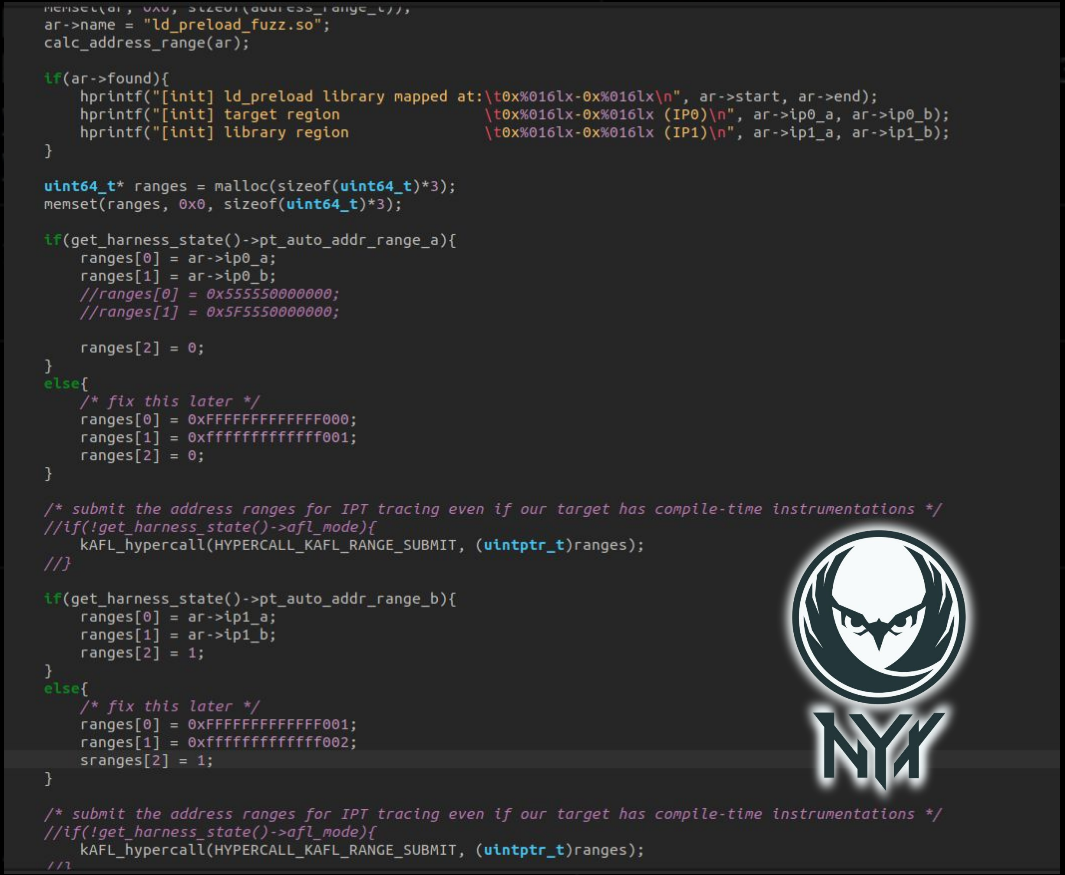


```
  1  [|                                     1.3%]   21 [||||||||||||||||||||||||||||||||||||||100.0%]   41 [||||||||||||||||||||||||||||||||||||||100.0%]   61 [||||||||||||||||||||||||||||||||||||||100.0%]
  2  [||||||||||||||||||||||||||||||||||||||100.0%]   22 [||||||||||||||||||||||||||||||||||||||100.0%]   42 [||||||||||||||||||||||||||||||||||||||100.0%]   62 [||||||||||||||||||||||||||||||||||||||100.0%]
  3  [||||||||||||||||||||||||||||||||||||||100.0%]   23 [||||||||||||||||||||||||||||||||||||||100.0%]   43 [||||||||||||||||||||||||||||||||||||||100.0%]   63 [||||||||||||||||||||||||||||||||||||||100.0%]
  4  [||||||||||||||||||||||||||||||||||||||100.0%]   24 [||||||||||||||||||||||||||||||||||||||100.0%]   44 [||||||||||||||||||||||||||||||||||||||100.0%]   64 [||||||||||||||||||||||||||||||||||||||100.0%]
  5  [||||||||||||||||||||||||||||||||||||||100.0%]   25 [||||||||||||||||||||||||||||||||||||||100.0%]   45 [||||||||||||||||||||||||||||||||||||||100.0%]   65 [||||||||||||||||||||||||||||||||||||||100.0%]
  6  [||||||||||||||||||||||||||||||||||||||100.0%]   26 [||||||||||||||||||||||||||||||||||||||100.0%]   46 [||||||||||||||||||||||||||||||||||||||100.0%]   66 [||||||||||||||||||||||||||||||||||||||100.0%]
  7  [||||||||||||||||||||||||||||||||||||||100.0%]   27 [||||||||||||||||||||||||||||||||||||||100.0%]   47 [||||||||||||||||||||||||||||||||||||||100.0%]   67 [||||||||||||||||||||||||||||||||||||||100.0%]
  8  [||||||||||||||||||||||||||||||||||||||100.0%]   28 [||||||||||||||||||||||||||||||||||||||100.0%]   48 [||||||||||||||||||||||||||||||||||||||100.0%]   68 [||||||||||||||||||||||||||||||||||||||100.0%]
  9  [||||||||||||||||||||||||||||||||||||||100.0%]   29 [||||||||||||||||||||||||||||||||||||||100.0%]   49 [||||||||||||||||||||||||||||||||||||||100.0%]   69 [||||||||||||||||||||||||||||||||||||||100.0%]
  10 [||||||||||||||||||||||||||||||||||||||100.0%]   30 [||||||||||||||||||||||||||||||||||||||100.0%]   50 [||||||||||||||||||||||||||||||||||||||100.0%]   70 [||||||||||||||||||||||||||||||||||||||100.0%]
  11 [||||||||||||||||||||||||||||||||||||||100.0%]   31 [||||||||||||||||||||||||||||||||||||||100.0%]   51 [||||||||||||||||||||||||||||||||||||||100.0%]   71 [||||||||||||||||||||||||||||||||||||||100.0%]
  12 [||||||||||||||||||||||||||||||||||||||100.0%]   32 [||||||||||||||||||||||||||||||||||||||100.0%]   52 [||||||||||||||||||||||||||||||||||||||100.0%]   72 [||||||||||||||||||||||||||||||||||||||100.0%]
  13 [||||||||||||||||||||||||||||||||||||||100.0%]   33 [||||||||||||||||||||||||||||||||||||||100.0%]   53 [||||||||||||||||||||||||||||||||||||||100.0%]   73 [||||||||||||||||||||||||||||||||||||||100.0%]
  14 [||||||||||||||||||||||||||||||||||||||100.0%]   34 [||||||||||||||||||||||||||||||||||||||100.0%]   54 [||||||||||||||||||||||||||||||||||||||100.0%]   74 [||||||||||||||||||||||||||||||||||||||100.0%]
  15 [||||||||||||||||||||||||||||||||||||||100.0%]   35 [||||||||||||||||||||||||||||||||||||||100.0%]   55 [||||||||||||||||||||||||||||||||||||||100.0%]   75 [||||||||||||||||||||||||||||||||||||||100.0%]
  16 [||||||||||||||||||||||||||||||||||||||100.0%]   36 [||||||||||||||||||||||||||||||||||||||100.0%]   56 [||||||||||||||||||||||||||||||||||||||100.0%]   76 [||||||||||||||||||||||||||||||||||||||100.0%]
  17 [||||||||||||||||||||||||||||||||||||||100.0%]   37 [||||||||||||||||||||||||||||||||||||||100.0%]   57 [||||||||||||||||||||||||||||||||||||||100.0%]   77 [||||||||||||||||||||||||||||||||||||||100.0%]
  18 [||||||||||||||||||||||||||||||||||||||100.0%]   38 [||||||||||||||||||||||||||||||||||||||100.0%]   58 [||||||||||||||||||||||||||||||||||||||100.0%]   78 [||||||||||||||||||||||||||||||||||||||100.0%]
  19 [||||||||||||||||||||||||||||||||||||||100.0%]   39 [||||||||||||||||||||||||||||||||||||||100.0%]   59 [||||||||||||||||||||||||||||||||||||||100.0%]   79 [||||||||||||||||||||||||||||||||||||||100.0%]
  20 [||||||||||||||||||||||||||||||||||||||100.0%]   40 [||||||||||||||||||||||||||||||||||||||100.0%]   60 [||||||||||||||||||||||||||||||||||||||100.0%]   80 [||||||||||||||||||||||||||||||||||||||100.0%]
  Mem[||||||||||||                                                                         7.11G/62.9G]   Tasks: 268, 171 thr; 80 running
  Swp[|||||||||||||||                                                                       302M/2.00G]   Load average: 57.42 26.66 16.60
                                                                                                          Uptime: 6 days, 19:02:49
```

Re-implementing things is hard

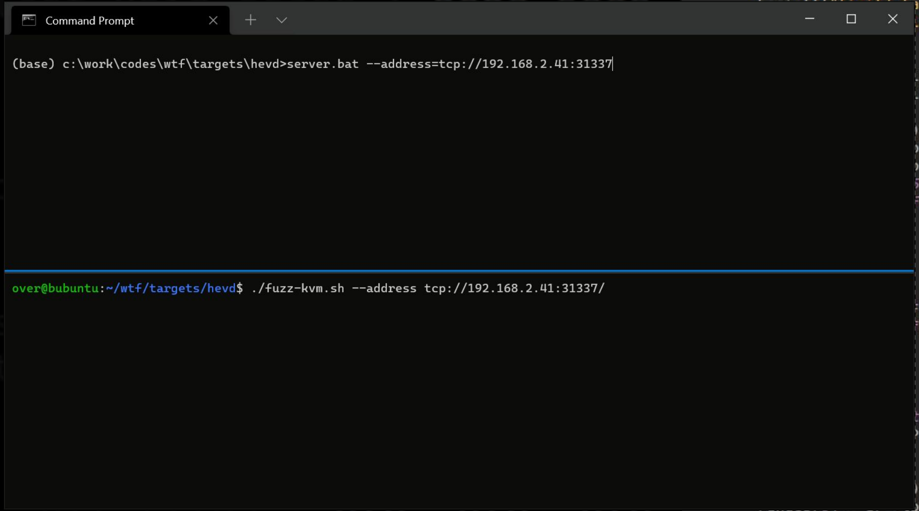
- Development cost and maintenance
- Re-evaluate techniques to decide if the improvement worths the effort
- Can we do better simply buying more core?
- Lack of production-ready engines for tracing/instrumentation of exotic targets

Hard targets



```
memset(ar, 0x0, sizeof(address_range_t));
ar->name = "ld_preload_fuzz.so";
calc_address_range(ar);

if(ar->found){
    hprintf("[init] ld_preload library mapped at:\t0x%016lx-0x%016lx\n", ar->start, ar->end);
    hprintf("[init] target region                \t0x%016lx-0x%016lx (IP0)\n", ar->ip0_a, ar->ip0_b);
    hprintf("[init] library region               \t0x%016lx-0x%016lx (IP1)\n", ar->ip1_a, ar->ip1_b);
}

uint64_t* ranges = malloc(sizeof(uint64_t)*3);
memset(ranges, 0x0, sizeof(uint64_t)*3);

if(get_harness_state()->pt_auto_addr_range_a){
    ranges[0] = ar->ip0_a;
    ranges[1] = ar->ip0_b;
    //ranges[0] = 0x555550000000;
    //ranges[1] = 0x5F5550000000;

    ranges[2] = 0;
}
else{
    /* fix this later */
    ranges[0] = 0xFFFFFFFFFFFFF000;
    ranges[1] = 0xfffffffffffff001;
    ranges[2] = 0;
}

/* submit the address ranges for IPT tracing even if our target has compile-time instrumentations */
//if(!get_harness_state()->afl_mode){
    kAFL_hypercall(HYPERCALL_KAFL_RANGE_SUBMIT, (uintptr_t)ranges);
//}

if(get_harness_state()->pt_auto_addr_range_b){
    ranges[0] = ar->ip1_a;
    ranges[1] = ar->ip1_b;
    ranges[2] = 1;
}
else{
    /* fix this later */
    ranges[0] = 0xFFFFFFFFFFFFF001;
    ranges[1] = 0xfffffffffffff002;
    sranges[2] = 1;
}

/* submit the address ranges for IPT tracing even if our target has compile-time instrumentations */
//if(!get_harness_state()->afl_mode){
    kAFL_hypercall(HYPERCALL_KAFL_RANGE_SUBMIT, (uintptr_t)ranges);
//}
```

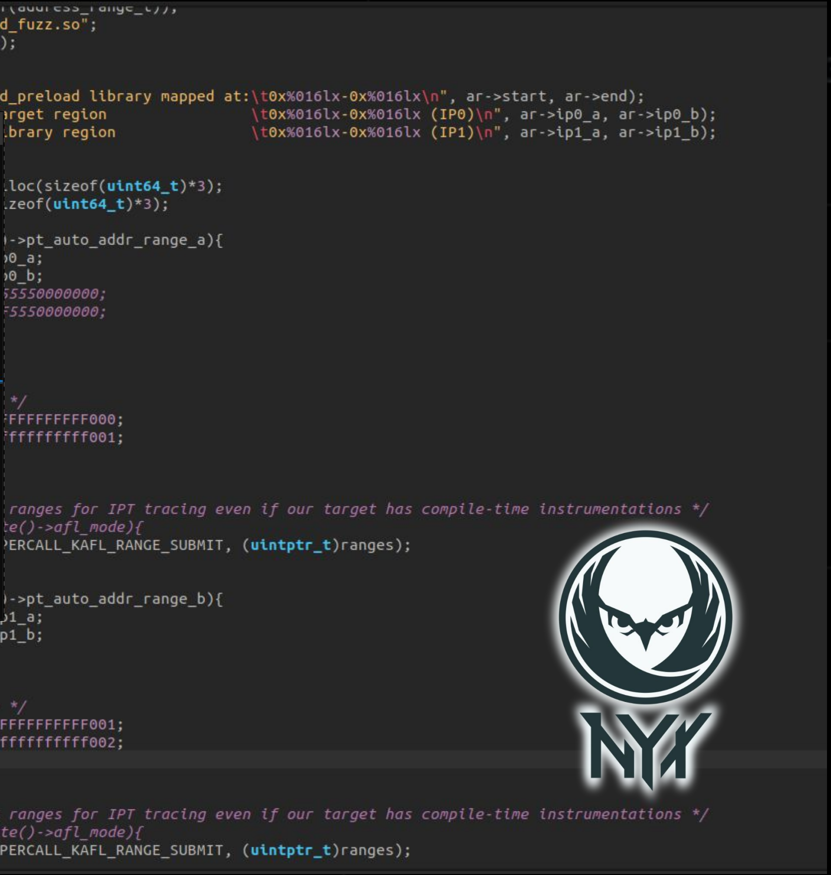
Hard targets



```
(base) c:\work\codes\wtf\targets\hevd>server.bat --address=tcp://192.168.2.41:31337
```

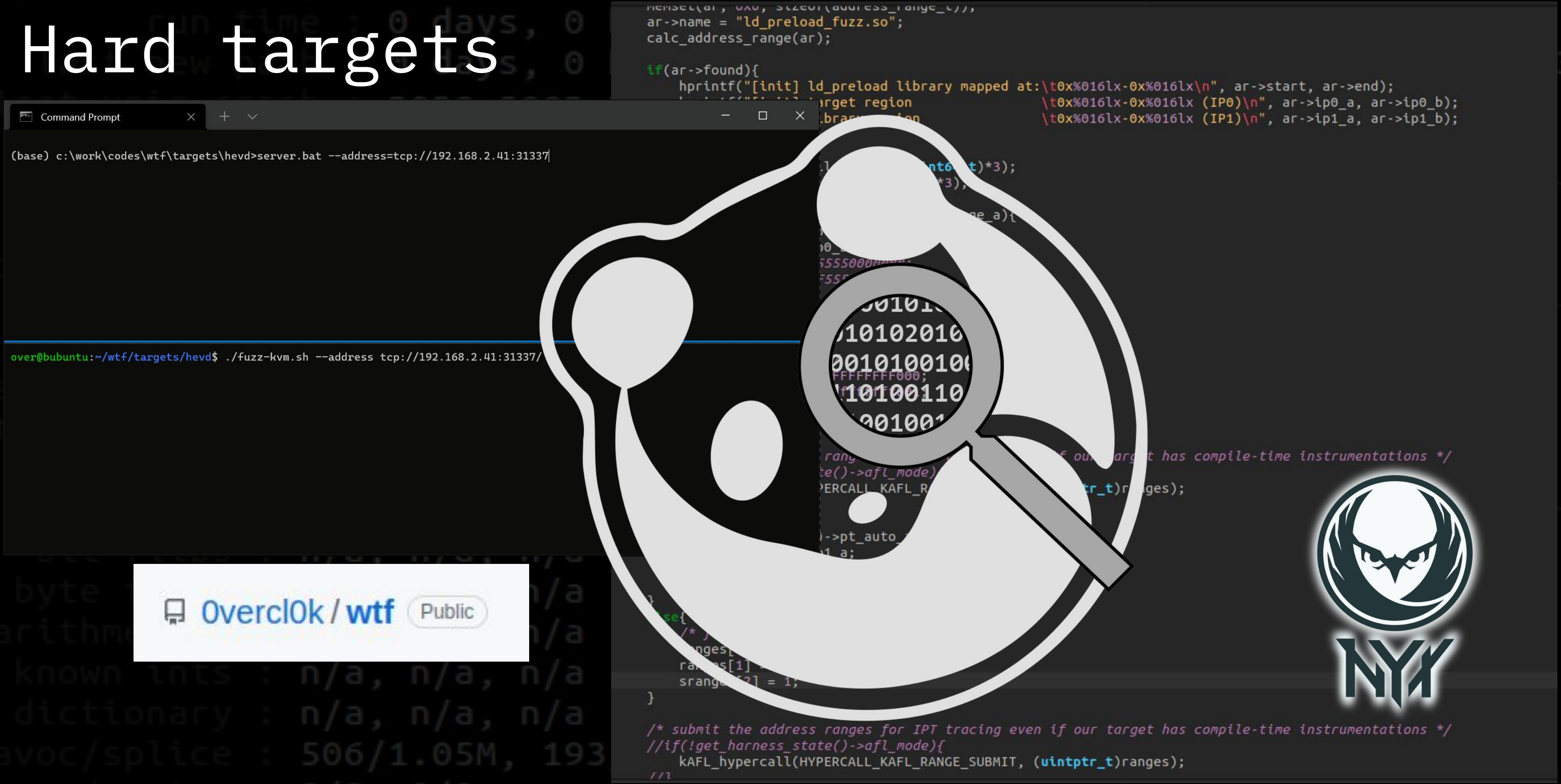
```
over@bubuntu:~/wtf/targets/hevd$ ./fuzz-kvm.sh --address tcp://192.168.2.41:31337/
```

Overcl0k / wtf Public



```
    ar->name = "ld_preload_fuzz.so";
    calc_address_range(ar);

    if(ar->found){
        hprintf("[init] ld_preload library mapped at:\t0x%016lx-0x%016lx\n", ar->start, ar->end);
                    rget region                  \t0x%016lx-0x%016lx (IP0)\n", ar->ip0_a, ar->ip0_b);
                    brary region                 \t0x%016lx-0x%016lx (IP1)\n", ar->ip1_a, ar->ip1_b);

                    loc(sizeof(uint64_t)*3);
                    zeof(uint64_t)*3);

                    ->pt_auto_addr_range_a){
                    0_a;
                    0_b;
                    5555000000;
                    F5550000000;



                     */
                    FFFFFFFFF000;
                    fffffffff001;



                     ranges for IPT tracing even if our target has compile-time instrumentations */
                    e()->afl_mode){
                    PERCALL_KAFL_RANGE_SUBMIT, (uintptr_t)ranges);

                    ->pt_auto_addr_range_b){
                    p1_a;
        ranges[1] = ar->ip1_b;
        ranges[2] = 1;
    }
    else{
        /* fix this later */
        ranges[0] = 0xFFFFFFFFFFFFF001;
        ranges[1] = 0xfffffffffffff002;
        sranges[2] = 1;
    }

    /* submit the address ranges for IPT tracing even if our target has compile-time instrumentations */
    //if(!get_harness_state()->afl_mode){
        kAFL_hypercall(HYPERCALL_KAFL_RANGE_SUBMIT, (uintptr_t)ranges);
```

Hard targets



```
(base) c:\work\codes\wtf\targets\hevd>server.bat --address=tcp://192.168.2.41:31337
```

```
over@bubuntu:~/wtf/targets/hevd$ ./fuzz-kvm.sh --address tcp://192.168.2.41:31337/
```

0vercl0k / wtf Public

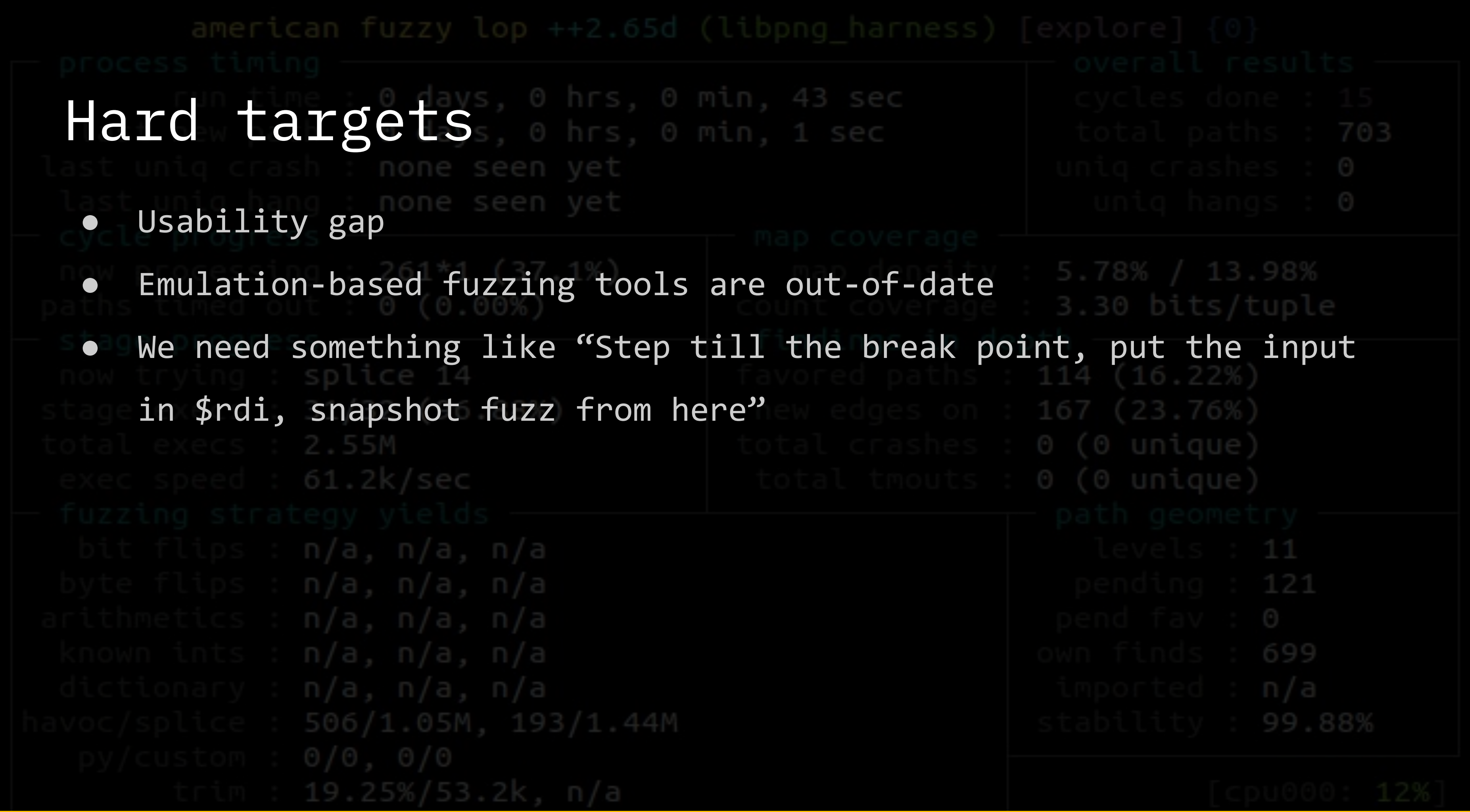
```
    ar->name = "ld_preload_fuzz.so";
    calc_address_range(ar);

    if(ar->found){
        hprintf("[init] ld_preload library mapped at:\t0x%016lx-0x%016lx\n", ar->start, ar->end);
                         arget region               \t0x%016lx-0x%016lx (IP0)\n", ar->ip0_a, ar->ip0_b);
                         brar                       \t0x%016lx-0x%016lx (IP1)\n", ar->ip1_a, ar->ip1_b);
```

```
    }
    else{
        ranges[1]
        sranges[2] = 1;
    }

    /* submit the address ranges for IPT tracing even if our target has compile-time instrumentations */
    //if(!get_harness_state()->afl_mode){
        kAFL_hypercall(HYPERCALL_KAFL_RANGE_SUBMIT, (uintptr_t)ranges);
```

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
  process timing                                         overall results
        run time : 0 days, 0
   havoc/splice : 506/1.05M, 193
      py/custom : 0/0, 0/0
           trim : 19.25%/53.2k, n/a
    known ints : n/a, n/a, n/a
    dictionary : n/a, n/a, n/a
                                                          [cpu000: 12%]
```

Hard targets

- Usability gap
- Emulation-based fuzzing tools are out-of-date
- We need something like “Step till the break point, put the input in $rdi, snapshot fuzz from here”



```
          american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
┌─ process timing ─────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 0 min, 43 sec        │  cycles done : 15    │
│   last new path : 0 days, 0 hrs, 0 min, 1 sec         │  total paths : 703   │
│ last uniq crash : none seen yet                       │ uniq crashes : 0     │
│  last uniq hang : none seen yet                       │   uniq hangs : 0     │
├─ cycle progress ───────────────────┬─ map coverage ───┴──────────────────────┤
│  now processing : 261*1 (37.1%)     │    map density : 5.78% / 13.98%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 3.30 bits/tuple        │
├─ stage progress ───────────────────┼─ findings in depth ─────────────────────┤
│  now trying : splice 14             │ favored paths : 114 (16.22%)            │
│ stage execs : [illegible]           │  new edges on : 167 (23.76%)            │
│ total execs : 2.55M                 │ total crashes : 0 (0 unique)            │
│  exec speed : 61.2k/sec             │  total tmouts : 0 (0 unique)            │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                          │    levels : 11         │
│  byte flips : n/a, n/a, n/a                          │   pending : 121        │
│ arithmetics : n/a, n/a, n/a                          │  pend fav : 0          │
│  known ints : n/a, n/a, n/a                          │ own finds : 699        │
│  dictionary : n/a, n/a, n/a                          │  imported : n/a        │
│havoc/splice : 506/1.05M, 193/1.44M                   │ stability : 99.88%     │
│  py/custom : 0/0, 0/0                                ├────────────────────────┘
│        trim : 19.25%/53.2k, n/a                      │          [cpu000: 12%]
```

```
american fuzzy lop ++2.65d (libpng_harness) [explore] {0}
process timing                                          overall results
       run time : 0 days, 0 hrs, 0 min, 43 sec           cycles done : 15
                                                         total paths : 703
last uniq crash : none seen yet                         uniq crashes : 0
 last uniq hang : none seen yet                           uniq hangs : 0
cycle progress
now processing : 261*1 (37.1%)            map density : 5.78% / 13.98%
paths timed out : 0 (0.00                                  3.30 bits/tuple
stage progress                                  depth
now trying : splice 14                               : 114 (16.22%)
stage execs : 31/32 (96.                             : 167 (23.76%)
total execs : 2.55M                                  : 0 (0 unique)
 exec speed : 61.2k/sec                              : 0 (0 unique)
fuzzing strategy yields                          path geometry
  bit flips : n/a, n/a,                              levels : 11
 byte flips : n/a, n/a,                             pending : 121
arithmetics : n/a, n/a,                            pend fav : 0
 known ints : n/a, n/a,                           own finds : 699
 dictionary : n/a, n/a,                            imported : n/a
havoc/splice : 506/1.05M,                          stability : 99.88%
  py/custom : 0/0, 0/0
       trim : 19.25%/53.2k, n/a                              [cpu000: 12%]
```

Ask more about fuzzing at

https://discord.gg/gCraWct

Thanks y’all