Fuzzing

Ronnie Eytchison 03/21/2023

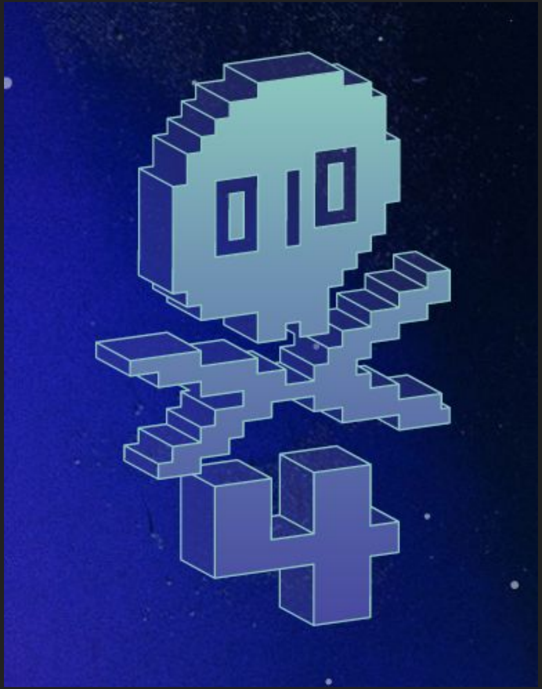
discord.osucyber.club

attend.osucyber.club

CYBER SECURITY CLUB
@ OHIO STATE

Announcements

- Hack-a-sat Quals April 1-2
- Finalists will hack a REAL ORBITING SATELLITE



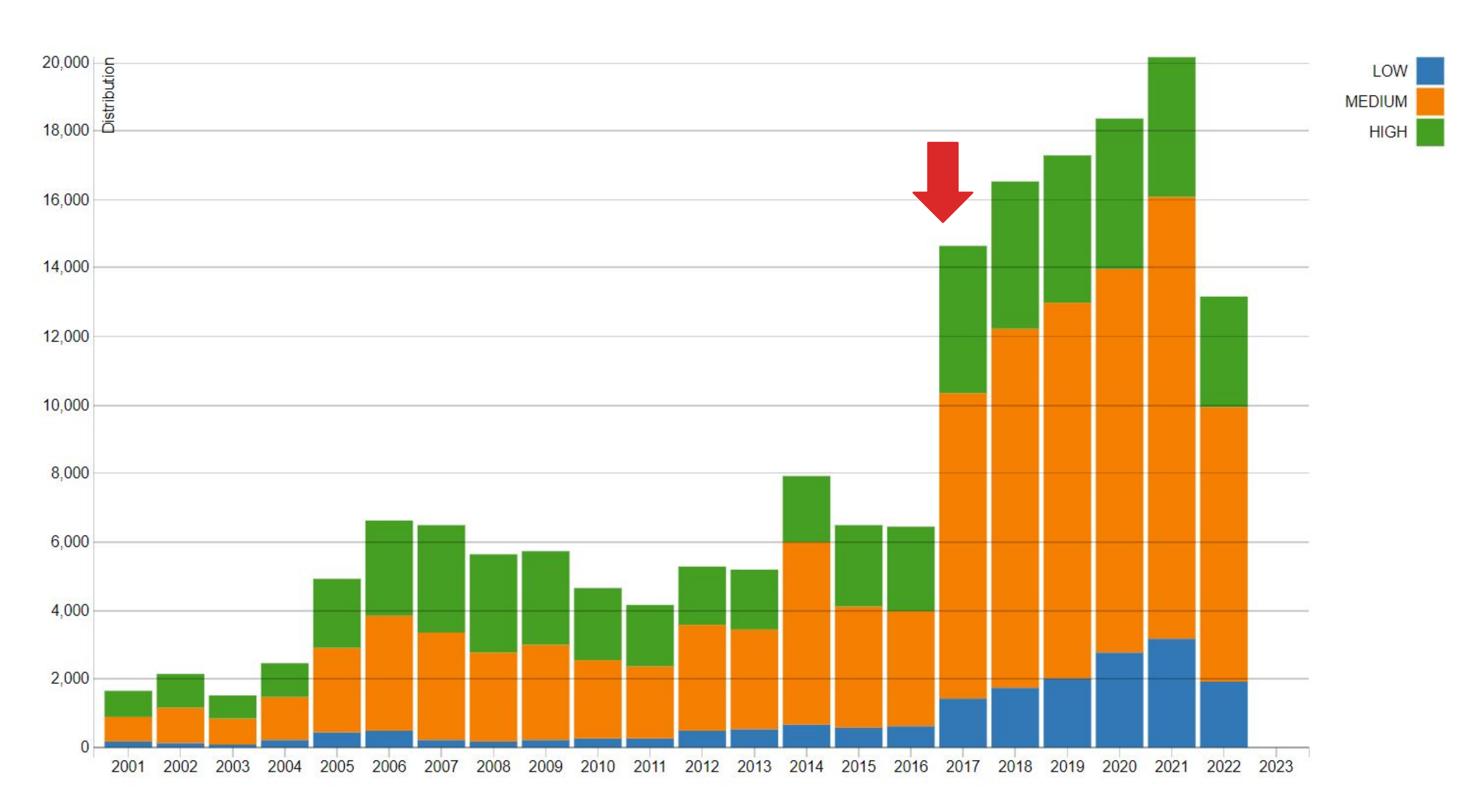
What is Fuzzing

- Automatically test programs with random inputs
  - Typically much smarter than purely random
- Feed inputs to program
- Observe program if misbehaves
- Save inputs which trigger bugs in the program

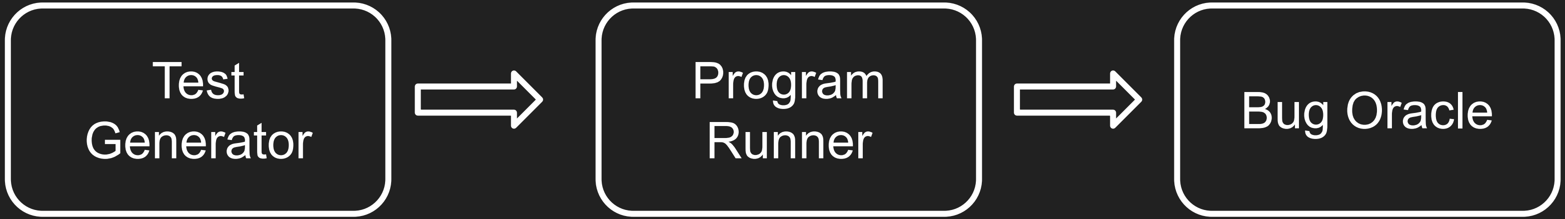
Why fuzzing?

- Humans can only create a limited number of test cases
- The inputs which cause bugs may be unintuitive
- Fuzzers have no false positives
  - Every flagged input crashed/caused a bug in the program
- Fuzzing works in the real world!

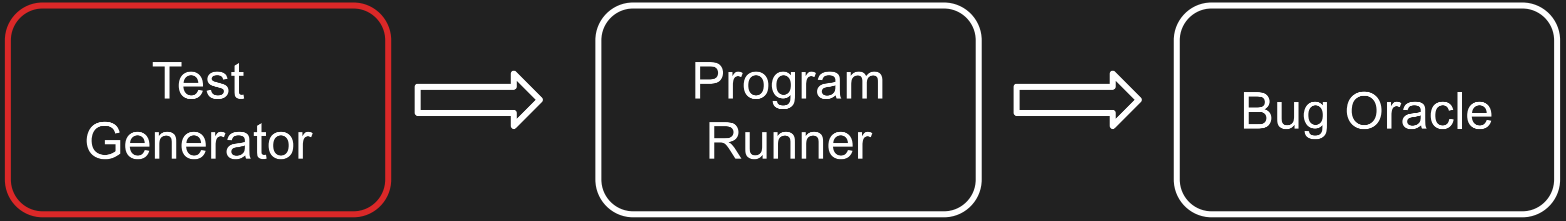
Impact of Fuzzers: More CVEs per year



Parts of a Fuzzer

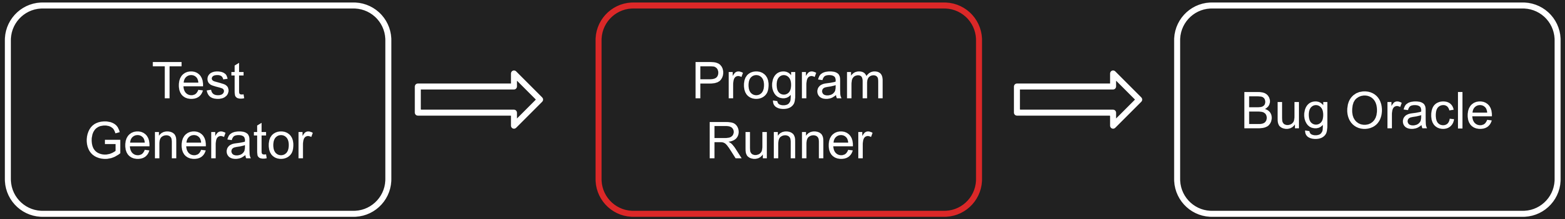


Parts of a Fuzzer: Test Generator



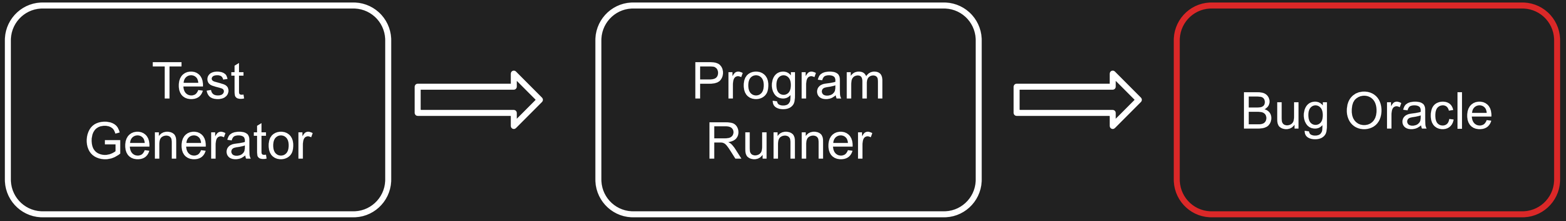
- Generate smart test cases (hint: not purely random)
- Techniques:
  - Mutate from existing inputs
  - Analyze program to synthesize inputs
  - Use heuristics to identify if a new input is “better”
  - E.g. new code coverage

Parts of a Fuzzer: Program Runner



- Run many tests quickly
- Different from the “harness” which is the actual binary being tested

Parts of a Fuzzer: Bug Oracle



- Identify when a bug occurs
- Many approaches:
  - Detect crashes via signals
  - Address Sanitizer
  - Incorrect output

Classes of Fuzzers

- Black-box: can only see output of program
- Grey-box: limited introspection of program state
  - E.g. from code instrumentation
- White-box: total knowledge of state
  - High overhead

AFL

- One of the most popular fuzzers
- Uses code coverage and genetic algorithm for test cases
- Grey-box fuzzer
- Identifies bugs with crashes
- Supports fuzzing stdin or file input

American Fuzzy Lop

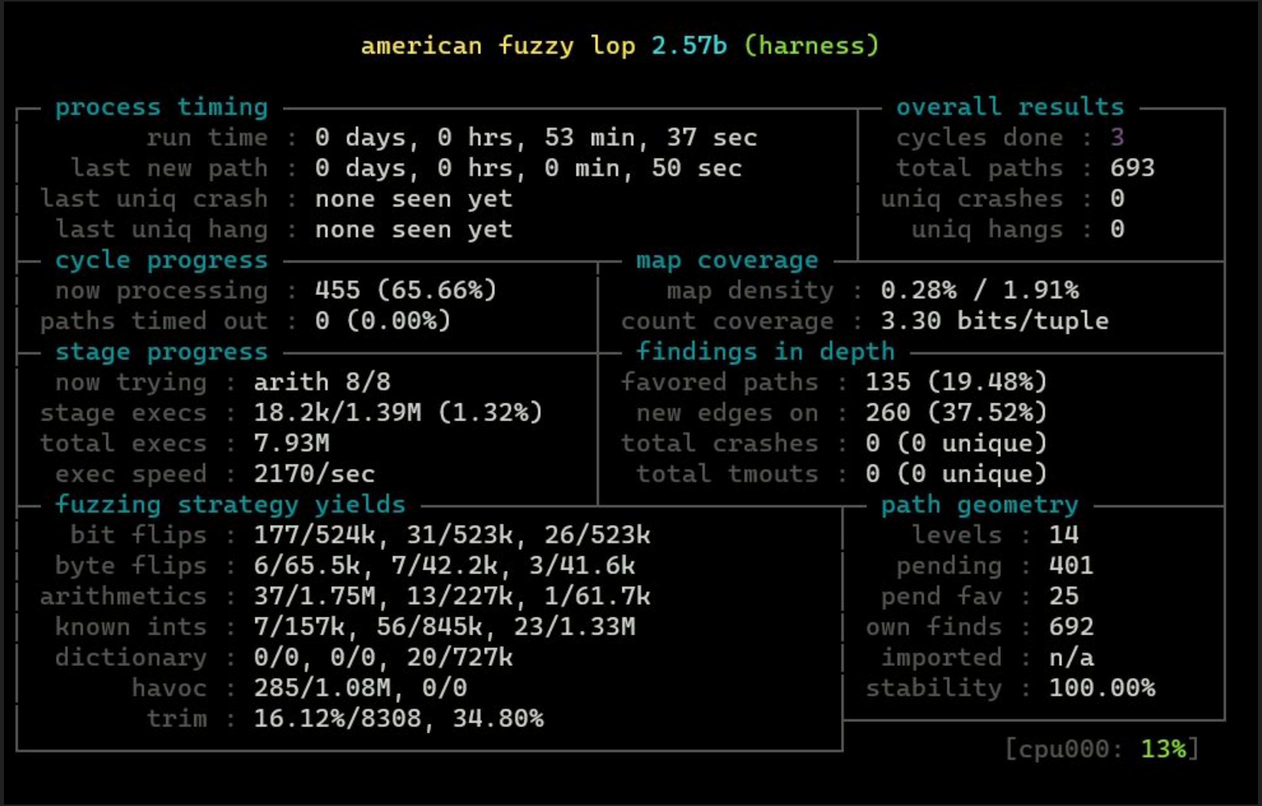
Using AFL

- Clone and install AFL:

```
git clone git@github.com:google/AFL.git
cd AFL && make && make install
```

- Compile and instrument your test program with afl-gcc:
  - Typically with: `CC=afl-gcc ./configure --disable-shared`
- Fuzz with afl-fuzz
  - Stdin: `afl-fuzz -i test_dir -o output_dir ./prog [prog args]`
  - File: `afl-fuzz -i test_dir -o output_dir ./prog [prog args] @@`

Sample AFL Interface



```
                  american fuzzy lop 2.57b (harness)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 53 min, 37 sec      │  cycles done : 3      │
│   last new path : 0 days, 0 hrs, 0 min, 50 sec       │  total paths : 693    │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 455 (65.66%)      │    map density : 0.28% / 1.91%         │
│ paths timed out : 0 (0.00%)         │ count coverage : 3.30 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : arith 8/8             │ favored paths : 135 (19.48%)           │
│ stage execs : 18.2k/1.39M (1.32%)   │  new edges on : 260 (37.52%)           │
│ total execs : 7.93M                 │ total crashes : 0 (0 unique)           │
│  exec speed : 2170/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 177/524k, 31/523k, 26/523k            │    levels : 14         │
│  byte flips : 6/65.5k, 7/42.2k, 3/41.6k             │   pending : 401        │
│ arithmetics : 37/1.75M, 13/227k, 1/61.7k            │  pend fav : 25         │
│  known ints : 7/157k, 56/845k, 23/1.33M             │ own finds : 692        │
│  dictionary : 0/0, 0/0, 20/727k                     │  imported : n/a        │
│       havoc : 285/1.08M, 0/0                        │ stability : 100.00%    │
│        trim : 16.12%/8308, 34.80%                   ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000: 13%]
```

Challenge: Find a real CVE with AFL!

- Download and extract fuzz-mtg-sp23.tar.gz from Discord
- Build zlib-1.2.12 archive

```
CC=afl-gcc ./configure --static
```

```
make
```

- Build target program:

```
afl-gcc -o minigzip-vuln -g -DZ_SOLO -I zlib-1.2.12/ minigzip-vuln.c zlib-1.2.12/libz.a
```

- Fuzz starting with seed.gz
  - Should take around 15 minutes to find a crash!

Afterword: Fuzzers are not magic!

- Larger seeds are much slower
- Finding root cause of a fuzzing bug takes time
- More challenging to fuzz programs without easy I/O interface
- Bypassing barriers like checksums and cryptography
- Tradeoff between “smart and slow” and “dumb and fast”
- Can take time to make a good harness
- Instrumentation from fuzzers can make bugs possible/impossible

Resources

- https://lcamtuf.coredump.cx/afl/technical_details.txt
- https://www.fuzzingbook.org/
- https://wcventure.github.io/FuzzingPaper/
- https://afl-1.readthedocs.io/en/latest/