The Art of Fuzzing Without Fuzzing

@rantyben

Kiwicon 2015

Fuzzing
(A History)

The year is 2002…

original photocomposite created by yourwallpaper.com
spike
Buffy
the vampire slayer
visit the official buffy site at http://www.buffy.com

Whitebox, Greybox, Polka Dots

- **SAGE** Godefroid / MS Research
- **EFS** DeMott
- **Sidewinder** Embleton, Sparks, Cunningham
- **Bunny the Fuzzer** Zalewski
- **Flayer** Ormandy
- **KLEE** Cadar et al
- … many more

Whitebox, Greybox, Polka Dots

1. Instrument Stuff
2. Do things
3. Measure
4. Do better things

Problems

- State Explosion
- Devil In Details
- Performance
- Limited Availability
- Fragile Code
- Artificial Results
- Require Source

Charlie Miller

The “Army of Monkeys” Talk

- CSW 2010
- Collected ~80k PDFs
- Traced for coverage, reduced to ~1500
- Fuzzed several parsers with “millerfuzz”
- ~7 million iterations TOTAL (!!)
- Drowned in bugs

+

STAR TREK

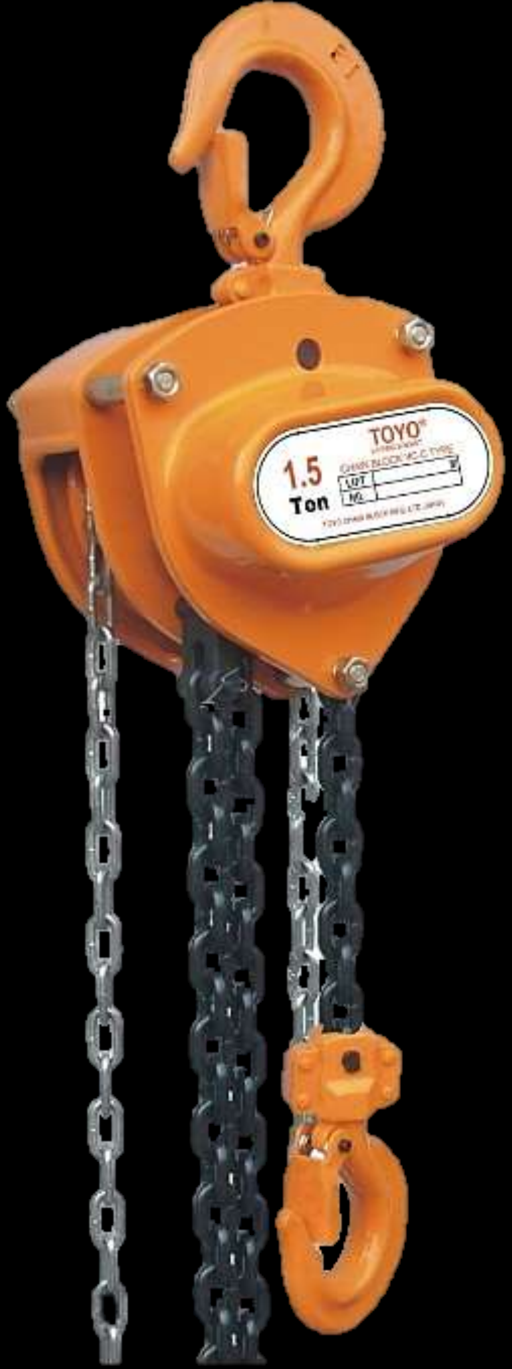
=

Prospector / BM3

- Me, 2010
- Scraper using Bing
- Tracer using Pin
- Distributed VM based fuzz cluster
- ~200 VMs, peak speeds ~50t/s
- Scales to 100s of Mt, works on Windows

Problems

- Scraping slow
- Office is slow
- VMs are slow
- “Real” files don’t exercise much code
- Millerfuzz is a pretty bad mutator
- Bug probability is constant


TOYO
CHAIN BLOCK VC-C TYPE
1.5
Ton
LOT
NO.
TOYO CHAIN BLOCK MFG. LTD. JAPAN

“Block” Based Fuzzing?

- Invented by Archimedes ~250BC
- Acts like Gears + Ratchet
- One Human easily lifts 1-2 tons

- Force Multiplier + Hill Climbing

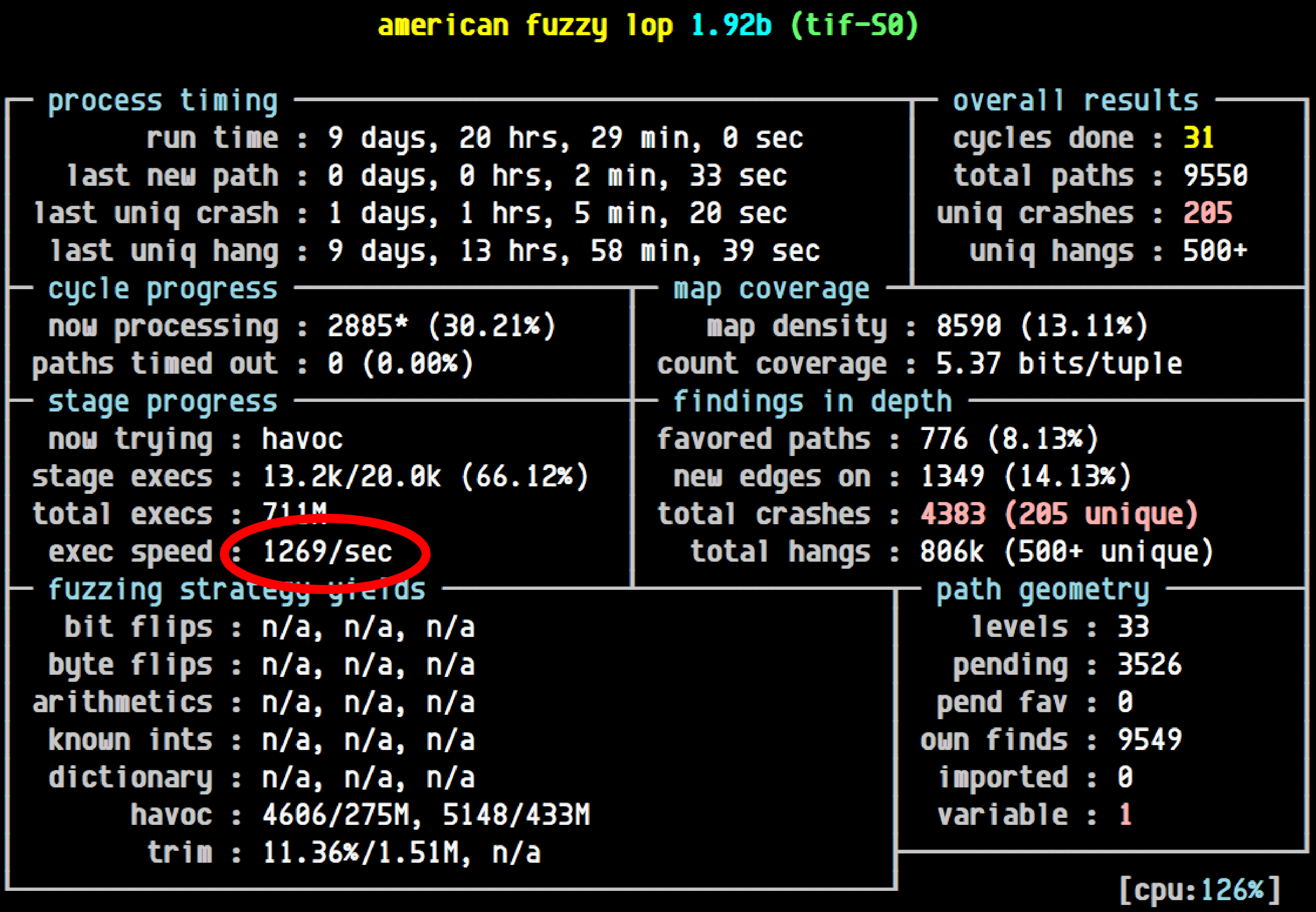
roar.

AFL - It’s What’s Up.

- Instrument source or binary
- Dumb fuzz, get coverage
- Tests that add coverage are kept
- Naturally evolves towards max coverage
- Can appear terrifyingly intelligent

AFL - It’s What’s Up.

- Novel coverage approach (bitmap)
- VERY VERY VERY FAST
- Well tested dumb mutators
- Zero configuration
- No academic silliness
- Breaks $#$%@* Everything



```
                     american fuzzy lop 1.92b (tif-S0)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 9 days, 20 hrs, 29 min, 0 sec       │  cycles done : 31     │
│   last new path : 0 days, 0 hrs, 2 min, 33 sec        │  total paths : 9550   │
│ last uniq crash : 1 days, 1 hrs, 5 min, 20 sec        │ uniq crashes : 205    │
│  last uniq hang : 9 days, 13 hrs, 58 min, 39 sec      │   uniq hangs : 500+   │
├─ cycle progress ─────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 2885* (30.21%)     │    map density : 8590 (13.11%)         │
│ paths timed out : 0 (0.00%)          │ count coverage : 5.37 bits/tuple       │
├─ stage progress ─────────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                  │ favored paths : 776 (8.13%)            │
│ stage execs : 13.2k/20.0k (66.12%)   │  new edges on : 1349 (14.13%)          │
│ total execs : 711M                   │ total crashes : 4383 (205 unique)      │
│  exec speed : 1269/sec               │   total hangs : 806k (500+ unique)     │
├─ fuzzing strategy yields ────────────┴──────────────┬─ path geometry ────────┤
│   bit flips : n/a, n/a, n/a                          │    levels : 33         │
│  byte flips : n/a, n/a, n/a                          │   pending : 3526       │
│ arithmetics : n/a, n/a, n/a                          │  pend fav : 0          │
│  known ints : n/a, n/a, n/a                          │ own finds : 9549       │
│  dictionary : n/a, n/a, n/a                          │  imported : 0          │
│       havoc : 4606/275M, 5148/433M                   │  variable : 1          │
│        trim : 11.36%/1.51M, n/a                      ├────────────────────────┘
└──────────────────────────────────────────────────────┘             [cpu:126%]
```

Welcome to the Gt Era

- < 10Mt is now insignificant
- 1Gt MINIMUM per parser
- < 24 hrs with a decent box
- …but Reader is still slow!

Corpus Driven Fuzzing

- MS15-024 / MS15-029 (lcamtuf)
- IE bugs handling JXR and PNG
- Found without ever fuzzing IE!

Corpus Driven Fuzzing

- Really good corpora are **AMAZING**
- AFL creates really good corpora
- Quickly*
- With Zero* Configuration
- Bonus - they’re re-useable!

Building Corpora with AFL

- NOT aimed at finding bugs
- Crashy targets are actually WORSE
- Focus is on exercising code / spec

AFL "Crash" Course

1. Build AFL
2. Build target with AFL compilers
3. Run AFL on instrumented target

Target Selection Tips

- Multiple engines
- Simple apps
  - converters
  - extractors
- Code examples
- Good / Complete code!

The Perfect AFL Target

- Complete protocol support
- Fine-grained error handling
- Exits when done
- Reads STDIN, outputs to STDOUT
- Accepts arbitrary filenames

Instrumentation Tricks

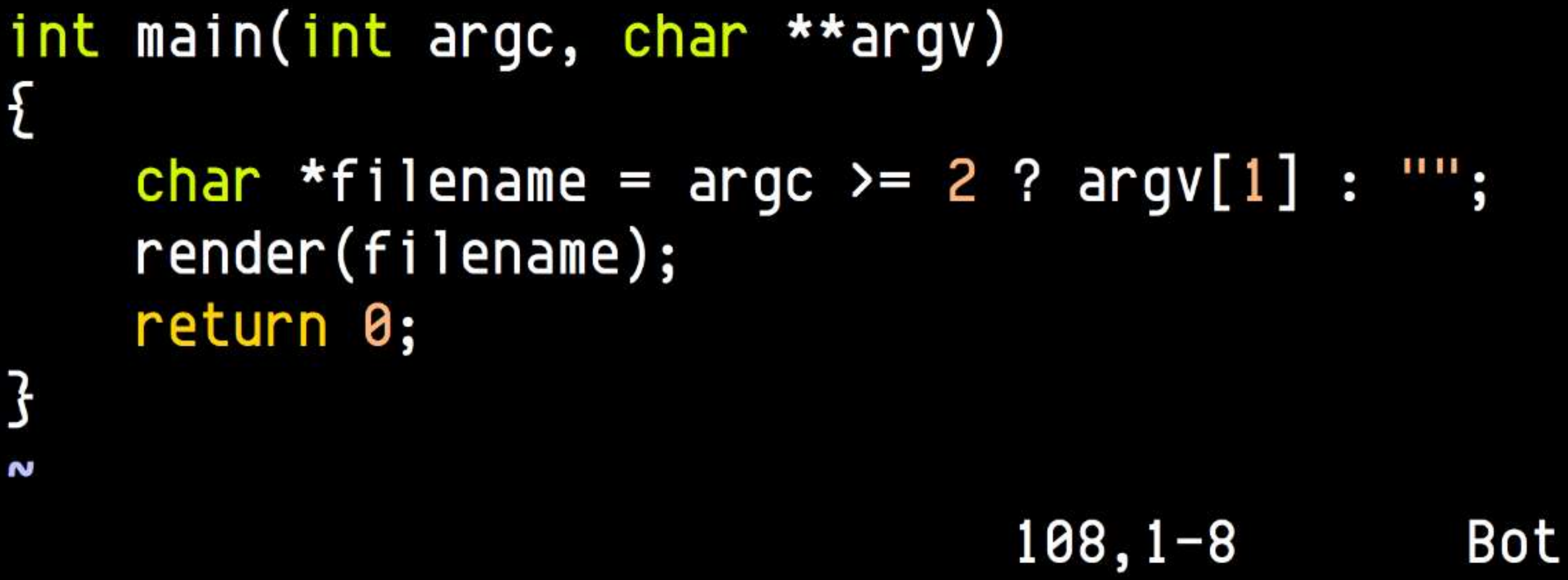
- Patch out
  - extension checking
  - checksum checks
  - disk writes
  - useless config file reads (strace)

Instrumentation Tricks

- Persistent Mode
- Deferred Forkserver (?)

- Speed gains can be HUGE (like 10x)

Persistent Mode - mupdf

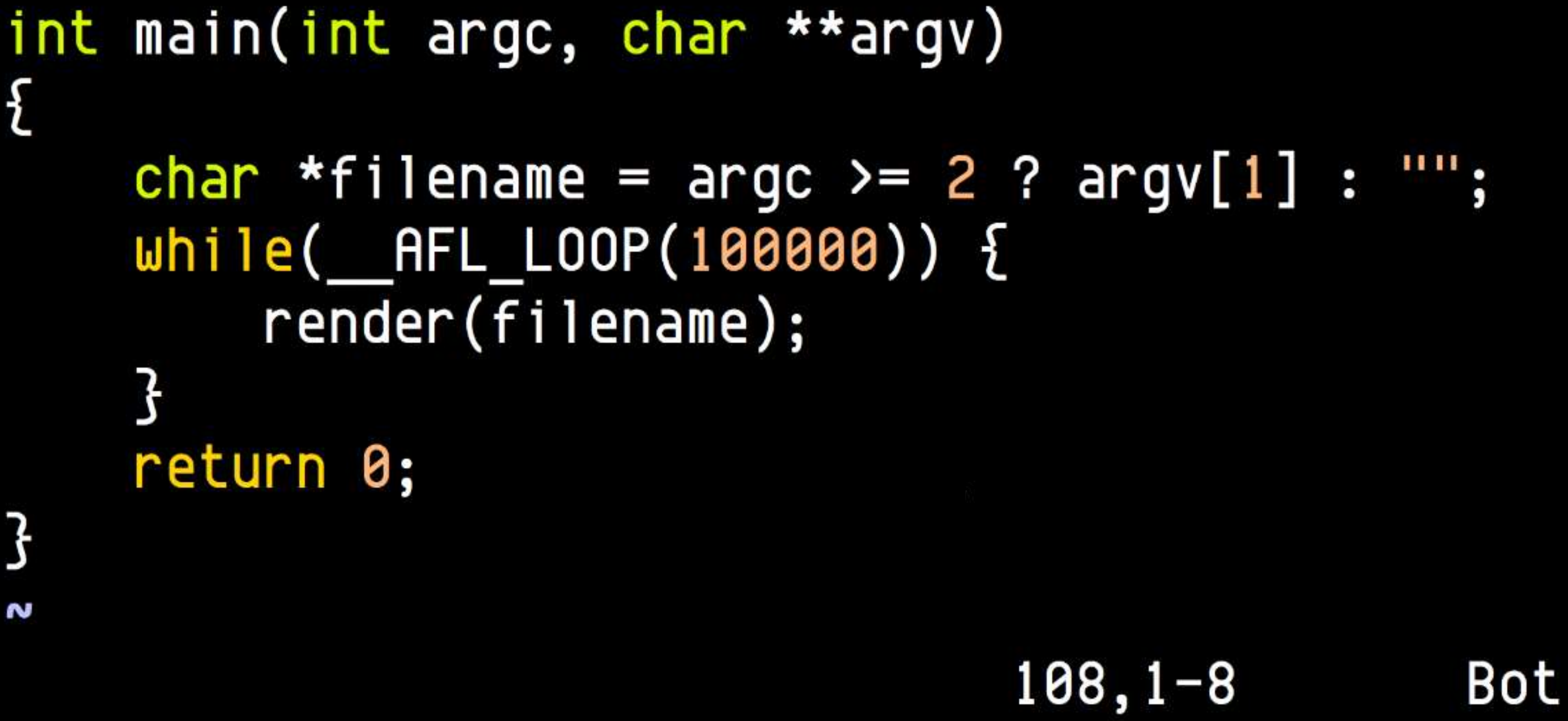


```
int main(int argc, char **argv)
{
    char *filename = argc >= 2 ? argv[1] : "";
    render(filename);
    return 0;
}
~
                                          108,1-8          Bot
```

COSEINC®

Solid Security.Verified.

Persistent Mode - mupdf



```
int main(int argc, char **argv)
{
    char *filename = argc >= 2 ? argv[1] : "";
    while(__AFL_LOOP(100000)) {
        render(filename);
    }
    return 0;
}
~
                                                108,1-8       Bot
```

Persistent Mode Tips

- Reinitialize ALL dirty variables
- __AFL_LOOP(xxx) can limit leaks
- Might even stabilize variable flow!
  - (pdfium example)

AFL Seeds

- Complex formats NEED seeds!
- 1 day x 48 cores with:
  - AFL Sample PDF - 13%
  - Corkami POC PDFS - 30%
  - Add Adobe Samples - 47%

Pollenation

- Targets in one root sync automatically

```
/path/to/target1/target-S0
/path/to/target1/target-S1
/path/to/target1/target-S2
```

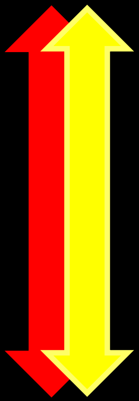
Pollenation

```
/path/to/target1/t1-S0
/path/to/target1/t1-S1
/path/to/target1/t1-S2

/path/to/target2/t2-S0
/path/to/target2/t2-S1
/path/to/target2/t2-S2
```

Pollenation

```
/path/to/target1/t1-S0
/path/to/target1/t1-S1
/path/to/target1/t1-S2
/path/to/target1/t2-S0.sync
```



```
/path/to/target2/t2-S0
/path/to/target2/t2-S1
/path/to/target2/t2-S2
/path/to/target2/t1-S0.sync
```

Pollenation

- For core-heavy farms, this is great
- For box-heavy farms try `roving`
- github.com/richoh/roving

“Low Hanging Fruit Trick”

- Run 1 fuzzer with 1 tiny sample
- Run another with varied seeds
- Pollenate after a day
- ~20% smaller total queue
- ~5% faster execution

Spring Cleaning

- AFL is stubborn about resumes
  - Thundering Herd -> timeouts
  - Some queue files ‘crash’
- Queue from another target is WORSE

Spring Cleaning

- `export AFL_SKIP_CRASHES=1`
- in config.h, try:

```
#define CAL_TMOUT_PERC      200
#define CAL_TMOUT_ADD       200
```

Spring Cleaning

- Use afl-cmin!
  - Automatically skips crashes
  - Skips timeouts
  - Bonus: actually reduces queue!

- Note: queue/ usually “reinflates” under fuzz

My Software

afl-launch

- Launch many afl instances
- Passes through all afl flags
- Redirects stdout to logfiles
- Template filename options

cwtriage

- Automatically repro and triage crashes
- OSX* and Linux
- Simple, plays well with others
- Output to TXT, JSON or protobuf
- Heavily tested (on the linux side)

cwtriage

- `-tidy` stashes files that don’t repro
- `-afl` automatically re-use AFL settings
- `-every` runs on a schedule
- `-file` uses template filenames
  - /dev/shm/xvghktsty.pdf
- `-workers` uses multiple cores

COSEINC®
Solid Security.Verified.

cwdump

- Dumps a cwtriage database
- Summarises crashes by stack hash

Initiating Demonstration…

```
~ cat > ~/bin/showbugs
```

```
#! /bin/sh

cwtriage -root . \
-workers $(nproc) -afl > /dev/null

cwdump ./crashwalk.db > triage.txt

less triage.txt
^D
```

```
~ cat > ~/bin/showbugs
^D
~ chmod a+x ~/bin/showbugs
~ cd /my/target && showbugs
```

afl-trivia

- afl-pause / afl-resume
- afl-pcmin
- afl-consolidate
- afl-pollenate

```
~ cd /rootdir/for/target
~ afl-pause .
~ showbugs
~ afl-resume .
```

afl-pcmin

- Like afl-cmin, but using GNU parallel
- Minimise a set of files for coverage

afl-pollenate

- Pollenates .sync dirs (shown earlier)
- Runs once an hour, by default

```
~ afl-pollenate /fuzz/pdf
```

afl-consolidate

- Consolidate all queue and crash files
- Named via SHA-1 (removes dups)
- Optionally add extension
- Do this to prep for repro stage

```
~ afl-consolidate . /pdfs .pdf
```

babysit.exe

- Run a Windows target with many files
  - `CreateProcess()`
  - `WaitForSingleObject()`
  - `GetExitCodeProcess()`
- YES it works for /GS violations etc

- BugId is better, but this is like 50x faster
  - github.com/SkyLined/BugId

h0w 2 cyb3rbugs

1. `afl-launch` several linux engines
2. `afl-pollenate` between them
3. `afl-consolidate` that corpus
4. `babysit` real target with those files
5. Use `bugid.py` on tasty exit codes
   - 0xc0000005
   - 0xc0000409

COSEINC®
Solid Security.Verified.

github.com/bnagy/afl-trivia
github.com/bnagy/afl-launch
github.com/bnagy/crashwalk
github.com/bnagy/babysit

Thanks

- lcamtuf for AFL
- Jonathan Foote for GDB exploitable
- ~~@rich0H for the SIGSTOP trick~~
- @snare

COSEINC

Solid Security.Verified.

A fuzzer is like
a finger
pointing a way
to the moon.
Don’t focus on
the fuzzer, or
you will miss all
that heavenly
glory.

come at me, bro!

@rantyben
freenode/#afl-users (bnagy)
github.com/bnagy

COSEINC
Solid Security.Verified.