Introduction to fuzzing

using American Fuzzy Lop

Giovanni Lagorio

giovanni.lagorio@unige.it
https://zxgio.sarahah.com

DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
University of Genova
Italy

December 18, 2017

Outline

Fuzz-testing, AKA Fuzzing

- Basic idea: throw random garbage to programs and make them crash ... in interesting ways ☺
- Extremely useful: gcc/clang address sanitizer `-fsanitize=address`
  - Comparison of memory tools: `https://github.com/google/sanitizers/wiki/AddressSanitizerComparisonOfMemoryTools`

American Fuzzy Lop (rabbit)

https://en.wikipedia.org/wiki/American_Fuzzy_Lop

American Fuzzy Lop (fuzzer)

AFL (`http://lcamtuf.coredump.cx/afl/`) by Michal Zalewski

- is a security-oriented fuzzer
- employs compile-time instrumentation
  - on Linux, optional QEMU mode allows black-box binaries to be fuzzed
- can wrap either gcc/g++ or clang/clang++
- uses genetic algorithms to discover interesting test cases, that trigger new internal states in the targeted binary

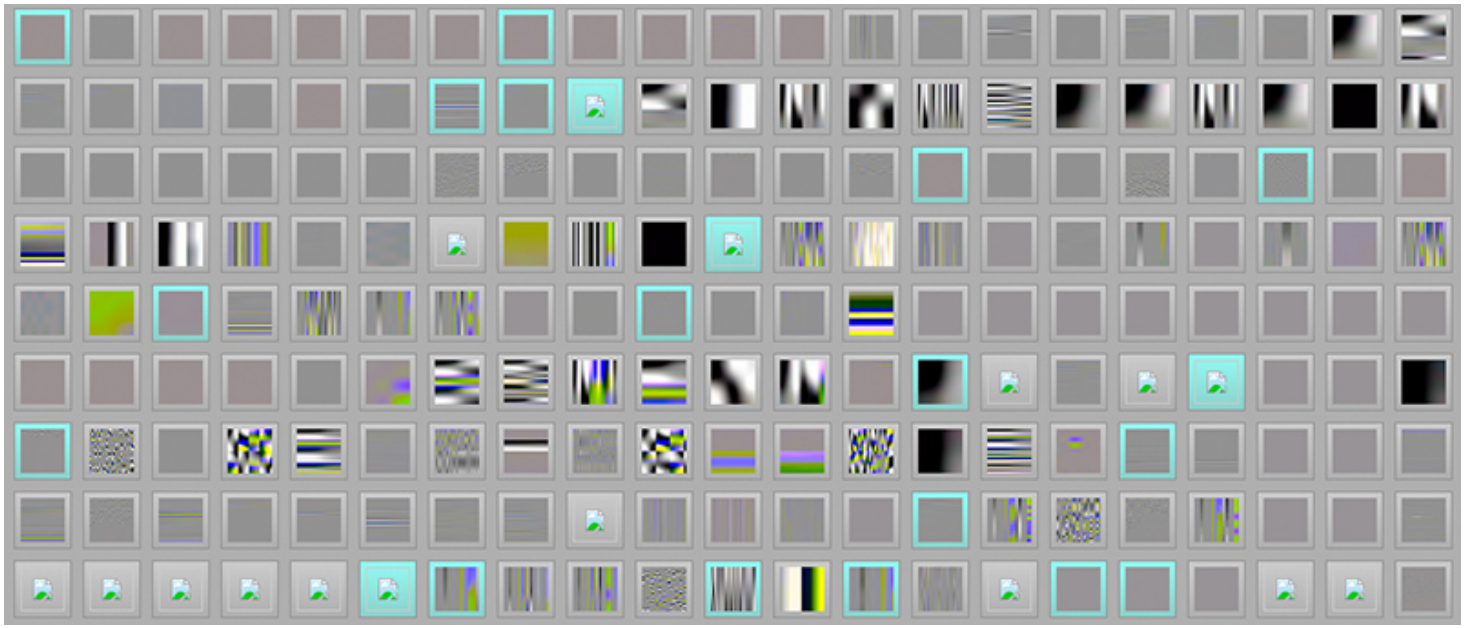
(Simplified) overall algorithm

1. load user-supplied initial test cases into the queue
2. take next input file from the queue
3. attempt to trim the test case to the smallest size that doesn't alter the measured behavior of the program
4. repeatedly mutate the file using a balanced and well-researched variety of traditional fuzzing strategies
5. if any of the generated mutations resulted in a new state transition recorded by the instrumentation, add mutated output as a new entry in the queue
6. go to (2)

The discovered test cases are also periodically culled to eliminate ones that have been obsoleted by newer, higher-coverage finds

Pulling JPEGs out of thin air

. . . created a text file containing just "hello" and asked the fuzzer to keep feeding it to a program that expects a JPEG image . . . The first image, hit after about six hours on an 8-core system, looks very unassuming . . . But the moment it is discovered, the fuzzer starts using the image as a seed - rapidly producing a wide array of more interesting pics . . .



`https://lcamtuf.blogspot.it/2014/11/pulling-jpegs-out-of-thin-air.html`

Test-cases and dictionaries

- the smaller, the better
- AFL comes with some examples of small test cases in various categories (images, archives, . . .) that can be leveraged

Fuzzing dictionaries

afl-fuzz mutation engine is optimized for compact data formats, like images, multimedia, compressed data. . .
It is somewhat less suited for languages with particularly verbose and redundant verbiage, like HTML, SQL, or JavaScript.
See `dictionaries/README.dictionaries` for more details

Outline

Installation

Download latest version:
`http://lcamtuf.coredump.cx/afl/releases/afl-latest.tgz`

- To build:
  ```
  tar xfz afl-latest.tgz && cd afl-* && make
  ```
- To build (experimental) QEMU mode:
  ```
  sudo apt install libtool-bin libglib2.0-dev automake
  cd qemu_mode && ./build_qemu_support.sh
  ```
- To install, either
  - `sudo make install`
  - update `PATH`, and set `ALF_PATH`; e.g.:
    ```
    # ~/.bashrc
    export AFL_PATH=~/bin/afl
    export PATH=$PATH:$AFL_PATH
    ```
- To force the use of *Address Sanitizer* set/export: `AFL_USE_ASAN=1`

Core files

Sending crash notifications to external utilities creates a significant delay between the actual crash and having this information relayed to AFL, which may misinterpret crashes as hangs

Core dumps - Ubuntu 14.x (and later?)

- `sudo systemctl stop apport.service`
- `ulimit -c unlimited`
- check the contents of `/proc/sys/kernel/core_pattern`; it should be `core`

CPU frequency scaling

Apparently, the scaling algorithm in the kernel is imperfect and can miss the short-lived processes spawned by afl-fuzz
To keep things moving, run these commands as root:

```
cd /sys/devices/system/cpu
echo performance | tee cpu*/cpufreq/scaling_governor
```

Go back to the original state by replacing `performance` with `ondemand` or `powersave`; see:
`/sys/devices/system/cpu/cpu*/cpufreq/scaling_available_governors`

RAM disks

Since AFL generates *a lot* of files, working on a RAM FS could

- make your SSD last longer
- (probably) speed the fuzzing process up

In Ubuntu just:

- `mkdir afl-ramdisk`
- `sudo mount -t tmpfs -o size=512M tmpfs afl-ramdisk`

A temporary file-system is just that!

Do remember to copy AFL findings *outside*

Outline

Compilation

Usually, assuming gcc compiler suite, it is enough to:

- `./configure CC="afl-gcc" CXX="afl-g++" --disable-shared`
- `make`, *or*
- `CC="afl-gcc" CXX="afl-g++" make`
  - you may need to add `LDFLAGS="-ldl"` if the compilation fails with an error like `undefined reference to symbol 'dlsym@@GLIBC_2.0'`

Example: GNU Indent

- download the sources: `https://mirror2.mirror.garr.it/mirrors/gnuftp/indent/indent-2.2.10.tar.gz`
- `tar xfz indent-2.2.10.tar.gz && cd indent-2.2.10`
- `./configure CC="afl-gcc" --disable-shared`
- `make`

In the output, you should get something like:

```
...
afl-as 2.52b by <lcamtuf@google.com>
[+] Instrumented 189 locations (64-bit, ASAN/MSAN mode, ratio 33%).
...
```

Test cases

Let's start with some very simple program:

```
int main() { int x,y[10]; if (x>0) { --x; } }
```

Running AFL

Syntax:

```
afl-fuzz -i test-cases -o findings -m none -- ./indent @@
```

Required arguments:

- `-i` — input directory, with test cases
- `-o` — output directory, for fuzzer findings

Optional arguments:

- `-f` *file* — location read by the fuzzed program (default: stdin)
- `-m` — memory limit; use `-m none` with *address sanitizer*
- `-Q` — use binary-only instrumentation (QEMU mode)
- ...

Timing

No hard rule, but expect most fuzzing jobs to run for *days or weeks*

Distributed mode

Every copy of afl-fuzz use one CPU core; to distribute:

- On a single system
  - create a new, shared output directory ("sync dir")
  - come up with a naming scheme; say, "fuzzer01", "fuzzer02", ...
  - run the *master* like this:

    ```
    ./afl-fuzz ...  -o sync_dir -M fuzzer01 ...
    ```
  - then, start up secondary instances:

    ```
    ./afl-fuzz ...  -o sync_dir -S fuzzer02 ...
    ./afl-fuzz ...  -o sync_dir -S fuzzer03 ...
    ```
- On different machines
  - see `docs/parallel_fuzzing.txt` in AFL