(Continuous) Automated C/C++ Fuzz Testing

Yevgeny Pats @fuzzitdev

April 2020, Moscow C++

Fuzzit

A bit about me

- @yevgenypats
- Founder & CEO at @fuzzitdev
- Security Researcher
- Serial entrepreneur
- Cyber Security @ IDF

Fuzzit

Agenda

- What is fuzzing?
- Types of Fuzzers
- libFuzzer
- Continuous Fuzzing
- Trophies (Case studies)
- What Fuzzing is not?
- The Future
- Q&A

Fuzzit

What is Fuzzing / Fuzz Testing

- Fuzzing - providing semi-random input in automated way to a program in order to uncover bugs and crashes.
- Is it only to find security/memory corruption vulnerabilities?
- In safe languages fuzzing helps find a LOT of bugs and improve stability and code coverage. Logic bugs with security impact as well.
- Run millions of unit-tests without writing them.

Fuzzit

Quick history & Types of fuzzers

- Traditional/Random
- Coverage-Guided
  - AFL
  - libFuzzer
  - go-fuzz (Initially developed by Dmitry Vyukov)
  - cargo-fuzz (rust - based on libFuzzer)
  - JQF (java)
  - jsfuzz
  - pythonfuzz
  - javafuzz

Fuzzit

ParseComplex Example

```
bool parse_complex(const char *src, size_t len) {
    if (len == 5) {
        if (src[0] == 'F' &&
            src[1] == 'U' &&
            src[2] == 'Z' &&
            src[3] == 'Z' &&
            src[4] == 'I' &&
            src[5] == 'T') {
            return true;
        }
    }
    return false;
}
```

Fuzzit

Random vs Coverage guided fuzzing Algorithm

```
// pseudo code
for {
        Generate random input
        Execute input
}
```

```
// pseudo code
Instrument program for code coverage
for {
        Choose random input from corpus
        Mutate input
        Execute input and collect coverage
        If new coverage/paths are hit add it to corpus
}
```

Fuzzit

Coverage Guided Fuzzing - Demo

```
package parser

extern "C" int LLVMFuzzerTestOneInput(const uint8_t * data, size_t size) {
   parse_complex((const char *)data, size);
   return 0;
}

// gclang++-8 -fsanitize=fuzzer,address -lsrc src/parse_complex.cpp fuzz/fuzz_parse_complex.cpp
// ./a.out
```

Time till Crash: 3 sec

Fuzzit

Data generation

“Sdlkfgnjk12 iv7$”

“Laksjdh2345 24ו4לך3יגחלדשר ”

“as(*&^&^%*&^%”

(The testcases that are saved in the corpus)

“”
“FAAAA”
“FUAAA”
“FUZAA”
“FUZZA”
“FUZZI” - Crash
??

Fuzzit

Property based fuzz testing

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t * data, size_t size) {
   memcmp(decode(encode((const char *)data, size)), data, size);
   return 0;
}
```

Fuzzit

Trophies

IJG jpeg [1]	libjpeg-turbo [1] [2]	libpng [1]
libtiff [1] [2] [3] [4] [5]	mozjpeg [1]	PHP [1] [2] [3] [4] [5] [6] [7] [8]
Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4] [5] [6] [7]	sqlite [1] [2] [3] [4] ...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1] [2] ...	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3] [4] [5]
PuTTY [1] [2]	ntpd [1] [2]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	JavaScriptCore [1] [2] [3] [4]
pdfium [1] [2]	ffmpeg [1] [2] [3] [4] [5]	libmatroska [1]
libarchive [1] [2] [3] [4] [5] [6] ...	wireshark [1] [2] [3]	ImageMagick [1] [2] [3] [4] [5] [6] [7] [8] [9] ...
BIND [1] [2] [3] ···	QEMU [1] [2]	lcms [1]
Oracle BerkeleyDB [1] [2]	Android / libstagefright [1] [2]	iOS / ImageIO [1]
FLAC audio library [1] [2]	libsndfile [1] [2] [3] [4]	less / lesspipe [1] [2] [3]
strings (+ related tools) [1] [2] [3] [4] [5] [6] [7]	file [1] [2] [3] [4]	dpkg [1] [2]
rcs [1]	systemd-resolved [1] [2]	libyaml [1]
Info-Zip unzip [1] [2]	libtasn1 [1] [2] ···	OpenBSD pfctl [1]
NetBSD bpf [1]	man & mandoc [1] [2] [3] [4] [5] ···	IDA Pro [reported by authors]
clamav [1] [2] [3] [4] [5] [6]	libxml2 [1] [2] [4] [5] [6] [7] [8] [9] ···	glibc [1]
clang / llvm [1] [2] [3] [4] [5] [6] [7] [8] ···	nasm [1] [2]	ctags [1]

Fuzzit

Mutations

- Bit-flipping, Byte-flipping, arithmetics,
- Sonar

Solutions and What Fuzzing is not

- Doesn’t replace unit-tests, integration tests.
- Secure design, threat modeling & attack surface reduction
  - Sandbox
  - Thread modeling
  - Up-to-date third-party-libraries
- As the developer you are responsible for writing the fuzz tests just as you write the unit-tests for your code. You are the best person to understand which parts of the code need to be fuzzed.

Fuzzit

Continuous Fuzzing

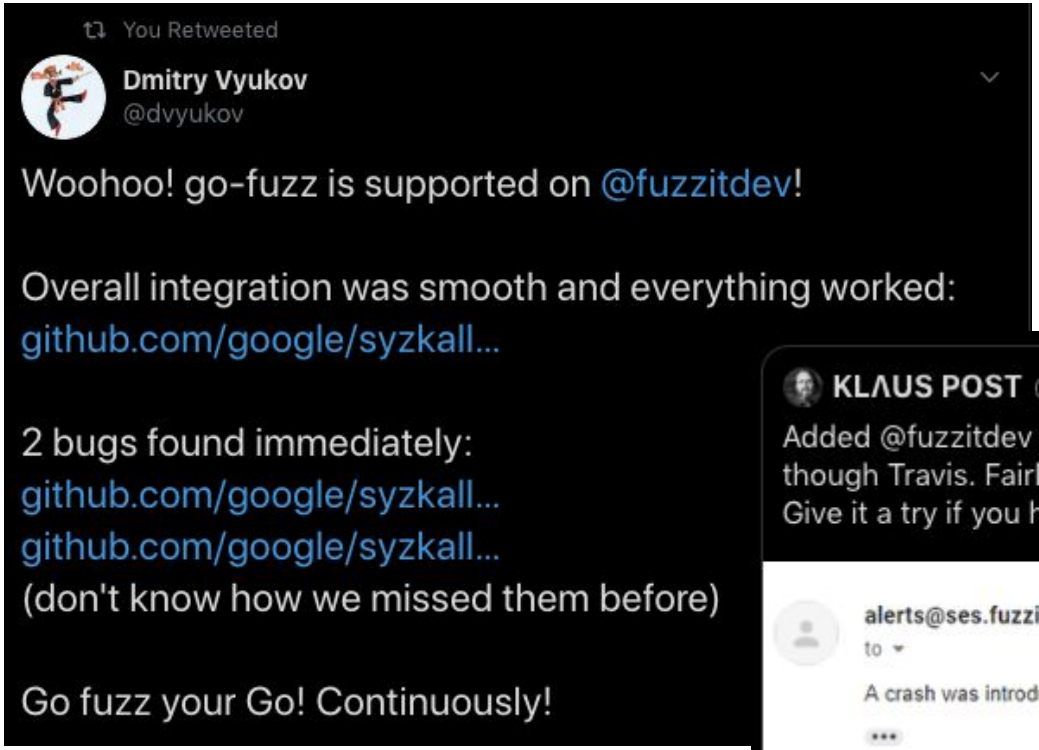
- Running a fuzzer once is nice and it will probably find bugs.
- Just like unit-tests, you want to run the fuzzers every time you push new code.
- Unlike unit-tests which are quick (usually), fuzzing can run indefinitely.
- How long should we fuzz? What version should we fuzz?

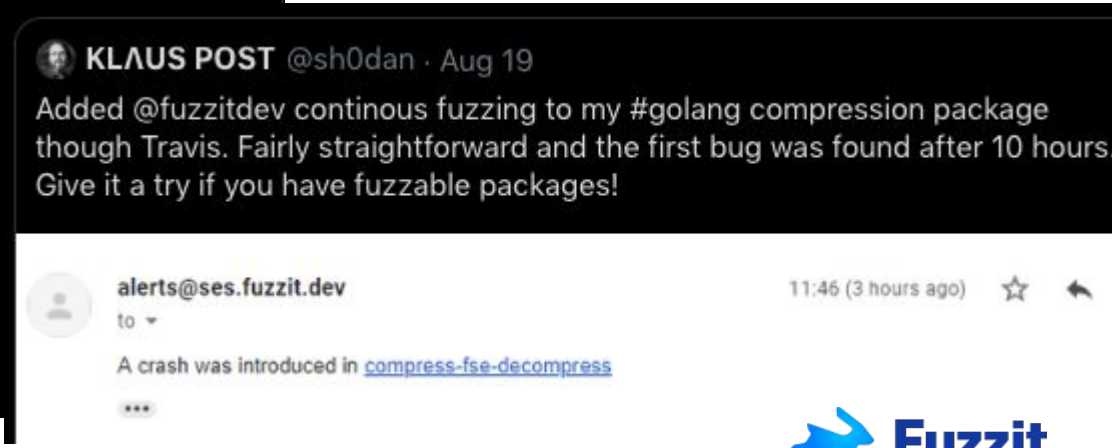
Fuzzit

Continuous Fuzzing Workflow

```
// Pseudo code
// Fuzzing workflow
for {
        Push new code to master/dev
        Build the fuzzers in the CI and upload to a server where you will run them.
        The fuzzer will run either until it finds a crash or until a new version of the fuzzer is uploaded
        Corpus is saved between runs
}
// Regression workflow
for {
        Open a Pull-Request
        Download the corpus
        Run the fuzzers through all the files available in the corpus (quick) - Free unit-tests!
}
```

Continuous Trophies





Future

- Structure Aware Fuzzing - https://github.com/google/fuzzing/blob/master/docs/structure-aware-fuzzing.md

Fuzzit

Q & A