OSS-Fuzz
Google's continuous fuzzing service for open source software

Kostya Serebryan <kcc@google.com>
USENIX Security 2017
Agenda

● Fuzzing-related archeology (paleontology?)

● libFuzzer demo

● OSS-Fuzz - continuous fuzzing service
Testing vs Fuzzing

```
MyApi(\textbf{Input1});
MyApi(\textbf{Input2});
MyApi(\textbf{Input3});

while (true)
  MyApi(\textbf{GenerateInput}());
```
Coverage-guided fuzzing

- Acquire the initial corpus of inputs for your API
- while (true)
  - Randomly mutate one input
  - Feed the new input to your API
  - new code coverage => add the input to the corpus
Coverage-guided fuzzing is not new

- Bunny-the-fuzzer (2007)
- “Automated Whitebox Fuzz Testing” (aka “SAGE”, 2008)
- ...

- 2013-11-14 “[asan] Poor man's coverage that works with ASan”
  - Used internally by the Google Security team
  - 2014/01/ffmpeg-and-thousand-fixes.html (and the following 500+ bugs)

- 2013-11-12: AFL released

- 2014-11-14: first bug found by libFuzzer (released: 2015-01-27)
Yet, the **Heartbleed**

- **2011-12-31**: Introduced into OpenSSL

- **2014-03**: Found independently by
  - Google's Neel Mehta: *code audit*
  - Codenomicon: *specialized fuzzer*

- **2015-04-07** (Hanno Böck):
  - AFL (out-of-process): 6 hours

- **2015-04-09** (Kostya Serebryany):
  - libFuzzer (in-process): 10 seconds
Why did Heartbleed exist for 2 years?

- OpenSSL not funded well?
- Fuzzing tools not widely available?
- **Fuzzing done by security researchers, not by code owners**
Why didn’t OpenSSL team fuzz until 2016?

- OpenSSL not funded well?
- Fuzzing tools not widely known (poorly documented, etc)?
- No infrastructure to automate continuous fuzzing!
Experimental fuzzing “service” (2015)

- 100-line bash script to automate fuzzing
- OpenSSL, BoringSSL, PCRE2, FreeType, LibXML, HarfBuzz
- One 8-core VM per project, running for 24/7
- Found bugs in every project, decided to make it bigger!
Fuzzing as a Service

- **2016-12-01**: OSS-Fuzz launched publicly
  - Collaboration between Chrome Security, Open Source, and Dynamic Tools teams

- Continuous automated fuzzing on Google’s VMs

- Uses **libFuzzer** and AFL, more fuzzing engines in pipeline
  - Also uses ASan/MSan/UBSan to catch bugs

- Available to important OSS projects for free
  - The project needs to have a large user base and/or be critical to Global IT infrastructure, a general heuristic that we are intentionally leaving open to interpretation at this stage (*)

- Same infrastructure is used to [fuzz Chrome](#) since 2015
Detour: libFuzzer and Fuzz Targets
libFuzzer

bool FuzzMe(const uint8_t *Data, size_t DataSize) { // fuzz_me.cc
    return DataSize >= 3 &&
            Data[0] == 'F' &&
            Data[1] == 'U' &&
            Data[2] == 'Z' &&
            Data[3] == 'Z';  // :-
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    FuzzMe(Data, Size);
    return 0;
}

% clang -g -fsanitize=address,fuzzer fuzz_me.cc && ./a.out

# Requires fresh clang
Fuzz Target

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  DoStuffWithYourAPI(Data, Size);
  return 0;
}

- Consumes any data: \{abort, exit, crash, assert, timeout, OOM\} == bug
- Single-process
- Deterministic (need randomness? Use part of the input data as RNG seed)
- Does not modify global state (preferably)
- The narrower the better (fuzz small APIs, not the entire application)
libFuzzer demo

tutorial.libFuzzer.info
Back to OSS-Fuzz
2000+ bugs

- heap buffer overflows: 11.5%
- global buffer overflows: 1.5%
- stack buffer overflows: 1.3%
- use after frees: 1.1%
- uninitialized memory: 2.2%
- stack overflows: 3.8%
- timeouts: 8.3%
- OOMs: 8.5%
- leaks: 7.9%
- other (e.g. assertions): 11.3%
- unknown crashes: 6.1%
- ubsan: 36.5%
In **60+ OSS projects** (showing top 30)
Example: Wireshark (~50 bugs)

Wireshark mailing list:
>> Timeouts. These are more severe as it causes a denial of service due to "infinite" loops
Ideal integration with OSS-Fuzz

- Every fuzz target:
  - Is maintained by code owners in their RCS (Git, SVN, etc)
  - Is built with the rest of the tests - no bit rot!
  - Has a seed corpus with good code coverage
  - Is continuously tested on the seed corpus with ASan/UBSan/MSan
  - Is fast and has no OOMs
  - Has fuzzing dictionary, if applicable

- Projects don’t have to have their own continuous fuzzing
  - But are welcome to!
Life of a 🐜

- The bot detects a bug and deduplicates it against other known bugs
- Reproducer input is minimized, “regression revision range” identified
- Private issue is reported with project owners in CC
- Owners fix the bug
  - Recommended: the reproducer is added to the seed corpus for regression testing
- (every 24 hours) the bot reruns on fresh trunk
  - If the bug is fixed, identifies “fixed revision range” and closes the bug
- The bug is made public:
  - 30 days after the fix or
  - 90 days after reporting (whichever is earlier)
### Issue 2445

**Issue Title:** gdal: Heap-buffer-overflow in PCIDSK::CBandInterleavedChannel::ReadBlock

**Reported by:** monor...@clusterfuzz-external.iam.gserviceaccount.com, Jul 1

**Project:** gdal

**Fuzzer:** libFuzzer_gdal_filesystem_fuzzer

**Fuzz target binary:** gdal_filesystem_fuzzer

**Job Type:** libfuzzer_asan_gdal

**Platform Id:** linux

**Crash Type:** Heap-buffer-overflow READ 4

**Crash Address:** 0x602000000055

**Project Member:** Reported by monor...@clusterfuzz-external.iam.gserviceaccount.com, Jul 1

**Detailed report:** [https://oss-fuzz.com/testcase?key=476641567039488](https://oss-fuzz.com/testcase?key=476641567039488)

**Type:** Bug-Security

- ClusterFuzz
- Stability-Memory-AddressSanitizer
- Reproducible
- ClusterFuzz-Verified
- Engine-libfuzzer
- OS-Linux

**Closed:** Jul 2

**Cc:**
- b_________@gmail.com
- m_________.net
- s__________@gmail.com
- e________i...@gmail.com

**Status:** Verified

**Owner:** ----

**Reported:** 2017-07-01

**Reproduced Testcase:** [https://oss-fuzz.com/download?testcase_id=476641567039488](https://oss-fuzz.com/download?testcase_id=476641567039488)

# Fuzzer statistics

<table>
<thead>
<tr>
<th>fuzzer</th>
<th>perf_report</th>
<th>logs</th>
<th>tests_executed</th>
<th>new_crashes</th>
<th>known_crashes</th>
<th>edge_cov</th>
<th>fune_cov</th>
<th>cov_report</th>
<th>corpus_size</th>
<th>corpus_backup</th>
<th>avg_exec_per_sec</th>
<th>new_units_added</th>
</tr>
</thead>
<tbody>
<tr>
<td>libFuzzer_boringssl_cxml</td>
<td>Performance</td>
<td>Logs</td>
<td>18,867,870,757</td>
<td>0</td>
<td>0</td>
<td>23.11% (1525/6599)</td>
<td>31.57% (340/1077)</td>
<td>Coverage</td>
<td>1348 (363 KB)</td>
<td>Download</td>
<td>4,103,811</td>
<td>79</td>
</tr>
<tr>
<td>libFuzzer_boringssl_client</td>
<td>Performance</td>
<td>Logs</td>
<td>3,684,787,202</td>
<td>0</td>
<td>0</td>
<td>30.17% (4708/15606)</td>
<td>40.89% (1144/2798)</td>
<td>Coverage</td>
<td>2895 (5 MB)</td>
<td>Download</td>
<td>737,793</td>
<td>206</td>
</tr>
<tr>
<td>libFuzzer_boringssl_pkey</td>
<td>Performance</td>
<td>Logs</td>
<td>17,013,559,702</td>
<td>0</td>
<td>0</td>
<td>31.70% (1390/4385)</td>
<td>40.47% (332/865)</td>
<td>Coverage</td>
<td>897 (164 KB)</td>
<td>Download</td>
<td>3,493,126</td>
<td>640</td>
</tr>
<tr>
<td>libFuzzer_boringssl_privkey</td>
<td>Performance</td>
<td>Logs</td>
<td>9,246,506,400</td>
<td>0</td>
<td>0</td>
<td>30.47% (1290/4233)</td>
<td>44.36% (291/656)</td>
<td>Coverage</td>
<td>1078 (264 KB)</td>
<td>Download</td>
<td>2,360,454</td>
<td>20</td>
</tr>
<tr>
<td>libFuzzer_boringssl_read_pem</td>
<td>Performance</td>
<td>Logs</td>
<td>59,809,056,058</td>
<td>0</td>
<td>0</td>
<td>29.92% (177/646)</td>
<td>23.44% (45/192)</td>
<td>Coverage</td>
<td>159 (396 KB)</td>
<td>Download</td>
<td>12,372,294</td>
<td>5</td>
</tr>
<tr>
<td>libFuzzer_boringssl_server</td>
<td>Performance</td>
<td>Logs</td>
<td>4,236,237,451</td>
<td>0</td>
<td>0</td>
<td>29.31% (421/14384)</td>
<td>42.23% (1051/2489)</td>
<td>Coverage</td>
<td>1474 (703 KB)</td>
<td>Download</td>
<td>897,857</td>
<td>1,712</td>
</tr>
<tr>
<td>libFuzzer_boringssl_session</td>
<td>Performance</td>
<td>Logs</td>
<td>30,566,193,178</td>
<td>0</td>
<td>0</td>
<td>29.49% (1212/4254)</td>
<td>29.50% (223/756)</td>
<td>Coverage</td>
<td>1029 (702 KB)</td>
<td>Download</td>
<td>6,038,652</td>
<td>84</td>
</tr>
<tr>
<td>libFuzzer_boringssl_upki</td>
<td>Performance</td>
<td>Logs</td>
<td>17,610,541,626</td>
<td>0</td>
<td>0</td>
<td>20.33% (827/4067)</td>
<td>38.53% (252/654)</td>
<td>Coverage</td>
<td>278 (17 KB)</td>
<td>Download</td>
<td>3,744,903</td>
<td>2</td>
</tr>
<tr>
<td>libFuzzer_boringssl_ssl_cder_api</td>
<td>Performance</td>
<td>Logs</td>
<td>338,393,427</td>
<td>0</td>
<td>568</td>
<td>17.55% (1804/10281)</td>
<td>24.98% (597/2360)</td>
<td>Coverage</td>
<td>1755 (255 KB)</td>
<td>Download</td>
<td>84,060</td>
<td>38</td>
</tr>
</tbody>
</table>
Coverage report

```c
if(w <= 0 || h <= 0 || av_image_check_size(w, h, &avctx) || s->byostream == s->byostream_end)
    return AVERAGE_INVALIDDATA;

avctx->width = w;
avctx->height = h;
if (avctx->pix_fmt != AV_PIX_FMT_MONOWHITE && avctx->pix_fmt != AV_PIX_FMT_MONOBLACK) {
    prct_get(s, buf1, sizeof(buf1));
    s->maxval = atoi(buf1);
    if (s->maxval <= 0 || s->maxval > UINT16_MAX) {
        av_log(avctx, AV_LOG_ERROR, "Invalid maxval: %d\n", s->maxval);
        s->maxval = 255;
    }
}
if (s->maxval >= 256) {
    if (avctx->pix_fmt == AV_PIX_FMT_GRAY8) {
        avctx->pix_fmt = AV_PIX_FMT_GRAY16;
    } else if (avctx->pix_fmt == AV_PIX_FMT_RGB24) {
        avctx->pix_fmt = AV_PIX_FMT_RGB48;
    } else if (avctx->pix_fmt == AV_PIX_FMT_YUV420P && s->maxval < 65536) {
        if (s->maxval < 512)
            avctx->pix_fmt = AV_PIX_FMT_YUV420P9;
        else if (s->maxval < 1024)
            avctx->pix_fmt = AV_PIX_FMT_YUV420P16;
        else
            avctx->pix_fmt = AV_PIX_FMT_YUV420P16;
    } else {
        av_log(avctx, AV_LOG_ERROR, "Unsupported pixel format\n");
        avctx->pix_fmt = AV_PIX_FMT_NONE;
        return AVERAGE_INVALIDDATA;
    }
```
How to participate in OSS-Fuzz

● Be an important OSS project (examples)

● Send a pull request to https://github.com/google/oss-fuzz
  ○ project.yaml - project information and maintainer e-mails (example)
  ○ Dockerfile - set up the build environment (example)
  ○ build.sh - build the fuzz targets (example)

● Improve over time
  ○ Fix bugs (including timeouts/OOMs)
  ○ Monitor coverage and extend seed corpus
Google’s **Patch Reward Program** (for OSS-Fuzz)

- $1,000 for initial integration with OSS-Fuzz
- *Up to* 20,000 for *ideal* integration

**Why are we doing this?**
- To make Google’s code safer (we use lots of OSS)
- To make Internet safer (no more Heartbleeds, please!)
- To popularize continuous fuzzing
Fuzz-Driven Development

- Kent Beck @ 2003 (?): **Test-Driven Development**
  - Great & useful approach (still, not used everywhere)
  - Drastically insufficient for security

- Kostya Serebryany @ 2017: Fuzz-Driven Development:
  - Every API is a Fuzz Target
  - Tests == “Seed” Corpus for fuzzing
  - Continuous Integration (CI) includes Continuous Fuzzing
  - Equally applicable to “safer” languages, see e.g. [rust-fuzz](https://rust-fuzz.io), [go-fuzz](https://golang.org/cmd/fuzz/)
Summary

- Coverage-guided fuzzing is easy

- Fuzzing must be
  - Continuous & Automated
  - Maintained by code owners

- OSS-Fuzz - a public fuzzing service for OSS
  - Goal: make common software infrastructure more secure by applying modern fuzzing techniques at large scale.
  - 2000+ bugs reported since Dec 2016, most fixed.
Q&A

https://github.com/google/oss-fuzz