Modern Fuzzing of Media-processing projects

Max Moroz, FOSDEM 2017

Agenda

- Fuzzing
  - what is fuzzing, why fuzz, fuzzing types
- How to fuzz
  - fuzz target, fuzzing engine, libFuzzer
- Media processing as a target
  - motivation, scary stories
- OSS-Fuzz
  - Fuzzing-as-a-Service for Open Source Software

What is Fuzzing

- Somehow generate a test input
- Feed it to the code under test
- Repeat

Why Fuzz

- Bugs specific to C/C++ that require the *sanitizers* to catch:
  - Use-after-free, buffer overflows, Uses of uninitialized memory, Memory leaks
- Arithmetic bugs:
  - Div-by-zero, Int/float overflows, bitwise shifts by invalid amount
- Plain crashes:
  - NULL dereferences, Uncaught exceptions
- Concurrency bugs:
  - Data races, Deadlocks
- Resource usage bugs:
  - Memory exhaustion, hangs or infinite loops, infinite recursion (stack overflows)
- Logical bugs:
  - Discrepancies between two implementations of the same protocol (example)
  - Assertion failures
- Timeouts and Out-Of-Memory **are BUGS** (*in most of the cases)
  - And super bad for fuzzing

Fuzzing Types

- Generation-based fuzzing
  - Usually a target-specific grammar-based generator
- Mutation-based fuzzing
  - Acquire a corpus of test inputs
  - Apply random mutations to the inputs
- **Guided mutation-based fuzzing**
  - Execute mutations with coverage instrumentation
  - If new coverage is observed the mutation is permanently added to the corpus

Fuzz Target

```
bool TargetAPI(const uint8_t* Data, size_t Size) {
    bool Result = false;
    if (Size >= 3) {
        Result = Data[0] == 'F' &&
                 Data[1] == 'U' &&
                 Data[2] == 'Z' &&
                 Data[3] == 'Z';
    }
    return Result;
}

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

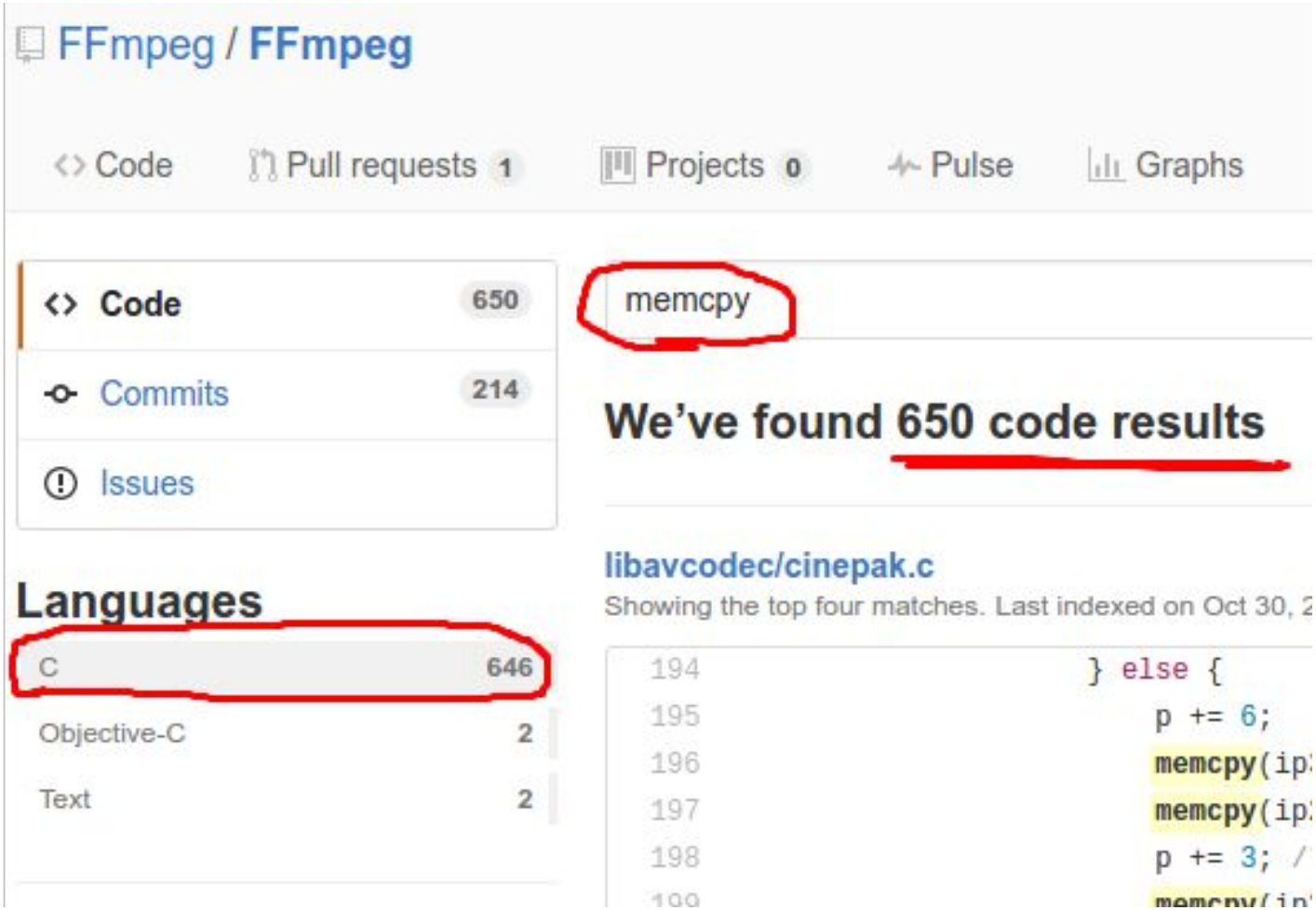
libFuzzer - an engine for guided in-process fuzzing

- libFuzzer: a library; provides main()
- Build your target code with extra compiler flags
- Link your target with libFuzzer
- Pass a directory with the initial test corpus and run

```
% clang++ -g my-code.cc libFuzzer.a -o my-fuzzer \
  -fsanitize=address -fsanitize-coverage=trace-pc-guard

% ./my-fuzzer MY_TEST_CORPUS_DIR
```

Media is a great target to Fuzz [1 / 2]

- Lots of code working with raw pointers



Media is a great target to Fuzz [2 / 2]

- Being used everywhere
  - Video hosting services
  - Media players
  - Mobile devices
  - Embedded entertainment systems
    - In planes
    - In cars
    - In space? :)
  - etc.

Example: *GStreamer in the living room and in outer space*, FOSDEM 2015

Recent security breaches

- FFmpeg and a thousand fixes, Jan 2014
- Stagefright, Apr 2015
- Viral Video, Nov 2015
- ImageTragick, Apr 2016
- A scriptless 0day exploit against Linux desktops, Nov 2016

Present Perfect → Present Continuous

- "The project X **has been** fuzzed, hence it is somewhat secure"

- False:
  - Bug discovery techniques evolve
  - The project X evolves
  - Fuzzing is CPU intensive and needs time to find bugs

- "The project X **is being continuously** fuzzed, the code coverage is monitored."
  - Much better!

Case Study from OSS-Fuzz: CVE-2017-3732 took **more than 1 CPU year** to find

OSS-Fuzz: Fuzzing-as-a-Service

- Based on ClusterFuzz, the fuzzing backend used for fuzzing Chrome components
  - Supported engines: libFuzzer, AFL, Radamsa, …
- Thousands of CPU cores for free
- https://github.com/google/oss-fuzz
  - 55+ projects
    - 180+ fuzz targets
  - 450+ bugs (~150 vulnerabilities)
    - 320+ fixed

Bug Report sample [1 / 2]

- Filed automatically:

ffmpeg: Stack-buffer-overflow in ff_htmlmarkup_to_ass

Project Member Reported by monor...@clusterfuzz-external.iam.gserviceaccount.com, Nov 9

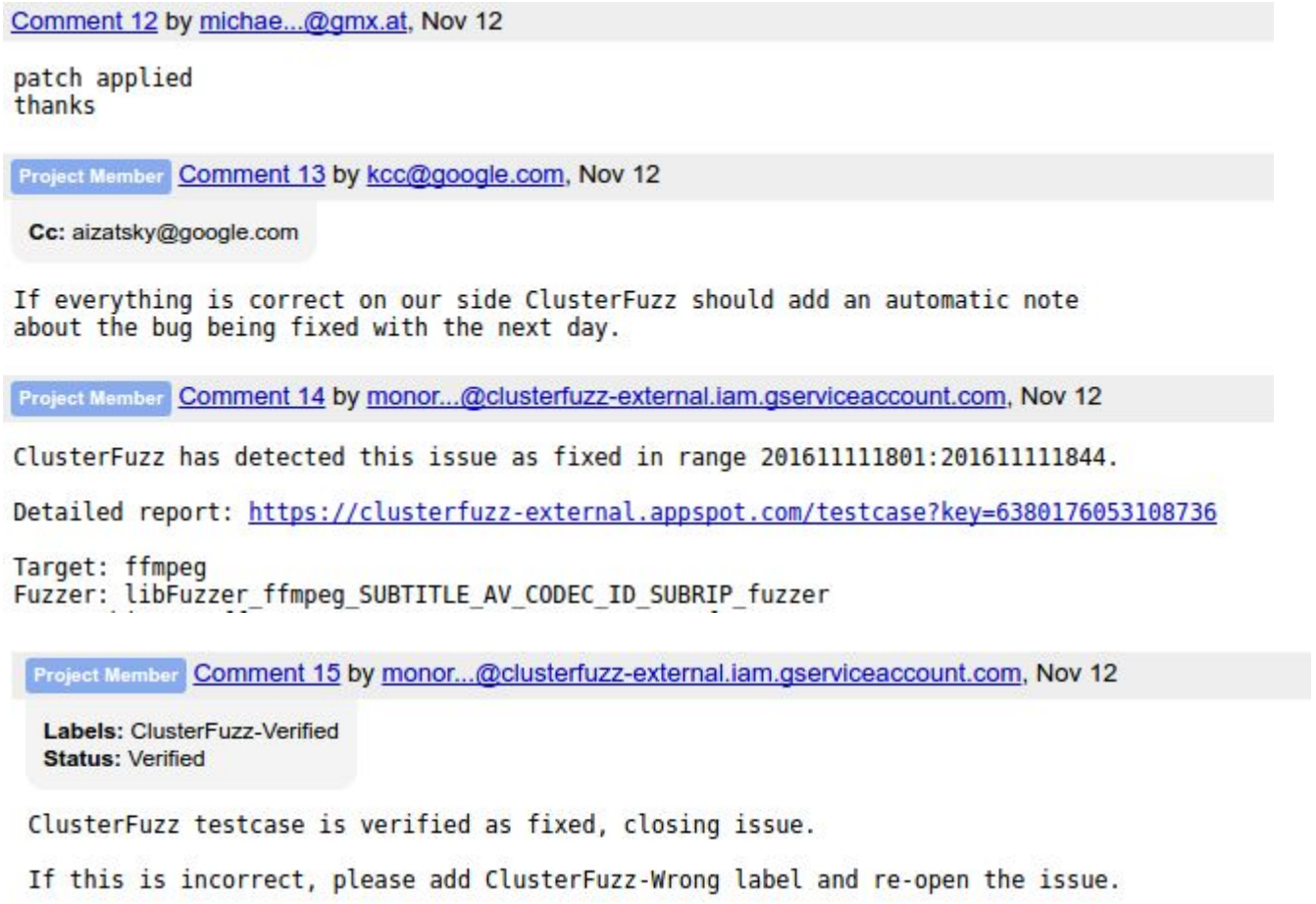
Detailed report: https://clusterfuzz-external.appspot.com/testcase?key=6380176053108736

```
Target: ffmpeg
Fuzzer: libFuzzer_ffmpeg_SUBTITLE_AV_CODEC_ID_SUBRIP_fuzzer
Fuzzer binary: ffmpeg_SUBTITLE_AV_CODEC_ID_SUBRIP_fuzzer
Job Type: libfuzzer_asan_ffmpeg
Platform Id: linux

Crash Type: Stack-buffer-overflow READ 1
Crash Address: 0x7f71190d38b0
Crash State:
  ff_htmlmarkup_to_ass
  srt_to_ass
  srt_decode_frame

Recommended Security Severity: Medium


Minimized Testcase (0.30 Kb): https://clusterfuzz-external.appspot.com/download/AMIfv94wZV8lr
wgbn_Khh1BhjVqfrKstRyHFO3i2VZYvDRM6zLYEGRFb738fwdRy4DD0443qck9RoF_mryo_P3eWhZsCGlg1fqJGYvG6aZ
CkB63AQDhBc9Q?testcase_id=6380176053108736

Issue filed automatically.

See https://github.com/google/oss-fuzz/blob/master/docs/reproducing.md for more information.
```

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=151

Bug Report sample [2 / 2]

- Verified automatically:



Comment 12 by michae...@gmx.at, Nov 12

patch applied
thanks

Project Member Comment 13 by kcc@google.com, Nov 12

Cc: aizatsky@google.com

If everything is correct on our side ClusterFuzz should add an automatic note
about the bug being fixed with the next day.

Project Member Comment 14 by monor...@clusterfuzz-external.iam.gserviceaccount.com, Nov 12

ClusterFuzz has detected this issue as fixed in range 201611111801:201611111844.

Detailed report: https://clusterfuzz-external.appspot.com/testcase?key=6380176053108736

Target: ffmpeg
Fuzzer: libFuzzer_ffmpeg_SUBTITLE_AV_CODEC_ID_SUBRIP_fuzzer

Project Member Comment 15 by monor...@clusterfuzz-external.iam.gserviceaccount.com, Nov 12

Labels: ClusterFuzz-Verified
Status: Verified

ClusterFuzz testcase is verified as fixed, closing issue.

If this is incorrect, please add ClusterFuzz-Wrong label and re-open the issue.

https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=151

Fuzzer Stats Dashboard

Job type: All | Group by: Fuzzer | Start date: 01/27/2017 | End date (inclusive): 02/02/2017 | SUBMIT

testcases_executed	new_crashes	known_crashes	edge_coverage	func_coverage	coverage_report	corpus_size	corpus_backup	quarantine_size	average_exec_per_sec	new_units_ad
2,252,311,127,196	0	0	79.02% (177/224)	97.56% (80/82)	Coverage	278 (145 KB)	Download	0 (0 B)	21,545.999	0
54,167,417,778	0	0	23.11% (1525/6599)	31.57% (340/1077)	Coverage	1314 (365 KB)	Download	0 (0 B)	4,632.038	316
6,930,653,986	0	0	30.16% (4708/15608)	40.87% (1144/2799)	Coverage	2854 (5 MB)	Download	0 (0 B)	587.329	1,092
33,730,087,948	0	0	31.70% (1390/4385)	48.47% (332/685)	Coverage	895 (168 KB)	Download	0 (0 B)	2,883.002	1,908
18,738,795,285	0	0	30.47% (1290/4233)	44.36% (291/656)	Coverage	1051 (257 KB)	Download	0 (0 B)	1,610.97	87
112,165,305,576	0	0	20.92% (177/846)	23.44% (45/192)	Coverage	158 (396 KB)	Download	0 (0 B)	9,373.312	13
8,524,671,634	0	0	29.27% (4217/14409)	42.19% (1050/2489)	Coverage	1439 (679 KB)	Download	0 (0 B)	733.28	5,292
77,425,677,573	1	19	28.44% (1212/4262)	29.42% (223/758)	Coverage	1022 (697 KB)	Download	0 (0 B)	6,727.741	374
42,280,886,830	0	0	20.33% (827/4067)	38.53% (252/654)	Coverage	254 (16 KB)	Download	0 (0 B)	3,601.079	12
1,237,164,540	3	1,237	17.54% (1804/10284)	25.01% (598/2391)	Coverage	1751 (257 KB)	Download	47 (1 MB)	151.543	4,205

Coverage Report

Coverage report for libFuzzer_ffmpeg_VIDEO_AV_CODEC_ID_H264_fuzzer

File	Coverage
Files with zero coverage are not shown.	
/src/ffmpeg/libavcodec/allcodecs.c	67%
/src/ffmpeg/libavcodec/avpacket.c	10%
/src/ffmpeg/libavcodec/bitstream.c	42%
/src/ffmpeg/libavcodec/cabac.c	75%
/src/ffmpeg/libavcodec/cabac_functions.h	35%
/src/ffmpeg/libavcodec/codec_desc.c	10%
/src/ffmpeg/libavcodec/error_resilience.c	68%
/src/ffmpeg/libavcodec/get_bits.h	4%
/src/ffmpeg/libavcodec/golomb.h	19%
/src/ffmpeg/libavcodec/h2645_parse.c	64%
/src/ffmpeg/libavcodec/h264_cabac.c	96%
/src/ffmpeg/libavcodec/h264_cavlc.c	90%
/src/ffmpeg/libavcodec/h264_direct.c	57%
/src/ffmpeg/libavcodec/h264_loopfilter.c	67%
/src/ffmpeg/libavcodec/h264_mb.c	73%
/src/ffmpeg/libavcodec/h264_mb_template.c	79%
/src/ffmpeg/libavcodec/h264_mc_template.c	73%
/src/ffmpeg/libavcodec/h264_mvpred.h	80%
/src/ffmpeg/libavcodec/h264_parse.c	70%
/src/ffmpeg/libavcodec/h264_picture.c	62%
/src/ffmpeg/libavcodec/h264_ps.c	79%
/src/ffmpeg/libavcodec/h264_refs.c	68%
/src/ffmpeg/libavcodec/h264_sei.c	62%
/src/ffmpeg/libavcodec/h264_slice.c	72%
/src/ffmpeg/libavcodec/h264addpx_template.c	84%
/src/ffmpeg/libavcodec/h264chroma.c	100%
/src/ffmpeg/libavcodec/h264dec.c	47%
/src/ffmpeg/libavcodec/h264dec.h	86%

Performance Analysis

Performance issues detected

Issue Name	Runs Affected	Experimental Priority Score	Description
speed	87.65%	0.32	Execution speed is one of the most important factors of an efficient fuzzing. Do your best to optimize the target function as any other code you are optimizing. Every fuzz target is expected to have average execution speed of at least 1,000 testcases per second.
coverage	56.79%	0.47	The fuzzer cannot find new 'interesting' inputs and increase code coverage. There are several ways to help the fuzzer: - Add or update the dictionary. - Add new testcases to the corpus (it can be manually generated testcases, some inputs from unit tests, valid files, traffic streams, etc depending on the target). - Update the target function to use different combinations of flags passed to the target. - Check `max_len` value, may be it is not appropriate for the target (too big for some data which cannot be too big, or too small for some data types which cannot be so small).
timeout	38.27%	24.49	The fuzzer hits timeout error. Usually, there are two possible reasons why timeouts happen during fuzzing: - A valid bug in the target like an infinite loop, some complex computations or anything else hanging the target. As it is not expected for the target application to hang for a long time, the root cause of the bug should be found out and fixed. - The target allows to hang it with some input. Assume that the target is an interpreter for some programming language. If we supply an input with an infinite loop there, the fuzzer will expectedly hang. But it is not a bug. Recommended solution is to improve source code of the fuzzer to avoid processing of such inputs leading to timeouts.
logging	28.40%	0.20	The fuzzer writes too many log messages. Excessive logging is a foe of efficient fuzzing. Often, it is possible to set up different levels of logging for a target. In that case, you need to modify the target function or compilation flags to use the lowest level of logging verbosity. If target does not provide a way to control logging levels or to disable logging in any other legitimate way, you can use `-close_fd_mask` option of libFuzzer.

Fuzz Targets examples

- Chromium: https://cs.chromium.org/search/?q=file:.*media.*fuzzer.*+package:%5Echromium$&type=cs

- OSS-Fuzz: https://git.ffmpeg.org/gitweb/ffmpeg.git/blob_plain/HEAD:/tools/target_dec_fuzzer.c

- Thousands of random examples: https://github.com/search?l=C%2B%2B&q=%22LLVMFuzzerTestOneInput%22&ref=searchresults&type=Code&utf8=%E2%9C%93

Q & A

Useful links:

- OSS-Fuzz project
- libFuzzer.info
- tutorial.libFuzzer.info
- libFuzzer workshop
  - Live at BSidesMunich'2017 on 3rd of April

Contacts:

- mmoroz@chromium.org
- twitter.com/Dor3s
- github.com/Dor1s