

OSS-Fuzz

Google's continuous fuzzing service for
open source software

Kostya Serebryany <kcc@google.com>

USENIX Security 2017

Agenda

- Fuzzing-related archeology (paleontology?)
- libFuzzer demo
- OSS-Fuzz - continuous fuzzing service

Testing vs Fuzzing

```
MyApi(Input1);
```

```
MyApi(Input2);
```

```
MyApi(Input3);
```

```
while (true)
```

```
    MyApi(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

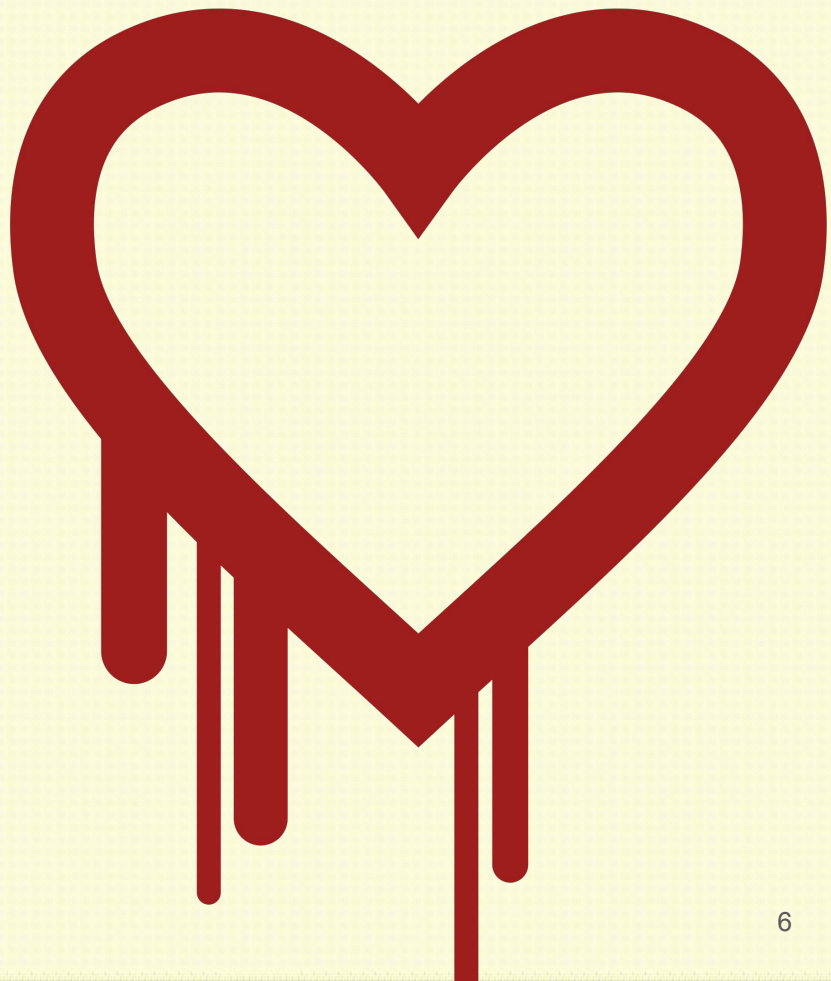
Coverage-guided fuzzing is not new

- 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

% ./fuzz-openssl



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;  
}
```

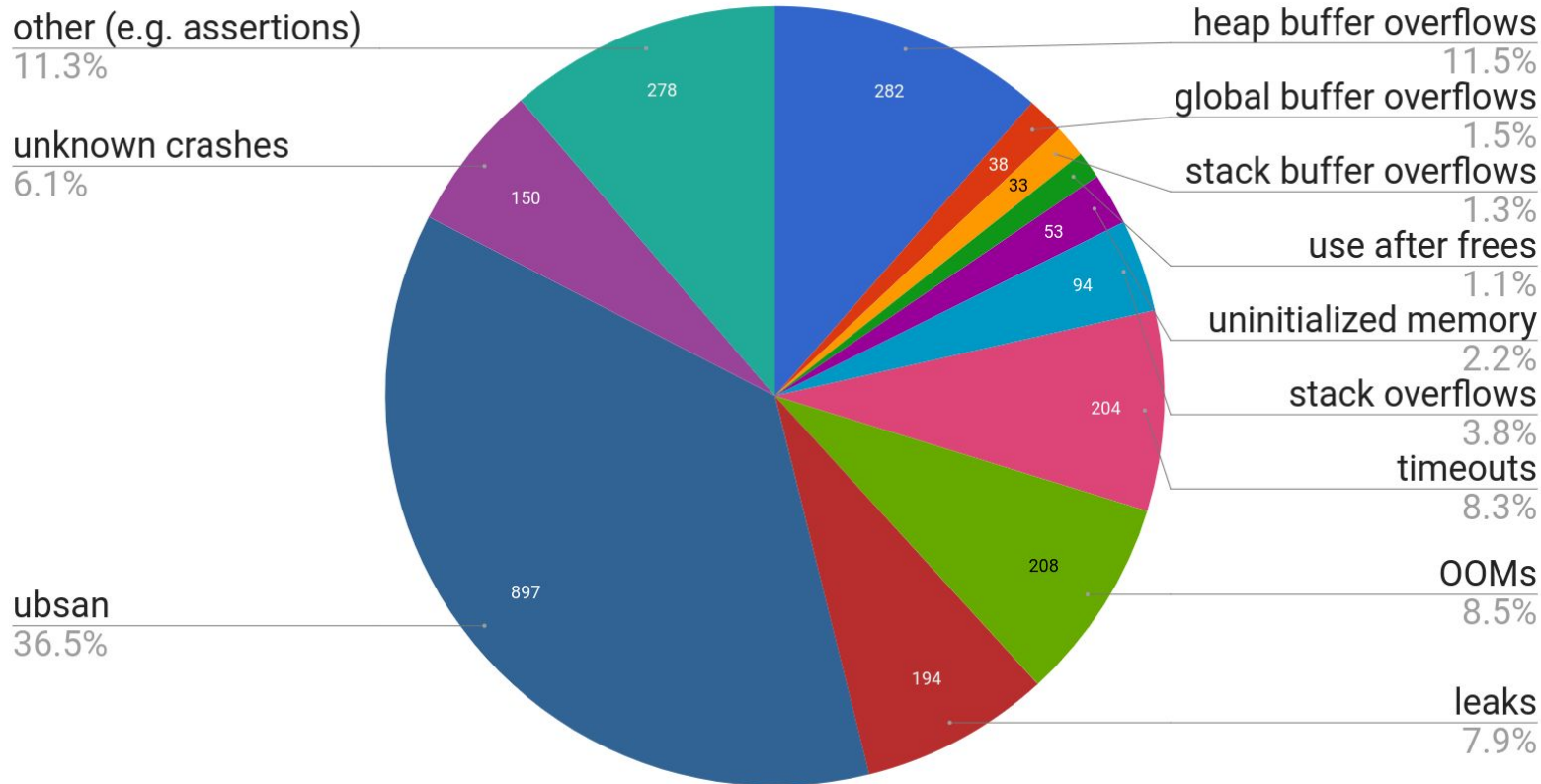
- Consumes any data: {abort,exit,crash,assert,timeout,OOM} == bug
- Single-process
- Deterministic (need randomness? Use part of the input data as RNG seed)
- Does not modify global state (preferably)
- The narrower the better (fuzz small APIs, not the entire application)

libFuzzer demo

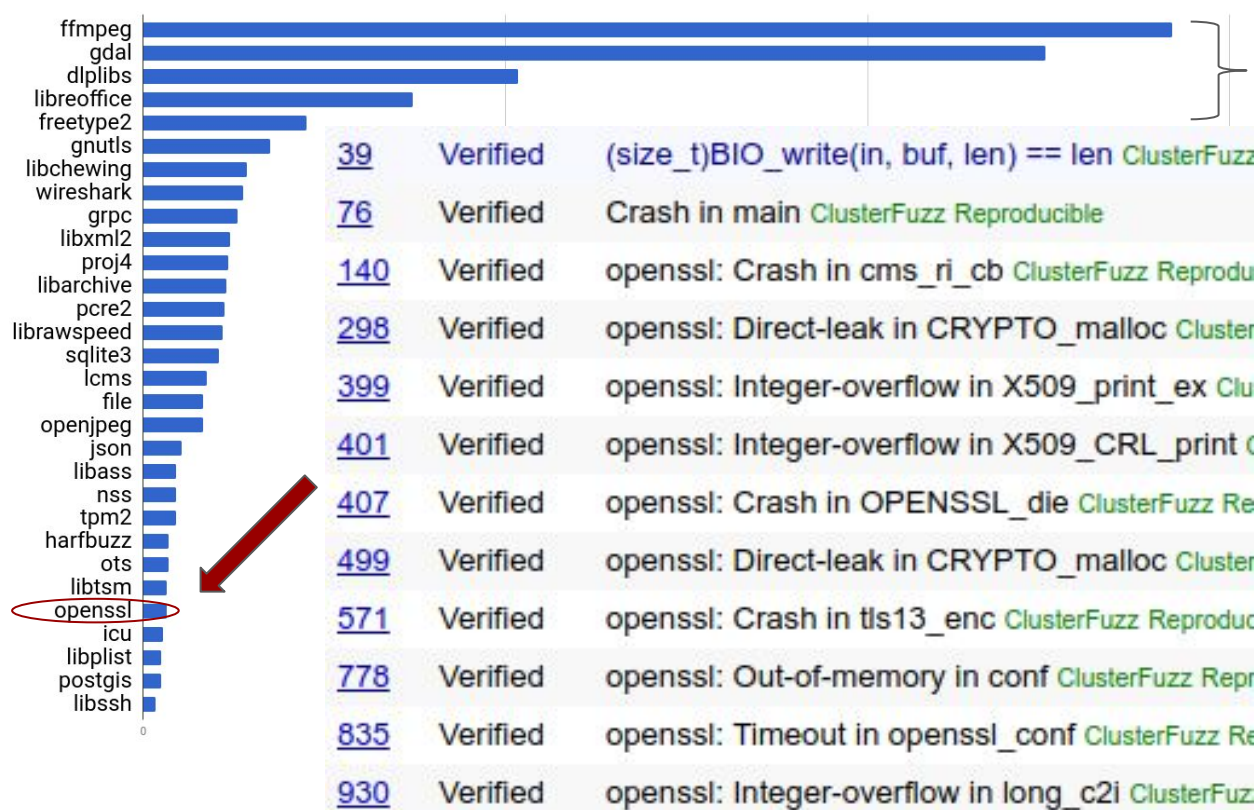
tutorial.libFuzzer.info

Back to OSS-Fuzz

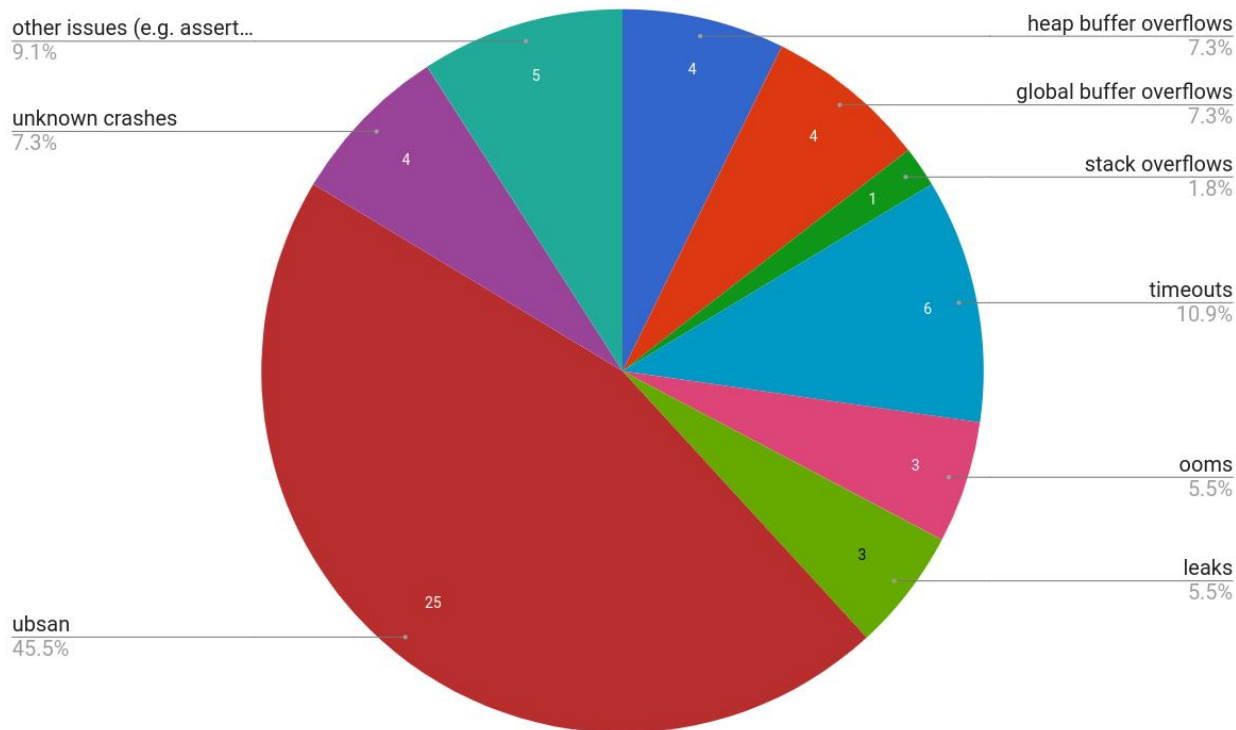
2000+ bugs



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)

Report example (automatically filed)

Issue 2445

Starred by 2 users


Status: Verified

Owner: ----

Closed: Jul 2

Cc:  [bis...@gmail.com](#)

 [m...@gmail.com](#)

 [s...@gmail.com](#)

 [e...@gmail.com](#)

Type: Bug-Security

ClusterFuzz

Stability-Memory-AddressSanitizer

Reproducible

ClusterFuzz-Verified

Engine-libfuzzer

OS-Linux

Proj-gdal

Reported-2017-07-01

gdal: Heap-buffer-overflow in PCIDSK::CBandInterleavedChannel::ReadBlock

Project Member Reported by [monor...@clusterfuzz-external.iam.gserviceaccount.com](#), Jul 1

Detailed report: <https://oss-fuzz.com/testcase?key=4766641567039488>

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: 0x602000008855

Crash State:

PCIDSK::CBandInterleavedChannel::ReadBlock

PCIDSK2Band::IRadBlock

GDALRasterBand::GetLockedBlockRef

Sanitizer: address (ASAN)

Recommended Security Severity: Medium

Regressed: https://oss-fuzz.com/revisions?job=libfuzzer_asan_gdal&range=201705291647:201705301648

Reproducer Testcase: https://oss-fuzz.com/download?testcase_id=4766641567039488

Fuzzer statistics

fuzzer	perf_report	logs	tests_executed	new_crashes	known_crashes	edge_cov	func_cov	cov_report	corpus_size	corpus_backup	avg_exec_per_sec	new_units_added
libFuzzer_boringssl_cert	Performance	Logs	18,867,870,757	0	0	23.11% (1525/6599)	31.57% (340/1077)	Coverage	1346 (363 KB)	Download	4,103.611	79
libFuzzer_boringssl_client	Performance	Logs	3,694,767,292	0	0	30.17% (4709/15606)	40.89% (1144/2798)	Coverage	2895 (5 MB)	Download	737.793	206
libFuzzer_boringssl_pkcs8	Performance	Logs	17,013,559,702	0	0	31.70% (1390/4385)	48.47% (332/685)	Coverage	897 (164 KB)	Download	3,493.126	640
libFuzzer_boringssl_privkey	Performance	Logs	9,246,506,460	0	0	30.47% (1290/4233)	44.36% (291/656)	Coverage	1076 (264 KB)	Download	2,360.454	20
libFuzzer_boringssl_read_pem	Performance	Logs	59,809,096,058	0	0	20.92% (177/846)	23.44% (45/192)	Coverage	159 (396 KB)	Download	12,372.294	5
libFuzzer_boringssl_server	Performance	Logs	4,236,237,451	0	0	29.31% (4219/14394)	42.23% (1051/2489)	Coverage	1474 (703 KB)	Download	897.857	1,712
libFuzzer_boringssl_session	Performance	Logs	30,566,193,178	0	0	28.49% (1212/4254)	29.50% (223/756)	Coverage	1029 (702 KB)	Download	6,038.692	84
libFuzzer_boringssl_spki	Performance	Logs	17,610,541,626	0	0	20.33% (827/4067)	38.53% (252/654)	Coverage	278 (17 KB)	Download	3,744.903	2
libFuzzer_boringssl_ssl_ctx_api	Performance	Logs	336,393,427	0	568	17.55% (1804/10281)	24.98% (597/2390)	Coverage	1755 (255 KB)	Download	84.069	38

Coverage report

/src/ffmpeg/libavcodec/libx265.c	0.3%
/src/ffmpeg/libavcodec/lpc.h	0.9%
/src/ffmpeg/libavcodec/lsp.c	0.17%
/src/ffmpeg/libavcodec/lzw.c	0.65%
/src/ffmpeg/libavcodec/mathops.h	0.2%
/src/ffmpeg/libavcodec/mdct_template.c	0.7%
/src/ffmpeg/libavcodec/mdec.c	0.10%
/src/ffmpeg/libavcodec/me_cmp.c	0.2%
/src/ffmpeg/libavcodec/mpegdec.c	0.46%
/src/ffmpeg/libavcodec/movtextdec.c	0.36%
/src/ffmpeg/libavcodec/mpeg12.c	0.31%
/src/ffmpeg/libavcodec/mpeg12.h	0.34%
/src/ffmpeg/libavcodec/mpeg12dec.c	0.62%
/src/ffmpeg/libavcodec/mpeg4video.c	0.06%
/src/ffmpeg/libavcodec/mpeg4video.h	0.27%
/src/ffmpeg/libavcodec/mpeg4videodec.c	0.82%
/src/ffmpeg/libavcodec/mpeg_er.c	0.70%
/src/ffmpeg/libavcodec/mpegaudio.c	100%
/src/ffmpeg/libavcodec/mpegaudio_tablegen.h	0.50%
/src/ffmpeg/libavcodec/mpegaudiodec_template.c	0.37%

```
if(w <= 0 || h <= 0 || av_image_check_size(w, h, 0, avctx) || s->bytestream >= s->bytestream_end)
    return AVERROR_INVALIDDATA;
avctx->width = w;
avctx->height = h;
if (avctx->pix_fmt != AV_PIX_FMT_MONOWHITE && avctx->pix_fmt != AV_PIX_FMT_MONOBLACK) {
    pnm_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_YUV420P10;
            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 AVERROR_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](#), [go-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>