Black Hat Europe 2019

ClusterFuzz

Fuzzing at Google Scale

Abhishek Arya
Oliver Chang

About us

- Chrome Security team (Bugs--)
- Abhishek Arya (@infernosec)
  - Founding Chrome Security member
  - Founder of ClusterFuzz
- Oliver Chang (@halbecaf)
  - Lead developer of ClusterFuzz
  - Tech lead for OSS-Fuzz

Fuzzing

- Effective at finding bugs by exploring **unexpected states**
- Recent developments
  - Coverage guided fuzzing
    - AFL started “smart fuzzing” (Nov’13)
  - Making fuzzing more accessible
    - libFuzzer - in-process fuzzing (Jan’15)
    - OSS-Fuzz - free fuzzing for open source (Dec’16)

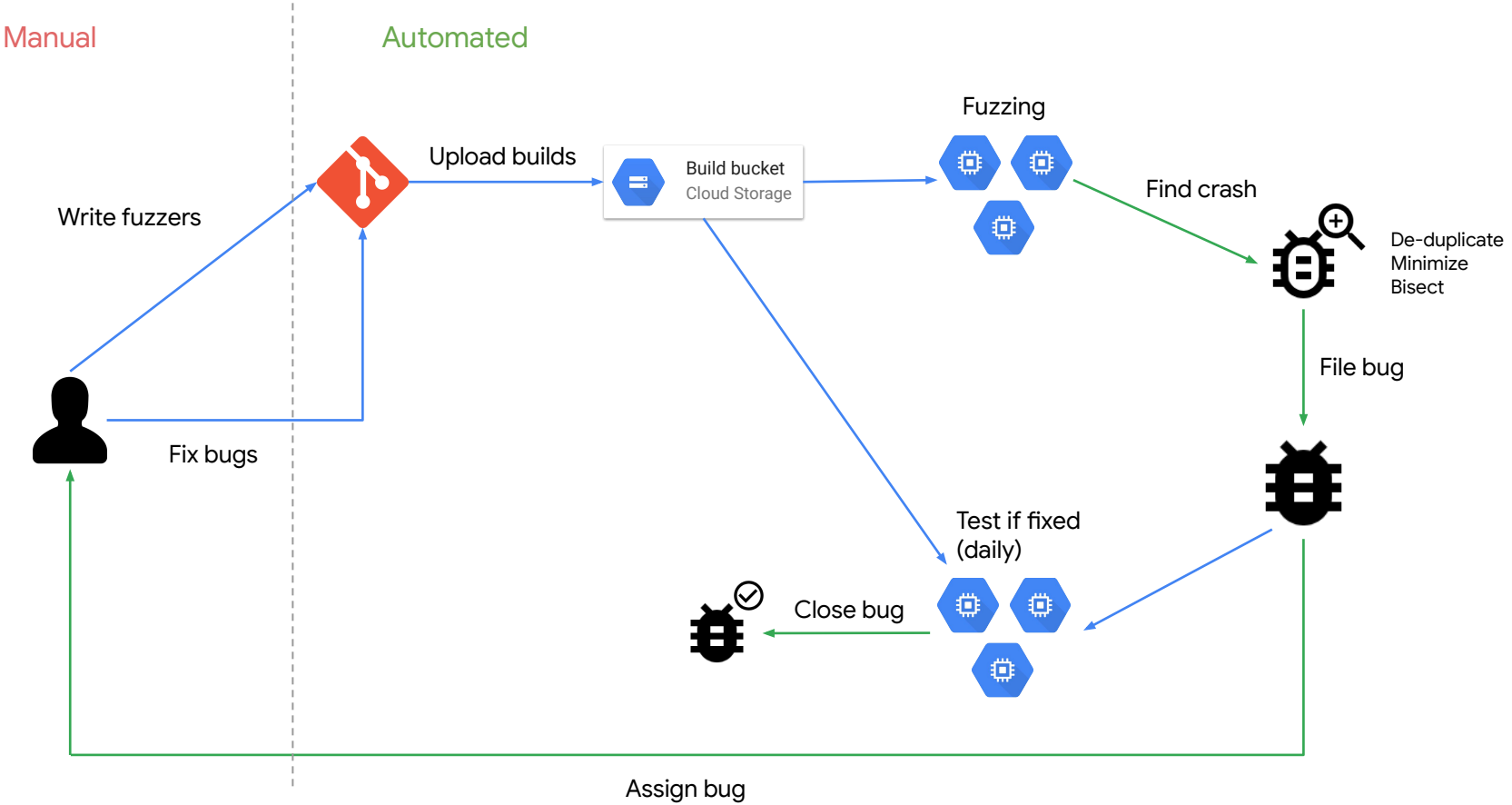
Fuzzing mythbusting

- Fuzzing is only for security researchers or security teams
- Fuzzing only finds security vulnerabilities
- We don’t need fuzzers if our project is well unit-tested
- Our project is secure if there are no open bugs

Scaling fuzzing

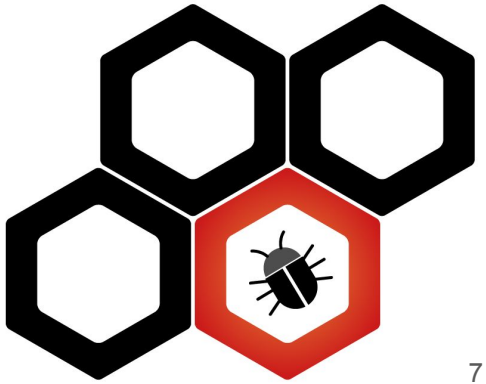
- How to fuzz effectively as a Defender?
  - Not just “more cores”
- Security teams can’t write all fuzzers for the entire project
  - Bugs create triage burden
- Should seamlessly fit in software development lifecycle
  - Input: Commit unit-test like fuzzer in source
  - Output: Bugs, Fuzzing Statistics and Code Coverage

Fuzzing lifecycle



ClusterFuzz

- Open source - https://github.com/google/clusterfuzz
- Automates **everything** in the fuzzing lifecycle apart from “fuzzer writing” and “bug fixing”
- Runs 5,000 fuzzers on 25,000 cores, can scale more
- Cross platform (Linux, macOS, Windows, Android)
- Powers OSS-Fuzz and Google’s fuzzing



Fuzzing lifecycle

1. Write fuzzers
2. Build fuzzers
3. Fuzz at scale
4. Triage crashes
5. Improving fuzzers

Step 1: Write fuzzers

Finding targets to fuzz

- Attack surface enumeration
- e.g. Chrome
  - Sandboxed renderer process
    - Direct untrusted input
  - Privileged processes:
    - Fuzz IPC boundaries
- Third party libraries
- Parsers, complicated processing of input data
- VRP reports

Black box fuzzing

- Generation or mutation based, with rules specific to a particular format
  - e.g. A script that generates valid but randomized HTML files
- Slow (few execs/sec)
- Significant effort to write (>1k LoC)

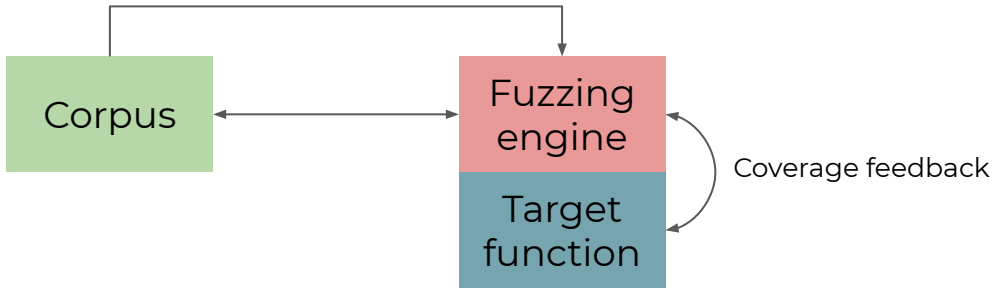
Black box fuzzing

- Chrome employs a number of custom black box fuzzers to do “integration” style testing
  - HTML/DOM fuzzers
  - JavaScript fuzzers
  - IPC fuzzers
- Gestures
- Not guided by coverage

```
for (let __v_2528 = 0; __v_2528 < 100; ++__v_2528) {
  const __v_2529 = __v_2528 % 20 + 15 | 0;

  try {
    __f_548(__v_2519.test(
    /* VariableMutator: Replaced __v_2529 with __v_2514 */
    __v_2514), __v_2529);
  } catch (e) {}
}

/* CrossOverMutator: Crossover from /usr/local/google/home/
try {
  Array.prototype.findIndex.call(undefined, function () {
    return true;
  }, {});
} catch (e) {}

/* VariableOrObjectMutator: Random mutation */
try {
  __callRandomFunction(__v_2514, 313164, null, 4294967297,
} catch (e) {}
```

Grey box fuzzing

- Coverage-guided fuzzers
  - AFL
  - libFuzzer
- Better for testing more focused parts of codebase, akin to unit tests



Grey box fuzzing

- Unit test-like stubs, called “fuzz targets” (as little as 5 LoC)
- Write once, run with multiple fuzzing engines (AFL, libFuzzer, etc)

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

Grey box fuzzing

- No need for mutation or generation logic
  - Fuzzing engine does mutation based on coverage feedback
  - Dictionaries/seed corpora can help a lot
- Written by **developers** to complement traditional unit testing
  - "Security is everyone’s job now, not just the security team’s."

    — Werner Vogels, Amazon CTO

Black box vs grey box

- When to use grey box fuzzing?
  - Smaller, more targeted components
  - Encourage developers to write these
  - Preferred
- Black box fuzzing still necessary
  - Larger components
  - Non-deterministic targets
  - Integration testing

Structure aware fuzzing

- Bridges some gaps between grey box and black box fuzzing
- Structure (protos) + rules = **libprotobuf-mutator**
- Manual, cumbersome, but equally rewarding
- Reference: Jonathan Metzman talk, Black Hat USA 2019

Structure aware fuzzing example

Structure

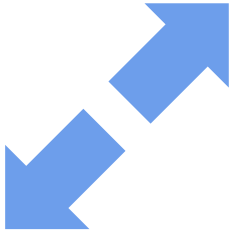
```
message SQLQueries {
    repeated CreateTable queries = 1;
}

message CreateTable {
    optional TempModifier temp_table = 1;
    required Table table = 2;
    required ColumnDef col_def = 3;
    repeated ColumnDef extra_col_defs = 4;
    repeated TableConstraint table_constraints = 5;
    required bool without_rowid = 6;
}

// Further definitions of TempModifier, Table,
ColumnDef, and TableConstraint.
```

Rules

```
std::string CreateTableToString(const CreateTable& ct) {
    std::string ret("CREATE TABLE ");
    if (ct.has_temp_table()) {
        ret += TempModifierToString(ct.temp_table());
        ret += " ";
    }
    ret += TableToString(ct.table());
    ret += "(";
    ret += ColumnDefToString(ct.col_def());
....

DEFINE_BINARY_PROTO_FUZZER(const SQLQueries& sql_queries)
{
    std::string queries = SQLQueriesToString(sql_queries);
    sql_fuzzer::RunSQLQueries(queries);
}
```

Scaling fuzzer writing

- Key to scaling is not through cores, but through educating **developers**
- Documentation and examples for writing grey box fuzzers
- Provide guidance on efficient fuzzing
  - Seed corpora, dictionaries
- Make grey box fuzzing a first class citizen (like unit tests)



Step 2: Build fuzzers

Building fuzzers

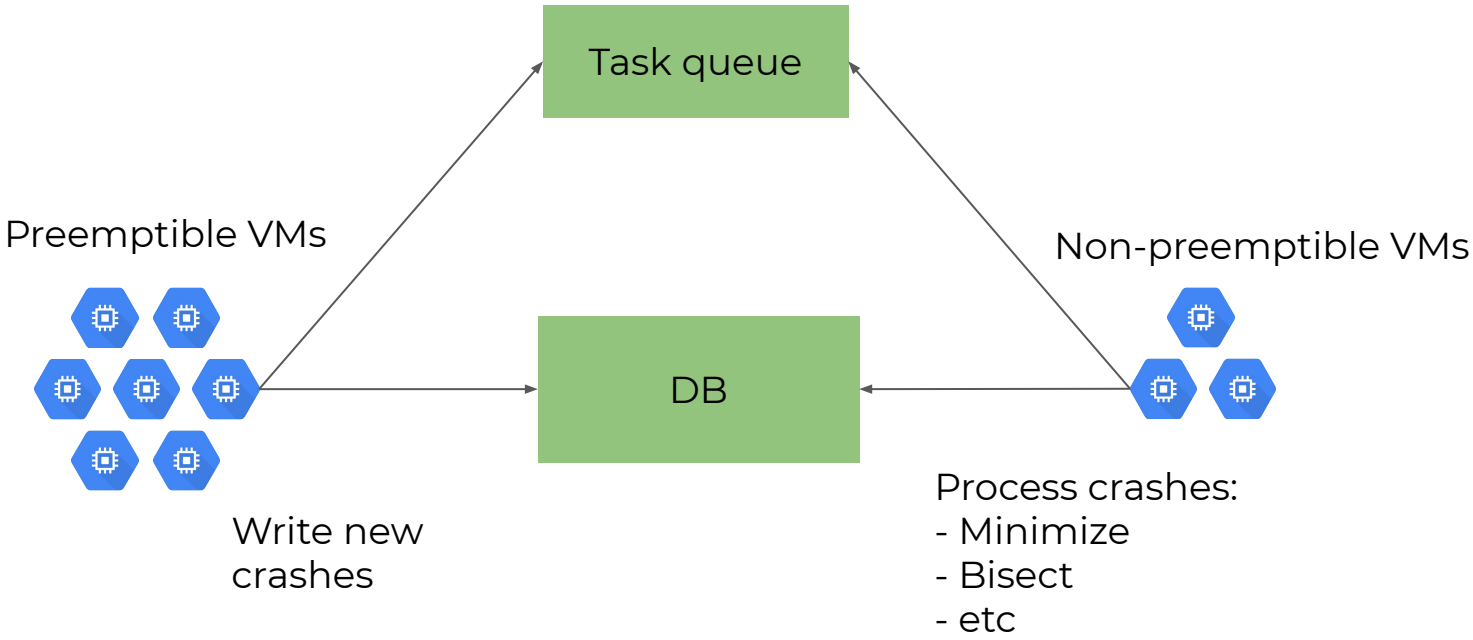
- Use compile-time instrumentation
  - **AddressSanitizer**, **MemorySanitizer**, etc...
  - Coverage instrumentation
  - 2x~ performance penalty
- Link with a fuzzing engine or driver
  - libFuzzer: `clang -fsanitize=address,fuzzer ...`

Building fuzzers (cont)

- Make sure that **release** version is fuzzed
  - Assertions etc are often noisy when fuzzing
  - (Optional) Add debug version for assertion coverage
- Optimization level matters
  - Speed vs more edges
- Builds should be continuous
  - Ideally produced as artifacts of existing CI infrastructure

Step 3: Fuzz at scale

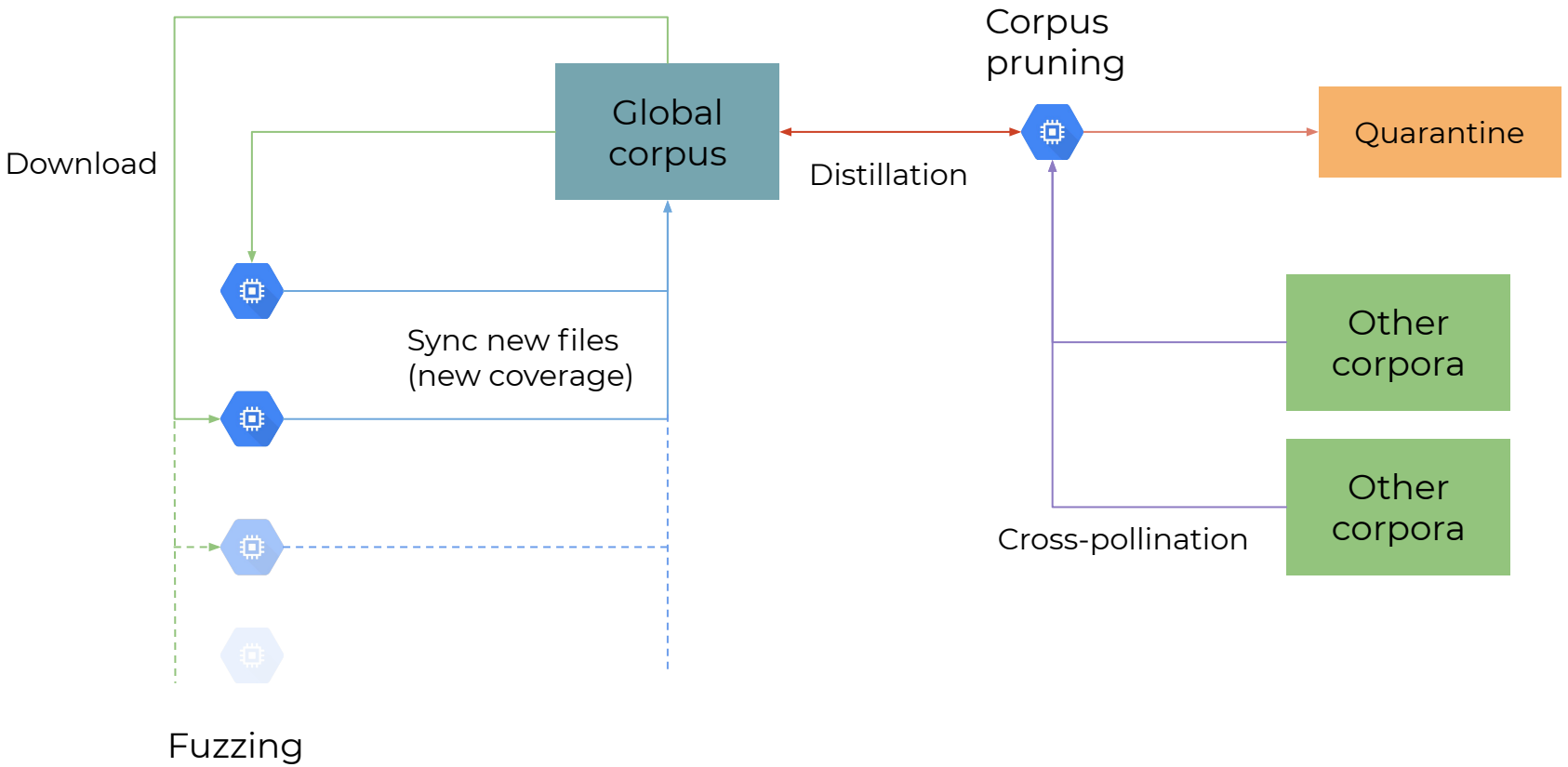
Fuzzing task management



Picking targets

- Large projects can have thousands of fuzz targets
- Automatic fuzz target discovery
- Prioritize based on fuzz target quality
  - Productive target > Unproductive target > Target with startup issues
- Prioritize based on sanitizer importance
  - ASan > MSan > Others (UBSan / CFI / TSAN)

Corpus management



Fuzzing strategies

- No **perfect** search heuristic
  - Corpus subset
  - Value profiling
  - Custom mutators
  - Limiting maximum length of inputs
- Corpus enhancement techniques
  - Radamsa mutator
  - Recurrent neural network mutator (ML-based)

Fuzzing strategy selection

- Multi-armed bandit (MAB)
  - Waste fewer resources on bad fuzzing strategies
  - Choose strategy combinations that improved coverage
  - Some runs use strategies with a default weight
    - Act as dynamic input to MAB model
  - Rest of runs use strategies based on MAB optimizations

Step 4: Triage crashes

De-duplication

- Based on stacktraces
  - (Crash type, Crash state, Security flag) tuple
- Pick top 3 “interesting” frames as the crash state
  - Include debug and release assertions
  - Exclude inline frames, common library and debug funcs
- Ignore stacktrace for OOMs and Timeouts
- Used for immediate de-duplication

De-duplication

```
[1:1:1030:FATAL:layout_inline.h(399)]Security DCHECK failed: !object || (object->IsLayoutInline()).
==1==ERROR: AddressSanitizer: ABRT on unknown address 0x053900000001 (pc 0x7f24f8426428 ...)
   #0 0x7f24f8426427 in gsignal/build/glibc-Cl5G7W/glibc-2.23/sysdeps/unix/sysv/linux/raise.c:54
   #1 0xb599eb3 in logging::LogMessage::~LogMessage()base/logging.cc:876:7
   #2 0x14171df5 in ToLayoutInlinethird_party/blink/renderer/core/layout/layout_inline.h:399:91
   #3 0x14171df5 in blink::LayoutBox::ContainingBlockLogicalWidthForPositioned(...)
   #4 0x1417b923 in blink::LayoutBox::ComputePositionedLogicalWidth(...)
```

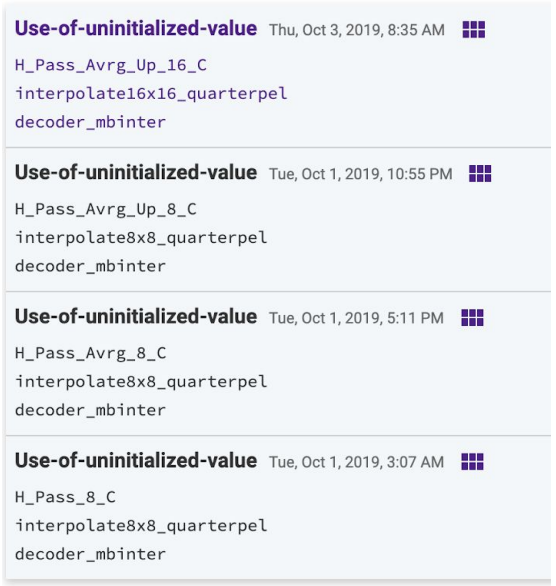
!object || (object->IsLayoutInline()) in layout_inline.h

blink::LayoutBox::ContainingBlockLogicalWidthForPositioned

blink::LayoutBox::ComputePositionedLogicalWidth

Grouping

- Second stage of de-duplication (slower)
- Same crash can manifest with a slightly different signature
- Use Levenshtein distance to group all similar crashes
- Works well with real world crashes

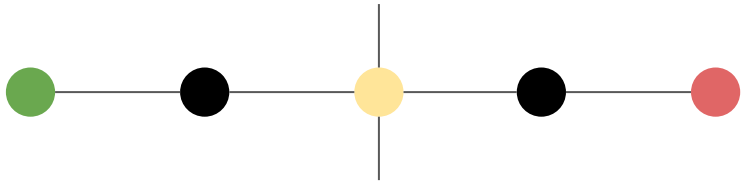


Testcase minimization

- Makes testcases less flaky and easier to root cause
- Grey box fuzzers
  - Often provide facilities for fast minimization
- Black box fuzzers
  - Delta debugging based testcase minimization
  - Slower out-of-process minimization, but parallelized

Bisection

- A large percent of bug finds are “regressions” (OSS-Fuzz: ~40%)
- Early reverts are far easier than CVEs
- Bisection based on simple binary search
- Re-use same builds used for fuzzing



REGRESSION REVISION RANGE

Chakra: 3ee8acae8ecca674d660baf52679b3a7c6169d88:93752a1fa83bd9c43520b992d49d9f7c0763fbeb

Variant analysis

- A crash input can manifest with different signatures across sanitizers, fuzzing engines, platforms, architectures
- Automate analysis across all possible configs
- Help with severity analysis

```
for (operand = 0, numBitsFound = 0, currentBit = 1 << ((opcodePTR->size * 8) - 1);
```

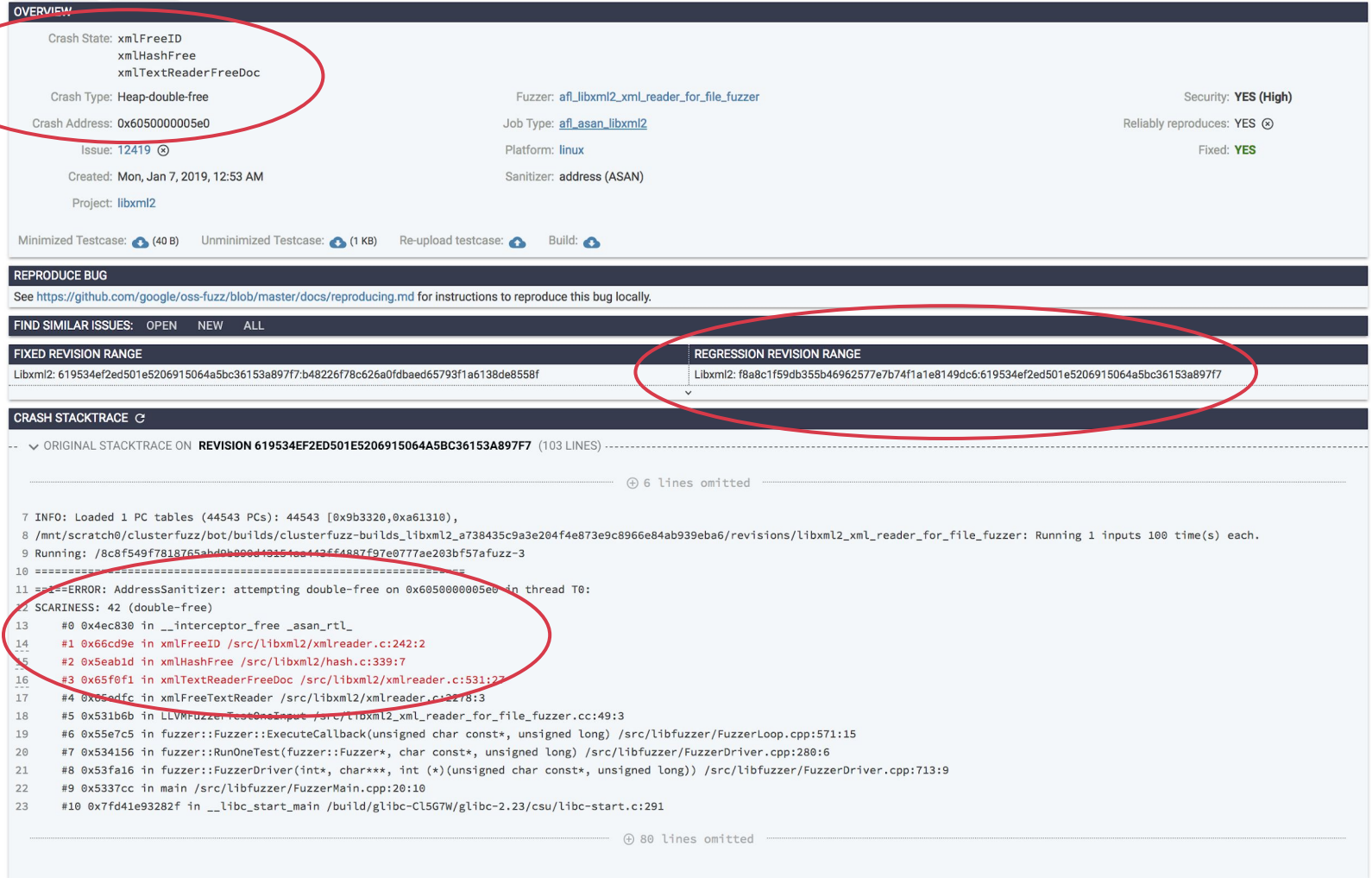
TESTCASE ANALYSIS ON OTHER JOBS							
JOB NAME	STATUS	REVISION	CRASH TYPE	CRASH STATE	SECURITY	SIMILAR	REPRODUCER
afl_asan_binutils	Reproducible	201910141554	Global-buffer-overflow READ 4	ripBits print_insn print_insn_xgate	true	true	Same
libfuzzer_ubsan_binutils	Reproducible	201910141554	Undefined-shift	ripBits print_insn print_insn_xgate	false	false	Same

Automatic bug filing

- Automatic assignment of owner based on bisect results
  - If failed, assign to sub-product area bug queue
- Provide minimized reproducer, detailed crash report
- “Fuzz-blocker” label if hurting fuzzer performance
- File ONLY reproducible crashes
  - Exception: frequent unreproducible crashes

Automatic bug filing

ID ▼	Type ▼	Component ▼	Status ▼	Proj ▼	Reported ▼	Owner ▼	Summary + Labels ▼
3122	Bug-Security	----	Verified	openssl	2017-08-22	----	openssl: Heap-buffer-overflow in X509v3_addr_get_afi ClusterFuzz Reproducible
3474	Bug-Security	----	Verified	openssl	2017-09-23	----	openssl: Index-out-of-bounds in tls1_set_ec_id ClusterFuzz Reproducible
7696	Bug-Security	----	Verified	openssl	2018-04-15	----	openssl/asn1: Heap-buffer-overflow in asn1_ex_i2c ClusterFuzz Reproducible
8241	Bug-Security	----	Verified	openssl	2018-05-12	----	openssl/server: Heap-use-after-free in ssl_get_prev_session ClusterFuzz Reproducible
15114	Bug-Security	----	Verified	openssl	2019-06-04	----	openssl/client: Stack-use-after-return in OSSL_PARAM_get_int32 ClusterFuzz Reproducible
16107	Bug-Security	----	Verified	openssl	2019-07-31	----	openssl/conf: Heap-buffer-overflow in OPENSSL_strlcpy ClusterFuzz Reproducible
17715	Bug-Security	----	New	openssl	2019-09-25	----	openssl:x509: Heap-buffer-overflow in CRYPTO_strdup ClusterFuzz Reproducible


OVERVIEW
Crash State: xmlFreeID
xmlHashFree
xmlTextReaderFreeDoc
Crash Type: Heap-double-free
Crash Address: 0x6050000005e0
Issue: 12419
Created: Mon, Jan 7, 2019, 12:53 AM
Project: libxml2
Fuzzer: afl_libxml2_xml_reader_for_file_fuzzer
Job Type: afl_asan_libxml2
Platform: linux
Sanitizer: address (ASAN)
Security: YES (High)
Reliably reproduces: YES
Fixed: YES
Minimized Testcase: (40 B)
Unminimized Testcase: (1 KB)
Re-upload testcase:
Build:
REPRODUCE BUG
See https://github.com/google/oss-fuzz/blob/master/docs/reproducing.md for instructions to reproduce this bug locally.
FIND SIMILAR ISSUES: OPEN NEW ALL
FIXED REVISION RANGE
Libxml2: 619534ef2ed501e5206915064a5bc36153a897f7:b48226f78c626a0fdbaed65793f1a6138de8558f
REGRESSION REVISION RANGE
Libxml2: f8a8c1f59db355b46962577e7b74f1a1e8149dc6:619534ef2ed501e5206915064a5bc36153a897f7
CRASH STACKTRACE
ORIGINAL STACKTRACE ON REVISION 619534EF2ED501E5206915064A5BC36153A897F7 (103 LINES)
6 lines omitted
INFO: Loaded 1 PC tables (44543 PCs): 44543 [0x9b3320,0xa61310),
/mnt/scratch0/clusterfuzz/bot/builds/clusterfuzz-builds_libxml2_a738435c9a3e204f4e873e9c8966e84ab939eba6/revisions/libxml2_xml_reader_for_file_fuzzer: Running 1 inputs 100 time(s) each.
Running: /8c8f549f7818765abd9b800d43154ae442ff4887f97e0777ae203bf57afuzz-3
==1==ERROR: AddressSanitizer: attempting double-free on 0x6050000005e0 in thread T0:
SCARINESS: 42 (double-free)
#0 0x4ec830 in __interceptor_free _asan_rtl_
#1 0x66cd9e in xmlFreeID /src/libxml2/xmlreader.c:242:2
#2 0x5eab1d in xmlHashFree /src/libxml2/hash.c:339:7
#3 0x65f0f1 in xmlTextReaderFreeDoc /src/libxml2/xmlreader.c:531:27
#4 0x65edfc in xmlFreeTextReader /src/libxml2/xmlreader.c:2278:3
#5 0x531b6b in LLVMFuzzerTestOneInput /src/libxml2_xml_reader_for_file_fuzzer.cc:49:3
#6 0x55e7c5 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) /src/libfuzzer/FuzzerLoop.cpp:571:15
#7 0x534156 in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) /src/libfuzzer/FuzzerDriver.cpp:280:6
#8 0x53fa16 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) /src/libfuzzer/FuzzerDriver.cpp:713:9
#9 0x5337cc in main /src/libfuzzer/FuzzerMain.cpp:20:10
#10 0x7fd41e93282f in __libc_start_main /build/glibc-Cl5G7W/glibc-2.23/csu/libc-start.c:291
80 lines omitted

Prioritization

- Don’t attempt to do deep analysis on bugs to figure out impact
  - Not scalable
- Assume **all** memory corruption are exploitable
- Rough automated prioritization based on
  - Type of crash (e.g. UaF vs null deref)
  - Where the crash occurs
    - E.g. if the crash occurs in a sandboxed process

Fix verification

- Verify that fix actually causes a crash to stop reproducing
- Perform a bisect to verify the commit which fixed the bug
  - Useful for comparing different bugs with same root cause
  - Sometimes, an unrelated patch can fix the issue
- Auto-close bugs once verified
- Human errors can be common

Vulnerability reward program

- External PoCs can be uploaded to the fuzzing infrastructure
- Get same benefits of automated triage
  - Deduplication, fix verification, etc
- Fuzzer reward program
  - Continuous bug reporting pipeline
  - High-quality reports

Step 5: Improving fuzzers

Fuzzer statistics

fuzzer	corpus_size	avg_exec_per_sec	fuzzing_time_percent	new_tests_added	new_features	regular_crash_percent	oom_percent	leak_percent	timeout_percent	startup_crash_percent
libFuzzer_aec3_config_json_fuzzer	4662 (1 MB)	1,200.7	14.1	2,142	29,858	92.6	0	0	0	0
libFuzzer_agc_fuzzer	2308 (37 MB)	319.4	98.7	5,586	429	0	0	0	0	0
libFuzzer_android_crazy_linker_zip_fuzzer	504 (5 MB)	19,540.5	98.2	3,439	438	0	0	0	0	0
libFuzzer_angle_translator_fuzzer	8168 (8 MB)	156.2	79.8	18,600	0	0	22.8	0	2.6	0
libFuzzer_apdu_fuzzer	265 (10 MB)	1,025.8	98.3	1,191	845	0	0	0	0	0
libFuzzer_appcache_fuzzer	5100 (98 MB)	3	96.9	124	171	0	15.7	0	0	0
libFuzzer_appcache_manifest_parser_fuzzer	6998 (163 MB)	40.6	97	6,607	232	0	3.1	0	0	0
libFuzzer_audio_decoder_fuzzer	9911 (23 MB)	36.6	64.4	24,559	713	0	33	9.4	0.9	0

Crash statistics

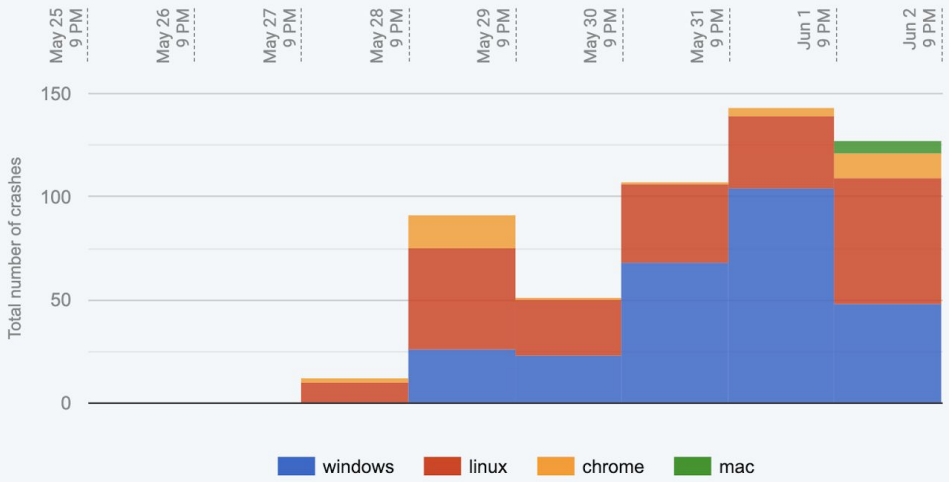
Heap-use-after-free READ 1

```
quic::QuicDataReader::PeekVarInt62Length
quic::HttpDecoder::ReadFrameType
quic::HttpDecoder::ProcessInput
```

Project chromium | Security | Reliably reproduces | New | Issue 981291

Time to crash: 1154.65s

Total count: 531



Code coverage

- Separate coverage instrumented build (Clang Source Code Cov)
- Run fuzz target with distilled corpus ->

  Per-fuzz target / Aggregate project report

Coverage Report

View results by: Directories | Files

PATH	LINE COVERAGE	FUNCTION COVERAGE	REGION COVERAGE
crypto/	35.58% (45125/126819)	38.07% (2224/5842)	29.81% (30811/103342)
engines/	5.15% (54/1049)	8.51% (4/47)	3.77% (24/637)
fuzz/	88.53% (571/645)	70.00% (28/40)	74.08% (403/544)
include/	76.00% (776/1021)	74.71% (130/174)	76.59% (373/487)
providers/	27.54% (2571/9334)	30.18% (185/613)	21.95% (1784/8127)
ssl/	44.41% (17425/39237)	42.44% (519/1223)	40.40% (13073/32360)
TOTALS	37.35% (66522/178105)	38.92% (3090/7939)	31.94% (46468/145497)

```
47        OPENSSL_LHASH *OPENSSL_LH_new(OPENSSL_LH_HASHFUNC h, OPENSSL_LH_COMPFUNC c)
48  22.8k {
49  22.8k     OPENSSL_LHASH *ret;
50  22.8k
51  22.8k     if ((ret = OPENSSL_zalloc(sizeof(*ret))) == NULL) {
52      0         /*
53      0          * Do not set the error code, because the ERR code uses LHASH
54      0          * and we want to avoid possible endless error loop.
55      0          * CRYPTOerr(CRYPTO_F_OPENSSL_LH_NEW, ERR_R_MALLOC_FAILURE);
56      0          */
57      0         return NULL;
58      0     }
59  22.8k     if ((ret->b = OPENSSL_zalloc(sizeof(*ret->b) * MIN_NODES)) == NULL)
60  22.8k         goto err;
61  22.8k     ret->comp = ((c == NULL) ? (OPENSSL_LH_COMPFUNC)strcmp : c);
62  22.8k     ret->hash = ((h == NULL) ? (OPENSSL_LH_HASHFUNC)OPENSSL_LH_strhash : h);
```

Other applications

Non-security bugs

- Correctness bugs via differential fuzzing, e.g. CryptoFuzz
  - Across different product implementations
  - Across different languages, compilers, optimizations
    - E.g. optimized ASM vs pure C
- Stability bugs
  - Denial-of-service attacks can be serious in many scenarios
  - Fixing stability bugs leads to a more productive fuzzer
    - Esp. leaks, ooms, timeouts, null-ptr dereferences

Design and development decisions

- Should we add this third party library to our project?
  - Require fuzzing or integration in OSS-Fuzz as prerequisite
  - How well are those endpoints fuzz tested? Coverage?
- Feedback to prioritize security mitigations
  - Sandboxing
  - Allocator hardening
  - etc

Results

Chrome, OSS and Google

- Overall **40,000** bugs found
- Vulnerabilities found: 5000+ in Chrome, 3500+ in OSS-Fuzz
- Vulnerabilities **fixed**: Chrome (**98.6%**), OSS (**91.4%**)
- Methodology works for both large projects (Chrome, Google) and smaller projects in OSS-Fuzz (250+)
- Developer evangelism - tech talks, contests, etc

Results outside of Google

WebKit

Available for: macOS Mojave 10.14.6, macOS High Sierra 10.13.6

Impact: Processing maliciously crafted web content may lead to arbitrary code execution

Description: Multiple memory corruption issues were addressed with improved memory handling.

CVE-2019-8710: found by OSS-Fuzz

CVE-2019-8743: zhunki from Codesafe Team of Legendsec at Qi'anxin Group

CVE-2019-8751: Dongzhuo Zhao working with ADLab of Venustech

CVE-2019-8752: Dongzhuo Zhao working with ADLab of Venustech

CVE-2019-8763: Sergei Glazunov of Google Project Zero

CVE-2019-8765: Samuel Groß of Google Project Zero

CVE-2019-8766: found by OSS-Fuzz

CVE-2019-8773: found by OSS-Fuzz

CVE-2018-5093: Buffer overflow in WebAssembly during Memory/Table resizing

Reporter OSS-Fuzz

Impact high

Description

A heap buffer overflow vulnerability may occur in WebAssembly during Memory/Table resizing, resulting in a potentially exploitable crash.

References

Bug 1415291

GNUTLS-SA-2017-3	CVE-2017-7869	Memory corruption	It was found using the OSS-FUZZ fuzzer infrastructure that decoding a specially crafted OpenPGP certificates could lead to (**A**) an integer overflow, resulting to an invalid memory write, (**B**) a null pointer dereference resulting to a server crash, and (**C**) a large allocation, resulting to a server out-of-memory condition. These affect only applications which utilize the OpenPGP certificate functionality of GnuTLS. The issues were fixed in 3.5.10. **Recommendation:** The support of OpenPGP certificates in GnuTLS is considered obsolete. As such, it is not recommended to use OpenPGP certificates with GnuTLS. To address the issues found upgrade to GnuTLS 3.5.10 or later versions.

CVE-2018-0739 (OpenSSL advisory) [Moderate severity] 27 March 2018:

Constructed ASN.1 types with a recursive definition (such as can be found in PKCS7) could eventually exceed the stack given malicious input with excessive recursion. This could result in a Denial Of Service attack. There are no such structures used within SSL/TLS that come from untrusted sources so this is considered safe. Reported by OSS-fuzz.

- Fixed in OpenSSL 1.1.0h (git commit) (Affected 1.1.0-1.1.0g)
- Fixed in OpenSSL 1.0.2o (git commit) (Affected 1.0.2b-1.0.2n)

Libarchive vulnerability can lead to code execution on Linux, FreeBSD, NetBSD

Bug discovered by Google. Impacts Linux and BSD distros, but not Windows and macOS.

v1.10.0

ChakraCore 1.10.0 includes more JavaScript and WebAssembly feature updates, performance enhancements, and JSRT APIs. See notable changes below.

Also shout-out to @rhuanjl, @fatcerberus, OSS-Fuzz, and Google Project Zero for their contributions during this release!

Sec Bug #77831 Heap-buffer-overflow in exif_iif_add_value in EXIF

Submitted: 2019-04-02 06:44 UTC **Modified:** 2019-04-15 06:53 UTC
From: stas@php.net **Assigned:** stas (profile)
Status: Closed **Package:** EXIF related
PHP Version: 7.1.27 **OS:** *
Private report: No **CVE-ID:** 2019-11035

View | Add Comment | Developer | Edit

[2019-04-02 06:44 UTC] stas@php.net

Description:

Another OSS-Fuzz bug: https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=13938

2018-04-30 7.0.7-29 Cristy <quetzlzacatenango@image...>
- Release ImageMagick version 7.0.7-29, GIT revision 14225:41edbdcea:20180430.

2018-03-26 7.0.7-29 <quetzlzacatenango@image...>
- Fixed numerous use of uninitialized values, integer overflow, memory exceeded, and timeouts (credit to OSS Fuzz).

2018-03-24 7.0.7-28 Cristy <quetzlzacatenango@image...>
- Release ImageMagick version 7.0.7-28, GIT revision 23615:edd71782e:20180325.

2018-03-21 7.0.7-28 <quetzlzacatenango@image...>
- Fixed numerous use of uninitialized values, integer overflow, memory exceeded, and timeouts (credit to OSS Fuzz).

2018-03-18 7.0.7-27 Cristy <quetzlzacatenango@image...>
- Release ImageMagick version 7.0.7-27, GIT revision 23466:734b146df:20180318.

2018-03-17 7.0.7-27 <quetzlzacatenango@image...>
- Fixed numerous use of uninitialized values, integer overflow, memory exceeded, and timeouts (credit to OSS Fuzz).

libxml2

Available for: Windows 7 and later

Impact: Multiple issues in libxml2

Description: Multiple memory corruption issues were addressed with improved input validation.

CVE-2019-8749: found by OSS-Fuzz

CVE-2019-8756: found by OSS-Fuzz

Future plans

Future plans

- Fuzzing as part of Continuous Integration (CI) to catch regressions before check-in
- Alternate solution to artificial fuzzer benchmarks (e.g. LAVA-M)
- Continue to improve fuzzing efficiency - e.g. more focused mutations using DataFlowSanitizer
- Support for more languages (Java, Python, etc)

Conclusion

- Fuzzing should be an integral part of developer workflows
  - Not just for security researchers
- Different fuzzing engines and strategies can be combined effectively at scale
- Large projects with thousands of developers can be fuzzed effectively with a small team
  - Smaller projects can use the same methodology