The Evolution of Fuzzing in Finding the Unknowns

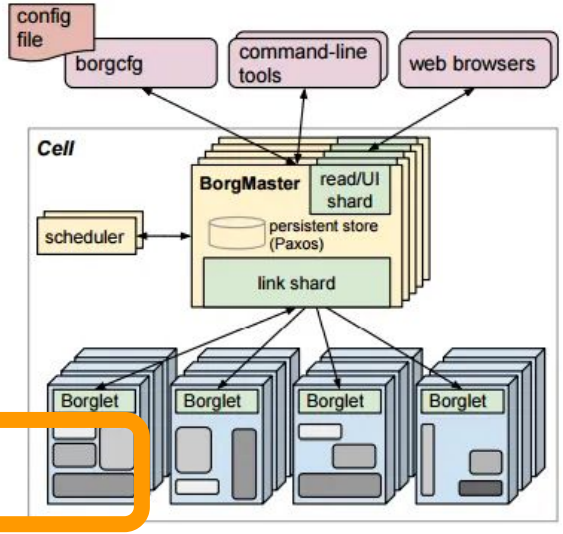
Abhishek Arya
OSST

About me!

- Googler
- Principal Engineer/Manager, Google Open Source Security Team (GOSST)
- TAC member, Open Source Security Foundation (OpenSSF)
- Founding Chrome Security member

2009-2010: It works!

- Prototype fuzzing at scale on Google Borg
- First example of “corpus distillation”
- Media parsers are hard to get right
- Simple mutations (e.g. bitflipping)



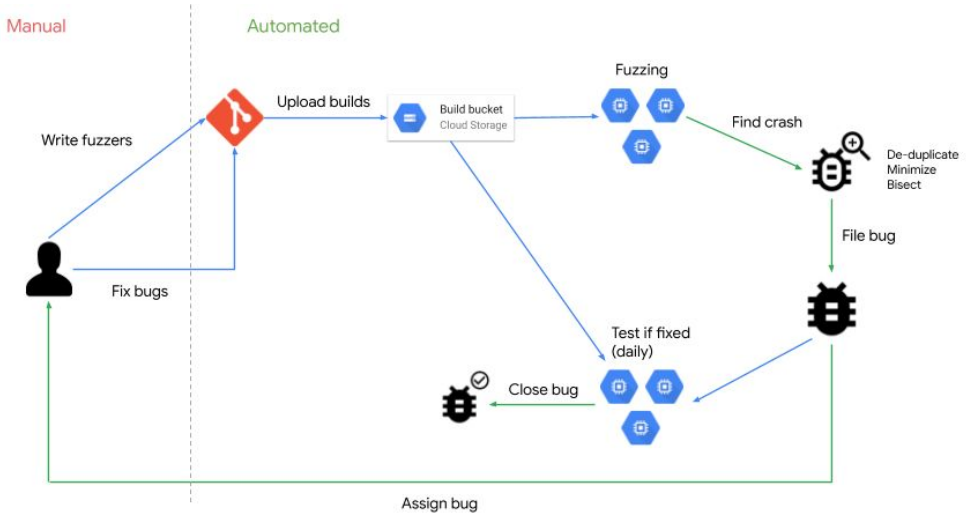
Fuzzing on free cycles

Figure 1: The high-level architecture of Borg. *Only a tiny fraction of the thousands of worker nodes are shown.*

Google

2011-2012: Figure out the pipeline

- Continuous fuzzing using ClusterFuzz, as part of SDLC
  - Build management
  - Task management
  - Test management
  - Crash management
  - Regression analysis
  - Fix verification



2012-2013: The rise of the Sanitizers

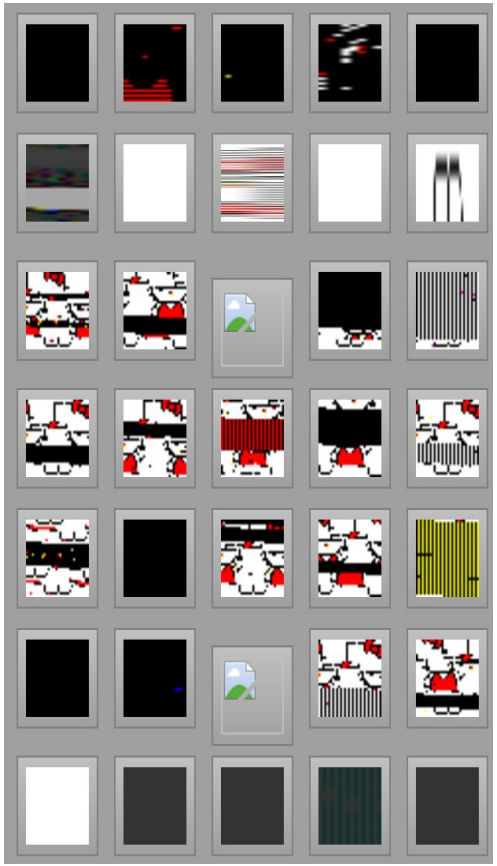
- Valgrind was impractical for efficient fuzzing
- Lack of instrumentation impacted reliability, usefulness
- Address, Memory, Thread, UndefinedBehavior Sanitizers
  - Fast (~1.5x slowdown), reproducible, insightful results
- Testcase Deduplication V1
- Security Asserts enabled builds

```
==158261==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60d0000023c0 at pc 0x55a4fea33789 bp 0x7ffef81
4b590 sp 0x7ffef814ad40
READ of size 72 at 0x60d0000023c0 thread T0
    #0 0x55a4fea33788 in __asan_memcpy /b/swarming/w/ir/kitchen-workdir/src/third_party/llvm/compiler-rt/lib/asan/a
san_interceptors_memintrinsics.cc:23:3
    #1 0x7f4f564790fd in exprCodeBetween third_party/sqlite/amalgamation/sqlite3.c:100476:11
    #2 0x7f4f56472410 in sqlite3ExprCodeTarget third_party/sqlite/amalgamation/sqlite3.c:100031:7
    #3 0x7f4f56470012 in sqlite3ExprCode third_party/sqlite/amalgamation/sqlite3.c:100305:13
    #4 0x7f4f56460f9c in sqlite3Insert third_party/sqlite/amalgamation/sqlite3.c:116578:9
    #5 0x7f4f56443301 in yy_reduce third_party/sqlite/amalgamation/sqlite3.c
    #6 0x7f4f5643e209 in sqlite3Parser third_party/sqlite/amalgamation/sqlite3.c:150805:15
    #7 0x7f4f5638b257 in sqlite3RunParser third_party/sqlite/amalgamation/sqlite3.c:151965:5
    #8 0x7f4f56439d0d in sqlite3Prepare third_party/sqlite/amalgamation/sqlite3.c:123386:5
    #9 0x7f4f56389c26 in sqlite3LockAndPrepare third_party/sqlite/amalgamation/sqlite3.c:123479:10
    #10 0x7f4f56375483 in chrome_sqlite3_prepare_v2 third_party/sqlite/amalgamation/sqlite3.c:123850:8
    #11 0x55a4fea9a689 in sql_fuzzer::RunSqlQueriesOnConnection(sqlite3*, std::__1::vector<std::__1::basic_string<c
har, std::__1::char_traits<char>, std::__1::allocator<char> >, std::__1::allocator<std::__1::basic_string<char, std
::__1::char_traits<char>, std::__1::allocator<char> > > >) third_party/sqlite/fuzz/sql_run_queries.cc:73:10
    #12 0x55a4fea9b1fb in sql_fuzzer::RunSqlQueries(std::__1::vector<std::__1::basic_string<char, std::__1::char_tr
aits<char>, std::__1::allocator<char> >, std::__1::allocator<std::__1::basic_string<char, std::__1::char_traits<cha
r>, std::__1::allocator<char> > > >) third_party/sqlite/fuzz/sql_run_queries.cc:130:3
    #13 0x55a4fea646c9 in TestOneProtoInput(sql_query_grammar::SQLQueries const&) third_party/sqlite/fuzz/sql_fuzze
r.cc:41:3
    #14 0x55a4fea64043 in LLVMFuzzerTestOneInput third_party/sqlite/fuzz/sql_fuzzer.cc:28:1
    #15 0x55a4feaaefb5 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) third_party/libFuzze
r/src/FuzzerLoop.cpp:571:15
    #16 0x55a4feaa626c in fuzzer::RunOneTest(fuzzer::Fuzzer*, char const*, unsigned long) third_party/libFuzzer/src
/FuzzerDriver.cpp:280:6
    #17 0x55a4feaa892b in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) third_p
arty/libFuzzer/src/FuzzerDriver.cpp:713:9
    #18 0x55a4feab4ca2 in main third_party/libFuzzer/src/FuzzerMain.cpp:20:10
```

Google

2014-2015: Smart coverage guided fuzzing

- American Fuzzy Lop (AFL)
  - Supports binary, source-code instrumentation
  - Out-of-process fuzzing (+later in-process)
- libFuzzer
  - Support source-code instrumentation only
  - In-process fuzzing
  - Developer focused, fuzzer unit-tests
  - Custom mutators for structure-aware fuzzing



Google

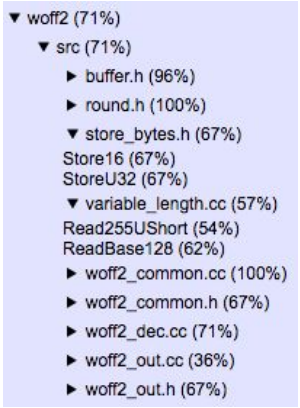
2014-2015: Smart coverage guided fuzzing (cntd)

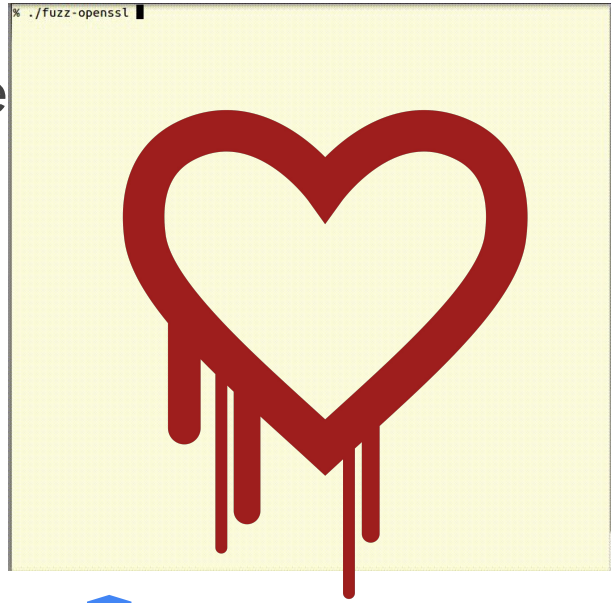
```
#include "libxml/parser.h"

extern "C" int LLVMFuzzerTestOneInput(
    const uint8_t *data, size_t size) {

  auto doc = xmlReadMemory(data, size,

"noname.xml",
                                NULL, 0);
  if (doc) {
    xmlFreeDoc(doc);
  }

  return 0;
}
```

	ID	Product	Summary
	756459	libxml2	memory leak in xmlNewDocElementContent
🔒	756479	libxml2	heap-buffer-overflow in xmlGROW
🔒	756525	libxml2	heap-buffer-overflow in xmlParseMisc
🔒	756527	libxml2	heap-buffer-overflow xmlParseXMLDecl
🔒	756528	libxml2	heap-buffer-overflow in xmlDictComputeFastQKe
🔒	756733	libxml2	tiny input takes 4+ seconds to process (xmlStrin
🔒	757711	libxml2	heap-buffer-overflow in xmlFAParsePosCharGrou
🔒	758549	libxml2	heap-buffer-overflow in xmlParseEndTag2
🔒	759573	libxml2	Heap-based buffer-underreads due to xmlParseN
	759671	libxml2	Heap-based buffer overread in xmlNextChar (fro
	761430	libxml2	xmlReadMemory causes undesired side effects
	756456	libxml2	heap-buffer-overflow in xmlParseConditionalSect

2016-2017: Scaling with the community

- OSS-Fuzz service launched to fuzz open-source
- Regressions reported in a few hours
  - Automation+Ease of use=90% fix rate
- Community input drove key features
  - Code coverage reports
  - Custom mutators
  - Ideal integrations





2016-2017: Scaling with the community (cntd)

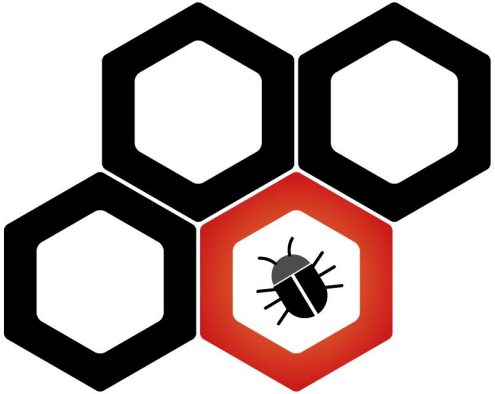
Internal FuzzIts

External OSS-Fuzz Rewards

To qualify for these rewards, a project needs to have a large user base and/or be critical to global IT infrastructure. **Eligible projects will receive $1,000 for initial integration, and up to $20,000 for ideal integration** (the final amount is at our discretion). You have the option of donating these rewards to charity instead, and Google will double the amount.

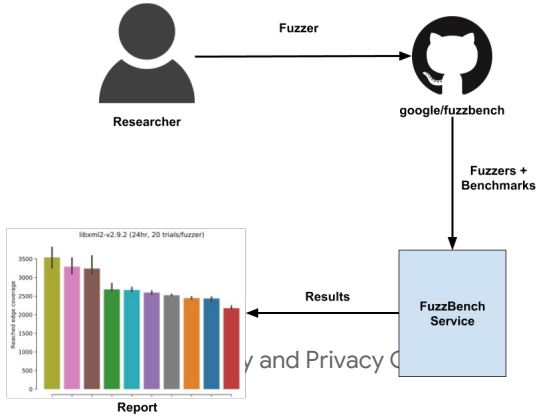
2018-2019: Open Sourced ClusterFuzz

- Testcase deduplication v2, high-quality automated filing
- Fuzz target performance analyzer
- Efficiency improvements
  - Corpus enhancements: cross-pollination, radamsa, etc
  - Distributed corpus sharing and pruning
- Cross-platform (Win, Linux, Mac, Android)
- First-class support for external fuzzers



2020-2021: Real-world benchmarking

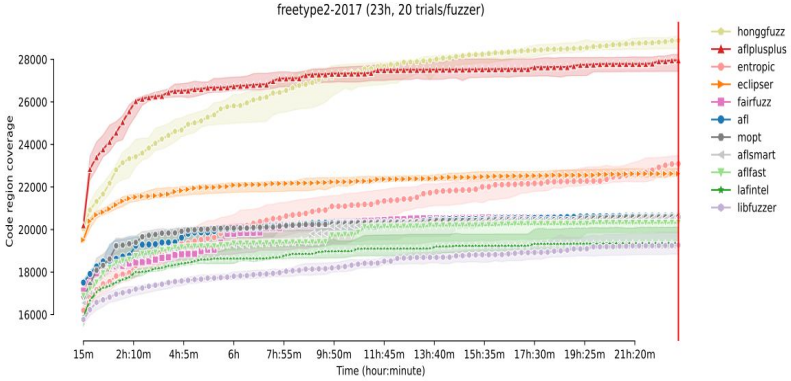
- FuzzBench service launched to compare fuzzer efficacy
- Why it worked well ?
  - Zero cost for large-scale experiments (4K cores)
  - Diverse, real world OSS-Fuzz benchmarks
  - Automated, easy to use workflows
  - Reproducible results
  - Support for private experiments



Google

2020-2021: Real-world benchmarking (cntd)

- Fuzzing engine improvements
  - libFuzzer, honggfuzz (interceptors, corpus size, input scheduling, etc)
  - **Developer testbed - AFL++**
- Validation for fuzzing research
  - Entropic, SymQEMU, StateAFL, WAFL, E9AFL, AFL-HIER, BigMap, etc



2022 and future: Predictions for the future

- Prioritized list of fuzzing chokepoints
- Coverage-guided property-based tests
- Non-memory corruption sanitizers
- Practical concolic execution
- Your ideas?

```
4 ks_release [function] [call site]
4 sam_hdr_sanitise [function] [call site]
  5 hts_log [function] [call site]
  5 sam_hdr_destroy [function] [call site]
  5 hts_log [function] [call site]
  5 hts_log [function] [call site]
  5 hts_log [function] [call site]
  5 sam_hdr_destroy [function] [call site]
  5 sam_hdr_destroy [function] [call site]
4 free [call site]
4 sam_hdr_destroy [function] [call site]
4 ks_free [function] [call site]
4 kh_destroy_s2i [function] [call site]
4 free [call site]
```

Questions?

- Reach out at
  - Email: aarya@google.com
  - Twitter: @infernosec
  - LinkedIn: linkedin.com/in/abhishek-arya-a565373/
  - OSS-Fuzz: oss-fuzz@google.com
  - FuzzBench: fuzzbench@google.com
  - OpenSSF fuzzing community meeting (monthly)