Fuzz or lose!
Why and how to make fuzzing a standard practice for C++

Kostya Serebryany, Google
CppCon 2017

Agenda

- Why fuzz?
- Fuzzing case studies
- OSS-Fuzz: continuous automated fuzzing
- Adoption challenges
- No implementation details

Testing vs Fuzzing

```
// Test
MyApi(Input1);
MyApi(Input2);
MyApi(Input3);
```

```
// Fuzz
while (true)
   MyApi(
      Fuzzer.GenerateInput());
```

Types of fuzzing engines

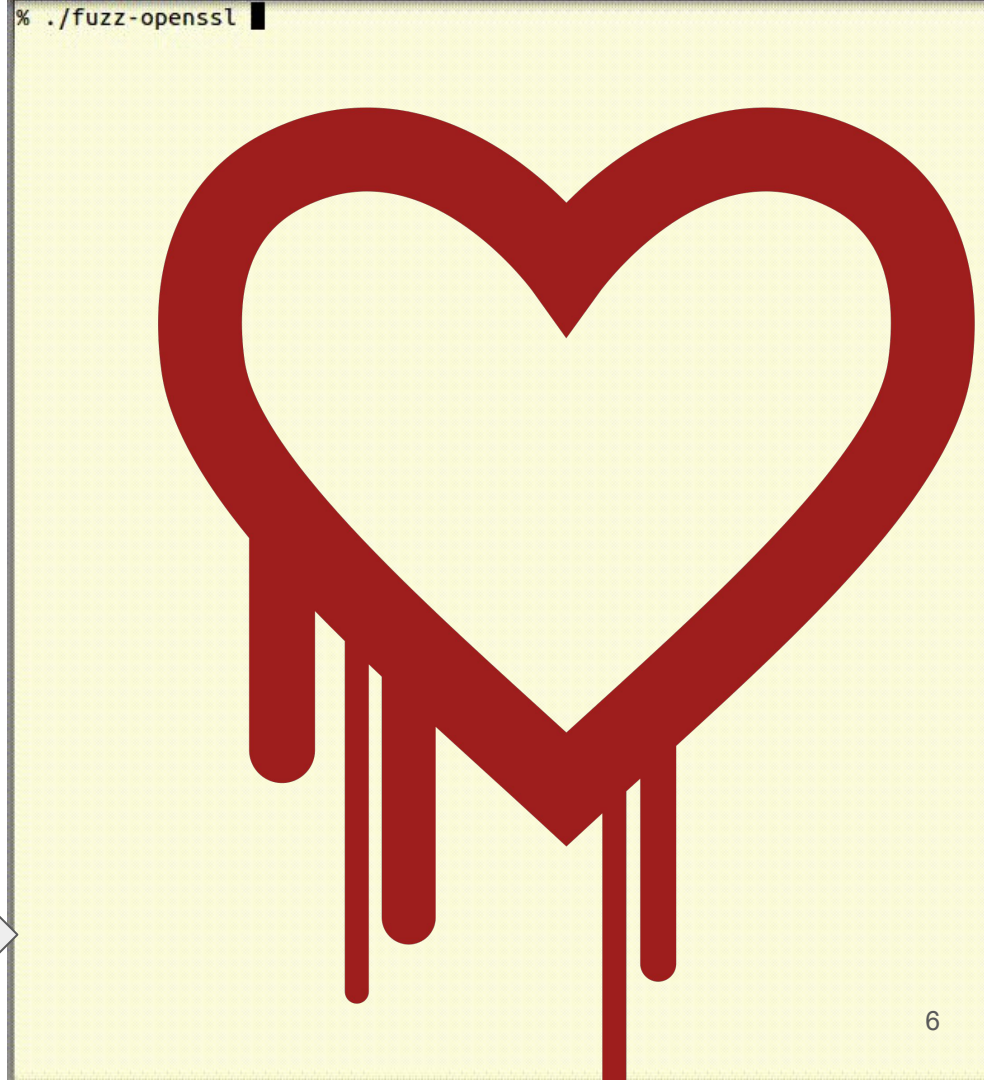
- Coverage-guided
- Generation-based
- Symbolic execution
- …

Why Fuzz C++ code?

Hackers love C / C++

(but not like you do)

libFuzzer finds Heartbleed



Boooo!

Did he just say C / C++?

C++ inherited memory safety bugs from C

- Buffer-overflow
  - Stack, heap, globals
- Heap-use-after-free
- Use of uninitialized memory
- …

Boooo!

“Modern C++” doesn’t have memory safety problems!

Can you spot the bug?

```
std::string s = "Hellooooooooooooooo ";
std::string_view sv = s + "World\n";
std::cout << sv;

// std::string_view is a C++17 feature!
```

Finding security bugs in C++17 code since 2011

```
std::string s = "Helloooooooooooooo ";
std::string_view sv = s + "World\n";
std::cout << sv;        // << OOPS
```

```
% clang++ -std=c++11 string_view_uaf.cc -stdlib=libc++ -fsanitize=address && ./a.out
ERROR: AddressSanitizer: heap-use-after-free
 READ of size 26 at 0x603000000010 thread T0
 0x603000000010 is located 0 bytes inside of 32-byte region ...
 freed by thread T0 here:
```

Even trickier

```
std::string s = "Hello ";
std::string_view sv = s + "World\n";
std::cout << sv;
```

Let’s fuzz some modern C++

https://github.com/google/woff2

- C++11
- Coding style
- Code review
- Unit tests
- CI
- STL containers
- Iterators
- Namespaces
- Bells & whistles

https://github.com/google/woff2

- C++11
- Coding style
- Code review
- Unit tests
- CI
- STL containers
- Iterators
- Namespaces
- ~~Bells & whistles~~
- **Fuzzing**

```
==22355==ERROR: AddressSanitizer: heap-buffer-overflow
WRITE of size 12960 [redacted]a thread T0
    #0 0x4bd026 in __asan_memcpy
    #1 0x6219d8 in woff2::Buffer::Read buffer.h:86:7
    #2 0x6219d8 in woff2::ReconstructGlyf woff2_dec.cc:500
    #3 0x6219d8 in woff2::ReconstructFont woff2_dec.cc:917
    #4 0x6219d8 in woff2::ConvertWOFF2ToTTF woff2_dec.cc:1282

0x627000006baa is located 0 bytes to the right of 12970-byte reg
allocated by thread T0 here:
    #0 0x4e7ddb in operator new[]
    #1 0x623d75 in woff2::ReconstructGlyf woff2_dec.cc:483:25
    #2 0x623d75 in woff2::ReconstructFont woff2_dec.cc:917
    #3 0x623d75 in woff2::ConvertWOFF2ToTTF woff2_dec.cc:1282
```

Woff2 Fuzz Target

```
extern "C" // fuzz.cpp
int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
    std::string buf;
    woff2::WOFF2StringOut out(&buf);
    out.SetMaxSize(30 * 1024 * 1024);
    woff2::ConvertWOFF2ToTTF(data, size, &out);
    return 0;
}
```

How to use libFuzzer

```
# Get *fresh* clang

% clang -g -O1 -fsanitize=address,fuzzer fuzz.cpp lib/*.cpp

% mkdir SEED_CORPUS_DIR  # and put some samples here

% ./a.out SEED_CORPUS_DIR
```

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)

Security + Stability >

Memory Safety

Rust: cargo fuzz

- `arith`: Arithmetic error, eg. overflows
- `logic`: Logic bug
- `loop`: Infinite loop
- `oom`: Out of memory
- `oor`: Out of range access
- `segfault`: Program segfaulted
- `so`: Stack overflow
- `uaf`: Use after free (In Rust ???)
- `unwrap`: Call to `unwrap` on `None` or `Err(_)`
- `utf-8`: Problem with UTF-8 strings handling
- `panic`: A panic not covered by any of the above
- `other`:

https://svn.boost.org/trac10/ticket/12818 (regex)

- 5 heap-buffer-overflows
- stack overflow
- assert failure
- use of uninitialized data
- SIGSEGV
- infinite loop
- undefined shift
- invalid enum value
- a bunch of memory leaks
- **in just half an hour**
- **reported 8 months ago** by Dmitry Vyukov

https://svn.boost.org/trac10/ticket/12818 (regex)

```
extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  try {
    std::string str((const char*)Data, Size);
    boost::regex e(str);
    boost::match_results<std::string::const_iterator> what;
    boost::regex_match(str, what, e,
        boost::match_default | boost::match_partial);
  } catch (const std::exception&) {} return 0;
}
```

boost::regex bugs were fixed …

… but **continuous fuzzing** was never set up :(

boost::regex added to OSS-Fuzz* 5 days ago

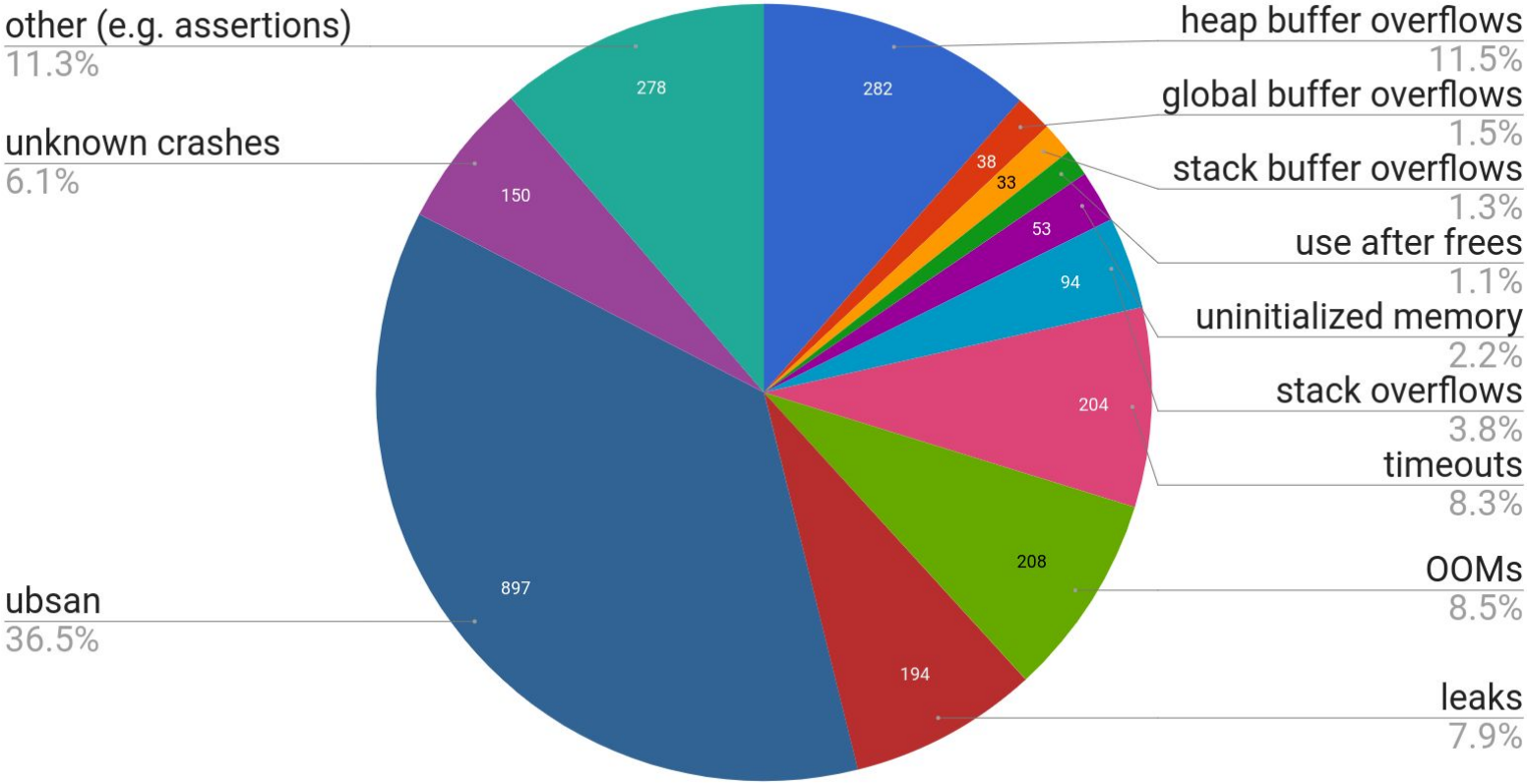
3460 boost: Integer-overflow in boost::re_detail_NUMBER::basic_regex_parser...
3464 boost: Integer-overflow in boost::re_detail_NUMBER::perl_matcher...
3469 boost: ASSERT: jmp->type == syntax_element_jump
3471 boost: Stack-overflow in boost::re_detail_NUMBER::basic_regex_parser...
3472 boost: Stack-overflow in boost::re_detail_NUMBER::perl_matcher…
3478 boost: **Stack-buffer-overflow** in boost::re_detail_NUMBER::perl_matcher...
3479 boost: Null-dereference READ in boost::re_detail_NUMBER::basic_regex...

(*) What's OSS-Fuzz?

OSS-Fuzz: Fuzzing as a Service

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

OSS-Fuzz: 2000+ bugs in 60+ OSS projects



What if my code is not open-source?

- OSS-Fuzz is for OSS-only
  - currently requires our approval
  - we won’t accept “toy” projects

- The tools are open-source
  - libFuzzer, AFL, Sanitizers
  - Linux & Mac are fully supported
  - Windows: YMMV
    - But: Microsoft Security Risk Detection

Back to Fuzzing

Structure-aware fuzzing

- Not every API consumes simple data
- Naive fuzzing creates invalid data => inefficient
- Fuzzing needs to be aware of the input structure

Case study:

let’s fuzz a C++ compiler (Clang)

Fuzzing a C++ compiler: naive

heap-buffer-overflow in clang::Lexer::SkipLineComment on a 4-byte input

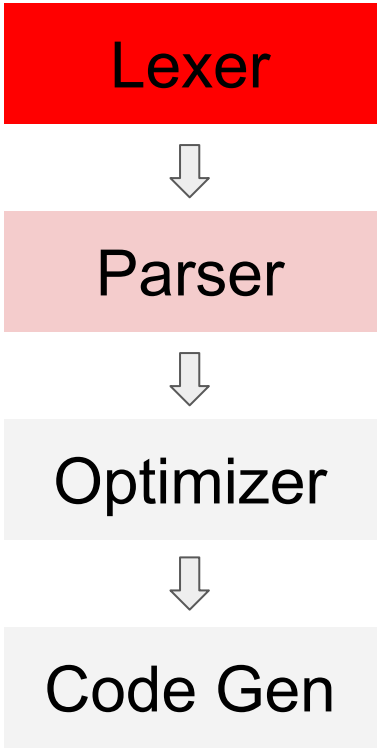
```
//\\
```

use-after-free or Assertion `Tok.is(tok::eof) && Tok.getEofData() == AttrEnd.getEofDat a()'.

```
cass⍰F{c<(F((F⍰F(;;)))))(
```

infinite CPU and RAM consumption on a 62-byte input

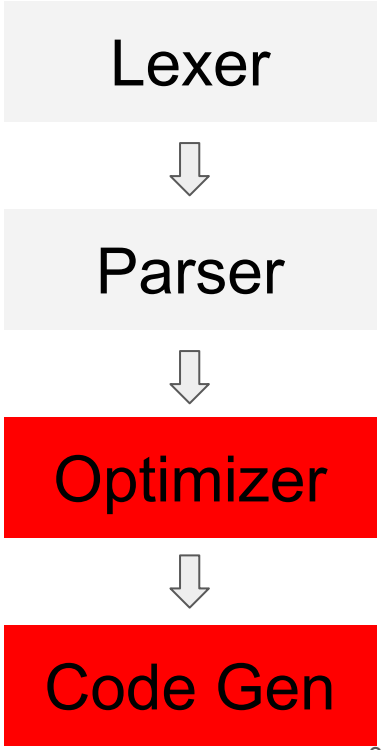
```
cFjass⍰⍰⍰F:⍰{⍰F*NFF(;F*⍰FF=F(JFF=F:
FFF.FFF-VFF,FFF-FFF'
```



Fuzzing a C++ compiler: structure-aware

clang hangs in llvm::JumpThreadingPass::ComputeValueKnownInPredecessors

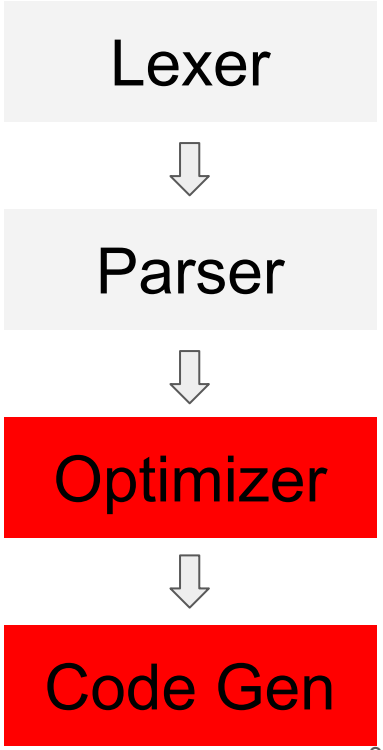
```
void foo(int *a) {
  while ((1 + 1)) {
    while ((a[96] * a[96])) {
      a[0] = (1024);
      while (a[0]) {
        while (a[0]) {
          (void)0;
          while ((a[96] * ((a[96] * a[96]) < 1))) {
            a[96] = (1 + 1);
          }
          a[0] = (a[0] + a[0]);
        }
      }
    }
```



Fuzzing a C++ compiler: structure-aware

use-after-poison in llvm::SelectionDAG::Combine

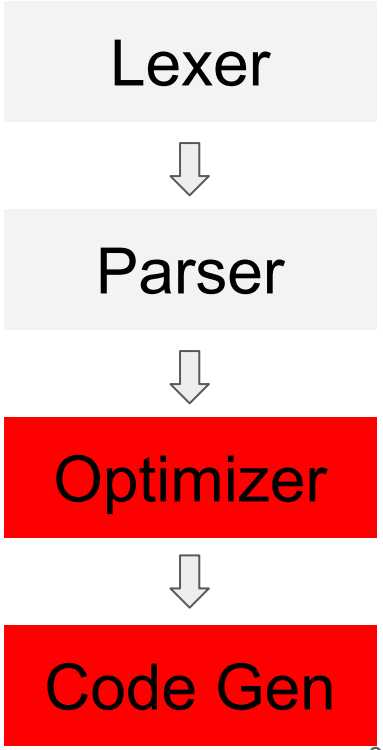
```
void foo(int *a) {
  while (1) {
    a[0] = (a[0] + (15134));
    while ((1 / a[6])) {
      (void)0;
    }
    a[0] = (a[0] + (1 + 1));
    a[8] =
        ((((((((((((a[63] % (-2147483648)) + a[0]) * a[0]) * a[0]) *
                  (-2147483648)) *
                 a[0]) +
                ((1 + 1) + (0))) -
               a[0]) *
              ((((((((a[63] % (-2147483648)) + a[0]) * a[0]) * a[0]) * a[0]) *
                 a[0]) +
                ((1 + 1) + (0))) *
               a[0])) -
             a[0]) *
```



Fuzzing a C++ compiler: structure-aware

fatal error: error in backend: Cannot select: t195: i1 = add t192, t194 (in HexagonDAGToDAGISel::Select)

```
void foo(int *a) {
  while ((
      (((a[0] -
         (((((((((1 * (((((1 + a[26]) * a[0]) + a[0]) * a[0]) * a[0])) * a[0]) *
                a[0]) *
               a[0]) *
              (((((((1 + (((((1 + a[26]) * a[0]) + a[0]) * a[0]) * a[0])) *
                    a[0]) +
                   a[0]) *
                  a[0]) *
                 a[0]) &
                1) -
               1)) &
              1) -
             1) *
            1) *
           a[26])) *
         a[0]) *
        a[0]) +
       a[0])) {
    a[0] = (((a[26] * 1) + a[0]) * 1);
```



Structure-aware fuzzing with libFuzzer

- Fuzzer needs a “Custom Mutator”
  - User-provided method to apply a single local mutation to an input

- https://github.com/google/libprotobuf-mutator
  - Custom mutator for protobufs
  - Can be used with non-protobuf inputs (e.g. Clang)

Fuzzing can find logical bugs too!

- Anything that has two implementations and a way to compare outputs: crypto, compression, rendering, …

- ```
  assert(OptimizedFoo(InputData) ==
          ReferenceFoo(InputData));
  ```

- CVE-2017-3732, a carry propagating bug in OpenSSL
  ```

Useful?

Simple?

Simple + Useful

!=

Widely Used

Adoption at Google

- 2000+ fuzz targets are continuously & automatically fuzzed
  - Server-side code, Chromium, OSS-Fuzz
  - And growing

- How did we do this?
  - We control the build system => made fuzzing builds super easy
  - Automated bug finding, reporting, and tracking
  - Held FuzzIts, Fuzzathons, and Fuzzing weeks
  - Advertized fuzzing in Google toilets worldwide, 3 times
    - Yes, really!

Adoption elsewhere: YMMV

☹?

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
  - Not specific to C++, see e.g. rust-fuzz

We need to make Fuzzing simpler

- Language
- IDEs
- Compilers, build systems
- ...

Proposal: C++ attribute

```
[[fuzz]]

void MyApi(const uint8_t *Data, size_t Size) {…}
```

Proposal: C++ attribute

```
[[fuzz]]
void MyApi(const std::string &s) {…}
[[fuzz]]
void AnotherApi(const std::vector<uint8_t> &v) {…}
```

Proposal: C++ attribute

```
[[fuzz]]

void MyApi(const T &Data) {…}
```

Proposal: C++ attribute

```
[[fuzz]]

void MyApi(const T &Data) {…}

   std::vector<uint8_t> serialize(const T&);

   T deserialize(const std::vector<uint8_t> &);

   T mutate(const T&);  // optional
```

C++ Memory Safety > Fuzzing

- Hardware-assisted memory safety
  - SPARC ADI
  - Intel {ENDBRANCH, CET, MPX}

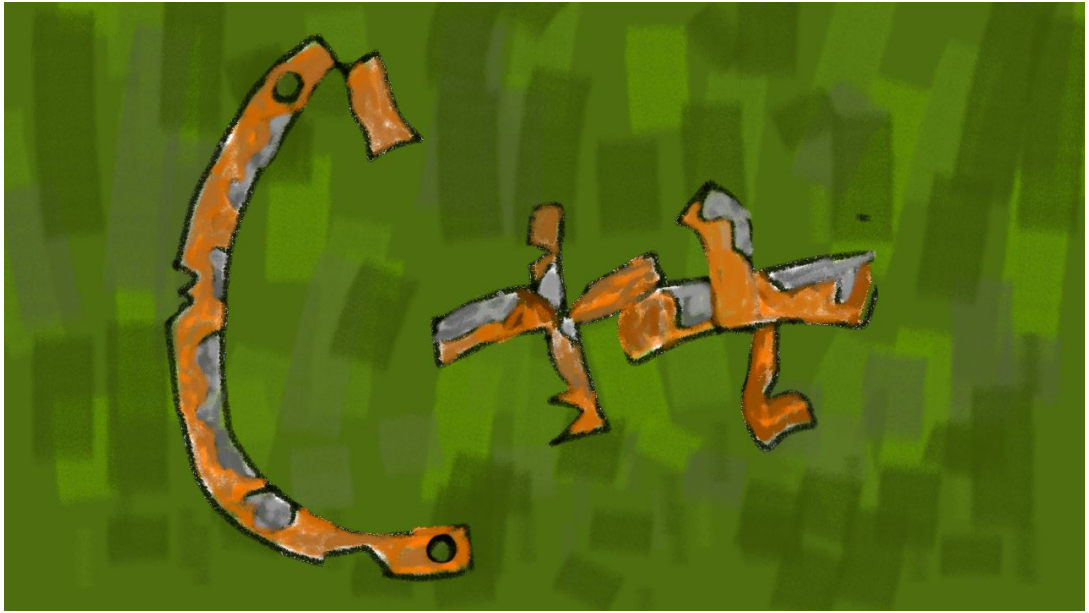
- Statically-verifiable safe subset of C++
  - C++ Core Guidelines?

Summary

- Fuzzing C++ code:
  - simple
  - prevents bugs

- We must make it:
  - even simpler
  - widely adopted

Fuzz, or



is Going to Swiftly **RUST**

Links

http://libfuzzer.info

http://tutorial.libfuzzer.info

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