Structure-aware fuzzing
for Clang and LLVM with libprotobuf-mutator

Kostya Serebryany, Vitaly Buka, Matt Morehouse; Google
October 2017

Agenda

- Fuzzing
- Fuzzing Clang/LLVM
- Fuzzing Clang/LLVM better (structure-aware)
  - llvm-isel-fuzzer
  - clang-proto-fuzzer

Testing vs Fuzzing

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

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

Types of fuzzing engines

- Coverage-guided
  - libFuzzer
  - AFL
- Generation-based
  - Csmith
- Symbolic execution
  - KLEE
- ...

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

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
```

Simple Fuzzers in LLVM

- clang-format-fuzzer
- clang-fuzzer
- llvm-dwarfdump-fuzzer
- llvm-as-fuzzer
- llvm-mc-assemble-fuzzer
- llvm-mc-disassemble-fuzzer
- llvm-demangle-fuzzer (llvm) & cxa_demangle_fuzzer (libcxxabi)
- ...

OSS-Fuzz + LLVM

- https://github.com/google/oss-fuzz
  - Continuous automated fuzzing for OSS projects
  - Usenix Security 2017

- TL;DR: fuzzers in, bug reports out

- LLVM: https://github.com/google/oss-fuzz/tree/master/projects/llvm/

cxa_demangle_fuzzer

```
extern "C"
int LLVMFuzzerTestOneInput(
    const uint8_t *data, size_t size) {
  char *str = new char[size+1];
  memcpy(str, data, size);
  str[size] = 0;
  free(__cxa_demangle(str, 0, 0, 0));
  delete [] str;
  return 0;
}
```

llvm_libcxxabi: Stack-buffer-overflow in std::__1::basic_string<char, std::__1::char_traits<char>, __cxxabiv1::malloc_

llvm_libcxxabi: Negative-size-param in std::__1::char_traits<char>::move ClusterFuzz Reproducible

llvm_libcxxabi: Out-of-memory in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Timeout in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Negative-size-param in std::__1::char_traits<char>::copy ClusterFuzz Reproducible

llvm_libcxxabi: Out-of-memory in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Timeout in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: ASSERT: k0 <= k1 && "parse_type() mutated the name stack" ClusterFuzz Reproducible

llvm_libcxxabi: ASSERT: k0 <= k1 && "parse_type() mutated the name stack" ClusterFuzz Reproducible

llvm_libcxxabi: Out-of-memory in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Timeout in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Timeout in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Out-of-memory in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Out-of-memory in llvm_libcxxabi_cxa_demTraceback ClusterFuzz Reproducible

llvm_libcxxabi: Use-of-uninitialized-value in __cxxabiv1::parse_nested_name ClusterFuzz Reproducible

llvm_libcxxabi: Null-dereference READ in __cxxabiv1::parse_nested_name ClusterFuzz Reproducible

llvm_libcxxabi: Sanitizer CHECK failure in ((IsAligned(reinterpret_cast<uptr>(p), page_size_))) != (0) (0, 0) ClusterFuzz

llvm_libcxxabi: Bad-free in std::__1::__vector_base<std::__1::vector<__cxxabiv1::Node*, __cxxabiv1::short_al Cluster

llvm_libcxxabi: ASSERT: FromPosition <= names.size() ClusterFuzz Reproducible

llvm_libcxxabi: Use-of-uninitialized-value in __cxxabiv1::Node::hasRHSComponent ClusterFuzz Reproducible

llvm_libcxxabi: Null-dereference READ in __cxxabiv1::Node::hasRHSComponent ClusterFuzz Reproducible

llvm_libcxxabi: Bad-free in std::__1::__vector_base<std::__1::vector<__cxxabiv1::Node*, __cxxabiv1::short_al Cluster

llvm_libcxxabi: ASSERT: FromPosition <= names.size() ClusterFuzz Reproducible

llvm_libcxxabi: Negative-size-param in __cxxabiv1::NodeArray __cxxabiv1::Db::makeNodeArray<std::__1::__wrap_it

llvm_libcxxabi: Out-of-memory in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Reproducible

llvm_libcxxabi: Heap-buffer-overflow in __cxxabiv1::parse_encoding ClusterFuzz Reproducible

llvm_libcxxabi: Heap-buffer-overflow in __cxxabiv1::parse_new_expr ClusterFuzz Reproducible

llvm_libcxxabi: Timeout in llvm_libcxxabi_cxa_demangle_fuzzer ClusterFuzz Unreproducible

clang-format-fuzzer

```
extern "C" int LLVMFuzzerTestOneInput(uint8_t *data, size_t size) {
  // FIXME: fuzz more things: different styles, different style features.
  std::string s((const char *)data, size);
  auto Style = getGoogleStyle(clang::format::FormatStyle::LK_Cpp);
  Style.ColumnLimit = 60;
  auto Replaces = reformat(Style, s, clang::tooling::Range(0, s.size()));
  auto Result = applyAllReplacements(s, Replaces);

  // Output must be checked, as otherwise we crash.
  if (!Result) {}
  return 0;
}
```

llvm-dwarfdump-fuzzer

```
extern "C"
int LLVMFuzzerTestOneInput(uint8_t *data, size_t size) {
  std::unique_ptr<MemoryBuffer> Buff = MemoryBuffer::getMemBuffer(
      StringRef((const char *)data, size), "", false);

  Expected<std::unique_ptr<ObjectFile>> ObjOrErr =
      ObjectFile::createObjectFile(Buff->getMemBufferRef());
  if (auto E = ObjOrErr.takeError()) {
    consumeError(std::move(E));
    return 0;
  }
  ObjectFile &Obj = *ObjOrErr.get();
  std::unique_ptr<DIContext> DICtx = DWARFContext::create(Obj);


  DIDumpOptions opts;
  opts.DumpType = DIDT_All;
  DICtx->dump(nulls(), opts);
  return 0;
}
```

llvm: ASSERT: result <= UINT32_MAX ClusterFuzz Reproducible
llvm: Heap-buffer-overflow in llvm::object::WasmObjectFile::pars
llvm: Heap-buffer-overflow in llvm::StringMapImpl::LookupBucke
llvm: Heap-buffer-overflow in llvm::identify_magic ClusterFuzz Repr
llvm: ASSERT: sizeof(Elf_Ehdr) <= Buf.size() && "Invalid buffer"
llvm: Out-of-memory in llvm_llvm-dwarfdump-fuzzer ClusterFuzz R
llvm: Heap-buffer-overflow in checkDylibCommand ClusterFuzz Re
llvm: Heap-buffer-overflow in readInitExpr ClusterFuzz Reproducible
llvm: Crash in llvm::DataExtractor::getUnsigned ClusterFuzz Reproc
llvm: Abrt in llvm::report_bad_alloc_error ClusterFuzz Reproducible
llvm: ASSERT: result <= INT32_MAX && result >= INT32_MIN C
llvm: Abrt in llvm::llvm_unreachable_internal ClusterFuzz Reproducil
llvm: Heap-buffer-overflow in llvm::DataExtractor::getU32 Clusterf
llvm: Heap-buffer-overflow in llvm::raw_svector_ostream::write_
llvm: Heap-buffer-overflow in llvm::DataExtractor::getCStr Cluster
llvm: Heap-buffer-overflow in llvm::DataExtractor::getU32 Clusterf
llvm: Heap-buffer-overflow in llvm::DataExtractor::getUnsigned C
llvm: Abrt in llvm::llvm_unreachable_internal ClusterFuzz Reproducil
llvm: Heap-buffer-overflow in llvm::DataExtractor::getCStr Cluster
llvm: Heap-buffer-overflow in llvm::StringMapImpl::FindKey Clust
llvm/llvm-dwarfdump-fuzzer: Heap-buffer-overflow in llvm::identi

clang-fuzzer

```
void clang_fuzzer::HandleCXX(const std::string &S,
                             const std::vector<const char *> &ExtraArgs) {
  llvm::InitializeAllTargets();
  llvm::InitializeAllTargetMCs();
  llvm::InitializeAllAsmPrinters();
  llvm::InitializeAllAsmParsers();

  llvm::opt::ArgStringList CC1Args;
  CC1Args.push_back("-cc1");
  for (auto &A : ExtraArgs)
    CC1Args.push_back(A);
  CC1Args.push_back("./test.cc");

  llvm::IntrusiveRefCntPtr<FileManager> Files(
      new FileManager(FileSystemOptions()));
  IgnoringDiagConsumer Diags;
  IntrusiveRefCntPtr<DiagnosticOptions> DiagOpts = new DiagnosticOptions();
  DiagnosticsEngine Diagnostics(
      IntrusiveRefCntPtr<clang::DiagnosticIDs>(new DiagnosticIDs()), &*DiagOpts,
      &Diags, false);
  std::unique_ptr<clang::CompilerInvocation> Invocation(
      tooling::newInvocation(&Diagnostics, CC1Args));
  std::unique_ptr<llvm::MemoryBuffer> Input =
      llvm::MemoryBuffer::getMemBuffer(S);
  Invocation->getPreprocessorOpts().addRemappedFile("./test.cc",
                                                    Input.release());
  std::unique_ptr<tooling::ToolAction> action(
      tooling::newFrontendActionFactory<clang::EmitObjAction>());
  std::shared_ptr<PCHContainerOperations> PCHContainerOps =
      std::make_shared<PCHContainerOperations>();
  action->runInvocation(std::move(Invocation), Files.get(), PCHContainerOps,
                        &Diags);
}
```

llvm: ASSERT: DelayedTypos.empty() && "Uncorrected typos!" ClusterFuzz Reproducible

llvm: ASSERT: ParmVarDeclBits.ScopeDepthOrObjCQuals == scopeDepth && "truncation!" ClusterFuzz Reproduc

llvm: ASSERT: CurPtr[-1] == '<' && CurPtr[0] == '#' && "Not a placeholder!" ClusterFuzz Reproducible

llvm: ASSERT: !isTokenSpecial() && "Should consume special tokens with Consume*Token" ClusterFuzz Reprodu

llvm: Stack-buffer-overflow in clang::Lexer::SkipLineComment ClusterFuzz Reproducible

llvm: ASSERT: Access != AS_none && "Access specifier is AS_none inside a record decl" ClusterFuzz Reproducib

llvm: ASSERT: CachedTokens[CachedLexPos-1].getLastLoc() == Tok.getAnnotationEndLoc() && "The a Cluste

llvm: ASSERT: !isNull() && "Cannot retrieve a NULL type pointer" ClusterFuzz Reproducible

llvm: ASSERT: ResultKind != Found || Decls.size() == 1 ClusterFuzz Reproducible

llvm: ASSERT: Tok.is(tok::eof) && Tok.getEofData() == AttrEnd.getEofData() ClusterFuzz Reproducible

llvm: ASSERT: Access == AS_private || Access == AS_protected ClusterFuzz Reproducible

llvm: ASSERT: RHS.U.VAL != 0 && "Divide by zero?" ClusterFuzz Reproducible

llvm: ASSERT: RHS.U.VAL != 0 && "Divide by zero?" ClusterFuzz Reproducible

llvm: ASSERT: !CodeSynthesisContexts.empty() ClusterFuzz Reproducible

llvm: ASSERT: Result.isInvalid() && "C++ binary operator overloading is missing candidates!" ClusterFuzz Reproc

llvm: Abrt in llvm::llvm_unreachable_internal ClusterFuzz Reproducible

llvm: Direct-leak in clang::Parser::ParseParameterDeclarationClause ClusterFuzz Reproducible

llvm: ASSERT: !Prev.isAmbiguous() && "Cannot have an ambiguity in previous-declaration lookup" ClusterFuzz

llvm: ASSERT: DD && "queried property of class with no definition" ClusterFuzz Reproducible

llvm: ASSERT: getContainingDC(DC) == CurContext && "The next DeclContext should be lexically c ClusterFuz

llvm: ASSERT: (OtherT->isIntegerType() && ConstantT->isIntegerType()) && "comparison with non- ClusterFuzz

llvm: ASSERT: Ancestor->getEntity() == CurContext && "ancestor context mismatch" ClusterFuzz Reproducible

llvm: ASSERT: CodeDC && !CodeDC->isFileContext() && "statement expr not in code context" ClusterFuzz Repr

llvm: ASSERT: BT->isInteger() ClusterFuzz Reproducible

llvm/clang-fuzzer: ASSERT: BitWidth && "bitwidth too small" ClusterFuzz Reproducible

libFuzzer’s default (generic) mutations

- Bit flip
- Byte swap
- Insert magic values
- Remove byte sequences
- …

clang-fuzzer (using generic mutations)

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

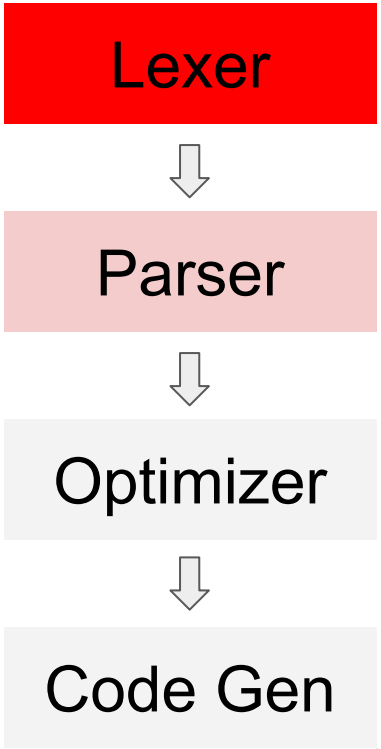
```
//\\
```

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'
```



Problem with generic mutations

- Some APIs consume highly structured data
- Generic mutations create invalid data that doesn’t parse

Structure-aware mutations

- Specialized solution for a given input type
- Parse one input, reject if doesn't parse
- Mutate the AST and/or the leaf nodes in memory

```
// Optional user-provided custom mutator.
// Mutates raw data in [Data, Data+Size) inplace.
// Returns the new size, which is not greater than MaxSize.
// Given the same Seed produces the same mutation.
size_t LLVMFuzzerCustomMutator(uint8_t *Data, size_t Size,
                               size_t MaxSize, unsigned int Seed);
// libFuzzer-provided function to be used inside LLVMFuzzerCustomMutator.
// Mutates raw data in [Data, Data+Size) inplace.
// Returns the new size, which is not greater than MaxSize.
size_t LLVMFuzzerMutate(uint8_t *Data, size_t Size, size_t MaxSize);
```

llvm-isel-fuzzer: structure-aware LLVM IR fuzzer

- Justin Bogner “Adventures in Fuzzing Instruction Selection” Euro LLVM ‘17
- libFuzzer + Custom Mutator:
  - Parse LLVM IR
  - Mutate IR in memory (`llvm/FuzzMutate/IRMutator.h`)
  - Feed the mutation to an LLVM pass

llvm-isel-fuzzer

https://bugs.chromium.org/p/**oss-fuzz**/issues/detail?id=3628

LLVM ERROR: VReg has no regclass after selection

```
source_filename = "M"

define void @f() {
BB:
  br label %BB1

BB1:              ; preds = %BB
  %G13 = getelementptr i16*, i16** undef, i1 false
  %A6 = alloca i1
  %A2 = alloca i1*
  %C1 = icmp ult i32 2147483647, 0
  store i1* %A6, i1** %A2
  store i1 %C1, i1* %A6
  store i16** %G13, i16*** undef
  ret void
}
```

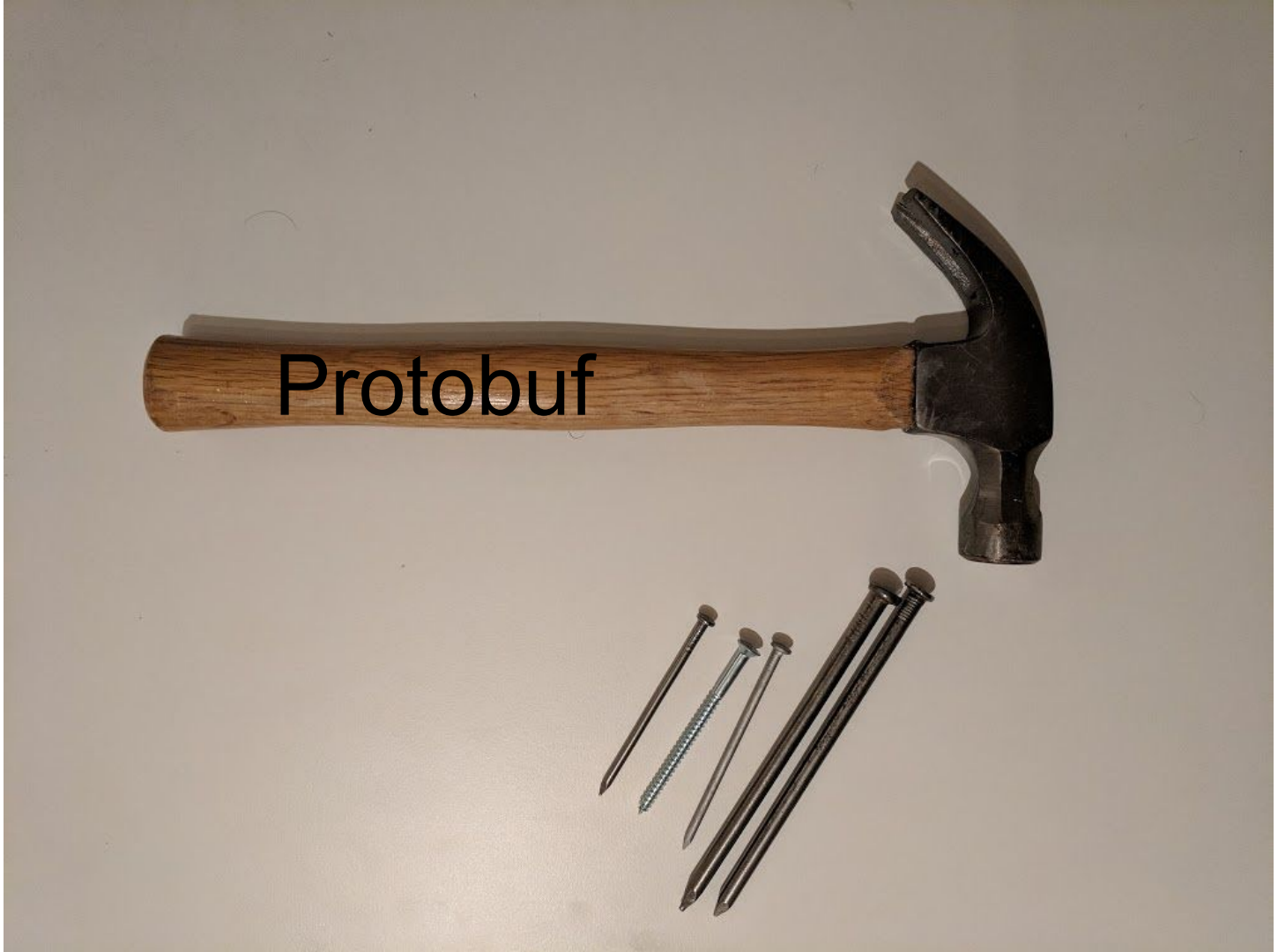
https://bugs.chromium.org/p/**oss-fuzz**/issues/detail?id=3629

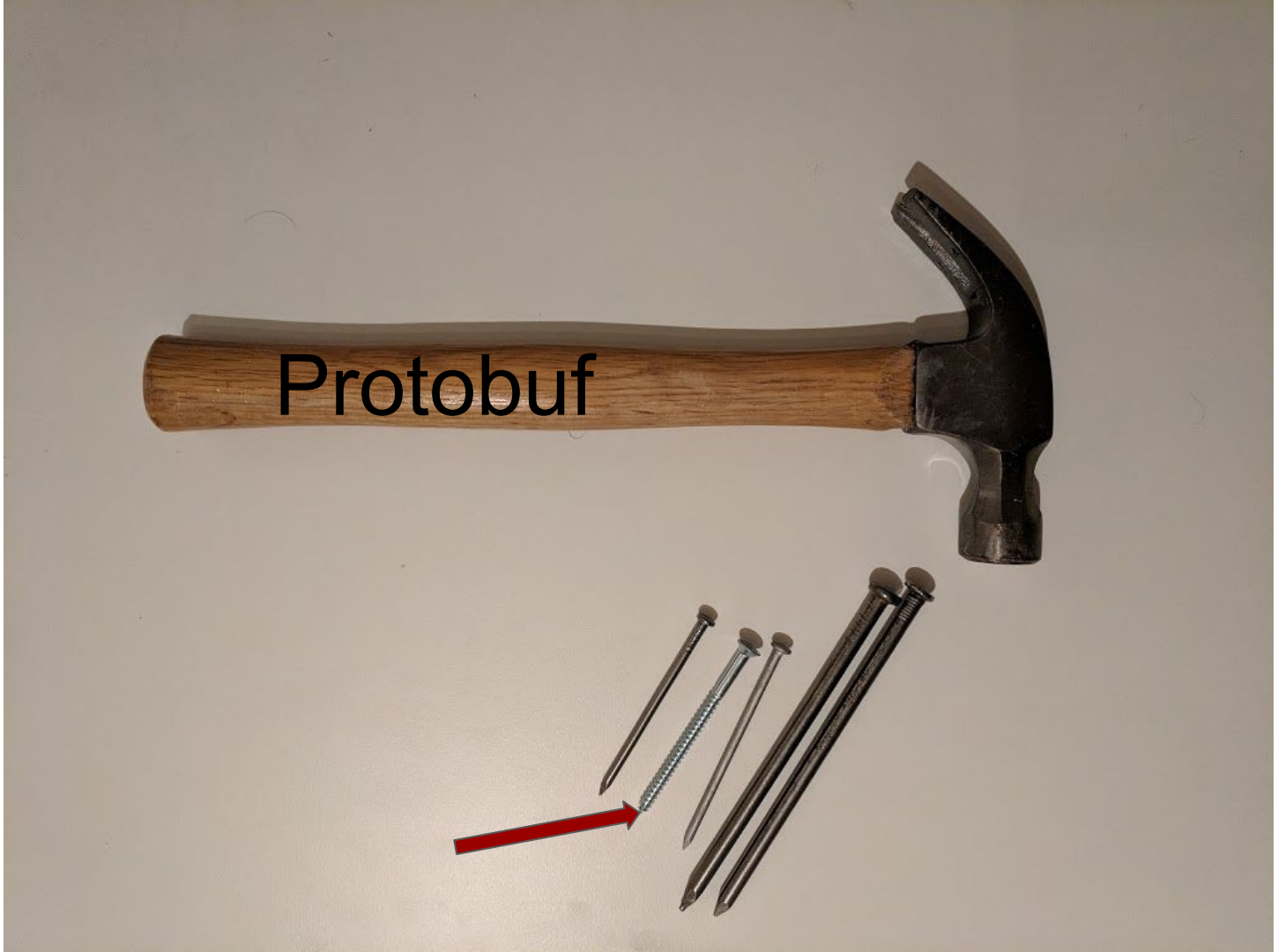
Assertion `Offset <= INT_MAX && "Offset too big to fit in int."' failed.

```
source_filename = "M"

define void @f() {
BB:
  %A11 = alloca i16
  %A7 = alloca i1, i32 -1
  %L4 = load i1, i1* %A7
  store i16 -32768, i16* %A11
  br label %BB1

BB1:              ; preds = %BB
  %C5 = icmp eq i1 %L4, %L4
  store i1 %C5, i1* undef
  store i16*** undef, i16**** undef
  ret void
}
```


Protobuf


Protobuf

https://github.com/google/protobuf

Protocol Buffers (a.k.a., protobuf) are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data

```
// Msg.proto
message Msg {
  string str = 1;
  int32  num = 2;
}
```

```
// orig.txt
str: “hello”
num: 42
```

https://github.com/google/libprotobuf-mutator

Applies a single random mutation to a protobuf message

Valid message in - valid message out

```
// Msg.proto
message Msg {
  string str = 1;
  int32  num = 2;
}
```

```
// orig.txt
str: “hello”
num: 42
```

```
// mut1.txt
str: “help”
num: 42
```

```
// mut2.txt
str: “help”
num: 911
```

https://github.com/google/libprotobuf-mutator

```
// my_api.cpp
void MyApi(const Msg &input) {
  if (input.str() == "help" && input.num() == 911)
    abort();  // bug
}
```

```
// my_api_fuzzer.cpp
DEFINE_PROTO_FUZZER(const Msg& input) {
  MyApi(input);
}
```

Fuzz clang/llvm via protobufs

- Define a protobuf type that represent a subset of C++
  - `message Function { ...`

```
// tools/clang-fuzzer/cxx_proto.proto
message BinaryOp {
 enum Op {
   PLUS = 0;
   MINUS = 1;  ...
 };
 required Op op = 1;
 required Rvalue left = 2;
 required Rvalue right = 3;
}
message Rvalue {
 oneof rvalue_oneof {
   VarRef varref = 1;
   Const cons = 2;
   BinaryOp binop = 3;
 }
}
message AssignmentStatement {
 required Lvalue lvalue = 1;
 required Rvalue rvalue = 2;
} ...
```

Fuzz clang/llvm via protobufs

- Define a protobuf type that represent a subset of C++
  - `message Function { ...`
- Implement a proto => C++ converter
  - `std::string FunctionToString(`
    `const Function &input);`

```
// tools/clang-fuzzer/proto-to-cxx/proto_to_cxx.cpp
std::ostream &operator<<(std::ostream &os,
                         const BinaryOp &x) {
 os << "(" << x.left();
 switch (x.op()) {
   case BinaryOp::PLUS: os << "+"; break;
   case BinaryOp::MINUS: os << "-"; break; ...
 }
 return os << x.right() << ")";
}
std::ostream &operator<<(std::ostream &os,
                         const Rvalue &x) {
   if (x.has_varref()) return os << x.varref();
   if (x.has_cons())   return os << x.cons();
   if (x.has_binop())  return os << x.binop();
   return os << "1";
}
std::ostream &operator<<(std::ostream &os,
            const AssignmentStatement &x) {
 return os <<x.lvalue() << "=" << x.rvalue() << ";\n";
}
```

// tools/clang-fuzzer/ExampleClangProtoFuzzer.cpp

Fuzz clang/llvm via protobufs

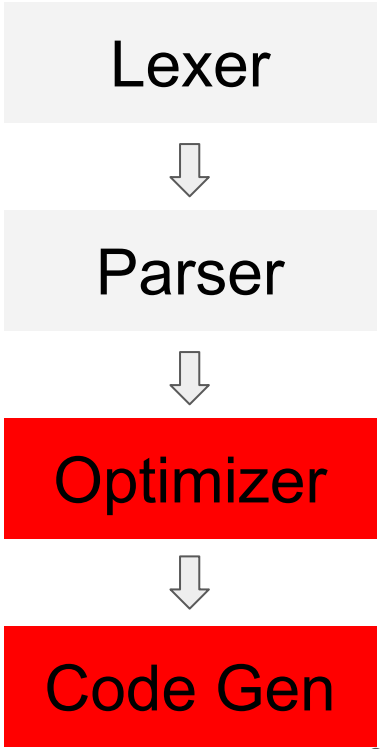
- Define a protobuf type that represent a subset of C++
  - `message Function { ...`
- Implement a proto => C++ converter
  - `std::string FunctionToString(const Function &input);`
- Implement a fuzz target
  - HandleCXX same as in clang-fuzzer
- Current state: toy prototype

```
DEFINE_BINARY_PROTO_FUZZER(
        const Function& input) {
  HandleCXX(
    FunctionToString(input));
}
```

clang-proto-fuzzer trophies

clang hangs in llvm::JumpThreadingPass::ComputeValueKnownInPredecessors

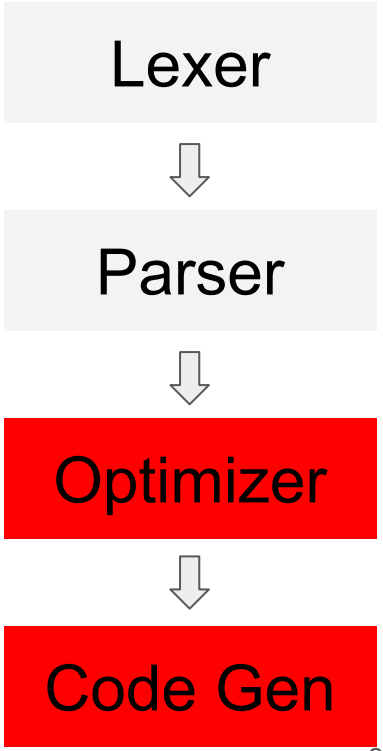
```
void foo(int *a) {
  while ((1 + 1)) {
    while ((a[96] * a[96])) {
      a[0] = (1024);
      while (a[0]) {
        while (a[0]) {
          (void)0;
          while ((a[96] * ((a[96] * a[96]) < 1))) {
            a[96] = (1 + 1);
          }
          a[0] = (a[0] + a[0]);
        }
      }
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



clang-proto-fuzzer trophies

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

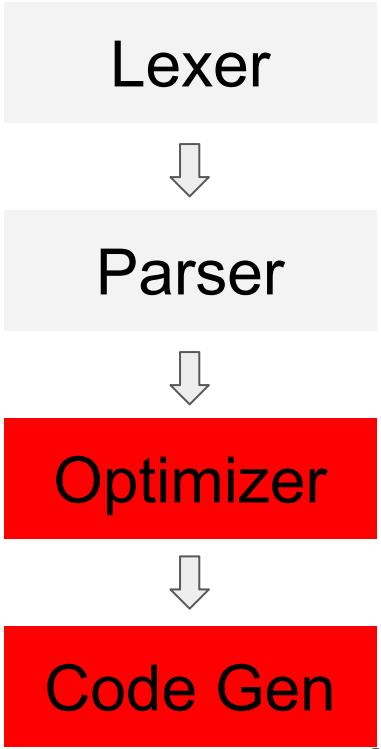
```
void foo(int *a) {
  while (1) {
    a[0] = (a[0] + (15134));
    while ((1 / a[6])) {
      (void)0;
    }
    a[0] = (a[0] + (1 + 1));
    a[8] =
        ((((((((((((a[63] % (-2147483648)) + a[0]) * a[0]) * a[0]) *
                   (-2147483648)) *
                  a[0]) +
                 ((1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



clang-proto-fuzzer trophies

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

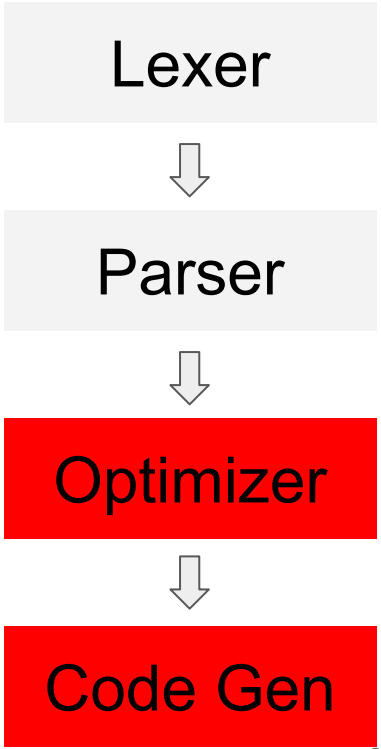
```
void foo(int *a) {
  while ((
      (((a[0] -
         ((((((((((1 * (((((1 + a[26]) * a[0]) + a[0]) * a[0]) * a[0])) * a[0]) *
                a[0]) *
               a[0]) *
              (((((((1 + (((((1 + a[26]) * a[0]) + a[0]) * a[0]) * a[0])) *
                    a[0]) +
                 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



clang-proto-fuzzer trophies

null deref in llvm::ScalarEvolution::getMulExpr

```
void foo(int *a) {
  while (1) {
    a[60] = ((1 + a[60]) + a[0]);
    while ((a[60] + a[0])) {
      a[0] = (a[0] + 1);
    }
  }
}
```



Custom IR mutator vs proto-mutator

- More work (but already done)
- Easier to reuse existing tests as corpus (?)
- No need to introduce one more IR
- Doesn’t involve Clang => faster

- Full C++ is hard to express in protobuf
- Easy to target a specific subset of C++
- Not LLVM-specific, can apply to other compilers and languages

Csmith?

- https://embed.cs.utah.edu/csmith/
- Generation-based, does not use coverage feedback
- Generates valid runnable programs
- Wish: Csmith + libFuzzer + protobuf-mutator == Csmith v2

Problems

- Bugs are being fixed too slow (if at all)
  - Not suitable for ‘starter’ projects due to code review latency
- Timeouts
- Clang/LLVM is very slow on small inputs
  - 5-20 inputs per second, w/o hitting timeouts

```
void foo(int *a0, int *a1, int *a2, int *a3, int *a4, int *a5, int *a6, int *a7,
         int *a8, int *a9, int n, int s) {
  int i0 = 0, i1 = 0, i2 = 0, i3 = 0, i4 = 0, i5 = 0, i6 = 0, i7 = 0, i8 = 0,
      i9 = 0;
  for (i5 = (-3); i5 < 3; i5 += 2) {
    for (i4 = n; i4 != n - 2; i4 += n + 1) {
      for (i8 = (-3); i8 < 3; i8 += 1) {
        for (i4 = n + 2; i4 != n - 2; i4 += n + 2) {
          a0[i3 - 8] = a0[i0 - 8] + a0[i0 + 0];
          a0[i0 + 0] = a0[i0 + 0] + a0[i0 + 8];
        }
        a0[i0 + 0] = a0[i0 - 8] + a0[i0 + 0];
      }
      a0[i3 - 8] = a0[i0 + 0] + a0[i0 + 0];
    }
  }
}
```

What's next

- clang-proto-fuzzer & llvm-isel-fuzzer run on OSS-Fuzz
  - let's observe
- How to contribute to the clang-proto-fuzzer prototype:
  - Try to express other/larger subset of C++ in a protobuf
    - Loop nests for to fuzz polly?
  - Try to make programs runnable (like csmith)
  - Try with other compilers
- How to contribute to fuzzing LLVM in general:
  - Fix crashes, timeouts, and OOMs and/or review the fixes
  - Developing a new feature? Create a dedicated fuzzer & add it to OSS-Fuzz

Q&A