Fuzzing Adoption at Facebook

Hasnain Lakhani & Dynamic Analysis team

12.13.2019

Agenda

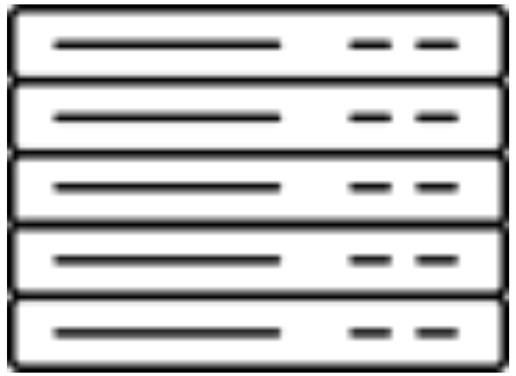
1) A walk down memory lane
2) Case study: Fuzzing for correctness
3) Driving proactive fuzzing usage

A walk down memory lane

Fuzzing at Facebook, circa 2013

Otto!

Native code at Facebook, circa 2018



Backend services

Cross-platform on mobile

Oculus, Portal, ...

Case Study: Fuzzing for Correctness

Fuzzing for correctness

- Folly contains a variety of core library components used extensively at Facebook.
- Tested the JSON parser for correctness
- Tested F14, a memory-efficient hash-table

https://github.com/facebook/folly
https://engineering.fb.com/developer-tools/f14/

Fuzzing folly::json

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* Data, size_t Size) {
 try {
   folly::StringPiece sp(reinterpret_cast<const char*>(Data), Size);
   folly::parseJson(sp);
 } catch (const std::runtime_error&) {
  // Throwing is ok.
 }
 return 0;
}
```

Fuzzing folly::json

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t* Data, size_t Size) {
 try {
   FuzzDataProducer producer(Data, Size);
   json::serialization_opts opts;
   // Use XOR so that 0 keeps the default setting
   opts.allow_non_string_keys ^= producer.produceBool();
   // 0 means default recursion limit
   opts.recursion_limit = 100 - producer.produceUint32Range(0, 100);
   folly::parseJson(producer.remainingBytes(), opts);
 } catch (std::runtime_error& e) {
   // Throwing is ok.
 }
 return 0;
}
```

Fuzzing F14, a memory-efficient hash-set

- What is an appropriate bug oracle?
  - Differential testing against `std::set`
- What operations do we need to find these bugs?
  - Standard set operations (insert, remove, etc)
- How do we make the fuzzer generate these?
  - Structure aware fuzzing!

Encoding operations

```
union SetOperation {
 1: InsertKeyMutation insertKeyMutation;
 2: InsertRangeMutation insertRangeMutation;
 3: EraseKeyMutation eraseKeyMutation;
 4: EraseRangeMutation eraseRangeMutation;
 5: ClearMutation clearMutation;
 6: ConstructRangeMutation constructRangeMutation;
 7: ConstructIListMutation constructIListMutation;
 8: LookupCheck lookupCheck;
 9: InvariantCheck invariantCheck;
}

struct SetOperations {
 1: required list<SetOperation> operations;
}
```

Encoding operations

```
struct InsertionKey {
 1: required i32 key;
 2: required bool throwOnCopy;
}

enum InsertKeyMethod {
 INSERT_COPY = 1,
 INSERT_MOVE = 2,
 EMPLACE = 3,
}

struct InsertKeyMutation {
 1: required InsertionKey key;
 2: required InsertKeyMethod method;
 3: optional i32 hint;
}
```

Writing the harness

```
DEFINE_THRIFT_FUZZER(SetOperations const& input) {
 SetOperationsInterpreter<
     BaseSet,
     F14VectorSet,
     SetOrdering::UNORDERED,
     SetExceptions::EXCEPTIONS>::go(input);
}
```

Executing an operation

```
void run(InsertRangeMutation const& op) {
  auto keys = FuzzKey::keys(op.keys, Exceptions);
  bool threw = false;
  try {
    base.insert(keys.begin(), keys.end());
  } catch (FuzzException const&) {
    threw = true;
  }
  try {
    test.insert(keys.begin(), keys.end());
  } catch (FuzzException const&) {
    threw = true;
  }
  if (threw) {
    base.clear();
    test.clear();
  }
  CHECK_EQ(base.size(), test.size());
}
```

Customizing mutations

```
mutator::Settings MutatorSettings() {
 mutator::Settings settings;
 // We aren't fuzzing the enums, we need them to be valid.
 settings.pInvalidEnum = 0;
 // Keep offset and size in reasonable ranges
 settings.setNumericRange(".operations.eraseRangeMutation.offset", 0, 1000);
 settings.setNumericRange(".operations.eraseRangeMutation.size", 0, 1000);
 // Don't throw too often
 settings.setPTrue(".operations.insertKeyMutation.key.throwOnCopy", 0.1);
 settings.setPTrue(".operations.insertRangeMutation.keys.throwOnCopy", 0.01);
 // Keep the keys in [0, 255] to promote collisions
 settings.setNumericRange(0, 255);
 return settings;
}
```

Does this actually find bugs?

- F14:
  - [folly][F14] Fix memory leak when exception is thrown
  - [folly] Improve sorted_vector_types standard compliance
  - [folly] Remove unnecessary copy in sorted_vector_types insert with hint
- JSON:
  - [folly] Remove unnecessary copies in dynamic::hash()
  - reduce key comparisons in map and set operator==
    - https://gcc.gnu.org/bugzilla/show_bug.cgi?id=91263
    - https://bugs.llvm.org/show_bug.cgi?id=42761

Hashing a folly::dynamic

```
std::size_t dynamic::hash() const {
 switch (type()) {
   case NULLT:
     return 0xBAAAAAAD;
   case OBJECT: {
     auto h = std::hash<std::pair<dynamic, dynamic>>{};
     return std::accumulate(
         items().begin(),
         items().end(),
         size_t{0x0B1EC7},
         [&](auto acc, auto item) { return acc + h(item); });
   }
   case ARRAY:
     return folly::hash::hash_range(begin(), end());
   case INT64:
     return std::hash<int64_t>()(getInt());
   case DOUBLE:
     return std::hash<double>()(getDouble());
   case BOOL:
     return std::hash<bool>()(getBool());
   case STRING:
     // keep consistent with detail::DynamicHasher
     return Hash()(getString());
 }
 assume_unreachable();
}
```

Hashing a folly::dynamic

```
  case OBJECT: {
    auto h = std::hash<std::pair<dynamic, dynamic>>{};
    return std::accumulate(
        items().begin(),
        items().end(),
        size_t{0x0B1EC7},
        [&](auto acc, auto item) {
          return acc + h(item);
        }
    );
  }
```

Implicit copies are evil

```
case OBJECT: {
  auto h = std::hash<std::pair<dynamic, dynamic>>{};
  return std::accumulate(
      items().begin(),
      items().end(),
      size_t{0x0B1EC7},
      [&](auto acc, auto const& item) {
        return acc + h(item);
      }
  );
}
```

Implicit conversions can result in implicit copies

```
case OBJECT: {
  auto h = std::hash<std::pair<dynamic const, dynamic>>{};
  return std::accumulate(
      items().begin(),
      items().end(),
      size_t{0x0B1EC7},
      [&](auto acc, auto const& item) {
        return acc + h(item);
      }
  );
}
```

Net result

- “I counted dynamic::destroy() calls in the fuzzer before this fix and there were 2120497 calls, now there are only 8633 calls, which is 245x less calls.”
- “Please apply this fuzzing magic to as much code as you can!”

Recap

- Team came to us at the right time
- “early enough” in the development cycle
- Adapted fuzzing to align with team’s goals
- Found (non-trivial) bugs

Driving proactive fuzzing usage

Driving proactive fuzzing usage

Lesson #1: Integrate into people's standard workflows

- Building harnesses is surprisingly hard
  - Mono-repos + a standard build environment make this easier
  - Simplify it: *build_harness.sh $HARNESS_NAME*
- Bug reports need to be clear and actionable
  - Simplify reproduction: *reproduce_crash.sh $BUG_ID*
  - Simplify debugging: *debug_crash.sh $BUG_ID*

Driving proactive fuzzing usage

Lesson #2: Focus on self-service

- “Fuzzing has a low/zero false positive rate”*
  - ***: If you have the same entry point as production**
- Code owners will **always** write better harnesses than you
- If it’s not documented, it might as well not exist
  - **It should be simple enough to not need documentation**

Driving proactive fuzzing usage

Lesson #3: Focus on a few power users first

- Iterate very quickly to keep momentum high
  - or, How to stop worrying and love CI/CD
- Turn them into **advocates**
- Where do you find power users?

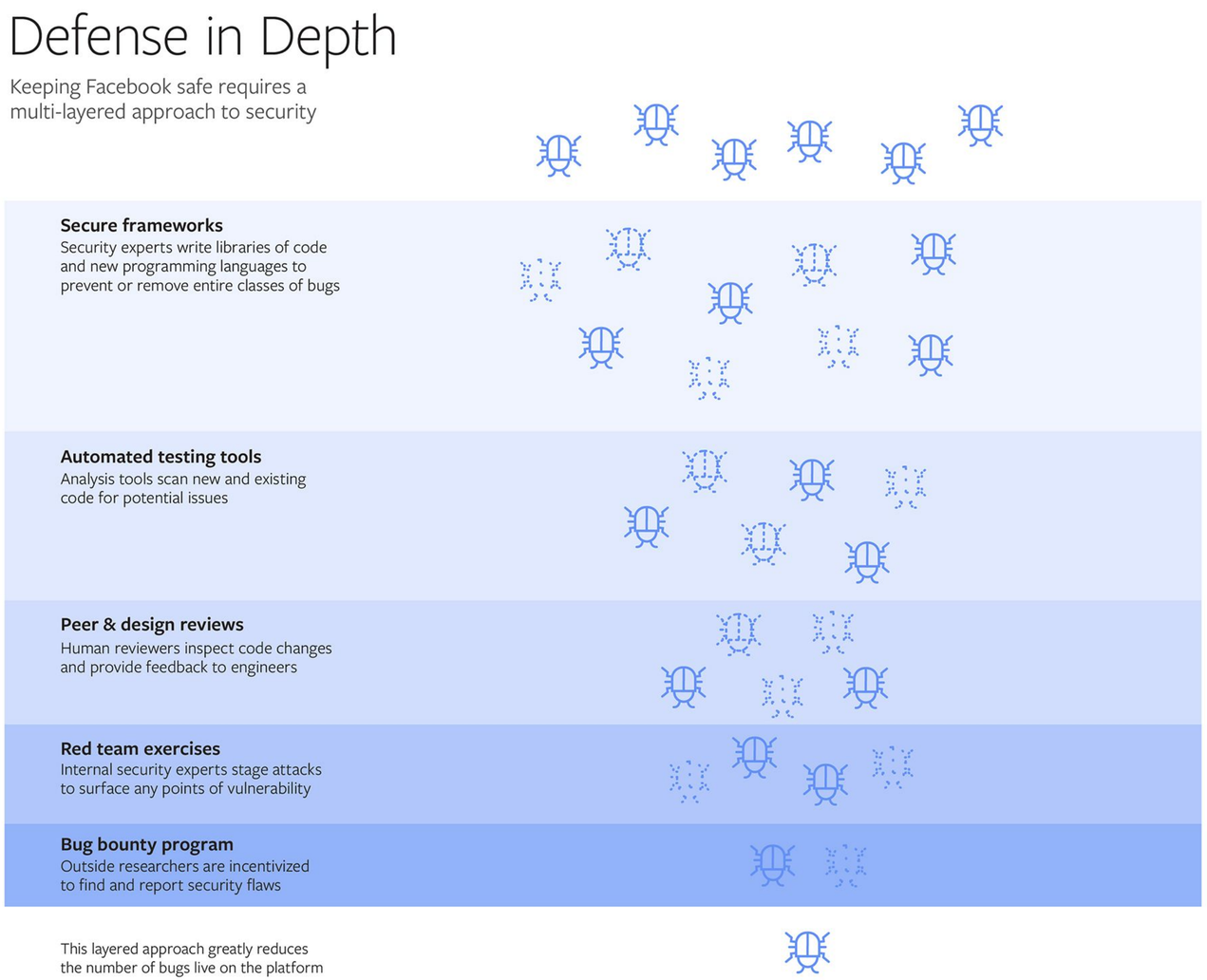
Driving proactive fuzzing usage

Lesson #4: Engage (application|product) security engineers

- Find code which needs fuzzing
- Write fuzzable code
- Go over **all** the active bugs, triage for security impact
- Find false negatives

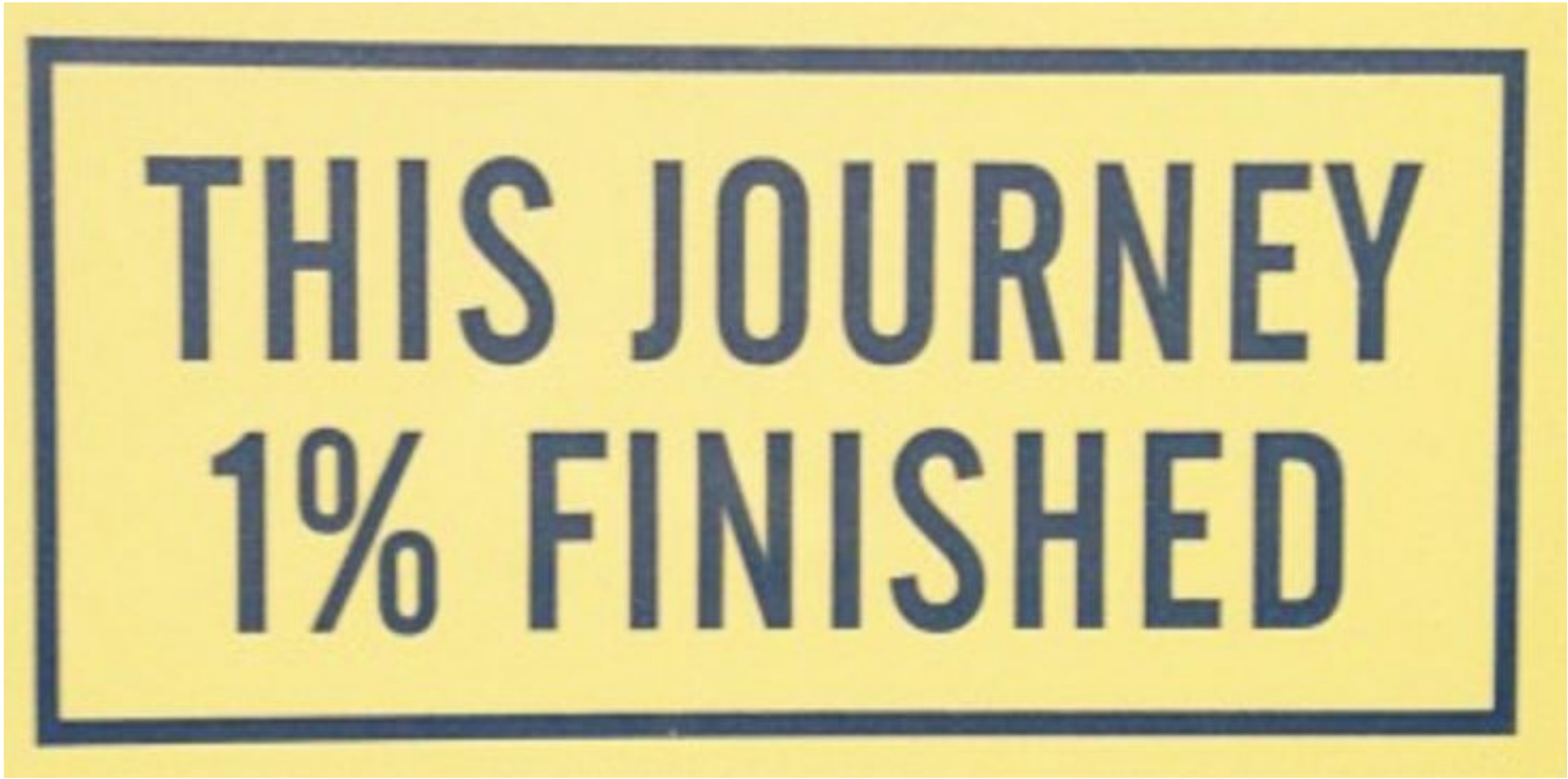
Recap

Facebook Product Security



https://about.fb.com/news/2019/01/designing-security-for-billions/

What’s next?



Thank you!