Making Your Library More Reliable with Fuzzing

Marshall Clow

Qualcomm

C++Now, May 10, 2018

About me

I have been contributing to Boost since about 2002, and am the author of the Boost.Algorithm library. (and the maintainer of a few others)

I also work on LLVM, and I am the "code owner" for libc++.

Contact info:

1. Email: mclow@boost.org
2. Slack: marshall
3. IRC: mclow

What is Fuzzing?

The basic idea is: Generate random inputs, and throw them at the code under test.

CPU cycles are very cheap these days.

What makes fuzzing better?

Detecting failures

1. Debug Builds
2. Assertions
3. Sanitizers

Guided Fuzzing

Most fuzzers let you start with a sample input and permute that.

However, just generating random inputs and trying them out seems ... inefficient.

Profile-guided Fuzzing

Basic Idea: use code-coverage information to "guide" the fuzzer.

After each input, look at the paths taken through the code, and try to figure out where to "go next" to exercise different parts of the code.

American Fuzzy Lop

From their web site: "American fuzzy lop is a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary."

How does it work?

AFL generates test cases, writes them to a file, and then calls your program to process the file. If your program crashes, it remembers that input as a bug, and continues with other tests.

After each test run, it examines the code coverage info generated during that run, and uses that information in generating the next test case.

libFuzzer

From their web site: "LibFuzzer is in-process, coverage-guided, evolutionary fuzzing engine."

How does it work?

libFuzzer generates test cases, and calls your entry point for each test case. If your code misbehaves (crashes), it notes that and starts again.

Since the code under test is linked into the program, it doesn't have to spawn a new process for each test case.

A libFuzzer skeleton

```
// A boring, do nothing LLVM fuzzer
extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *data, size_t sz)
{
//  Do something with data and sz

    return 0;  //  Always return zero
//  Non-zero return values are reserved for future use.
}
```

```
$ clang++ -g -O1 -fsanitize=fuzzer,address mytarget.cpp
$ ./a.out
```

A more interesting example

```
bool is_even(uint8_t val) { return val % 2 == 0; }

extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *data, size_t sz)
{
  vector<uint8_t> working(data, data + size);
  auto iter = partition(working.begin(), working.end(), is_even);

  assert(all_of (working.begin(), iter, is_even));
  assert(none_of(iter,   working.end(), is_even));
  assert(is_permutation(data, data + size, working.cbegin()));
  return 0;
}
```

OSS-Fuzz

OSS-Fuzz implements "Fuzzing as a service". It runs on Google's compute cloud, and fuzzes code continuously. It uses the same in-process interface as libFuzzer to fuzz code.

You provide instructions on how to build and fuzz your open source project, and a contact email. When you update the open source project, they will pull the updated code, and test that.

Errors are opened as issues in OSS-Fuzz's bugzilla, and you (as the contact point) receive an email with details (including a reduced test case). These bugs are kept private for 90 days, and then made public.

OSS-Fuzz and libc++

Algorithms in libc++ currently being fuzzed:

1. sort
2. stable_sort
3. partition
4. partition_copy
5. stable_partition
6. unique
7. unique_copy
8. nth_element
9. partial_sort
10. partial_sort_copy
11. make_heap
12. push_heap
13. pop_heap
14. regex_ECMAScript
15. regex_POSIX
16. regex_extended
17. regex_awk
18. regex_grep
19. regex_egrep
20. search

OSS-Fuzz and Boost

I am working on a general, boost-wide process to interface with OSS-Fuzz.

Individual library authors will have to write the test cases themselves, and (probably) provide build instructions for non header-only libraries, but the rest of the setup should happen automatically.

I hope to announce this on the developer list later this summer.

Questions?

Thank you

1. American Fuzzy Lop: http://lcamtuf.coredump.cx/afl/
2. libFuzzer: https://llvm.org/docs/LibFuzzer.html
3. OSS-Fuzz: https://github.com/google/oss-fuzz