Design and Implement Binary Fuzzing based on LibFuzzer

Chun-Ying Huang, Wei-Chieh Chao, Si-Chen Lin, Yi-Hsien Chen
Department of Computer Science, College of Computer Science, National Chiao Tung University

***Abstract*—Libfuzzer is a coverage-guided fuzzing engine. But not like AFL, which can do the fuzzing without source code, Libfuzzer needs source code of the target to compile and do the fuzzing. This paper presents a way to use Libfuzzer to fuzz the binary without source code. Our implementation link the IO of binary with Libfuzzer, and use qemu to collect runtime information.**

I. Introduction

Sample citations [1], [2], [3], [4].

II. Related Work

III. Methodology

A. Link IO together

Normally, when using Libfuzzer, we put the source code need to be test inside `LLVMFuzzerTestOneInput` as describe in the introduction section. But when we want to run a binary, we don't have source code. Hence, what we do is actually putting `execv("our-binary", ..., ...)` inside the `LLVMFuzzerTestOneInput` to invoke our binary. And before execute `execv` we need to use dup2 to link

1. fuzzer test input → binary stdin.
2. binary stdout → `/dev/null`.
3. binary stderr → `/dev/null`

The simplified code is something like below

```
extern "C" int LLVMFuzzerTestOneInput(const
    uint8_t *Data, size_t Size) {

    int P_IN[2]; pipe(P_IN);
    if(Size) write(P_IN[1], Data, Size);
    ...
    int pid = fork();
    if(pid == 0) {
        dup2(P_IN[0], STDIN_FILENO);
        dup2(dev_null, STDOUT_FILENO);
        dup2(dev_null, STDERR_FILENO);
        ...
        execv(..., ..., ...);
    }
    ...
}
```

B. Collect Runtime Information

After we link IO together, the binary should get the input of the libfuzzer test input now. Though it is runnable, Libufuzzer don't have any runtime information like code coverage and will stop running after a few rounds. First, we need to figure out what Libfuzzer collect while running. Libfuzzer collect runtime information through **Clang SanitizerCoverage**, which provides simple code coverage instrumentation and has hook function for customization. Now we use qemu to collect right information for those hook function implemented by LibFuzzer.

1) trace pc guard:

IV. Evaluation

V. Conclusion

References

[1] K. Serebryany, "libFuzzer a library for coverage-guided fuzz testing," LLVM project, 2015, https://llvm.org/docs/LibFuzzer.html.
[2] R. Swiecki, "honggfuzz," online, 2010, http://honggfuzz.com/.
[3] T. Petsios, J. Zhao, A. D. Keromytis, and S. Jana, "Slowfuzz: Automated domain-independent detection of algorithmic complexity vulnerabilities," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2155–2168.
[4] K. Serebryany, "OSS-Fuzz - Google's continuous fuzzing service for open source software," 2017, USENIX Security. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/serebryany