

Introduction to Fuzzing

Jack McPherson
Security Engineer & Developer
Sigma Prime

Australia Conference 2024 - Day Two
13 February 2024

Contents

1. Introduction
 - a. Intended audience
 - b. Motivation
 - c. Reasoning about software
 - d. Software testing
2. Fuzzing theory
3. Fuzzing practice

Introduction

Intended audience

- Sigma Prime, of course!
 - Experienced security researchers
 - Novice security researchers
 - Lighthouse developers
- Largely language agnostic but with a (strong) Rust bias

Motivation

- Why fuzzing?
 - Easy
 - Concrete
 - Effective
 - Applicable

Reasoning about software

- Static analysis
 - Linting
 - Symbolic execution*
 - Model checking (SMT, et. al.)
- Dynamic analysis
 - Testing
 - Unit testing
 - Fuzzing

Software testing

- Process
 - Executing the SUT with certain inputs
 - Checking to see if *failures* occur
- What's a *failure*?
 - Assertions being false at time of evaluation
 - Memory safety violations
 - Crashes
 - Segmentation faults
 - Panics
 - Exceptions

Fuzzing Theory

Fuzzing – in theory

- Big idea
 - Generate inputs (pseudo)randomly
- How to generate inputs?
 - Dumb fuzzing
 - Completely randomly (well, as randomly as computers can achieve, of course)
 - Smart fuzzing
 - Structurally
 - Mutation-based
 - We provide the structure of the input and the fuzzer mutates this accordingly
 - Coverage-guided
 - Fuzzer is aware of program's internal structure (i.e., both code and data)

Fuzzing – in theory (cont.)

- Fuzzing visibility
 - Black-box
 - No knowledge of program internals
 - No code
 - No recompilation
 - No relinking
 - Grey-box
 - Some knowledge of program internals
 - Code
 - No recompilation
 - No relinking
 - White-box
 - Full knowledge of program internals

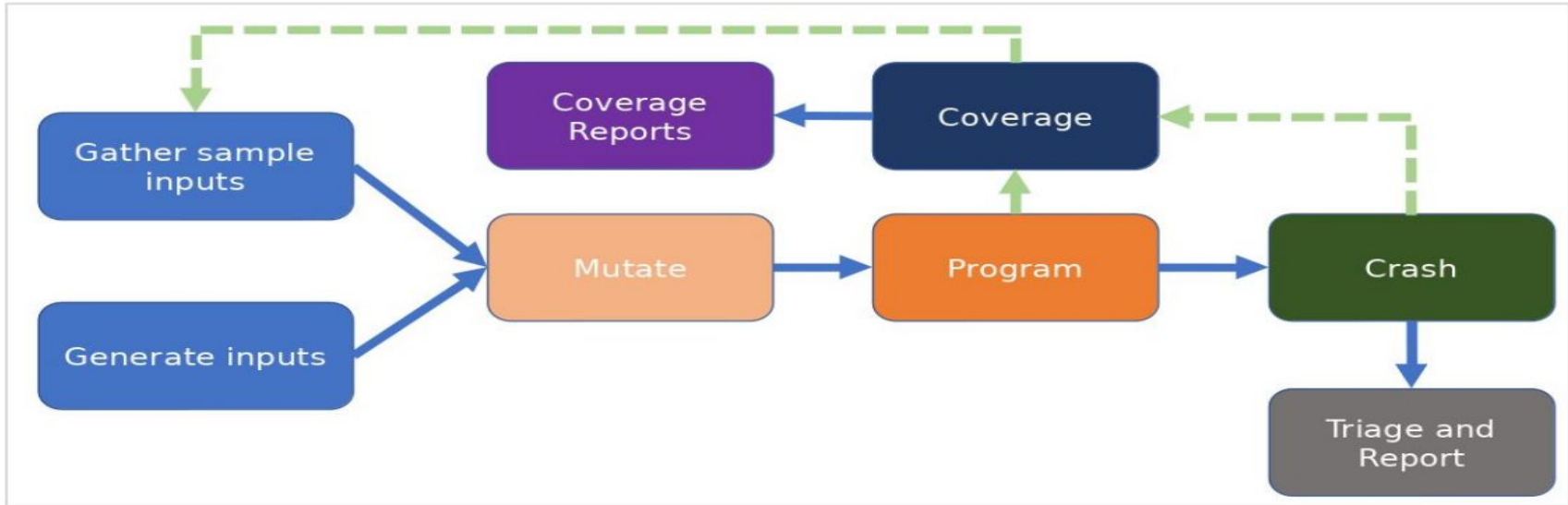
Fuzzing – in theory (cont.)

- Additional flavours
 - Differential
 - Failures become deviations from a given implementation
 - Obvious applications to Ethereum client development
 - Both EL and CL!
 - Snapshot
 - Capture both program and machine state and fuzz from there
 - Restore state after each fuzzing case

Fuzzing – in theory (cont.)

- Components
 - Fuzzer
 - Produces inputs
 - Detects failures
 - Triage failures
 - Minimises fuzzing cases
 - Manages corpus
 - Harness
 - Gets inputs into program
 - Implements custom, application-specific logic
 - Ignores any irrelevant failures
 - SUT

Fuzzing – in theory (cont.)



Fuzzing – in theory (cont.)

- Corpus
 - Collection of fuzzing inputs produced by the fuzzer
- Corpus minimisation
 - Corpi get very large very quickly
 - Very attractive feature of a fuzzer *corpus minimisation*
- Case minimisation
 - A given test case produced by the fuzzer that actually induces failure may be huge
 - Likely that the bug is reachable via drastically smaller input
 - Very attractive feature of a fuzzer is *case minimisation*

Fuzzing Practice

Fuzzing – in practice

- Nothing new under the Sun!
 - Everything is just three fuzzers under the hood
 - `libfuzzer`
 - AFL
 - `libafl`
 - Honggfuzz
- Rust
 - `cargo-fuzz` (Cargo frontend for `libfuzzer`)
 - `afl.rs` (Cargo frontend for AFL)
 - `honggfuzz-rs` (Cargo frontend for Honggfuzz)
 - `cargo-libafl` (Cargo frontend for `libafl`)

Fuzzing – in practice (cont.)

- Solidity
 - Forge!
 - Echidna
- Generalised EVM
 - None practical
 - ILF
 - ???

Fuzzing – in practice (cont.)

Learning to Fuzz from Symbolic Execution with Application to Smart Contracts

Jingxuan He
ETH Zurich, Switzerland
jingxuan.he@inf.ethz.ch

Mislav Balunović
ETH Zurich, Switzerland
mislav.balunovic@inf.ethz.ch

Nodar Ambroladze
ETH Zurich, Switzerland
anodar@ethz.ch

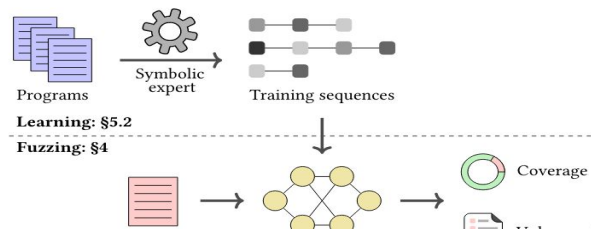
Petar Tsankov
ETH Zurich, Switzerland
petar.tsankov@inf.ethz.ch

Martin Vechev
ETH Zurich, Switzerland
martin.vechev@inf.ethz.ch

ABSTRACT

Fuzzing and symbolic execution are two complementary techniques for discovering software vulnerabilities. Fuzzing is fast and scalable, but can be ineffective when it fails to randomly select the right inputs. Symbolic execution is thorough but slow and often does not scale to deep program paths with complex path conditions.

In this work, we propose to learn an effective and fast fuzzer from symbolic execution, by phrasing the learning task in the framework of imitation learning. During learning, a symbolic execution expert generates a large number of quality inputs improving coverage on



Fuzzing – in practice (cont.)

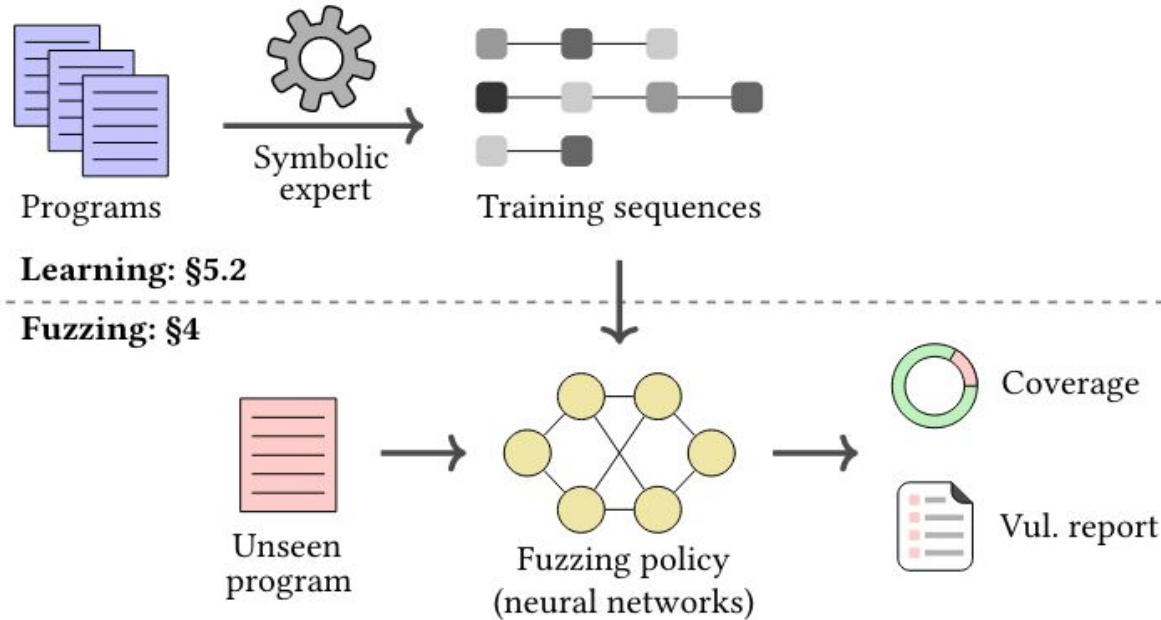


Figure 1: Learning to fuzz from symbolic execution.

Fuzzing – in practice (cont.)

- Go
 - Built-in!
- C/C++
 - The ones we've already listed
 - Much more diverse tooling
 - Can target intermediate representations too
 - LLVM, MLIR, etc.
 - Good compiler support
 - GCC
 - Clang

Fuzzing – in practice (cont.)

- Obligations as testers
 - Set up fuzzing environment (installation, configuration, etc.)
 - **Identify fuzzing target**
 - Write test harness
 - (Optionally) write effective dictionaries to massage fuzzer into reasonable directions
 - Inherent tradeoffs though!
 - Run fuzzer against target
 - Monitor coverage, failures, case minimisation, corpus size
 - Do this as much as you can!

Fuzzing – in practice (cont.)

- Identifying good fuzzing targets is where the magic happens
 - This is what differentiates expert fuzzing users from novices
 - Can be very difficult in practice
 - Some good starting points
 - `fn(Vec<u8>, ...) -> Whatever`
 - `fn(&[u8], ...) -> Whatever`
 - `fn(JsonBlob, ...) -> Whatever`
 - `fn(SszBlob, ...) -> Whatever`
 - `fn parse_msg_from_network(data_from_untrusted_actor: Vec<u8>) -> Result<Msg, SomeError>`
 - `fn parse_str_from_network(string_from_untrusted_actor: String) -> Result<Msg, SomeError>`
 - Any automated means to do this is very welcome!
 - I have thought of a few but would like Better Ones™

Fuzzing – in practice (cont.)

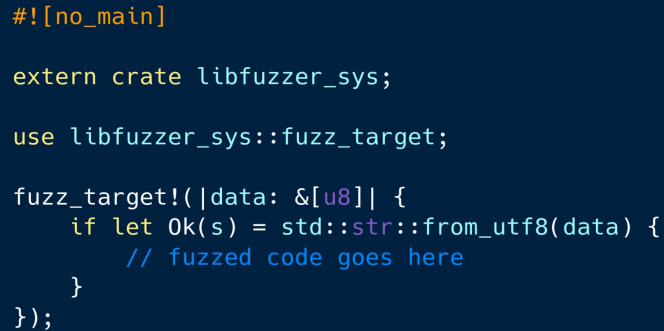


```
#![no_main]
```

```
use libfuzzer_sys::fuzz_target;
```

```
fuzz_target!(|data: &[u8]| {  
    // fuzzed code goes here  
});
```

Fuzzing – in practice (cont.)



```
#![no_main]

extern crate libfuzzer_sys;

use libfuzzer_sys::fuzz_target;

fuzz_target!(|data: &[u8]| {
    if let Ok(s) = std::str::from_utf8(data) {
        // fuzzed code goes here
    }
});
```


Fuzzing – in practice (cont.)

```
#19913 NEW cov: 1834 ft: 4548 corp: 397/2090b lim: 11 exec/s: 19799 rss: 128Mb L: 10/11 MS: 2 PersAutoDict-CopyPart- DE: "\01\000"-
#19868 NEW cov: 1834 ft: 4550 corp: 398/2098b lim: 11 exec/s: 19868 rss: 128Mb L: 8/11 MS: 4 ChangeByte-ShuffleBytes-ChangeBit-ChangeBit-
#19874 REDUCE cov: 1834 ft: 4550 corp: 398/2097b lim: 11 exec/s: 19874 rss: 128Mb L: 6/11 MS: 1 EraseBytes-
#19885 NEW cov: 1838 ft: 4554 corp: 399/2100b lim: 11 exec/s: 19885 rss: 128Mb L: 3/11 MS: 1 EraseBytes-
#19910 NEW cov: 1838 ft: 4555 corp: 400/2110b lim: 11 exec/s: 19910 rss: 128Mb L: 10/11 MS: 5 ChangeByte-ChangeByte-ChangeByte-CopyPart-Shuffle
Bytes-
#19912 NEW cov: 1838 ft: 4567 corp: 401/2120b lim: 11 exec/s: 19912 rss: 128Mb L: 10/11 MS: 2 ChangeBit-InsertRepeatedBytes-
#19983 NEW cov: 1838 ft: 4568 corp: 402/2128b lim: 11 exec/s: 19983 rss: 128Mb L: 8/11 MS: 1 CopyPart-
#19984 NEW cov: 1838 ft: 4574 corp: 403/2136b lim: 11 exec/s: 19984 rss: 128Mb L: 8/11 MS: 1 CMP- DE: "(((("-
#20015 NEW cov: 1842 ft: 4581 corp: 404/2144b lim: 11 exec/s: 20015 rss: 129Mb L: 8/11 MS: 1 CMP- DE: "\012\032"-
#20136 NEW cov: 1842 ft: 4583 corp: 405/2152b lim: 11 exec/s: 20136 rss: 129Mb L: 8/11 MS: 1 CrossOver-
#20177 NEW cov: 1842 ft: 4598 corp: 406/2162b lim: 11 exec/s: 20177 rss: 129Mb L: 10/11 MS: 1 InsertByte-
#20225 REDUCE cov: 1842 ft: 4598 corp: 406/2161b lim: 11 exec/s: 20225 rss: 129Mb L: 3/11 MS: 3 EraseBytes-CopyPart-EraseBytes-
#20248 REDUCE cov: 1842 ft: 4598 corp: 406/2160b lim: 11 exec/s: 20248 rss: 129Mb L: 3/11 MS: 3 CopyPart-ShuffleBytes-EraseBytes-
#20406 NEW cov: 1842 ft: 4600 corp: 407/2170b lim: 11 exec/s: 20406 rss: 130Mb L: 10/11 MS: 3 ChangeByte-CrossOver-ShuffleBytes-
#20645 NEW cov: 1842 ft: 4602 corp: 408/2180b lim: 11 exec/s: 20645 rss: 131Mb L: 10/11 MS: 4 InsertByte-CrossOver-CopyPart-InsertByte-
#20646 NEW cov: 1844 ft: 4610 corp: 409/2191b lim: 11 exec/s: 20646 rss: 131Mb L: 11/11 MS: 1 CopyPart-
#20789 NEW cov: 1844 ft: 4616 corp: 410/2202b lim: 11 exec/s: 20789 rss: 131Mb L: 11/11 MS: 3 EraseBytes-ChangeByte-InsertRepeatedBytes-
#20795 NEW cov: 1844 ft: 4621 corp: 411/2209b lim: 11 exec/s: 20795 rss: 132Mb L: 7/11 MS: 1 CopyPart-
#20798 NEW cov: 1853 ft: 4630 corp: 412/2220b lim: 11 exec/s: 20798 rss: 132Mb L: 11/11 MS: 3 ChangeByte-ChangeByte-ChangeBit-
#20826 NEW cov: 1854 ft: 4631 corp: 413/2231b lim: 11 exec/s: 20826 rss: 132Mb L: 11/11 MS: 3 ShuffleBytes-CopyPart-ChangeByte-
#20883 REDUCE cov: 1854 ft: 4631 corp: 413/2230b lim: 11 exec/s: 20883 rss: 132Mb L: 4/11 MS: 2 CopyPart-EraseBytes-
#20964 NEW cov: 1854 ft: 4644 corp: 414/2240b lim: 11 exec/s: 20964 rss: 133Mb L: 10/11 MS: 1 ChangeBinInt-
#20975 NEW cov: 1855 ft: 4645 corp: 415/2248b lim: 11 exec/s: 20975 rss: 133Mb L: 8/11 MS: 1 ChangeByte-
#20992 REDUCE cov: 1855 ft: 4645 corp: 415/2247b lim: 11 exec/s: 20992 rss: 133Mb L: 9/11 MS: 2 CopyPart-EraseBytes-
#21003 NEW cov: 1856 ft: 4647 corp: 416/2258b lim: 11 exec/s: 21003 rss: 133Mb L: 11/11 MS: 1 PersAutoDict- DE: "\001\000\000$" "-
#21119 NEW cov: 1856 ft: 4726 corp: 417/2268b lim: 11 exec/s: 21119 rss: 133Mb L: 10/11 MS: 1 CopyPart-
NEW_FUNC[1/1]: 0x55e7bf173b60 in core::ptr::drop_in_place<LT$regex_syntax..ast..ClassSetBinaryOp$GT$::hd2e73ff1abe462d4 /rustc/30dfb9e046ae
b878db04332c74de76e52fb7db10/library/core/src/ptr/mod.rs:507
#21171 NEW cov: 1862 ft: 4733 corp: 418/2277b lim: 11 exec/s: 21171 rss: 134Mb L: 9/11 MS: 2 PersAutoDict-InsertByte- DE: ")))\032"-
#21185 NEW cov: 1862 ft: 4737 corp: 419/2287b lim: 11 exec/s: 21185 rss: 134Mb L: 10/11 MS: 4 CrossOver-CMP-PersAutoDict-CrossOver- DE: ",\000"
"- ?= "-
#21312 NEW cov: 1862 ft: 4745 corp: 420/2293b lim: 11 exec/s: 21312 rss: 134Mb L: 6/11 MS: 2 InsertByte-PersAutoDict- DE: "0()")-
#21506 NEW cov: 1862 ft: 4747 corp: 421/2302b lim: 11 exec/s: 21506 rss: 135Mb L: 9/11 MS: 4 ChangeBinInt-EraseBytes-CMP-InsertRepeatedBytes- D
E: " ?= "-
#21622 REDUCE cov: 1871 ft: 4756 corp: 422/2311b lim: 11 exec/s: 21622 rss: 135Mb L: 9/11 MS: 1 InsertRepeatedBytes-
#21793 NEW cov: 1871 ft: 4758 corp: 423/2322b lim: 11 exec/s: 21793 rss: 136Mb L: 11/11 MS: 1 CMP- DE: "?!"-
#21899 REDUCE cov: 1871 ft: 4758 corp: 423/2321b lim: 11 exec/s: 21899 rss: 136Mb L: 8/11 MS: 1 EraseBytes-
#21946 REDUCE cov: 1879 ft: 4767 corp: 424/2328b lim: 11 exec/s: 21946 rss: 136Mb L: 7/11 MS: 2 PersAutoDict-CopyPart- DE: "\001\000\0000"-
#21972 REDUCE cov: 1879 ft: 4767 corp: 424/2326b lim: 11 exec/s: 21972 rss: 137Mb L: 4/11 MS: 1 EraseBytes-
^C=16392= libFuzzer: run interrupted; exiting
jmcph4@villain:~/dev/work/sigp/fuzzing-course/regex/fuzz$
```

Fuzzing – in practice (cont.)

- How long do I run the fuzzer?
 - It depends!
 - Arguably akin to the halting problem
 - Good heuristic: until the fuzzer stalls
- The importance of coverage
 - Coverage is our measure of how the fuzzer is progressing
 - If the rate of coverage growth gets sufficiently low (chosen w.l.o.g.)
 - Fuzzer has probably stalled
 - Good fuzzers send us heartbeats

Fuzzing – in practice (cont.)



Fuzzing – in practice (cont.)

Live demonstration! (potentially)

Resources

- [The Fuzzing Book](#)
- [Fuzzing section of ToB's *Testing Handbook*](#)
- [*An empirical study of the reliability of UNIX utilities*](#)
- [cpuu/awesome-fuzzing](#)
- [libfuzzer LLVM documentation](#)
- [\(Some of\) my raw notes from the fuzzing course](#)
- [FuzzingLabs Rust fuzzing course](#)
- [*Learning to Fuzz from Symbolic Execution with Application to Smart Contracts*](#)