FUZZING

CS 361S

Spring 2020

Seth James Nielson

WHAT IS FUZZING?

- **Definition 1 (Fuzzing).** Fuzzing is the execution of the Program Under Test [PUT] using input(s) sampled from an input space (the "fuzz input space") that protrudes the ***expected*** input space of the PUT.

IN ENGLISH PLEASE

- **Testing a program with unusual inputs**

THREE MORE DEFINITIONS

Definition 2 (Fuzz Testing). Fuzz testing is the use of fuzzing to test if a PUT violates a security policy.

Definition 3 (Fuzzer). A fuzzer is a program that performs fuzz testing on a PUT

Definition 4 (Fuzz Campaign). A fuzz campaign is a specific execution of a fuzzer on a PUT with a specific security policy.

WHY CAMPAIGNS?

Goal: Find a bug that violates the security policy

Example: Bug causes a crash

Any security policy that is ***EM-Enforcable***

EM-ENFORCABILITY

- Fred Schneider: *Enforceable Security Policies*
- The practicality of any security policy depends on whether that policy is
  - enforceable
  - and at what cost
- EM: Enforcement by **E**execution **M**onitoring

EXECUTION MONITORING

… the class of enforcement mechanisms that work by **monitoring execution steps** of some system, herein called the target, and **terminating** the target's execution if it is about to violate the security policy being enforced. We call this class EM, for Execution Monitoring. EM includes security kernels, **reference monitors**, firewalls, and most other operating system and hardware-based enforcement mechanisms that have appeared in the literature

Not exactly what is described in the Fuzzing paper.

BUG ORACLE

Definition 5 (Bug Oracle). A bug oracle is a program, perhaps as part of a fuzzer, that determines whether a given execution of the PUT violates a specific security policy.

FUZZ CONFIGURATION

Definition 6 (Fuzz Configuration). A fuzz configuration of a fuzz algorithm comprises the parameter value(s) that control(s) the fuzz algorithm.

TAXONOMY OF FUZZERS

- **Black Box:** No access to internals
- **White Box:** Based on relatively complete internal information
- **Grey Box:** Based on limited or partial information

ALGORITHM 1: Fuzz Testing

```
Input: ℂ, t_limit
Output: 𝔹 // a finite set of bugs
𝔹 ← ∅
ℂ ← PREPROCESS(ℂ)
while t_elapsed < t_limit ∧ CONTINUE(ℂ) do
    conf ← SCHEDULE(ℂ, t_elapsed, t_limit)
    tcs ← INPUTGEN(conf)
    // O_bug is embedded in a fuzzer
    𝔹′, execinfos ← INPUTEVAL(conf, tcs, O_bug)
    ℂ ← CONFUPDATE(ℂ, conf, execinfos)
    𝔹 ← 𝔹 ∪ 𝔹′
return 𝔹
```

GENERIC FUZZ ALGORITHM

FUZZ TESTING ROUTINES

Preprocess: modified configuration, such as instrumentation

Schedule: limit time of execution and updates current iteration

InputGen: produce input based on configuration

InputEval: generates output and tests with a bug oracle

ConfUpdate: potentially update configurations based on output

Continue: determines whether testing should continue

PREPROCESS: INSTRUMENTATION

Static and Dynamic options

Execution Feedback

In-Memory Fuzzing

Thread Scheduling

PREPROCESS: SEED SELECTION

Seed Selection Problem

Minimal seeds for maximal coverage

PREPROCESS: MISC

Seed Trimming: Reduce the size of a seed

Driver Preparation: Prepare for test

SCHEDULING

- More than just a for loop
- Goal: pick the next configuration with best outcome
  - Example: maximum unique bugs
  - Example: best code coverage
- Fuzz Configuration Scheduling (FCS): ***exploration*** v ***exploitation***
- Variations based on time elapsed v time budget
- Gray box has more information available than black box

INPUT GENERATION

Generation-based (generally model-based)

- Pre-defined model
- Inferred model
- Encoder model

Mutation-based (generally model-less)

- Bit flipping
- Arithmetic mutation
- Block-based mutation
- Dictionary-based mutation

WHITEBOX FUZZERS

- Either model or model-less
- Dynamic symbol execution
- Guided fuzzing
- PUT mutation

INPUT EVALUATION: BUG ORACLES

- Memory and Type Safety
- Undefined behaviors
- Input validation
- Semantic difference

INPUT EVALUATION: TRIAGE

Deduplication

Prioritization and Exploitability

Test case minimization

INPUT EVALUATION: EXECUTION OPTIMIZATIONS

- Improve repeat execution performance
- Shorten start-up times
- Potential relationship with in-memory fuzzing

CONFIGURATION UPDATE

- Evolutionary Seed Pool Update
- Maintaining a miniset