Vulnerability Search Problem and Methods

Bhargava Shastry

ibags

whoami

- Security Engineer at Ethereum Foundation
- Independent security researcher
- Enjoy finding software vulnerabilities

Introduction

Vulnerabilities are expensive

Damage Caused by Vulnerabilities

- Wannacry worm
  - Cost $8 billion [Reuters17]
  - Crippled healthcare system
- Router Vulnerabilities
  - Wormable

Vulnerability Search Problem

Systematic examination of a system to identify vulnerabilities.

What is a Vulnerability?

“Flaw or weakness in a system’s design, implementation, or operation and management that could be exploited to violate the system’s security policy.”

- IETF Security Glossary

Vulnerability Dimensions

System

- Software
- Hardware
- Network

Security Policy

- Confidentiality
- Integrity
- Availability

Vulnerability Origin

- Design
- Implementation
- Management

Challenges

Vulnerabilities are rare

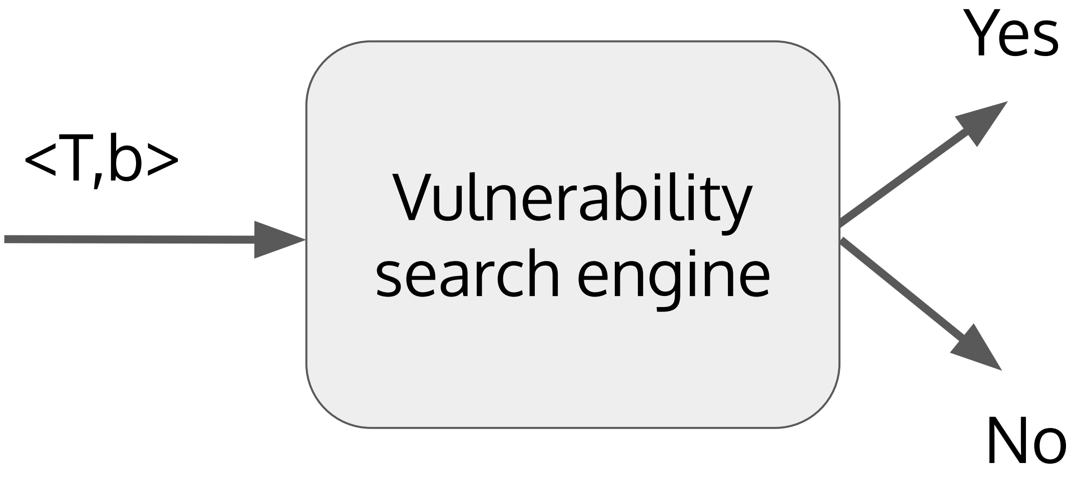
How Rare are Vulnerabilities?

- 6 bugs per 10K LoC [Coverity14]
- Chromium bug tracker
  - 1 in 5 bugs a vulnerability

1.2 vulnerabilities per 10000 LoC

Finding a needle in the haystack

Undecidability



Does buffer overflow?

Credit: https://www.coopertoons.com/education/haltingproblem/haltingproblem.html

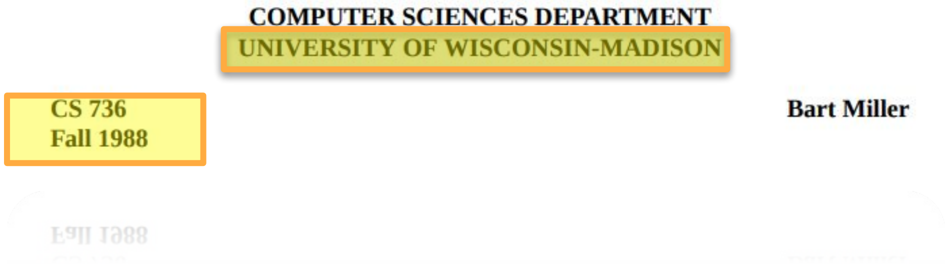
Methods

Method Overview

- Partial solution
  - False negatives permitted
  - False positives rare
- Try to be fast
  - > 100 executions per second

Origin of Fuzz Testing

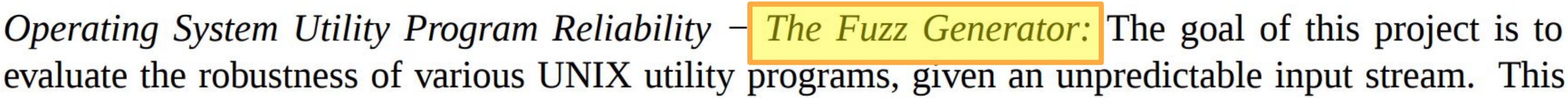
TL;DR: Throw corner-case input at a program until it breaks



COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN-MADISON

CS 736
Fall 1988

Bart Miller



Operating System Utility Program Reliability − *The Fuzz Generator:* The goal of this project is to evaluate the robustness of various UNIX utility programs, given an unpredictable input stream. This

How are Test Inputs Generated?

- Late 80’s: Randomly
- Early 00’s: Based on a specification
- Late 00’s: Based on program behavior

Random Test Generation

Overview

- Random mutation of initial input (seed)
- Mutation
    - Tweak bits
    - Add/remove bytes
    - Apply transformation `f(i) -> j`

Howto?

```
$ while true; do echo -n "\xd4\xc3\xb2\xa1" | radamsa |
 tcpdump -vr -; done
tcpdump: unknown file format
tcpdump: unknown file format
tcpdump: truncated dump file; tried to read 4 file header bytes,
only got 0
tcpdump: unknown file format
```

Observations

- Effectiveness depends on
  - Quality of initial input (`echo -n "\xd4\xc3\xb2\xa1"`)
  - Relevance of mutations to program under test (target)
    - Random mutations are of marginal utility to a target like tcpdump
- Speed
  - Very fast (typically, hundreds of executions per second)

Example: tcpdump

- 1000 tests in under 4 seconds
- Poor quality of tests

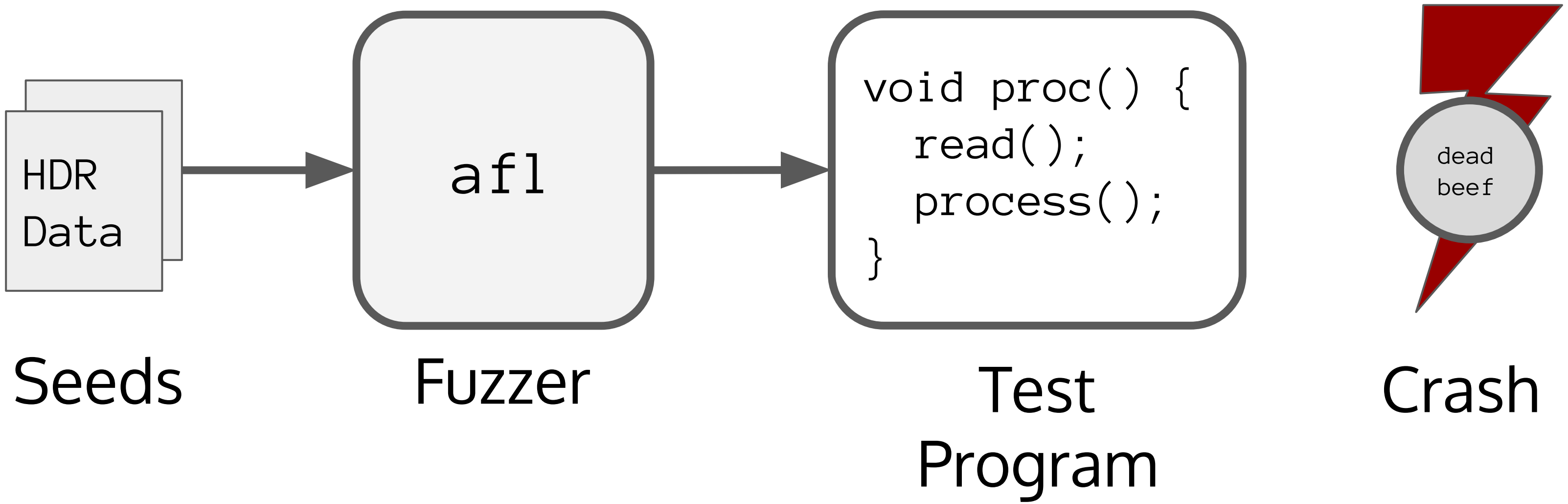
```
    973 tcpdump: unknown file format

    26 tcpdump: truncated dump file; tried to read 4 file header
bytes, only got 0

      1 tcpdump: truncated dump file; tried to read 4 file header
bytes, only got 3
```

Coverage Guided Test Generation

Fuzzing Parsers



```
void proc() {
  read();
  process();
}
```

Howto?

```
$ afl-fuzz -i pcap_seeds -o fuzz_out -- tcpdump -vr @@
```

Under the hood

- Mutate input
- Build bitmap of tcpdump branches covered
- Use bitmap to decide whether to fuzz input further

Observations

- Effectiveness depends on
  - Quality of seeds
  - Program coverage being a good “guide”
- Speed
  - Slower than random testing due to instrumentation overhead
  - Still, typically hundreds of executions per second

Specification Guided Test Generation

Howto? (1/2)

1. Read specification of pcap file format

Global Header	Packet Header	Packet Data	Packet Header	Packet Data	Packet Header	Packet Data	...

2. Map specification to a fuzzy grammar

```
message Pcap{
    required GlobalHeader gh = 1;
    required PacketHeader ph = 2;
```

Howto? (2/2)

3. Write a converter from grammar to file format

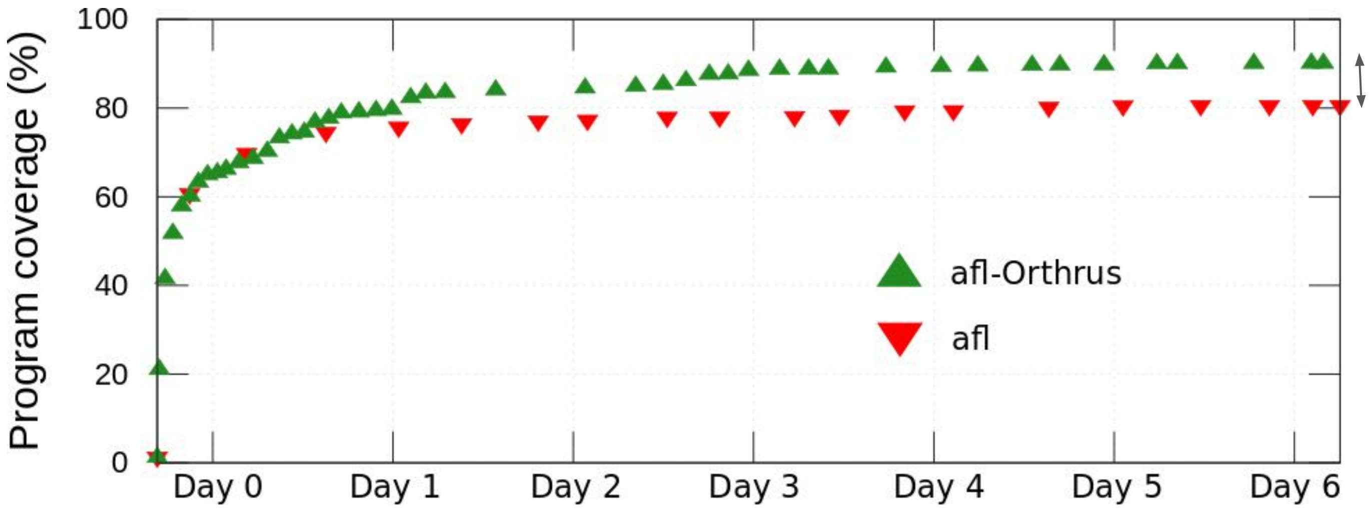
```
void converter::convertPcap(const Pcap& pcap)
{
    convertGlobalHeader(pcap.gh());
    convertPacketHeader(pcap.ph());
    …
}
```

Observations

- Effectiveness depends on
    - Quality of specification
- Speed
    - Slower than coverage-guided test generation
        - Added overhead of converting grammar to concrete input

Results

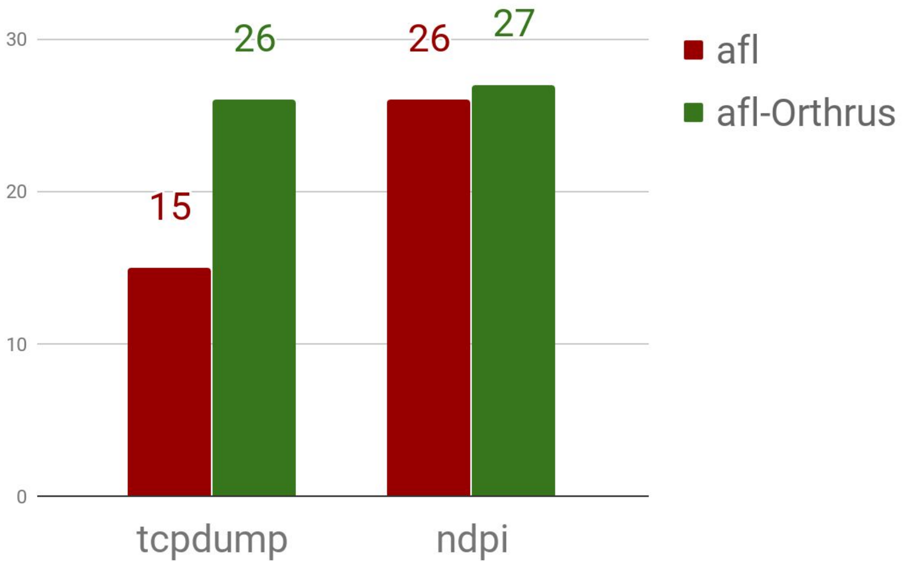
Test Coverage: afl vs afl-Orthrus



+9.7%

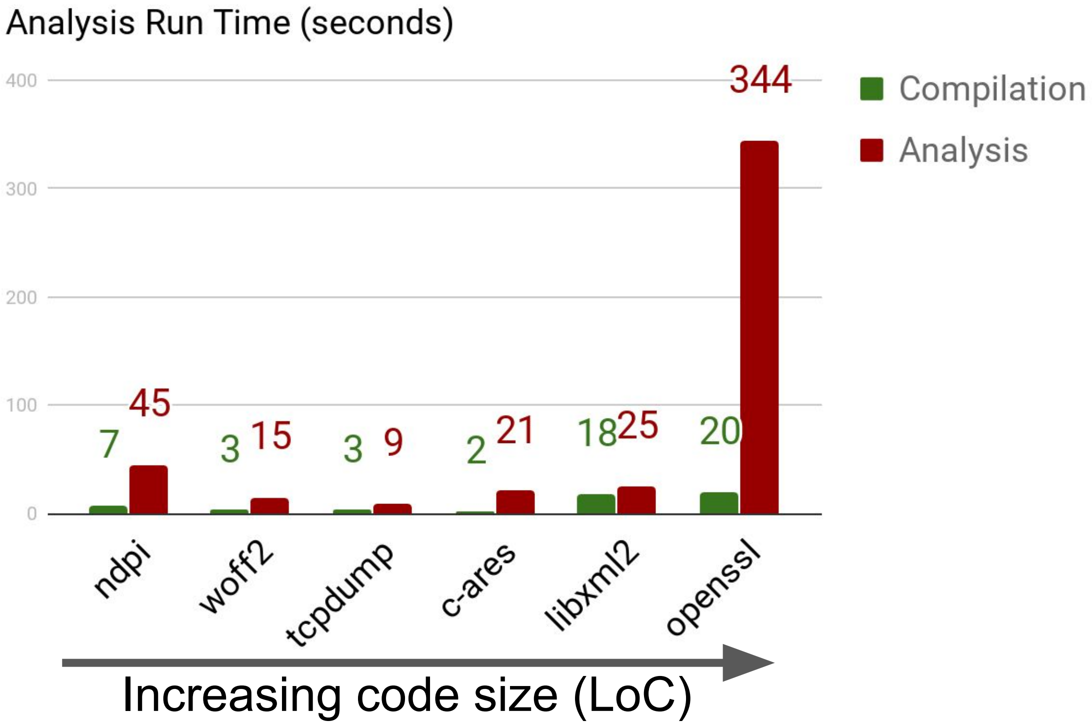
tcpdump

Number of Discovered Vulnerabilities



Orthrus finds 14 new vulnerabilities

Analysis Run Time



Impact: tcpdump 4.9.2

- Fuzzed by eight independent teams
- 92 CVEs discovered in total
- We discovered 43 CVEs using Orthrus

We found **just under 50%** of them!

Open Problems

Stateful Fuzzing

- Traditionally, each “fuzz” tests a program in isolation
- But consider a stateful firewall
  - Action depends on
    - Previous + current packet

What is Good Feedback?

- Feedback drastically improves bug finding ability
- What is good feedback?
  - Traditionally program coverage
  - What else?
    - Probably depends on target

Automatic Generation of Spec

- Specifications are useful but hard to write
- Can they be automatically generated?
    - E.g., based on a set of inputs

Talk Summary

Conclusions

- Vulnerability: A bug that violates security policy
- Vulnerability search problem generally undecidable
- Fuzz testing offers a partial solution
    - Very effective in practice
- Fuzzing techniques have different trade-offs
    - Precision, speed
    - Depends on program under test

References

- [Radamsa] https://gitlab.com/akihe/radamsa
- [afl-fuzz] http://lcamtuf.coredump.cx/afl/
- [libFuzzer] https://llvm.org/docs/LibFuzzer.html
- [StructuredFuzz] https://github.com/google/fuzzer-test-suite/blob/master/tutorial/structure-aware-fuzzing.md