An Introduction to Fuzzing

By Allison Marie Naaktgeboren

Email: naak@pdx.edu

Shakticon discord: allisonn

Portland State
Computer Science

Old Code Monkey, New Researcher

- There's more than one Allison Naaktgeboren!
- New Graduate Research Assistant, Portland State University
  - Security Lab
  - Studying influence of the initial seed corpus on fuzzer performance
- Old Code Monkey
  - Signal Sciences, Mozilla, FactSet, Amazon, Cisco, RI Biorobotics Laboratory, Coding with Kids
  - Carnegie Mellon University, SCS, BS in CS
    - Research in robotics
- CTF
  - President, cofounder PSU Security Club; void* vikings
  - Captain, cofounder QultoftheQuantumQapybaras
  - Minion, Samurai
- Community: OWASP PDX, WWC PDX, First Robotics, Blackhoodie
- Speaking: DEF CON HHV, BSides PDX, Linux Security Summit

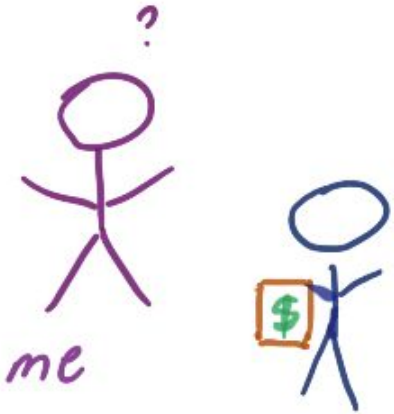
Remember in person CTFs?
DC27 Finals, credit: atlas

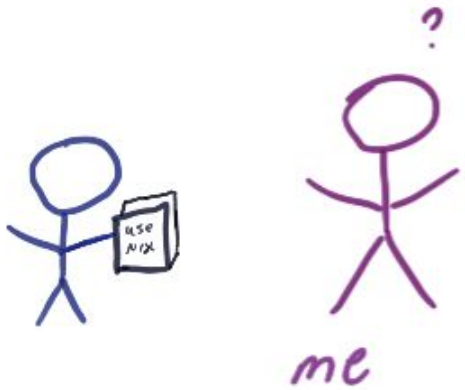
What's on the docket?

Some local friends and their spirit fuzzers

1. What is Fuzzing?
2. Understanding fuzzer differences
3. Understanding fuzzer limitations
4. Who's interested in fuzzing & why?
5. AFL, (some) AFL pitfalls
6. Hands on: Find Heartbleed!

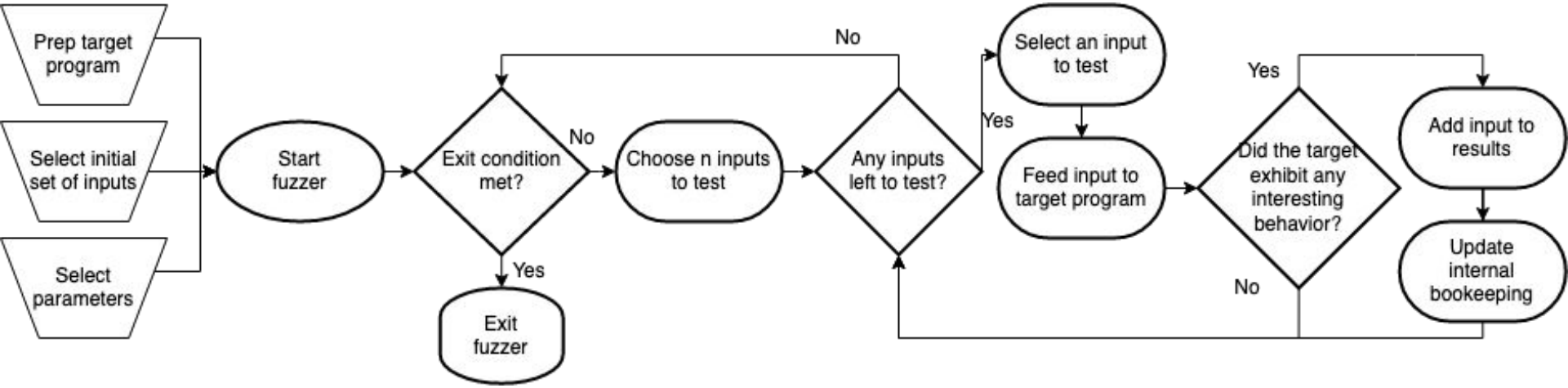
What's this new fuzzing thing I keep hearing about ???





Fuzzing, Fuzz Testing

- A dynamic stochastic software-testing technique
- Its goal is to thoroughly explore the input space of the target program looking for inputs (seeds) that cause ***interesting*** behavior
  - ‘Interesting’ is not universally defined
  - Requires the ability to reach any input in the input space
  - Symbolic execution(e.g. angr) is sometimes considered part of fuzzing
- First published by Bart Miller & students in December, 1990
  - An empirical study of the reliability of UNIX utilities | Communications of the ACM
- Terminology Cheat sheet:
  - https://github.com/anaaktge/talks/blob/main/fuzzingtalkglossary.md



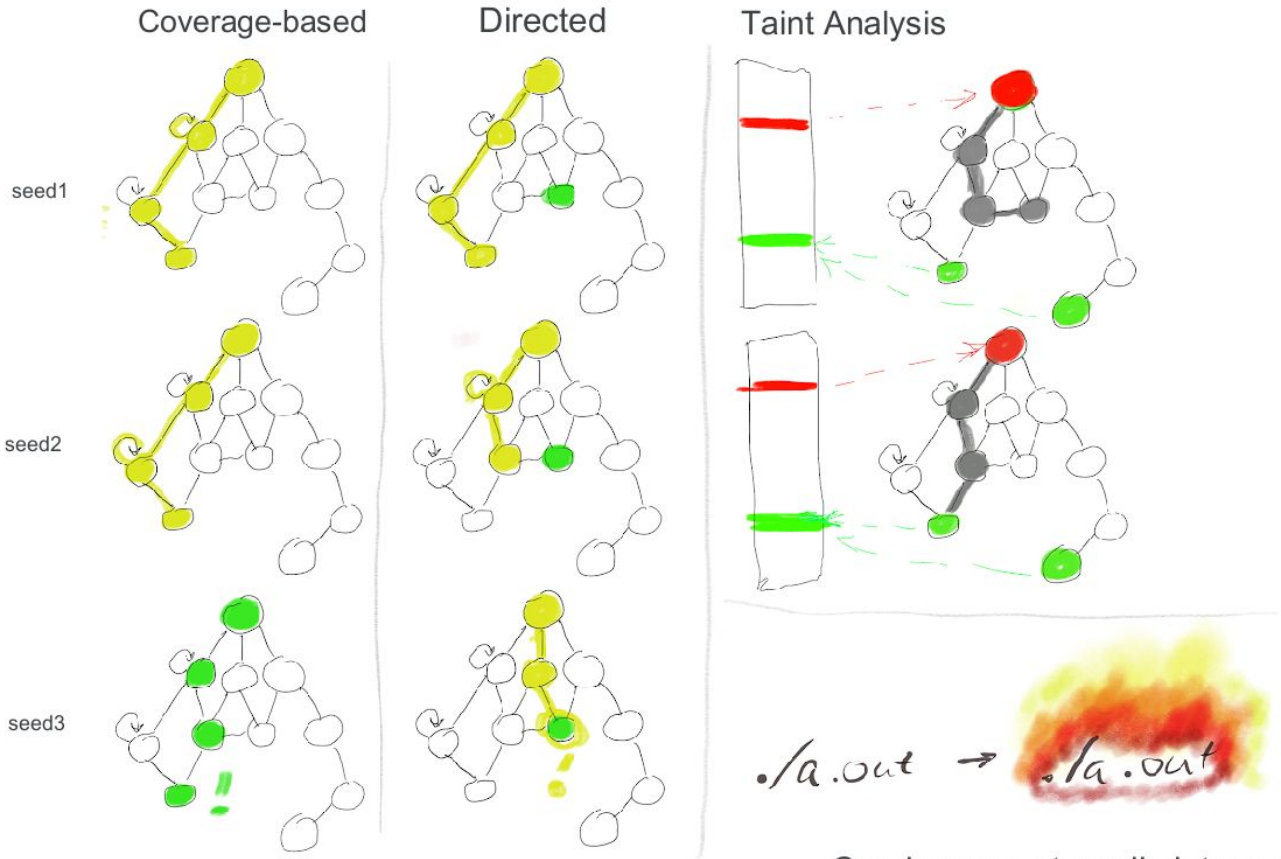
- Setup
  - Prep target program
  - Select initial inputs (seeds)
  - Select parameters
- Start fuzzer

- While exit condition is not satisfied:
  - Get a set of inputs (seeds) to test
  - For each input in set:
    - Feed target program input and run
    - If interesting behavior happens
      - Store input and result
      - Update bookkeeping
    - Else move on

Understanding Fuzzer Differences

1. How do you define interesting?
2. How do you get your seeds (inputs)?
3. What’s your exit condition?
4. How much do you know about the target source?
5. What’s your focus/speciality?
6. Do you have a secret sauce?

1 - How do you define interesting?



Crashes are eternally interesting...

2 - What's your exit condition?

Scheduled Duration

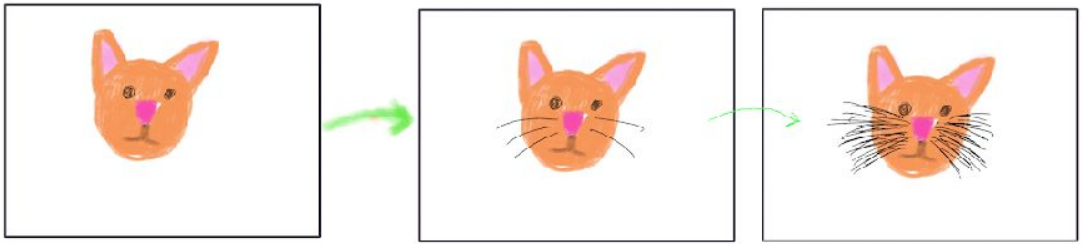
Out of inputs to test

User Initated Exit

One and Done

3 - Where do you get your seeds (inputs) ?

Starting Seed

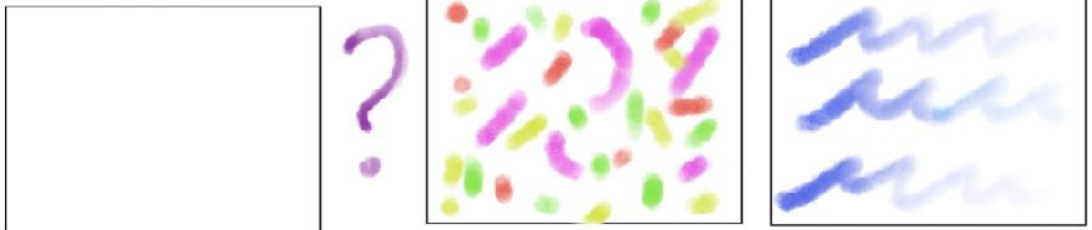


Mutation

Starting Algorithm

Generation

Starting ???



Classic, Naive

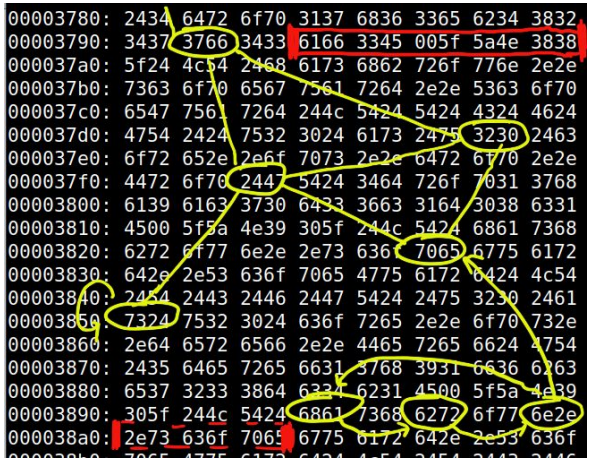
4 - What do you know about the target source?

```
105  // returns a reference
106  pub fn get_info(&self, id: usize) -> Result<String, String> {
107      dbg!("get info on id: {}", id);
108      if id < self.functions.len() {
109          let f = &self.functions[id];
110          let basic_info = f.to_string();
111          let callees = f.calls_ids().fold("Makes calls to: ".t
112              a + "(fn " + &i.to_string() + ": " + self.functio
113          });
114          let body = if let Some(b) = f.body_as_ref() {
115              b.to_string()
116          } else {
117              "[no body; this is an import]".to_string()
118          };
```

Whitebox

```
00003780: 2434 6472 6f70 3137 6836 3365 6234 3832
00003790: 3437 3766 3433 6166 3345 005f 5a4e 3838
000037a0: 5f24 4c54 2468 6173 6862 726f 776e 2e2e
000037b0: 7363 6f70 6567 7561 7264 2e2e 5363 6f70
000037c0: 6547 7561 7264 244c 5424 5424 4324 4624
000037d0: 4754 2424 7532 3024 6173 2475 3230 2463
000037e0: 6f72 652e 2e6f 7073 2e2e 6472 6f70 2e2e
000037f0: 4472 6f70 2447 5424 3464 726f 7031 3768
00003800: 6139 6163 3730 6433 3663 3164 3038 6331
00003810: 4500 5f5a 4e39 305f 244c 5424 6861 7368
00003820: 6272 6f77 6e2e 2e73 636f 7065 6775 6172
00003830: 642e 2e53 636f 7065 4775 6172 6424 4c54
```

Blackbox

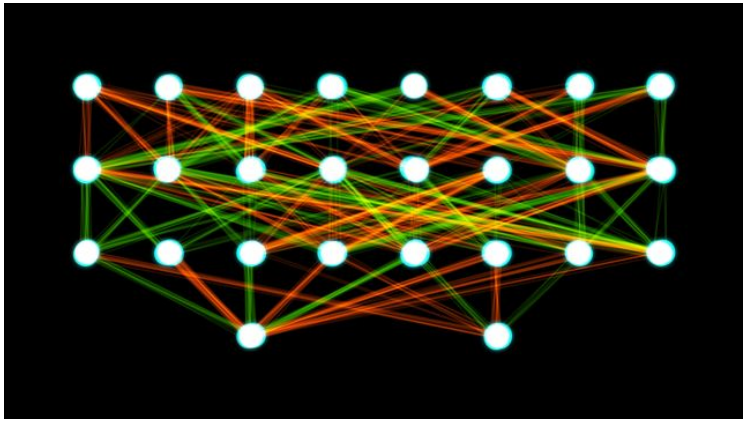


Graybox

5 - What's your focus or speciality?

tux, win, html5 from wikimedia.org

6 - Do you have a secret sauce?



REDQUEEN: Fuzzing with Input-to-State Correspondence

FairFuzz: A Targeted Mutation Strategy for Increasing Greybox Fuzz Testing Coverage

Images from wikimedia.org, project github, or papers

Understanding Fuzzer Differences tl;dr

- How do you define interesting?
  - ➢ Coverage based
  - ➢ Directed
  - ➢ Taint Analysis
  - ➢ Crashes
- What's your exit condition?
  - ➢ Time out
  - ➢ ^C
  - ➢ empty seed set
  - ➢ first crash found
- How do you get your seeds (inputs)?
  - ➢ Mutational
  - ➢ Generational
  - ➢ Classic (naive, random)
- How much do you know about the target source?
  - ➢ White box - full knowledge, source
  - ➢ Greybox - instrumented binary
  - ➢ Blackbox - no knowledge
- What's your focus/speciality?
  - ➢ Protocols, Kernels
  - ➢ Webpages, IoT
  - ➢ cars
- Do you have a secret sauce?
  - ➢ Super Awesome Optimization™
  - ➢ Combined with symbolic execution (hybrid fuzzing)
  - ➢ ML/AI/DNN

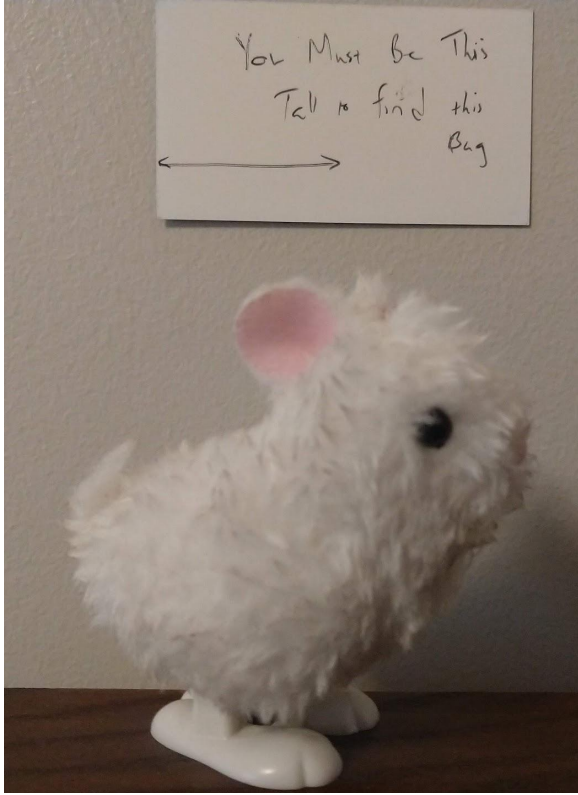
Fuzzing is not magic!

Nor is limitless

Limited by the cold bars of reality...

Fuzzing Limitations

- Hard on your machine
  - CPU intensive, RAM intensive
  - Consumes a lot of power and $$
- Relies on probability.
  - Probability is not always your friend
- Fuzzing does not tell you the ‘why’
  - Understanding the results can be nontrivial
- Subproblems in fuzzing are nontrivial
  - Code coverage algorithms are not absolute
- Figuring out how long to run a fuzzer is nontrivial
  - When does it stop making progress?
  - Going minimum is 24 hours
  - Magma results suggest that a week (168 hours) might be better



Sometimes, you’re just not a good fit..

Is it a Good Fuzzing Target?

Yes

- Obvious parsers
- Stealth parsers (load/store)
- Command line utilities
- Text based APIs
- Libraries

No

- GUI only access
  - Files are fine
- Deep, complex, non visible state
- Complex sequences & exchanges
- Difficult for probability
- Classic stumbling blocks
  - Kernels
  - Network protocols
  - Magic #s, Tight branches

Who’s at the Fuzzing Party?

...and why are some of them really angry?

Fuzzing Party: Offensive Security

You might be a… Vulnerability Researcher, Penetration Tester, Bug Bounty Hunter

Your goals are…

- Finding at least one vulnerability (exploitable bug) or chain
- ASAP!
- Preferably before anyone else
- Don’t care about ‘why’

Features to look for...

- Vulnerabilities
- Fast to first find
- Find tricky bugs other fuzzers have missed
- Source code not required

Fuzzers you might like…

- libFuzzer
- MOpt-AFL

Fuzzing Party: Defensive Security

You might be a… Blue teamer, Application Security Engineer, Security Consultant

Your goals are…

- Find all the vulnerabilities
- Find them all before release
- Care about ‘why’

Features to look for...

- Probe as much of the product as possible
- Complement other security practices, tools

Fuzzers you might like…

- AFLplusplus
- honggfuzz
- AFLGo: Directed Greybox Fuzzing

Fuzzing Party: Software Owners

You might be a… Engineering Manager, Lead Software Engineer, QA Engineer

Your goals are…

- Find vulnerabilities
- Find all the bugs
- Find them before release
- Really care about ‘why’

Features to look for....

- Target specific commits
- CI/CD integration
- Stability
- Finds more than just vulnerabilities
- Code coverage might interest you

Fuzzers you might like…

- classic fuzz
- TOFU: Target-Oriented FUzzer
- Microsoft OneFuzz

Fuzzing Party: Academic Researchers

You might be a...Phd Student, Faculty, Masters Student

Your goals might be…

- A previous group's goals
- Studying Software Testing
- Studying another aspect of CS

Features to look for...

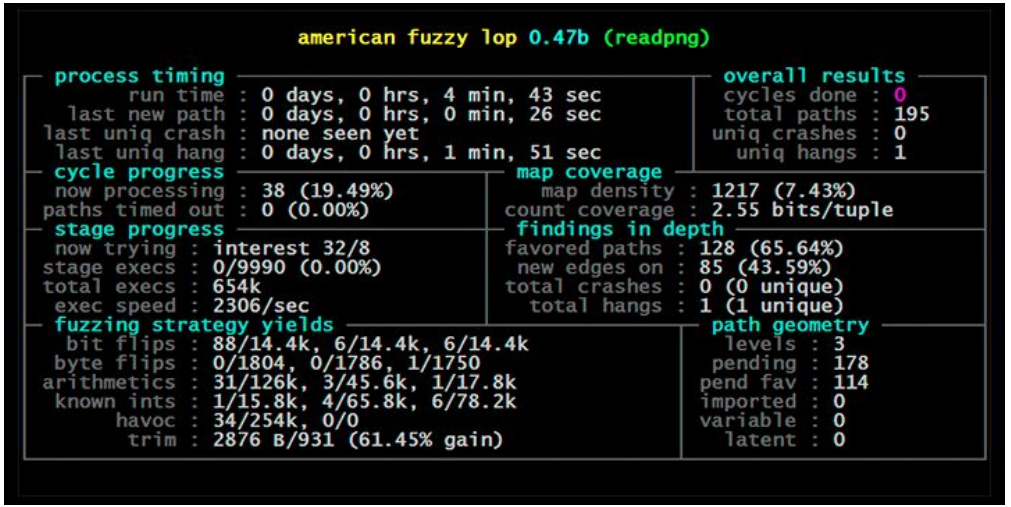
- Benchmark
- Good substrate for testing new ideas
- Publishable
- Citable

Fuzzers you might like…

- AFL
- Developing your own to advance the State of the Art (SOTA)

AFL (American Fuzzy Lop)

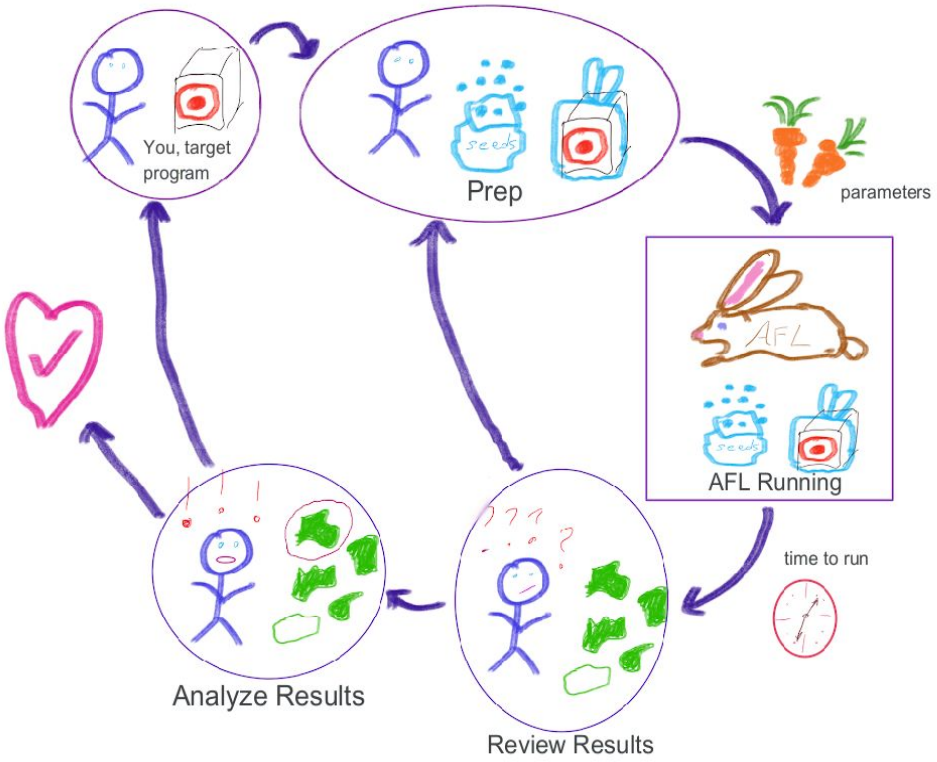
- Famous in research and industry
  - https://lcamtuf.coredump.cx/afl/
- Characteristics
  - Target Knowledge: Greybox
  - Seed Generation: Mutational
  - Interesting: Code coverage
  - Interesting: Crashes, hangs
  - Exit condition: time, ^C, empty
  - Driver: AFL speaks only stdin(ish)
- Lots of knobs to tune
  - Use them!
  - ***An hour to start, a lifetime to master***
- Posse of helper tools
  - afl-min, afl-plot, etc



The iconic main screen
Credit: https://lcamtuf.coredump.cx/afl/

AFL Gotchas

- Start
  - Don’t have source to instrument
- Prep
  - Didn’t create a good initial corpus
  - Didn’t develop drivers/harnesses
    - One call is unlikely to probe the entire program
  - Didn’t instrument everything in binary (including the shared libraries)
- AFL Parameter Configuration
  - Didn’t tune parameters
  - Abused the @ parameter
- Running AFL
  - Need root access to set processor settings
  - Didn’t run the fuzzer long enough (1 to 7 days)
  - Might need to repeat the fuzzer run
- Reviewing, Analyzing the Results
  - Crash != vulnerability.
    - Might not have security value
  - Unique crashes may not be unique
    - 20+ crashes might only be 1 source



*“In my experience as a vuln researcher, the two most important things are a **good harness** and **good seeds**”*

- *A Very Accomplished Vulnerability Researcher*

How do I get a good initial seed corpus for AFL ?

- We don't really know what an 'optimal' corpus is in the general case
- Get to know the target:
  - Run the target a few times
  - Examine source or binary
- Things I've done to get an initial corpus
  - Raid the unit test/test cases for the project
  - Run 'strings' on the target binary and edit it down
  - Run the program a few times and find a few inputs
  - Do some RE (Reverse Engineering) on the target!
- Running afl-min is a very good idea!

wikiMedia, navy bean

Acknowledgements & Further Reading

- Shakticon organizers
- Dr. Wu-chang Feng, advisor
- National Science Foundation
- The Security Lab, Portland State
- The movie crew/talk guinea pigs

- Invention of fuzzing: An empirical study of the reliability of UNIX utilities | Communications of the ACM
- 30 years later: The Relevance of Classic Fuzz Testing: Have We Solved This One?
- Magma Fuzzing Benchmark
- FairFuzz: FairFuzz: a targeted mutation strategy for increasing greybox fuzz testing coverage
- AFLFast: AFLFast (extends AFL with Power Schedules)
- REDQUEEN: Fuzzing with Input-to-State Correspondence

Practice Time!

- https://codelabs.cs.pdx.edu/cs492/
  - Select the AFL codelab
  - Uses docker
- You can do these locally or on Google Cloud
  - *I've never tried on windows
- Contact
  - naak@pdx.edu
  - Shakticon discord: allisonn

wikiMedia Heartbleed