A Gentle Introduction to Understanding Fuzzers

Allison Naaktgeboren
Phd Researcher,
Portland State University

Track 2
Rooms 329-327

Portland State
Computer Science

Agenda

1. **Introductions & Goals**
2. What fuzzing is & isn't
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
    - Craft: Optimizing Seeds for Mutational Fuzzing
6. Craft: Integrating Fuzzing into Software Lifecycle
7. Lab: Finding Heartbleed using AFL

Who I Am: Allison Naaktgeboren

Doctoral Researcher, studying fuzzers, Portland State, 2020-Present

- Advisor: Dr. Andrew Tolmach (verification, compilers, PL)
- 2023 Draper Labs Intern, VMF fuzzer I2S feature
- Quals Research: “Sowing the Seeds of Fuzz: Does the Influence of the Initial Seed Corpus Follow a Universal Law?” Jun 22
- CTF teams: void * vikings: founder + advisor. 侍 : minion
- Instructor, Lecturer, TA: Intro to Systems, Malware, Intro CS
    - + all legal obligations
- Masters Degree, Cybersecurity Grad Cert

BS in CS, Carnegie Mellon University, 2004-2008 (SCS ‘08)

- Undergraduate research in robotics
- CS TA, roboclub, CMU KGB, W@SCS

Previous Life: Senior Code Monkey, Hiring Manager, etc ~2009-2019

- Cisco, Amazon, Factset, Mozilla (Firefox), Signal Sciences (now Fastly)

The more fabric on your robe, the higher your rank.

The best regalia are not fancy sleeves and velvet, but minions who bring you donuts and coffee

Who I Think You Are:

- You're probably fried from 2 days of con awesomeness
  - Thanks for coming to the last talk <3
  - Focus on big picture ideas and most common fuzzers
- You've at least heard of fuzzing
- You're likely relatively new to infosec
- You probably have some developer experience
  - or have had least a CS class or two
  - Experts may be bored

Img from https://cheezburger.com/18381573/sleeper-memes-for-the-sleepiest-of-jokesters-that-are-constantly-exhausted

Goal of Talk

1. Frame your thinking about fuzzers
    - And hopefully avoid being prescriptive
2. Learn something
    - And hopefully ~~be mildly entertained~~ not fall asleep
3. Understand that fuzzing is still more craft than science
    - And hopefully how to apply some of that craft

Agenda

1. Introductions & Goals
2. **What fuzzing is & isn't**
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
    - Craft: Optimizing Seeds for Mutational Fuzzing
6. Craft: Integrating Fuzzing into Software Lifecycle
7. Lab: Finding Heartbleed using AFL

Automation to the Rescue? Fuzzing Better than Santa Claus?

The Register®
Biting the hand that feeds IT

Security

Linus Torvalds lauds fuzzing for improving Linux security

But he's not at all keen on Santa Claus or fairies

By Simon Sharwood, APAC Editor 16 Oct 2017 at 07:03

ZDNet MENU US

MUST READ A POPULAR VIRTUAL KEYBOARD APP LEAKS 31 MILLION USERS' PERSONAL DATA

Linux security: Google fuzzer finds ton of holes in kernel's USB subsystem

A Google-developed kernel fuzzer has helped locate dozens of Linux security flaws.

By Liam Tung | November 8, 2017 -- 12:43 GMT (04:43 PST) | Topic: Security

From Allison's lecture, CS 492/592 Malware Reverse Engineering

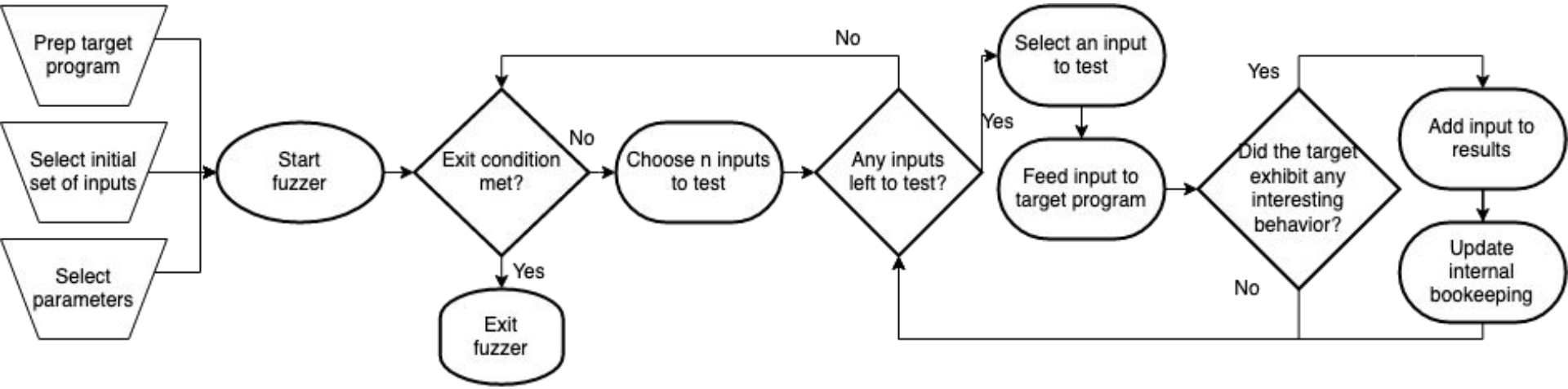
It was a dark and stormy night in 1989…

AI Generated : https://www.prompthunt.com

Fuzzing's Origin Story

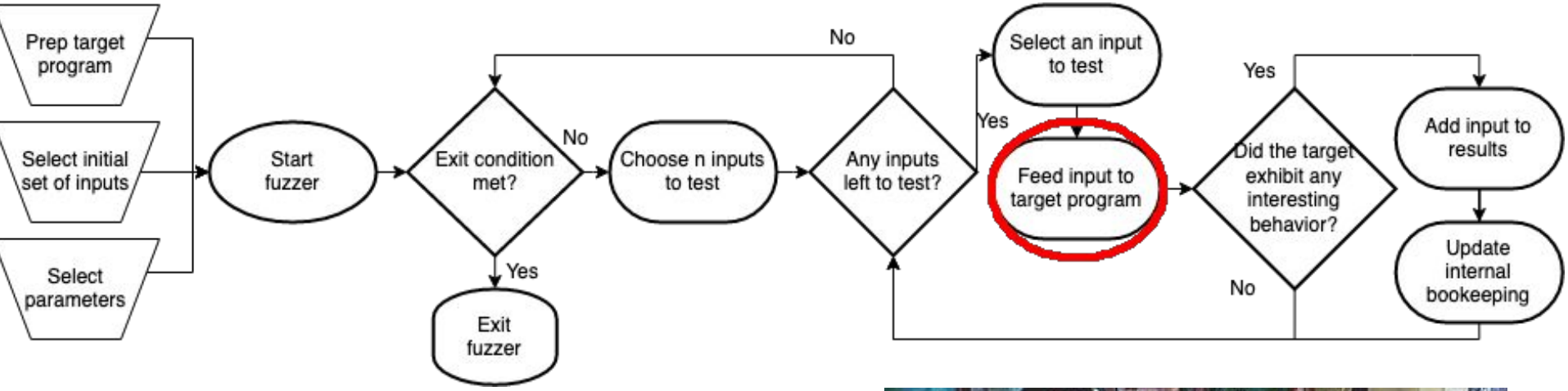
- Well, CS students always need final projects!
  - Getting students to solve your problems is a time honored tradition in academia
- 1990: Public invention of fuzzing:
  - An empirical study of the reliability of UNIX utilities in CACM https://dl.acm.org/doi/abs/10.1145/96267.96279
  - 2 student groups attempt the project (src: Dr. Bruce Irvin, Bart Miller's student)
- CS Reception is poor…
  - Dr. Miller is told it sucks so much he should leave CS
  - but the Vulnerability Researchers see the potential…
  - https://www.cs.wisc.edu/2021/01/14/the-trials-and-tribulations-of-academic-publishing-and-fuzz-testing/
- AFL puts fuzzing on the map (though it's certainly not first)
  - Various theories as why: Impressive trophy case, easy to set up, status screen, right place, right time?
- Also worth a read : The Relevance of Classic Fuzz Testing: Have We Solved This One? (30 year retrospective)

Overview of Fuzzing



- Aims to thoroughly explore the input space of the fuzzee (victim, target, SUT, PUT) looking for inputs (seeds) that cause interesting behavior
- Definition of “Interesting” varies, and is a key feature of fuzzer taxonomy
- A stochastic (probabilistic) dynamic software-testing technique (gotta run code)

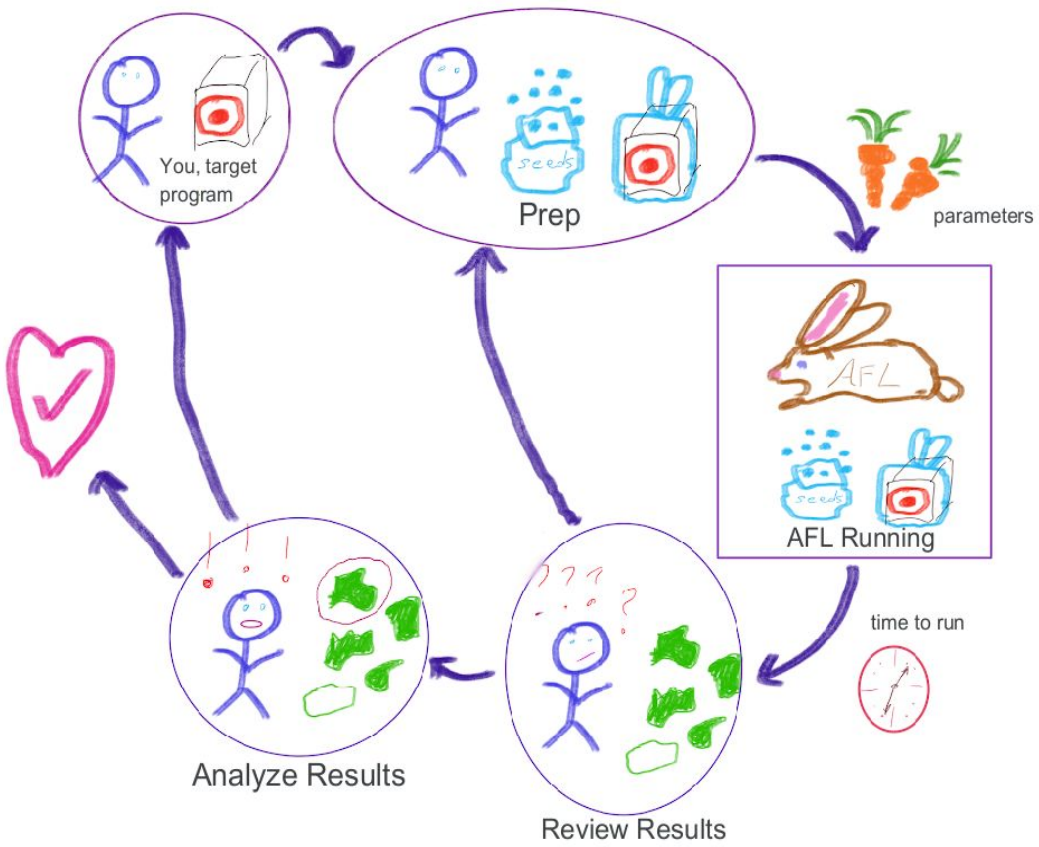
Fuzzers, Fuzzees, & Drivers



- Drivers (harnesses) ‘translate’ between fuzzer output and fuzzee input
- Not all fuzzer-fuzzee pairs need one
- Can be a nontrivial amount of engineering work

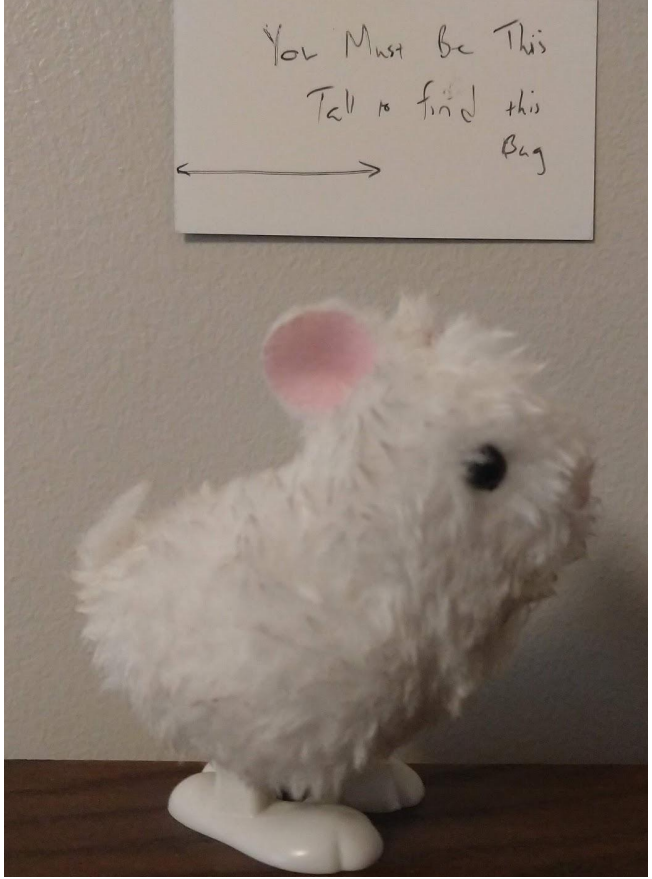
The General Fuzzer workflow

1. Set up
2. Debug with short fuzz jobs
3. Fuzz for real
4. Review & Check results
   - “Crash triage problem”
   - “Reproducibility problem”
5. (Optionally) Analyze
   - Fuzzers rarely know “why”
   - “Root cause problem”
6. Do something with the results
   - Attack, report, tune fuzzer, etc
7. Repeat



Fuzzing Limitations

- Hard on your machine
  - CPU intensive, RAM intensive
  - Consumes a lot of power and $$
- Probability is not always your friend
  - Guess x in “if x == 1337”
- Not good for truly dangerous code
  - If you’re afraid to run it, that’s a problem
- Fuzzing does not know the ‘why’
  - Can have a very bad signal to noise ratio
  - Understanding the results is resource expensive
- Relies on approximations and assumptions
  - A clean fuzz job is not proof there are no bugs
  - “Absence of evidence is not evidence of absence”



“You must be this tall to find this bug”

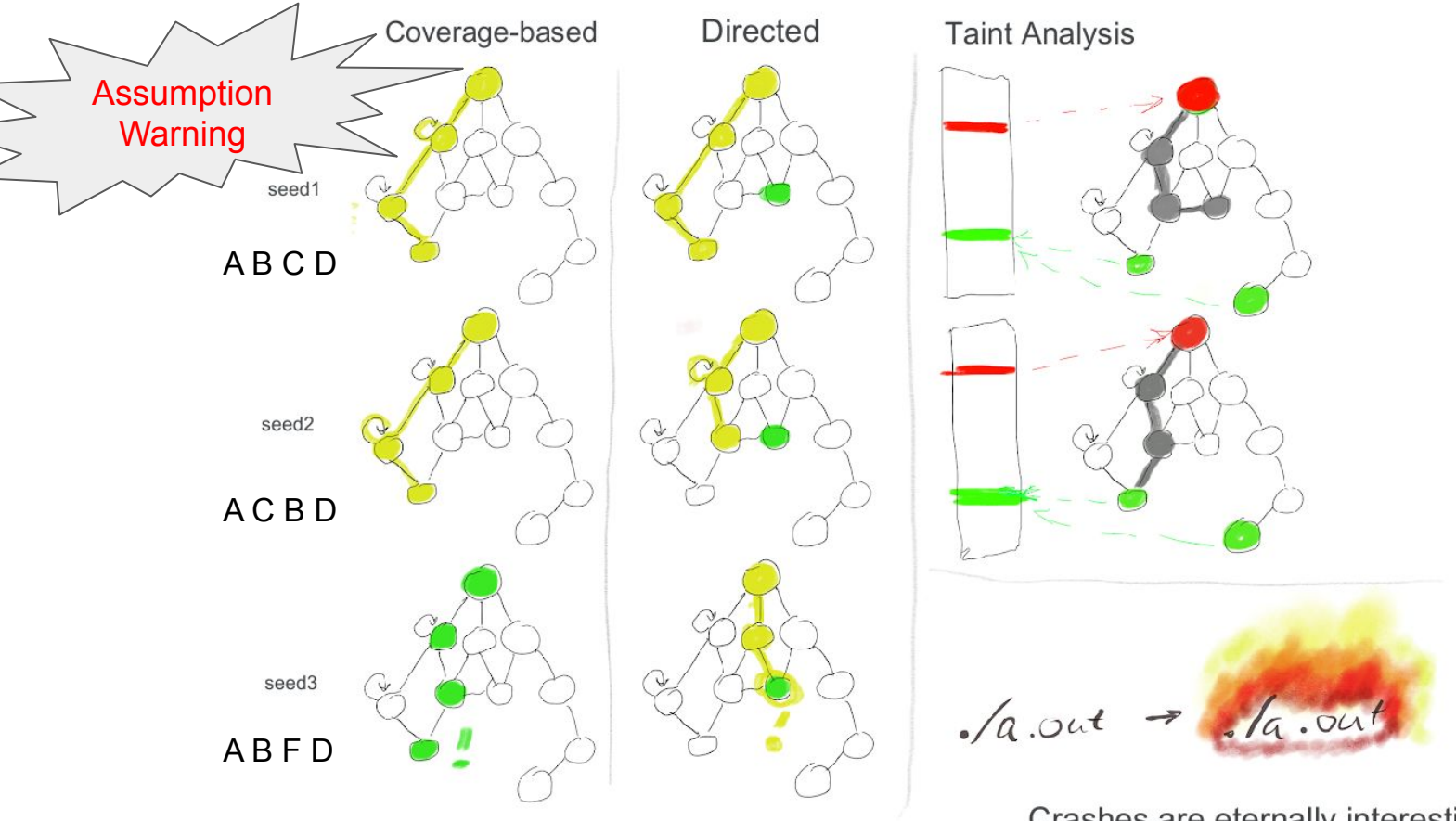
Agenda

1. Introductions & Goals
2. What fuzzing is & isn't
3. **Breaking down a fuzzer (5 questions)**
    - **Example: breaking down AFL++**
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
    - Craft: Optimizing Seeds for Mutational Fuzzing
6. Craft: Integrating Fuzzing into Software Lifecycle
7. Lab: Finding Heartbleed using AFL

Key Features of Fuzzer Taxonomy

1. What is interesting behavior to this fuzzer?
2. How does the fuzzer get new inputs (seeds)?
3. What is the exit condition?
4. How much does the fuzzer know about the fuzzee source?
5. Does the fuzzer have a speciality, focus, magic pixie dust?

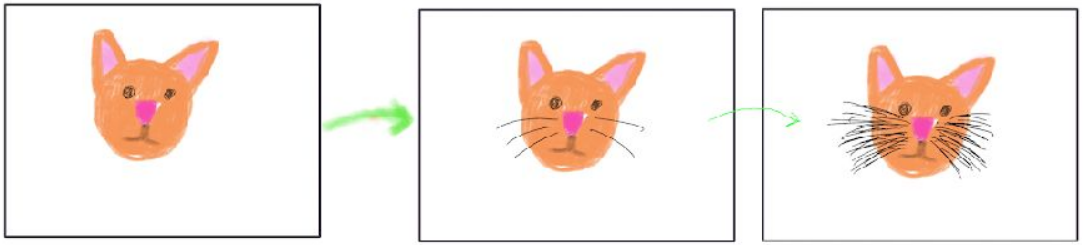
1 - What is interesting behavior?



Crashes are eternally interesting...

2 - Where do new seeds (inputs) come from ?

Starting Seed

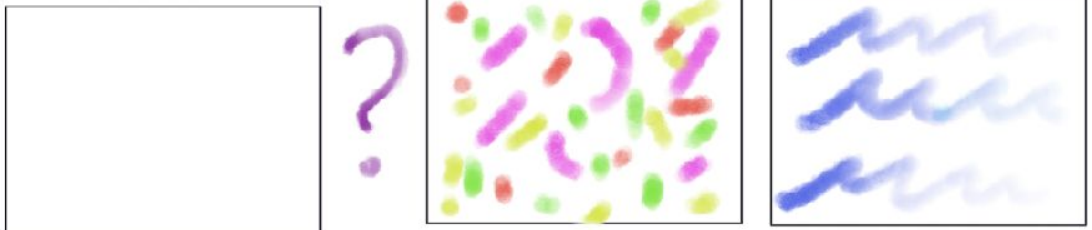


Mutation

Starting Algorithm

Generation

Stdin works



Classic, Naive

3 - What's the exit condition?

Scheduled Duration

Out of inputs to test

User Initated Exit

One and Done

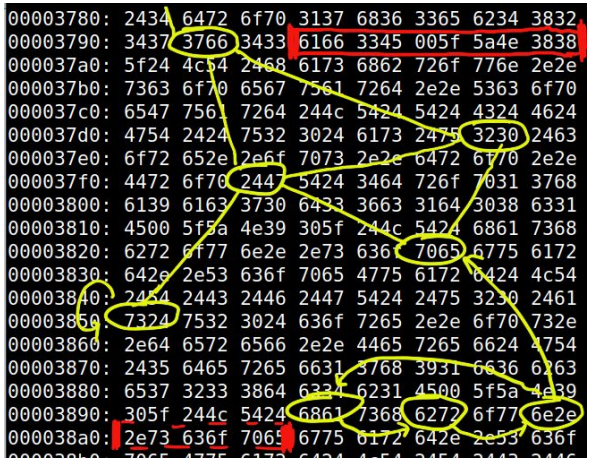
4 - What does the fuzzer know about the fuzzee source?

```
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 the focus or speciality?

boofuzz

HTML

syzkaller - kernel fuzzer

tux, html5 from wikimedia.org

The 5 questions, applied to AFL++

1. What is interesting behavior to this fuzzer?
   a. System crashes, seg fault
   b. New coverage approximated by tuples
2. How does the fuzzer get new inputs (seeds)?
   a. Multiple mutation strategies on prior inputs in a priority queue
3. What is the exit condition?
   a. User exit (can script a time out)
   b. Errors, e.g. OS or cloud service kills it for resource hogging
4. How much does the fuzzer know about the fuzzee source?
   a. Greybox: heavily instrumented binary
   b. Usually built from source
5. Does the fuzzer have a speciality, focus, magic pixie dust?
   a. Compiled userland binary applications
   b. ~ 20 research papers worth of ideas have been rolled into it
   c. “An hour to set up, a lifetime to master”

Agenda

1. Introductions & Goals
2. What fuzzing is & isn't
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. **Different strokes for different folks: Who uses a fuzzer and why**
    - **How to frame picking a fuzzer for your needs**
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
6. Craft: Optimizing Seeds for Mutational Fuzzing
7. Craft: Integrating Fuzzing into Software Lifecycle
8. Lab: Finding Heartbleed using AFL

Before you can pick fuzzer(s), you must figure out what **your** needs and goals are.

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...

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

Fuzzers you might like…

- libAFL
- honggfuzz
- MOpt-AFL
- Your own secret fuzzer

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
- Vulns > bugs, but bugs ok-ish

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 too
- Find them before release
- Really care about ‘why’

Features to look for....

- Target a specific commit or area (directed fuzzers)
- CI/CD integration
- Stability
- Finds **bugs and vulns**
- Snapshotting, time travel
- Code coverage interests you

Fuzzers you might like…

- TSFFS on SIMICS (libAFL internals)
- classic fuzz
- libFuzzer
- TOFU: Target-Oriented FUzzer
- Microsoft OneFuzz

Fuzzing Party: Academic Researchers

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

Your goals might be…

- A previous group's goals
- Studying Software Testing
- Studying another aspect of CS (parsers, synthesis)

Features to look for...

- Known Benchmarks
- Good substrate for testing new ideas
- Publishable
- Citable

Fuzzers you might like…

- LibAFL
- VMF
- **Developing your own** to advance the State of the Art (SOTA)

Agenda

1. Introductions & Goals
2. What fuzzing is & isn't
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. **Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)**
    - Craft: Optimizing Seeds for Mutational Fuzzing
6. Craft: Integrating Fuzzing into Software Lifecycle
7. Lab: Finding Heartbleed using AFL

Optimizing Fuzzer performance, Generally

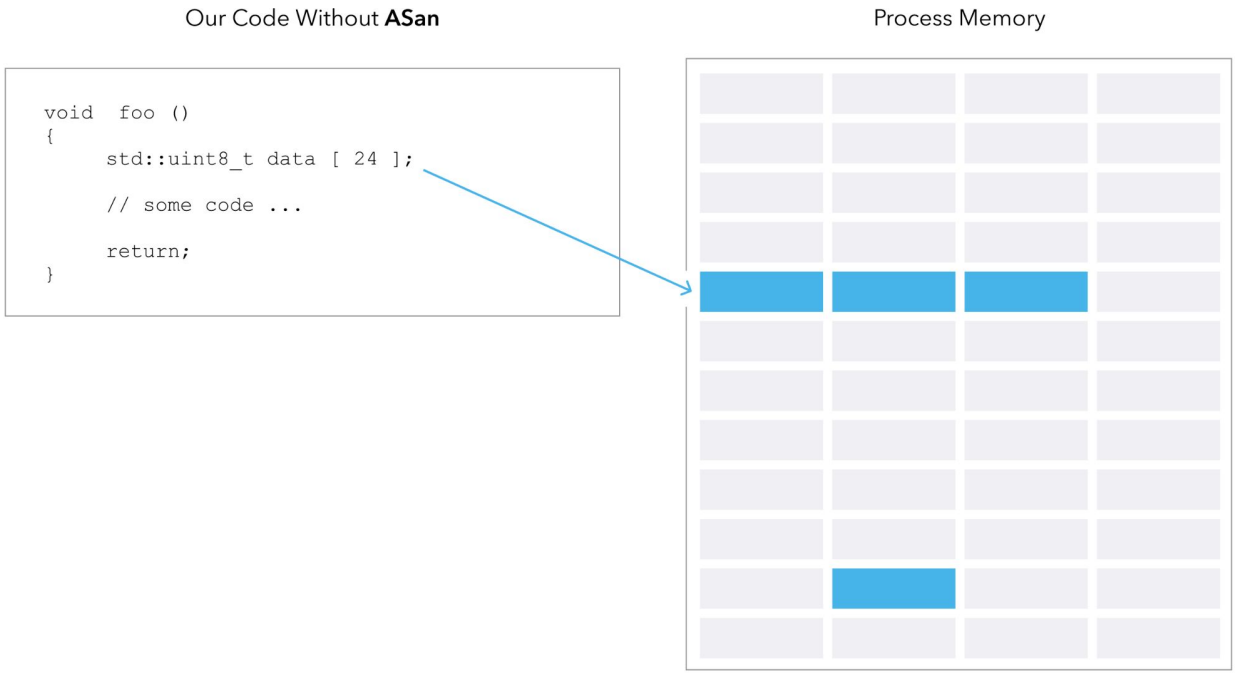
- Harness (drivers) matter
  - Help the fuzzer help you
  - Don’t skimp
  - You may need more than one.
- Spend time on the parts that make the fuzzer tick
  - What powers Slides 15-20?
- If nothing else, focus on what powers exploration
  - Generational? Focus on nailing all parts of a the generative algo (e.g. grammar)
  - Mutational? Focus on what gets mutated
  - Does it have special features that can help you? (wordlists, dictionaries, etc)
- Makes notes on what tuning works with what fuzzees
- Make more things crash is generally good (crashes are universally interesting)
  - Not all vulns crash by default
  - Using debug builds with asserts left on if possible
  - **If nothing else, Use as many sanitizers as possible**

Wait, this is an intro talk!

What is address sanitization?

https://dilbert.fandom.com/wiki/The_Garbageman

Helping Your Fuzzer Crash: Address Sanitization



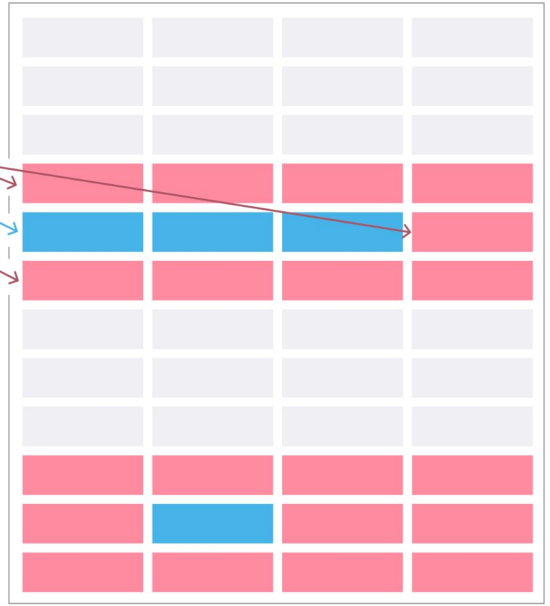
https://m-peko.github.io/craft-cpp/posts/be-wise-sanitize-keeping-your-cpp-code-free-from-bugs/

Helping Your Fuzzer Crash: Address Sanitization

Our Code With **ASan**

```
void  foo ()
{
     std::uint8_t redzone_1 [ 32 ];
     std::uint8_t data      [ 24 ];
     std::uint8_t redzone_2 [  8 ];
     std::uint8_t redzone_1 [ 32 ];

     // some code ...

     int * shadowBase  MemToShadow( redzone_1 );
     shadowBase[0] = 0xffffffff; // poison redzone_1
     shadowBase[0] = 0xffffffff; // poison redzone_2
     shadowBase[0] = 0xffffffff; // poison redzone_3

     // some code ...
     if ( IsPoisones( &data ) )
     Crash();

     // unpoison all
     shadowBase[0] = 0;
     shadowBase[1] = 0;
     shadowBase[2] = 0;


     return;
}
```

Shadowed Process Memory



https://m-peko.github.io/craft-cpp/posts/be-wise-sanitize-keeping-your-cpp-code-free-from-bugs/

Agenda

1. Introductions & Goals
2. What fuzzing is & isn't
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
    - **Craft: Optimizing Seeds for Mutational Fuzzing**
6. Craft: Integrating Fuzzing into Software Lifecycle
7. Lab: Finding Heartbleed using AFL

Optimizing Mutational Fuzzing

- Currently the dominant paradigm and the one beginners encounter first
- All the general advice applies
- But inputs (seeds) are a big focus for mutational fuzzers
  - They are what gets mutated
  - How, when, and with what are they mutated
- Mutation can be micro-tuned
  - Mutate the whole input or just part of it?
  - Mutate the keyword list? (auto token feature in AFL++)
  - Craft: Various strategies work better for certain fuzzees
- Initial corpus (initial set of inputs, typically provided by the user)
  - Have to have something to start mutations with
  - How much work should you put in?
    - There are two dominant theories on the internet

#1: Conventional Theory:

The Initial Corpus is always very important.
If you don't have one, you'll lose out.

Coverage-guided fuzzers like libFuzzer rely on a corpus of sample inputs for the code under test. This corpus should ideally be seeded with a varied collection of valid and invalid inputs for the code under test; for example, for a graphics library the initial corpus might hold a variety of different small PNG/JPG/GIF files. The fuzzer generates random mutations based around the sample inputs in the current corpus. If a mutation triggers execution of a previously-uncovered path in the code under test, then that mutation is saved to the corpus for future variations.

https://llvm.org/docs/LibFuzzer.html#corpus

#2: Zalewski Theory:
Not worth the bother.
Just fuzz a little longer and the results will be about the same.

Building the input corpus

Every fuzzer takes a carefully crafted test cases as input, to bootstrap the first mutations. The test cases should be short, and cover as large part of code as possible. Sadly - I know nothing about netlink. How about we don't prepare the input corpus...

Instead we can ask AFL to "figure out" what inputs make sense. This is what Michał did back in 2014 with JPEGs and it worked for him. With this in mind, here is our input corpus:

```
mkdir inp
echo "hello world" > inp/01.txt
```

https://blog.cloudflare.com/a-gentle-introduction-to-linux-kernel-fuzzing/, 2019

So who's right?

- Theyre both right. Just not all the time.
  - I have my theories and data, but it's hardly airtight
  - Could be structure level, size of control surface, topology of fuzzee, or something else
- The more structured the input, the more it seems to matter
  - Sqlite3 on a "hello corpus" might as well have been powered off
- The less structured the input, closer to binary data, the less it seems to matter, especially for multiday runs
  - > 1 day, it really didn't matter for tiff files in libtiff
- **You don't lose anything by spending 30 minutes making an initial corpus, but you can gain things**
  - At the very least, run "strings" on the fuzzee and use those as seeds
  - If open source, raid the unit tests/test case directories.
  - Run the fuzzee or do some RE looking for interesting areas and grab strings/values from there

Agenda

1. Introductions & Goals
2. What fuzzing is & isn't
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
    - Craft: Optimizing Seeds for Mutational Fuzzing
6. **Craft: Integrating Fuzzing into Software Lifecycle**
7. Lab: Finding Heartbleed using AFL

Fuzzing in the SDLC (Fuzz Left)

- No Free Lunches and No Silver Bullets
  - Budget for server time and user time each cycle
- Probably going to need more than one fuzzer
  - Finding all the bugs is harder than finding one
  - Different parts of a system frequently need different fuzzers
    - Fuzz binary client with libFuzzer
    - Fuzz web apis with FFuF or boofuzz
  - Different fuzzers find different bugs, even on the same fuzzee
- Probably Fuzz for at least 24 hours, if not a week
  - Perhaps on alpha branch ?
  - Your adversaries are, so you probably should too
- Invest in seed re-use for at least one of the fuzzers
  - Re-use output corpus from yesterday's job as today's input corpus
- Dedicate real time to triage, analysis, & messaging
  - To get bugs fixed, they need to be understood first
  - Indecipherable reports get ignored by devs

Image by
https://bishopfox.com/blog/fuzzing-aka-fuzz-testing

Agenda

1. Introductions & Goals
2. What fuzzing is & isn't
3. Breaking down a fuzzer (5 questions)
    - Example: breaking down AFL++
4. Different strokes for different folks: Who uses a fuzzer and why
    - How to frame picking a fuzzer for your needs
5. Craft: Optimizing a Mutational Coverage-Guided Greybox Fuzzer (AFL family)
    - Craft: Optimizing Seeds for Mutational Fuzzing
6. Craft: Integrating Fuzzing into Software Lifecycle
7. **Lab: Finding Heartbleed using AFL**

Thanks for Listening! Questions & Practice Time

- https://codelabs.cs.pdx.edu/cs492/
  - Select the AFL codelab
  - Uses docker
  - From PSU CS 492/592, Malware Reverse Engineering
    - CS 201 prerequisite
    - https://thefengs.com/wuchang/courses/cs492/
- You can do these locally* or on Google Cloud*
  - *I've never tried on windows
- Contact
  - Email: naak@pdx.edu
  - Bsidespdx discord: anaaktge
  - (don't really use linkedin)
  - Talks at https://github.com/anaaktge/talks
  - Conversation I want to have: What is hard to fuzz for in binaries? What's useful but nondestructive? Heap leaks?

wikiMedia Heartbleed

Extra Time:
Research Slides

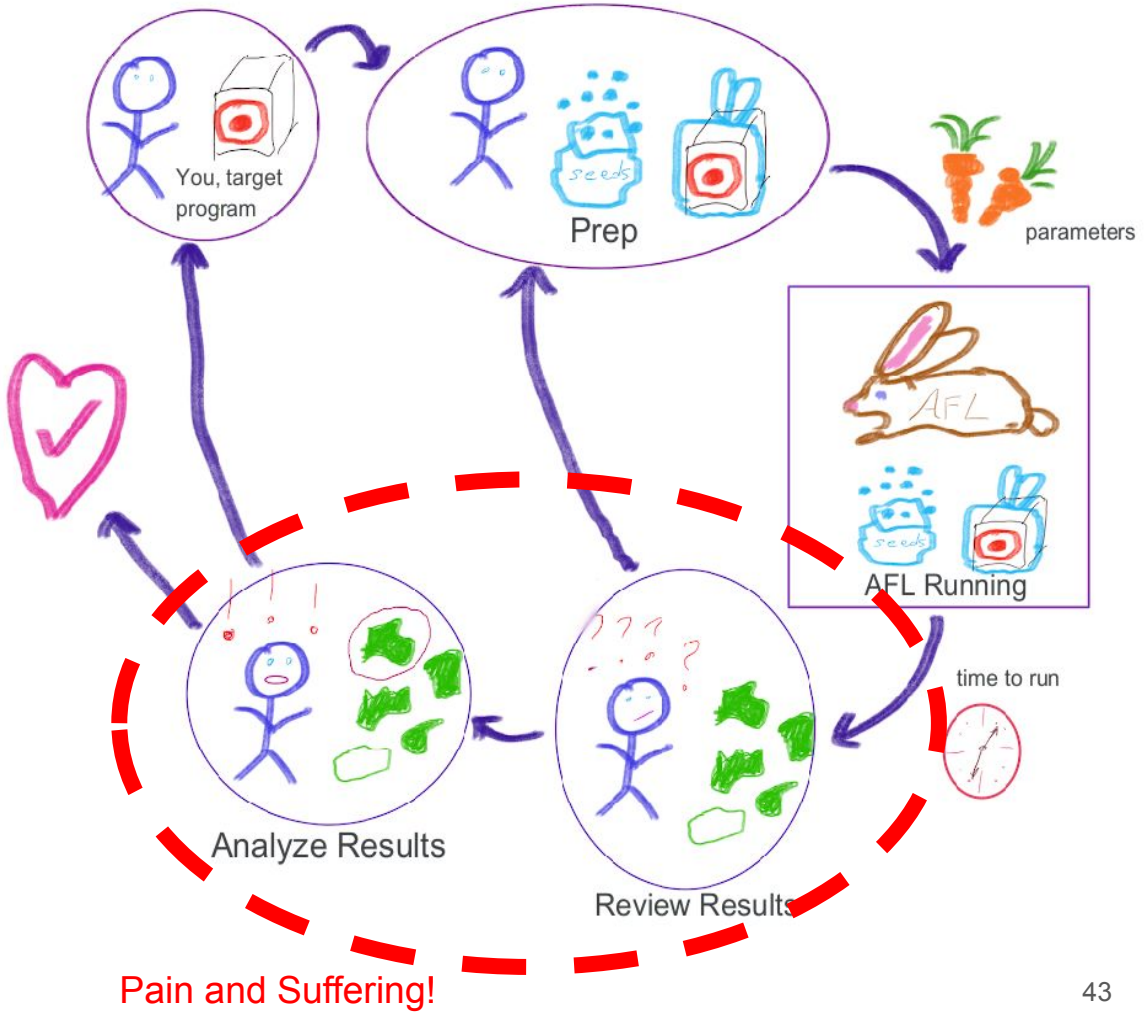
The Problem of Root Cause

"Developers generally appreciate bug reports, but they can sometimes be a bit **less enthusiastic about a flood of reports from automated fuzzing** systems." https://lwn.net/Articles/904293/

"..you have an ethical and moral responsibility to do some of the work to narrow down and **identify the cause of the failure**, not just throw them at someone to do all the work" https://lwn.net/Articles/917762/

How bad is it? (audience participation)
If a fuzzer produced 3,020 crashes, how many bugs do you think there were?

Ans: there were 15 bugs (FuzzerAid)



PIPE (Programmable Interlocks for Policy Enforcement)

- Is a Tag based hardware security reference monitor
  - ISA extension
- A tag represents arbitrary metadata associated with some data
  - Ex “This is a pointer to object a”, “This is low security data”, “The active function is f”, “this instruction can only be used by high security data’
- All data have tags
  - Check relevant tag rules on each execution step
  - if policy is violated (maybe) failstop
- PIPE, doesn’t care about language, high or low
- Problem: hardware doesn’t entirely exist yet, and what does isn’t widely available .

TaggedC

- Built on Compcert's Interpreter
  - Give semantics for all undefined behavior
- PIPE policies are encoded at source level
  - Developers rarely speak assembly or machine code
- PL-y magic happens here
  - Sean Anderson's dissertation work
- Formalizing Stack Safety as a Security Property
  - [2105.00417] Formalizing Stack Safety as a Security Property, https://web.cecs.pdx.edu/~apt/csf23.pdf
  - Sean is presenting it at a conference later this month
  - TaggedC in more detail: Flexible Runtime Security Enforcement with Tagged C https://web.cecs.pdx.edu/~apt/rv23.pdf
- Verified compilation part
  - Towards formally verified compilation of tag-based policy enforcement https://dl.acm.org/doi/10.1145/3437992.3439929