Portland State
Computer Science

An Introduction to Fuzzing
PSU Security Club
Guest Talk

Allison Naaktgeboren
naak@pdx.edu
https://github.com/anaaktge/talks
https://web.cecs.pdx.edu/~naak/

Introductions

Who I Am: Allison Naaktgeboren

Doctoral Researcher, PSU, 2020-Present

- Advisor: Dr. Andrew Tolmach (verification, compilers, PL)
  - Fellowship: Draper Scholars
  - 2023 Draper Labs phd Intern
- CTF:
  - void * vikings: founder, officer emeritus.
  - 侍 : minion
  - BSIDESPDX: CTF Organizer
- Instructor, Lecturer, TA: Intro to Systems, Malware, Intro CS
- PSU Masters Degree, Cybersecurity Grad Cert
- Guest Speaker at Linux Security Summit 2020

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

- Undergraduate research in robotics
- 2008 new grad resume available to current students

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

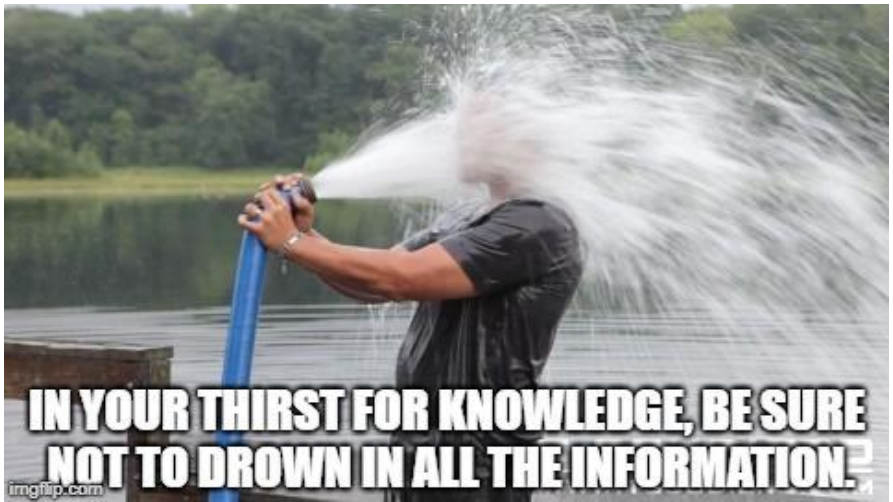
- Cisco, Amazon, Factset, Mozilla (Firefox), Signal Sciences (now Fastly)
- Hiring Manager, Triage Lead, interviewed 100+ job candidates

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

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

Who I Think You Are:

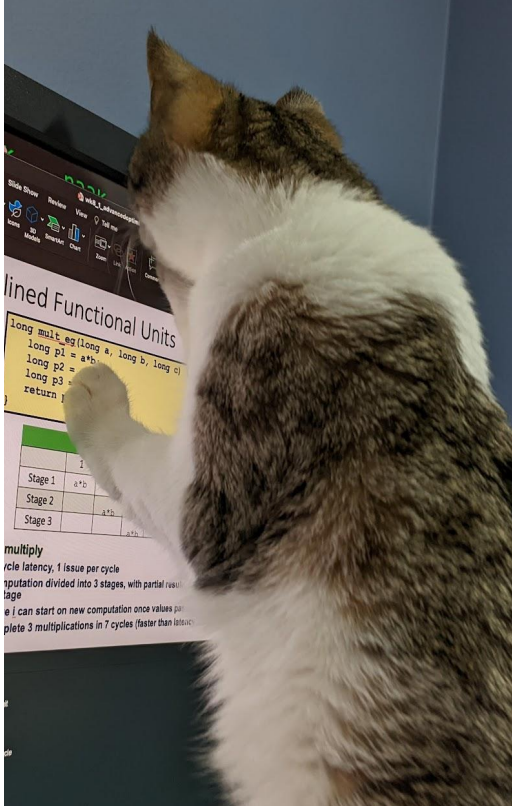
- You’re a PSU CS student interested in cybersecurity and/or development
  - We hope you’ll think about joining the security club
- You’ve at least heard the term fuzzing, but you might not know what it is



Credit: unknown.

How I hope you'll benefit from this talk:

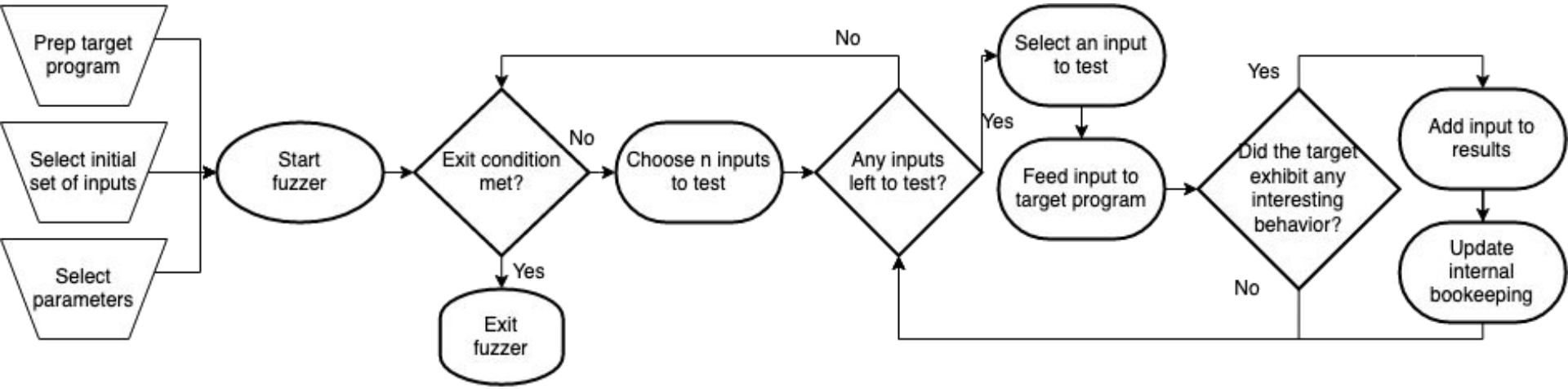
1. Learn something
2. Frame your thinking about fuzzers
3. Understand that fuzzing is still more craft than science.
4. Understand that when dealing with fuzzing output as a developer, patience is a requirement not a virtue.



Requires further study

Fuzzing & its Origins

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)

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: Students publish their final project, the 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 (especially in the OS community)…
  - Dr. Miller is told it sucks so much he should leave CS
  - but the Vulnerability Researchers (Offensive security) see the potential…
    - https://www.cs.wisc.edu/2021/01/14/the-trials-and-tribulations-of-academic-publishing-and-fuzz-testing/
    - The Relevance of Classic Fuzz Testing: Have We Solved This One? (30 year retrospective)
- 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?
  - Or maybe we just like making jokes about bunnies, cats, and other fluffy things

Fuzzing Strengths & Weaknesses

1 bug per 100 lines of code*?! Yikes! Automation to the Rescue?

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

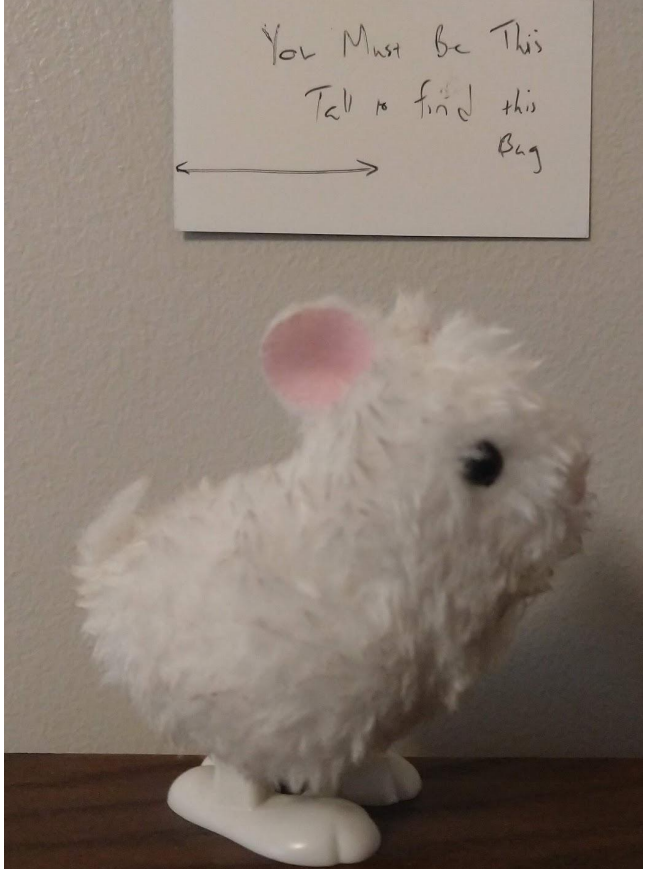
*"For a newly added project, there is usually an initial burst of new reports at 2-3 bugs per day. However, after that initial burst, and after weeding out most of the existing bugs, we still get **a constant rate of 3-4 bug reports per week**."*

https://mboehme.github.io/paper/CCS21.pdf

*https://edu.google.com/intl/ALL_us/why-google/customer-stories/portland-state-cybersecurity-cloud/

Fuzzing Limitations

- Fuzzee has to (frequently) accept input
  - not always easy
- Hard on your machine
  - CPU intensive, RAM really intensive
  - Consumes a lot of power and $$
- Probability is not always your friend
  - Tight branches; i.e. “if x == 1337”
  - Dependent state (checksums) Deep state (hoffman tree)
- Not good for truly dangerous code
  - If you’re afraid to run it, that’s a problem
- Fuzzing does not know the ‘why’ (implicit oracle)
  - Often a very bad signal to noise ratio
  - Understanding the results is resource expensive
- Relies on approximations and assumptions
  - “Absence of evidence is not evidence of absence”



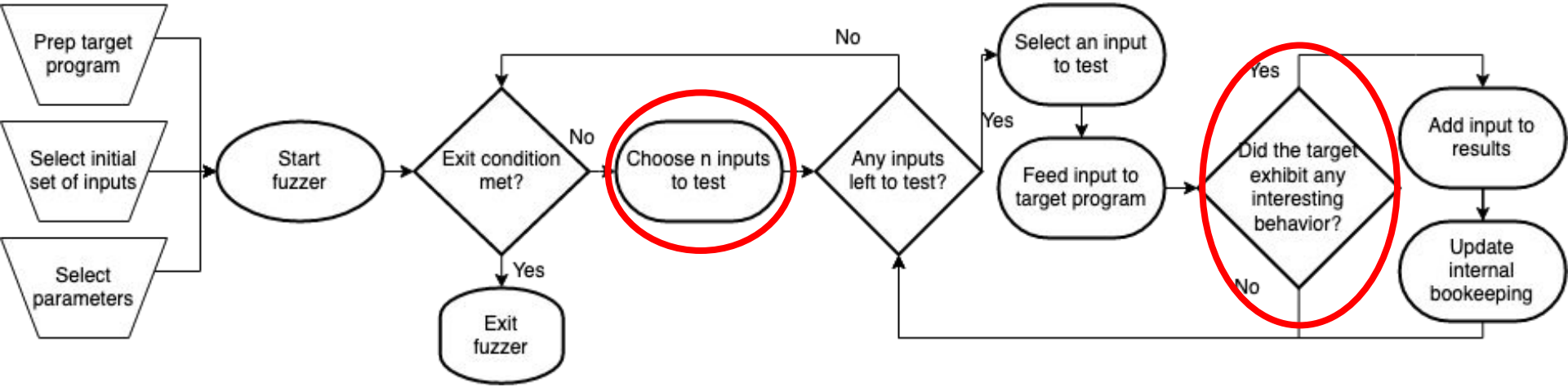
“You must be this tall to find this bug”

Fuzzer Taxonomy

Key Features of Fuzzer Taxonomy

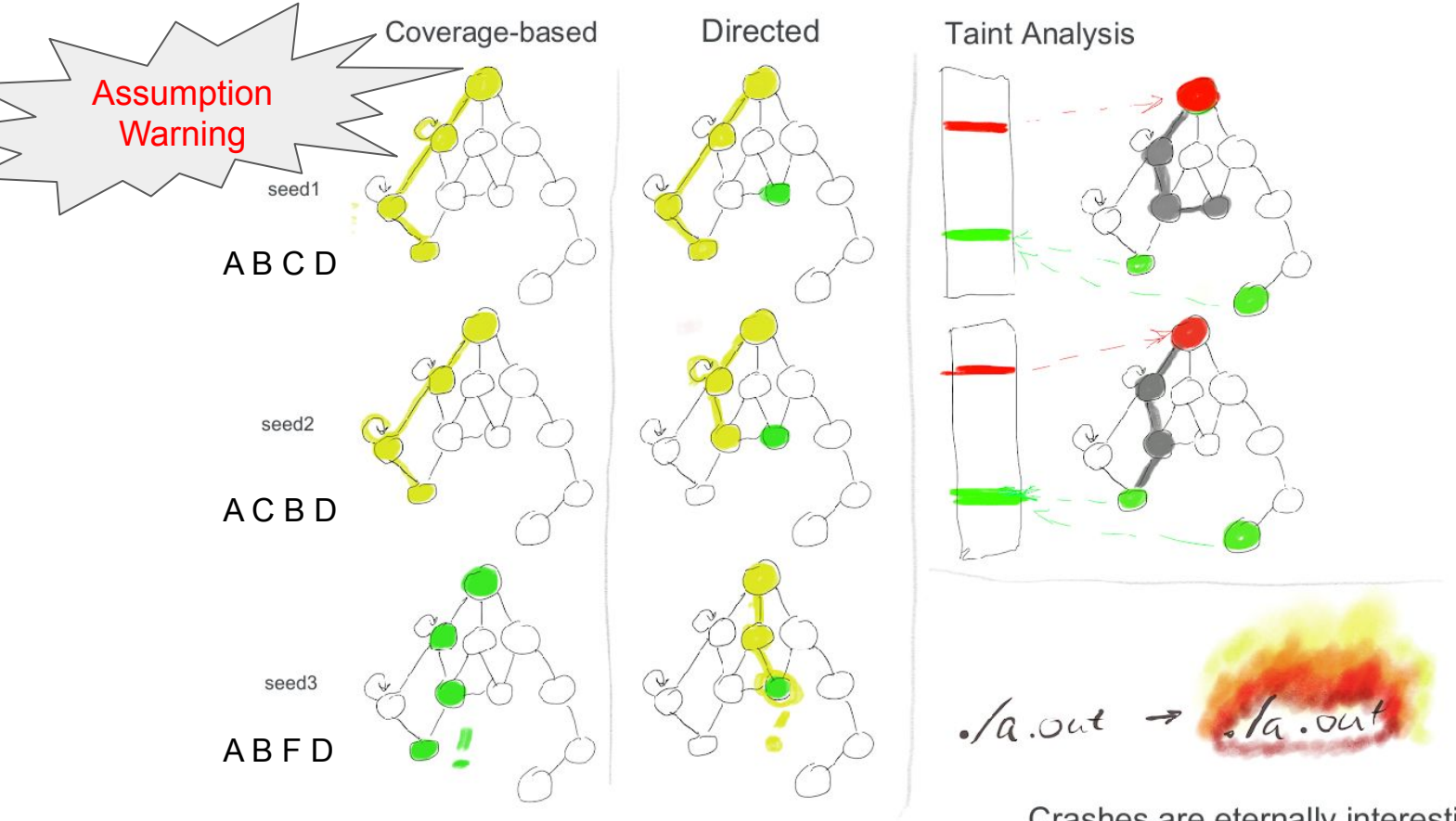
1. What is interesting behavior to this fuzzer?
2. How does the fuzzer get new inputs (seeds)?
3. What is the exit condition?
4. How much does the fuzzer know about the fuzzee source?
5. Does the fuzzer have a speciality, focus, or magic feature?

1 - What is interesting behavior?



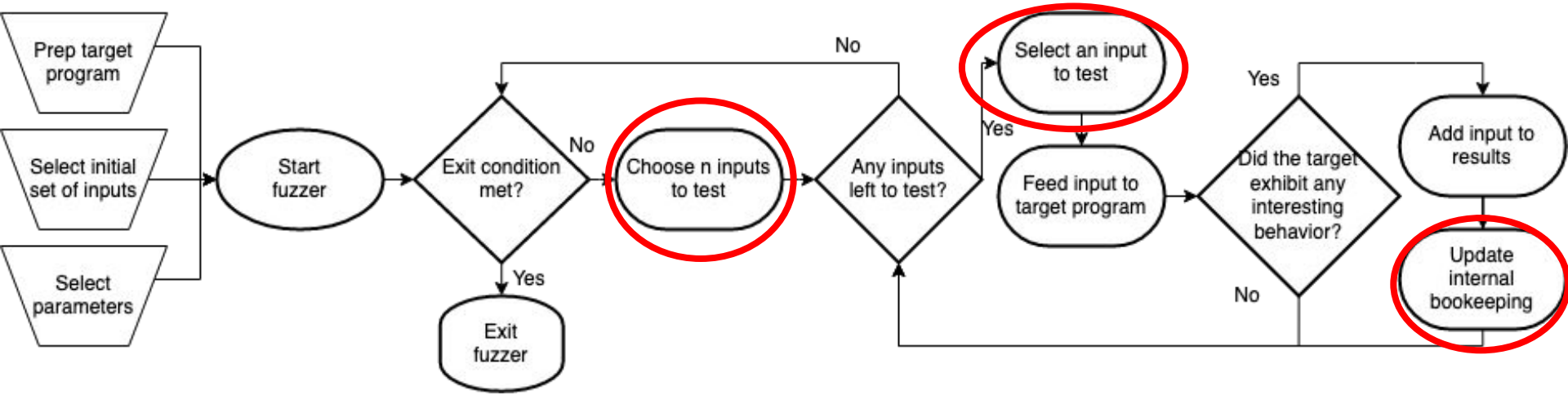
- What is the property you are searching for?
- How will you know when you find it?
- How will it influence what else gets tested?

1 - What is interesting behavior?



Crashes are eternally interesting...

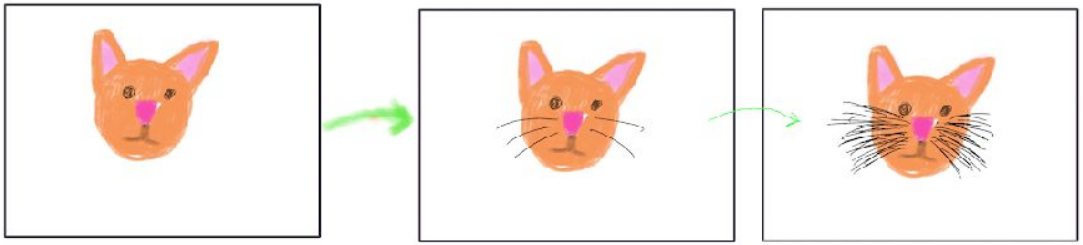
2 - Where do new seeds (inputs) come from ?



- Is there a hierarchy of “interesting”?
  - E.g. what is interesting if a crash did not happen?
- How will you prioritize what gets tested next?
- How will you make or find new inputs?

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

Starting Seed



Mutation

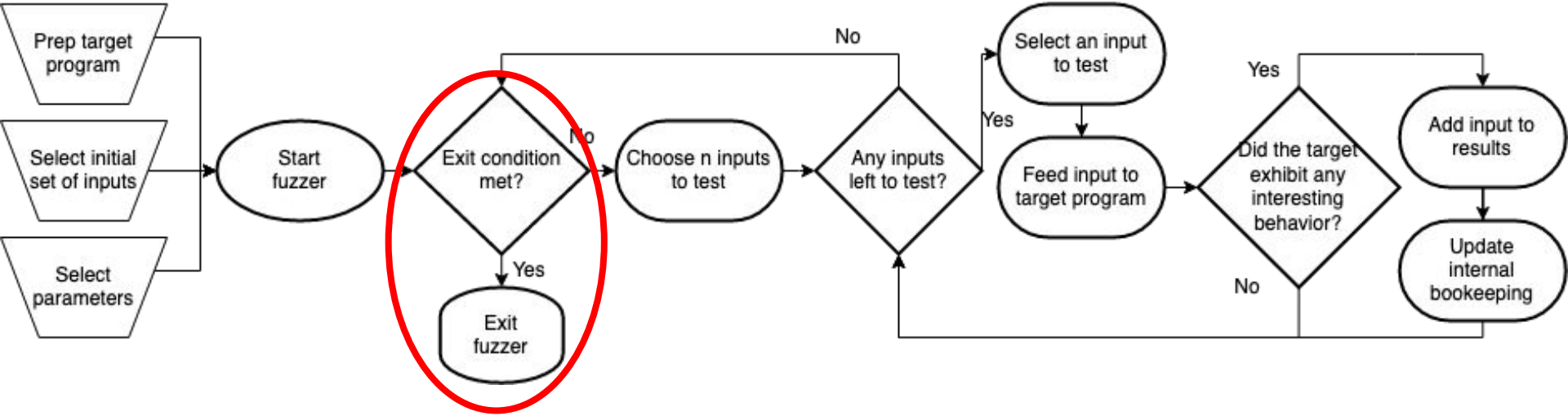
Starting Algorithm

Generation

Stdin works

Classic, Naive

3 - What's the exit condition?



- There can be more than one

3 - What's the exit condition?

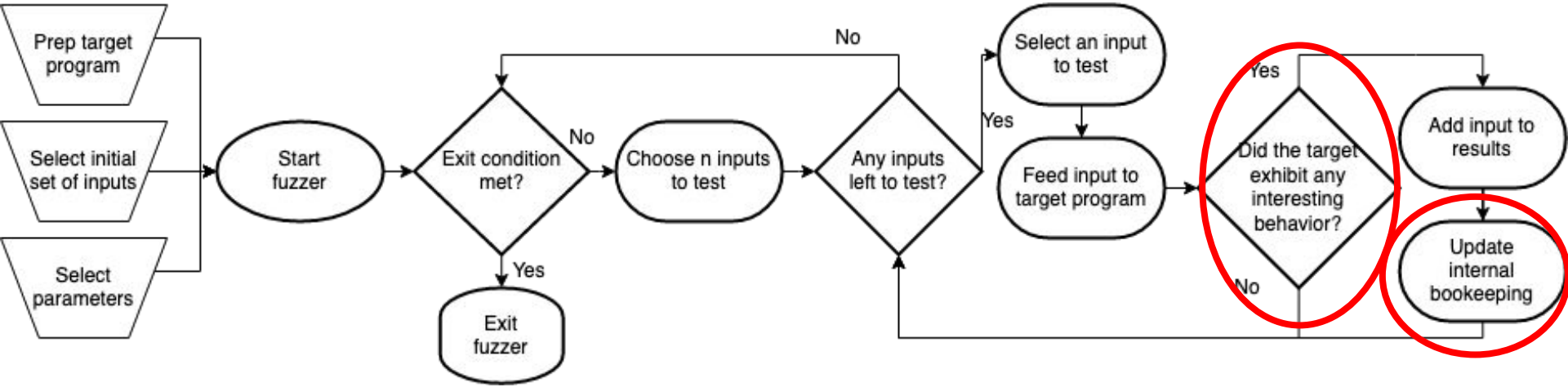
Scheduled Duration

Out of inputs to test

User Initated Exit

One and Done

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



- Tightly coupled to the definition of interesting
- How much you know about the fuzzee limits detection of interesting behavior

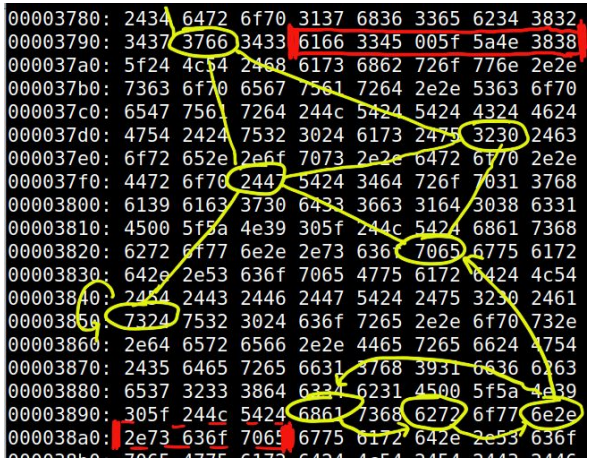
4 - What does the fuzzer know about the fuzzee source?

```
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

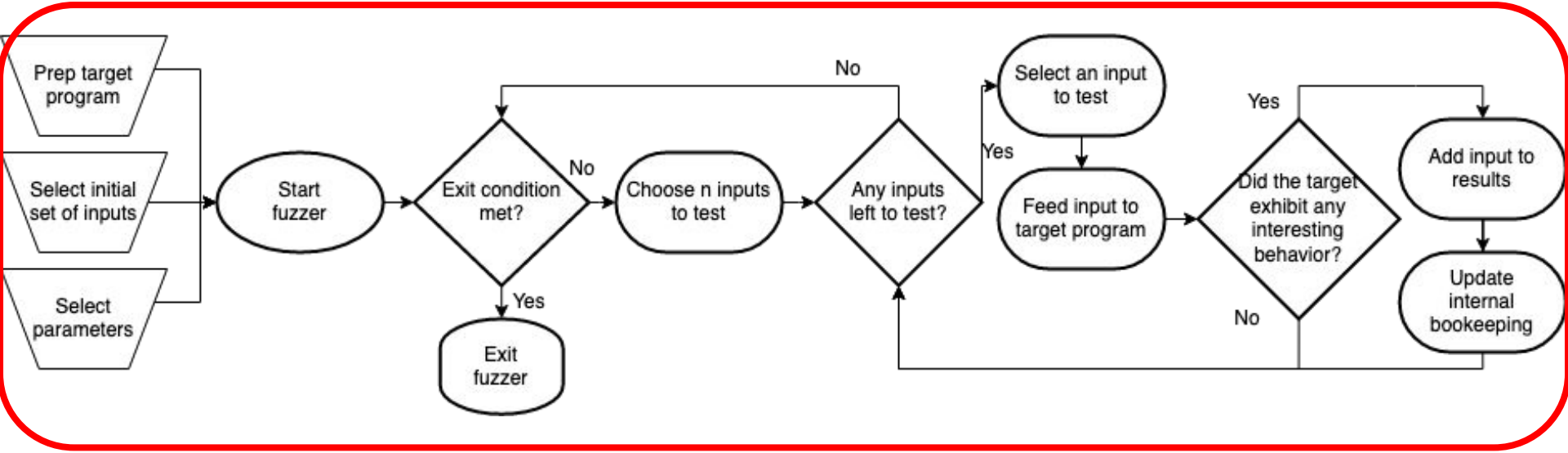
```
00003780: 2434 6472 6f70 3137 6836 3365 6234 3832
00003790: 3437 3766 3433 6166 3345 005f 5a4e 3838
000037a0: 5f24 4c54 2468 6173 6862 726f 776e 2e2e
000037b0: 7363 6f70 6567 7561 7264 2e2e 5363 6f70
000037c0: 6547 7561 7264 244c 5424 5424 4324 4624
000037d0: 4754 2424 7532 3024 6173 2475 3230 2463
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?



- Any or all components are fair game
- Can tie into any of the other taxonomy questions

5 - What's the focus or speciality?

boofuzz

syzkaller - kernel fuzzer

HTML

tux, html5 from wikimedia.org

The 5 questions, applied to AFL/AFL++ (intro to malware)

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*
    b. *Lots of settings to tune*
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”*

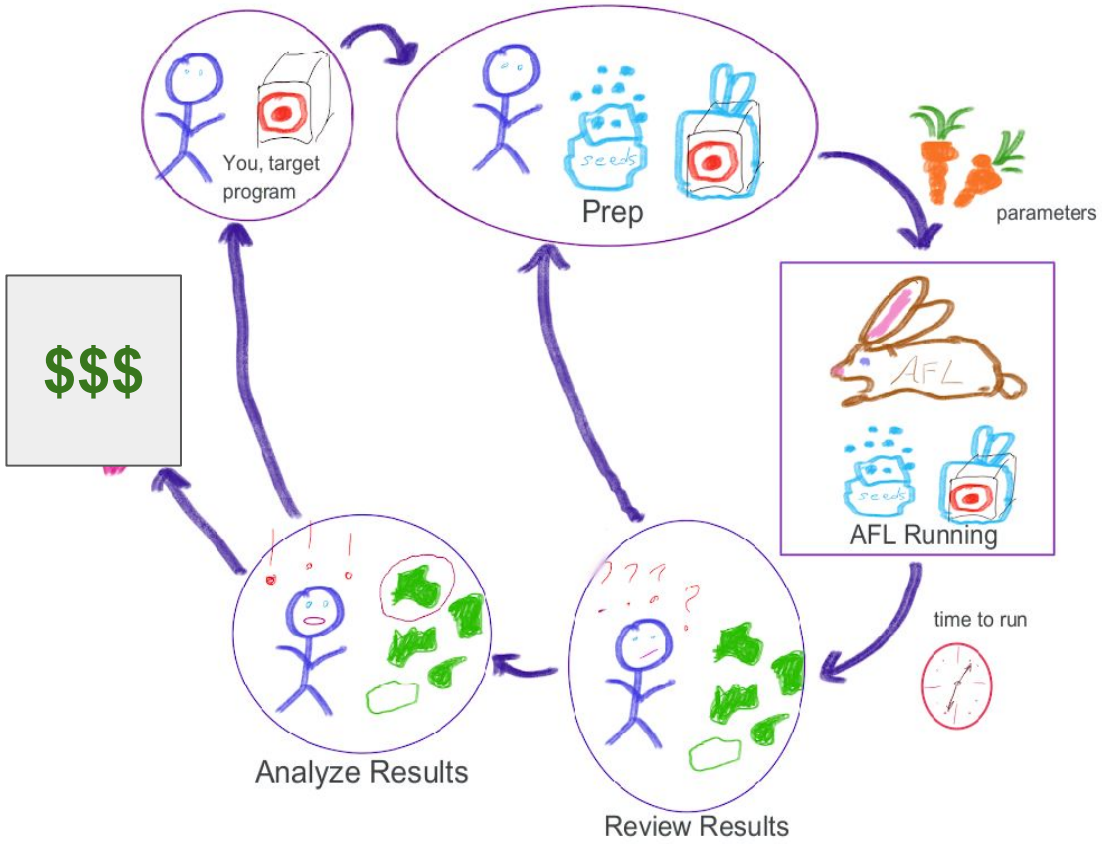
The 5 questions, applied to boofuzz

1. What is interesting behavior to this fuzzer?
   a. *Internet protocols*
   b. *Violations of the protocol you define*
2. How does the fuzzer get new inputs (seeds)?
   a. *Generative:* https://groups.google.com/g/boofuzz/c/48aaGjzbSWk?pli=1
3. What is the exit condition?
   a. *Programmable timer*
4. How much does the fuzzer know about the fuzzee source?
   a. *Whitebox network fuzzer*
   b. *“Health of session”*
5. Does the fuzzer have a speciality, focus, magic pixie dust?
   a. *Network Protocol Fuzzing for Humans, successor of Sulley fuzzer*
   b. *Written in python*
   c. *Support for arbitrary communications mediums.*
   d. *Built-in support for serial fuzzing, ethernet- and IP-layer, UDP broadcast.*
   e. *https://github.com/jtpereyda/boofuzz*

Groups Using Fuzzing

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: Attack, **report,** tune fuzzer, etc
7. Repeat



Biggest Faction at 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
- 0fg about ‘why’. Don’t owe victim a ‘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+mods
- Blackbox fuzzers
- MOpt-AFL
- Your own secret fuzzer

Understanding Freelance VR: Not Free

OSS Contributors:

Infosec Freelancers:

Credit: not sure where I got this one. reddit?

Fuzzing Party: Defensive Security

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

Your goals are…

- Find all the **vulnerabilities**
  - **bugs** are a dev problem
- Find them all before release
- Care about ‘why’
  - Have to prioritize & triage

Features to look for...

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

Fuzzers you might like…

- AFLplusplus
- honggfuzz
- AFLGo: Directed Greybox Fuzzing

Fuzzing Party: Software Owners (or Developers)

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…

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

Fuzzing Party: Academic Researchers

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

Your goals might be…

- Goals from prior group
- Studying Software Testing
- Studying another aspect of CS (parsers, synthesis)

Features to look for...

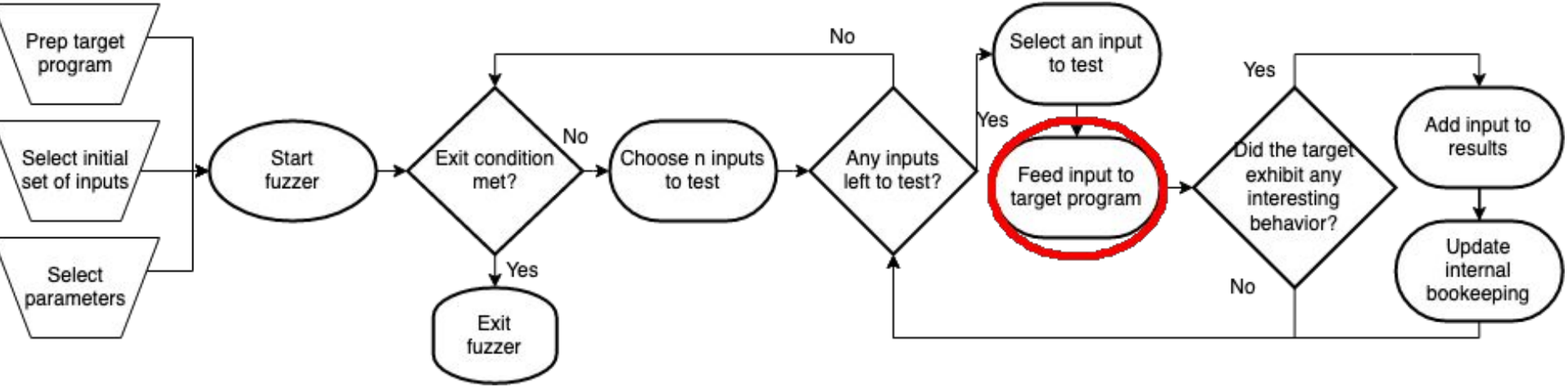
- Known Benchmarks
- Good substrate for testing new ideas
- Publishable
  - Lends itself to easy metrics
- Citable

Fuzzers you might like…

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

Challenges for Developers

Integration is hard: Drivers are Nontrivial



- Drivers (harnesses) ‘translate’ between fuzzer output and fuzzee input
- Can be an insane amount of engineering work to write
- Requires expertise in both fuzzer and fuzzee
- You might need a lot of them

Optimizing Fuzzer performance, Generally

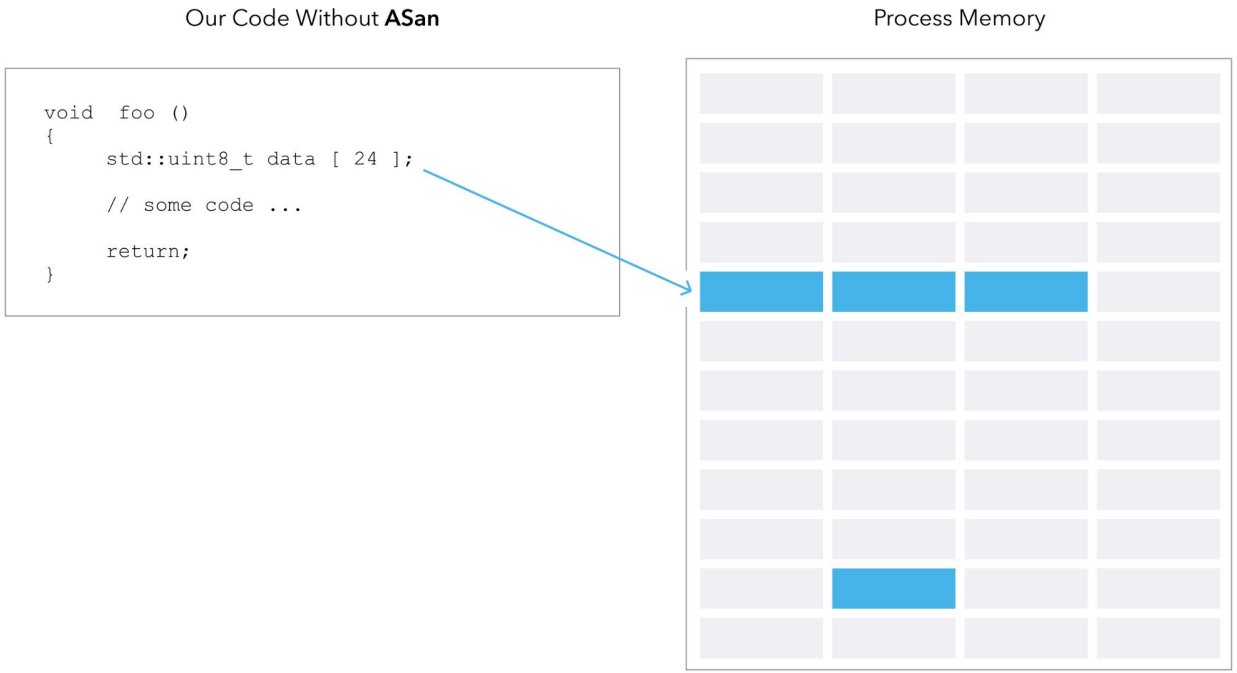
- Harness (drivers) matter
  - Help the fuzzer help you
  - Don’t skimp
  - You may need more than one.
- Spend time on the parts that make the fuzzer tick
- If nothing else, focus on what powers exploration
  - Generational? Focus on nailing all parts of a the generative algo (e.g. grammar)
  - Mutational? Focus on what gets mutated
  - Does it have special features that can help you? Use ‘em! (wordlists, dictionaries, etc)
- Makes notes on what tuning works with what fuzzers
  - Learn from past experience
- 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**
    - **This probably means running the fuzzer multiple times with different ones**

Wait, this is an intro talk!

What is sanitization ?

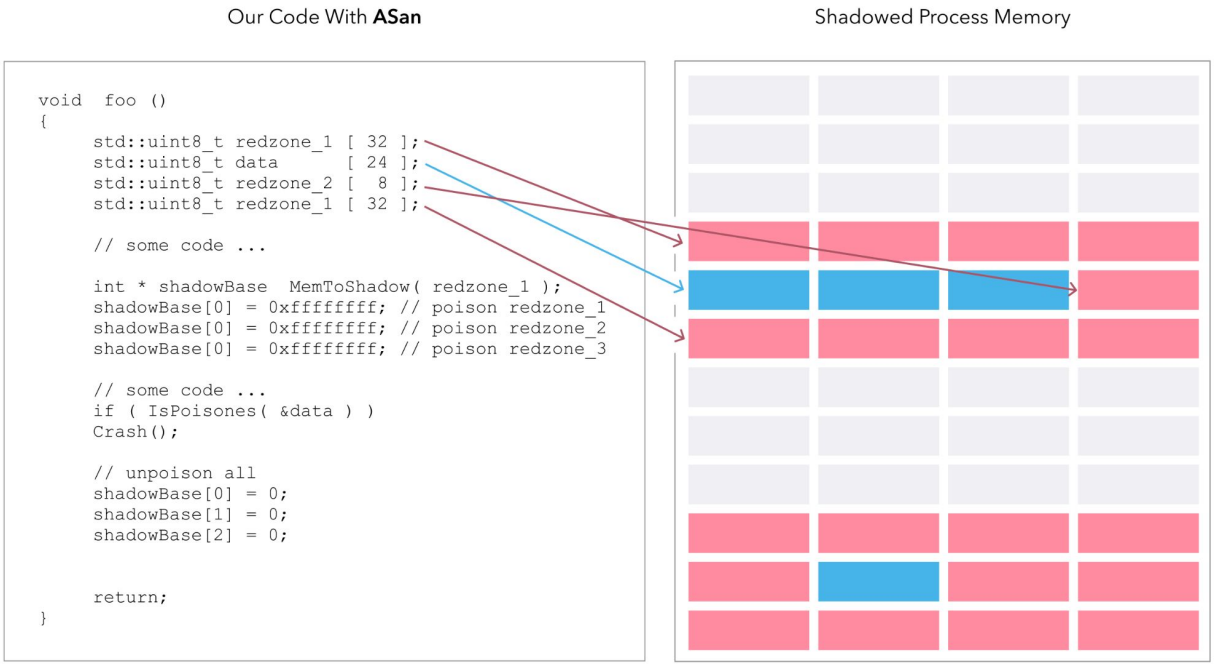
https://dilbert.fandom.com/wiki/The_Garbageman

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



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

Beware Snake Oil Salesmen

- Fuzzing does not replace your security team(s)
  - It does not replace regular tests
  - It does not replace expert code review by security team
  - **fuzzing does not replace development or security best practices in any way shape or form**
- Fuzzing is not a quick win
  - Probably Fuzz for at least 24 hours, if not a week
  - Your adversaries are fuzzing for weeks, so you probably should too
  - Most fuzzers do not fit well in an CI/CD pipeline on every commit
- Fuzzing is not Free (but likely cheaper than the alternative)
  - In a cloud system this can really add up. Electricity. servers cost $$
  - User time to analyze the results is usually expensive
- **Indecipherable bug reports are ignored reports**

Giulio Carrara
2y

Dear recruiters,
if you are looking for:

- Java, Python, PHP
- React, Angular
- PostgreSQL, Redis, MongoDB
- AWS, S3, EC2, ECS, EKS
- *nix system administration
- Git and CI with TDD
- Docker, Kubernetes

That's not a Full Stack Developer.
That's an entire IT department.

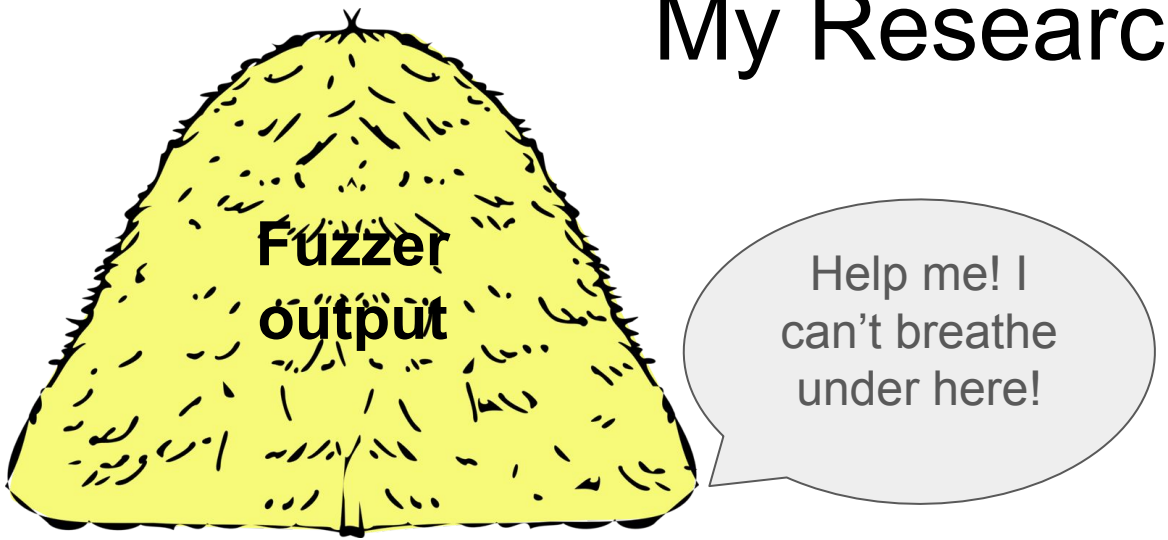
Yours truly

You can't do this in security either. Don't try.
Credit: ancient depths of my harddrive

Challenges Include but are not limited to..

- Integrating a fuzzer can be a lot of work
- Long runtimes. Don’t easily fit into CI/CD pipelines
- Expensive at corporate scale
- Optimization of one can be a lifetime’s work
- Requires time to triage, analysis, & root cause
  - Most fuzzers have no idea why something breaks
  - To get bugs fixed, they need to be understood first
  - Indecipherable reports are ignored reports
- But that’s not the worst one!

My Research



Buried under the Haystack & other problems

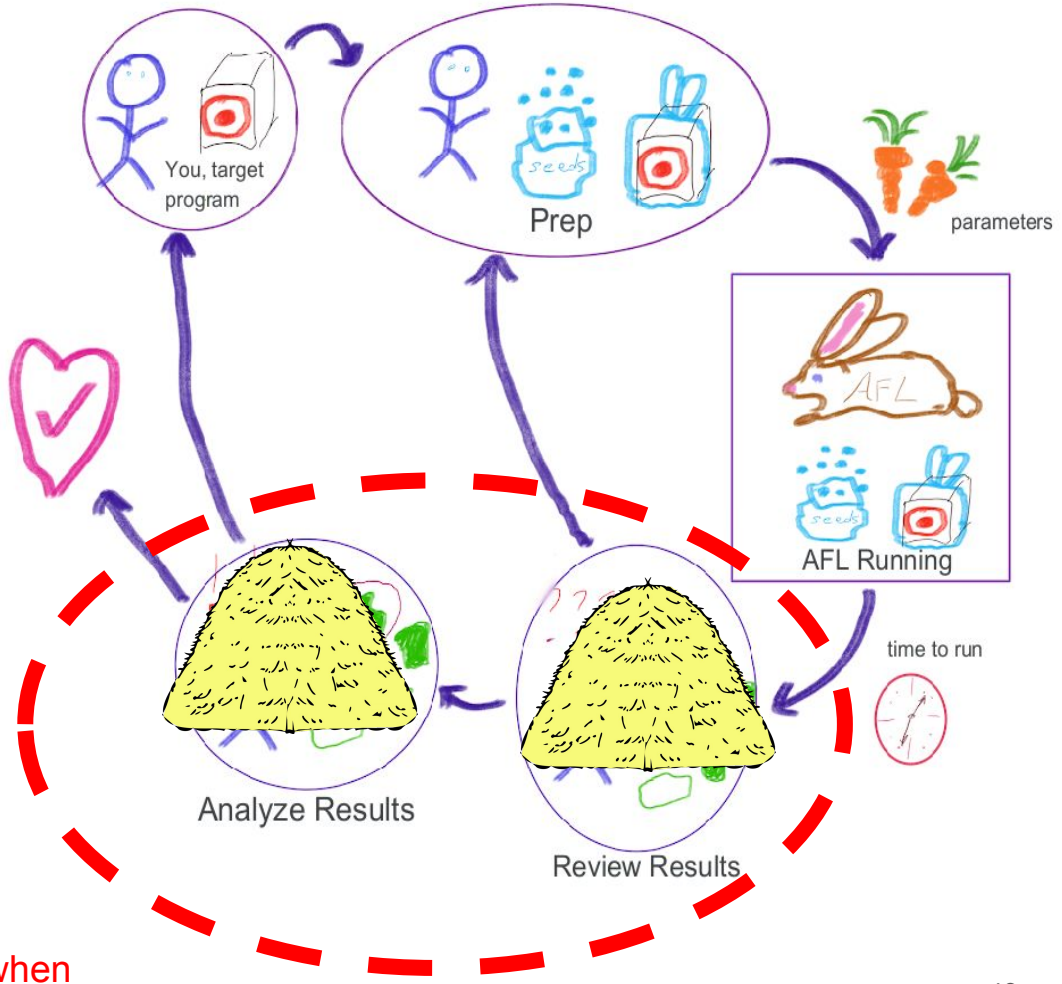
"Developers generally appreciate bug reports, but they can sometimes be a bit **less enthusiastic about a flood of reports from automated fuzzing** systems." https://lwn.net/Articles/904293/

"..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"

"for a large class of [fuzz] reports the stack trace does not contain the relevant information" https://lwn.net/Articles/917762/

If a fuzzer produced 3,020 crashes, how many bugs do you think there were?

15 root causes. There is pain and suffering when you have to worry about *why* it broke



```c
#include <stdlib.h>
#include <stdio.h>
#define MAX_SIZE 80 /*1024*/
int foo (int n) { return (n*n)+1;}
int foo2 (int n) {return 2*n;}

int main() {
   int* x = (int*) malloc(sizeof(int));
   *x = 3;
   char* input = (char*) malloc(MAX_SIZE * sizeof(char));
   printf("enter some input:");
   fgets(input, MAX_SIZE, stdin);
   printf("You entered %s. Hope it doesn't have a problem!", input);
   int flag = (char) input[1];
   if (!(flag % 2)) { free (input);}
   else { free (input); }
   input[0] = foo2(foo(input[0]));
   if (!(flag % 3)) { free (input); }
```

Double Free of var input

Would you rather look at 1000s of these…

Or (maybe) worse, miss it entirely if you didn't have a sanitizer running…

Or a few of these…..

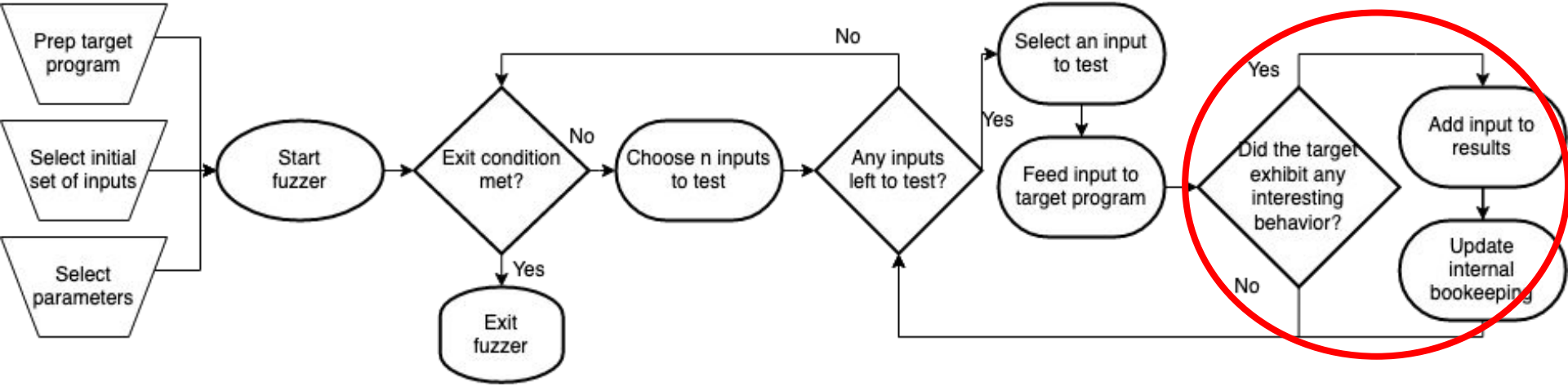
```
--------------------------------
Output Summary

Unique Sec Policy Failures: 3
Total (nonunique) Failures: 2633
Unique Standard Crashes   : 0
Total (nonunique) Crashes : 0
Unique Hangs              : 0
Total (nonunique) Hangs   : 0
Total Saved Testcases     : 1259
Total Testcases Executed  : 5765
Results saved in base dir : output-cc-multi/0118_175124
--------------------------------
PIPE Security Policy Violations Details
        For each violation, only the first detected test case is saved.
        Total records tagged in storage: 3

Problem Root-cause:
        Policy Violated: DoubleFree.
        Failed Rule: FreeT detects two frees.
        Memory first freed at location …/tests/foo.c:81 was freed again at location
…/tests/foo.c:83
TC Filename/ID : 3
TC buffer size : 4
Testcase/Input : 22p
```

[Target file had 3 unique DFs. Other two omitted for slide space]

What if something could help with the ‘why’ and the Haystack?!

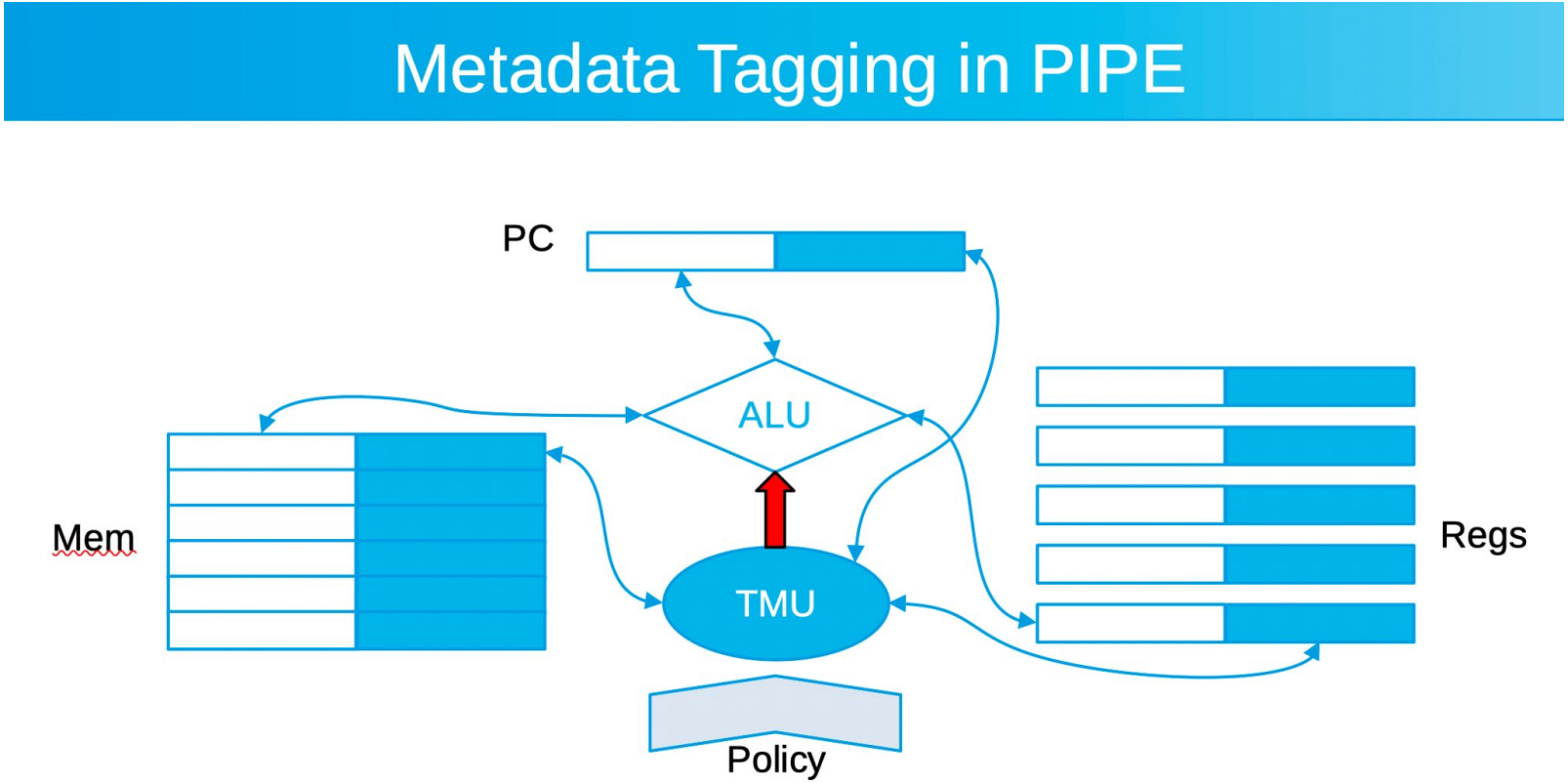


- Root-cause proxy: What if there was a system that could tell the fuzzer why a input failed/crashed/faulted?
  - Bonus: you could look for faults that don’t crash processes?
- Deduplication: What if the fuzzer could use that information to only keep needles?
  - instead of haystack + needles?
- Customization: Definition of “Interesting” varies, and is a key feature of fuzzer taxonomy
  - What if you could write your own without writing a whole new fuzzer?
  - Bonus: tailor the fuzzer to find the types of bug/vulns you’re interested without inconveniencing other groups

Runtime Reference Monitor, PIPE, & Tagged C

- Runtime Reference Monitor: Essentially the security referee of a system, monitors processes for misbehavior.
  - Well established in security literature, comes in many flavors
- PIPE (Programmable Interlocks for Policy Enforcement) Is a Tag-based hardware security reference monitor intended to be implemented as an ISA extension
  - Cousins include CHERI, MTG
  - Tags are checked at points
- Developers rarely speak hardware; most don’t speak assembly either
- Tagged C: PIPE implementation that developers put in and write rules at source level

Possible Future: PIPE Implementation in Hardware



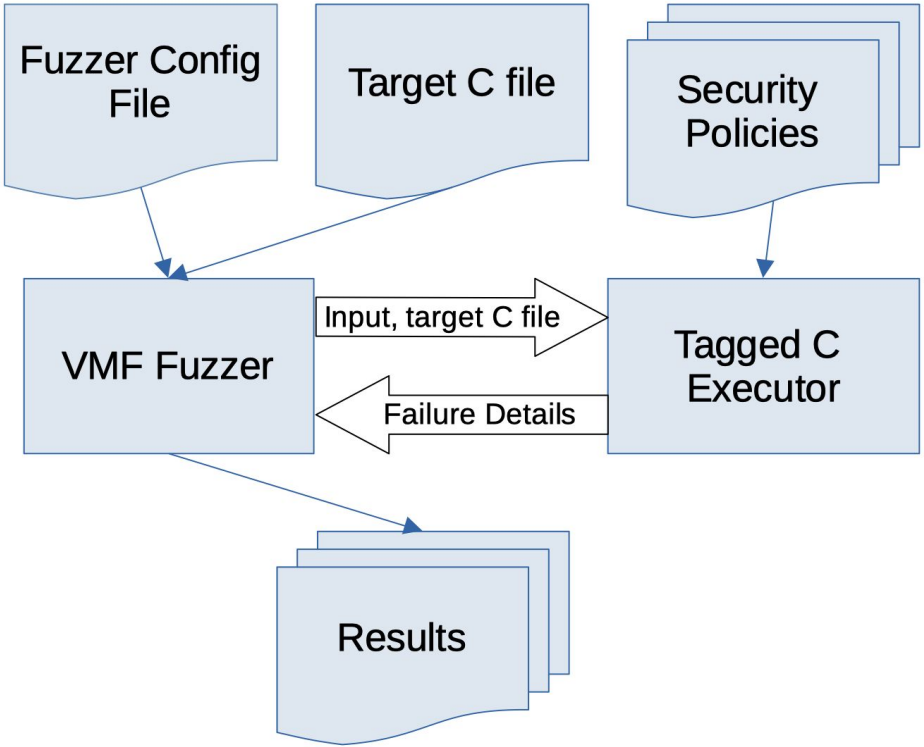
My research is in the software side, but someday…

img credit: Sean Anderson

The Dissertation Fuzzer: PIPE-Cleaner

- Next version will be compiled, not interpreted, code
- Tagged C Policies designed for detection and diagnostics
  - Don’t just find the bug, tell the fuzzer what it is
  - A “Personal Sanitizer” or “Personal Bug Oracle” without having to write a new compiler pass
    - change what to fuzz for without changing the fuzzer
  - Policies can better represent things like SQL injection that don’t make sense as segfaults
- Fuzzer searches for Policy Violations rather than crashes
  - Fuzzer has root cause at time of fault, segfault doesn’t
- Ideally, first fuzzer to have no false positives by construction*
  - Imagine instead of 3,020 crash reports, you only have to read 15 reports that tell you what happened?
  - Instead of consigning fuzzer report to oblivion or summer interns, bugs might get fixed
  - Focus resources on only the bug classes you want to find

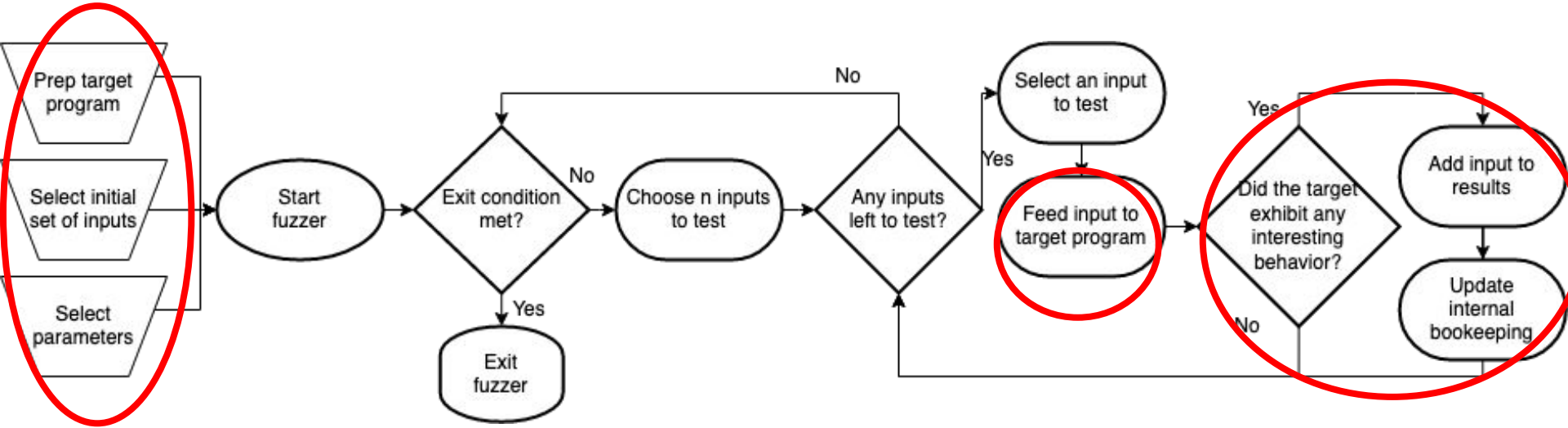
PIPE-Cleaner Tagged C Interpreter Prototype System



https://arxiv.org/abs/2411.00261

[IMO] the Future of Fuzzing

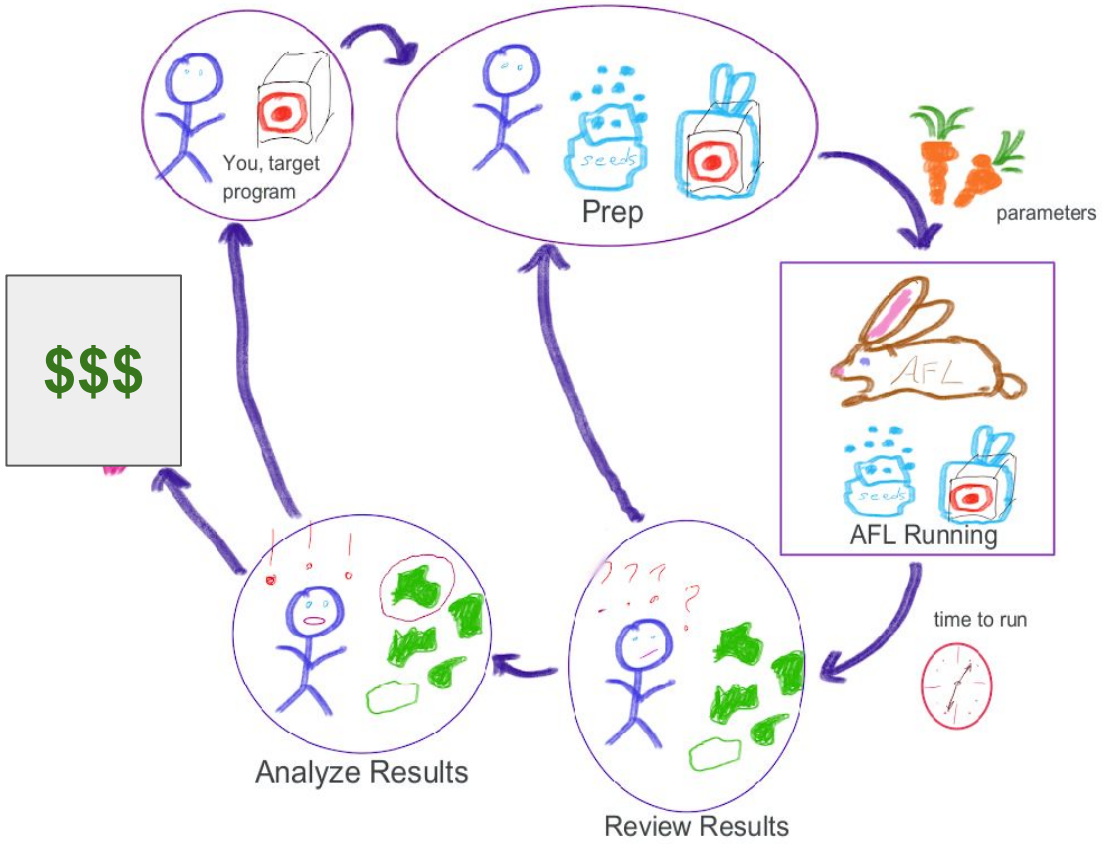
Future Fuzzing Directions



- Spectrum between Property Based Testing & Fuzzing
- Hybrid fuzzing - sym exec + fuzzing
- Bug Oracles

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
7. Repeat



Thanks for Listening <3

- Classes that teach fuzzing @ PSU
  - Malware expected Spring 2025
  - Intro to Network Security, expected ??
  - Independent study (By arrangement)
- PSU Security Club General Meetings
  - Fall 2024: Tuesdays 1-2pm
  - Join our discord
- PSU Security Club Winter Social:
  - Tuesday Dec 3th, 1-2pm
  - See an officer or the discord for more info
- Social Lunch at the library food carts after this talk
  - Brains need fuel

Hacker Cat