+99999 Performance Point!

Ways to Bench ✔ Fuzzers

...

@AlligatorCon 2024
By: 0xMard

whoami

- 🇰🇷 Half-drunk Security Researcher

whoami

- 🇰🇷 Half-drunk Security Researcher
- Blog.mard.kr / 0xMard

whoami

- 🇰🇷 Half-drunk Security Researcher
- Blog.mard.kr / 0xMard
- Researching hardware, hypervisors, fuzzers, CPU, algorithms @ Mardlab

whoami

- 🇰🇷 Half-drunk Security Researcher
- Blog.mard.kr / 0xMard
- Researching hardware, hypervisors, fuzzers, CPU, algorithms @ Mardlab
- Thanks @Gamozolabs for help!

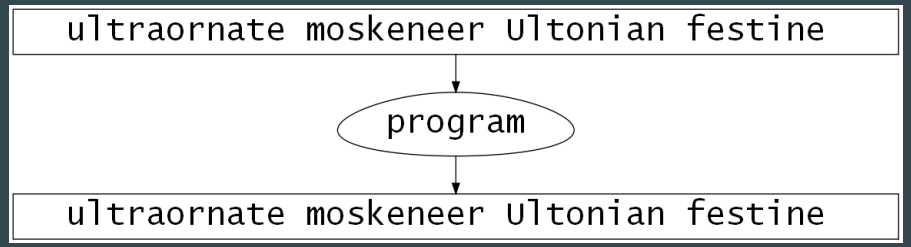
Table of Contents

Table of Contents

Fuzzing Basics

What is Fuzzing?

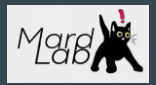
- Software testing technique for finding software bugs
- Simple and popular way to find security bugs
- Generate and mutate the inputs for a program
- Having an intention to break the program
  - Program crash
  - Errors
  - Hangs



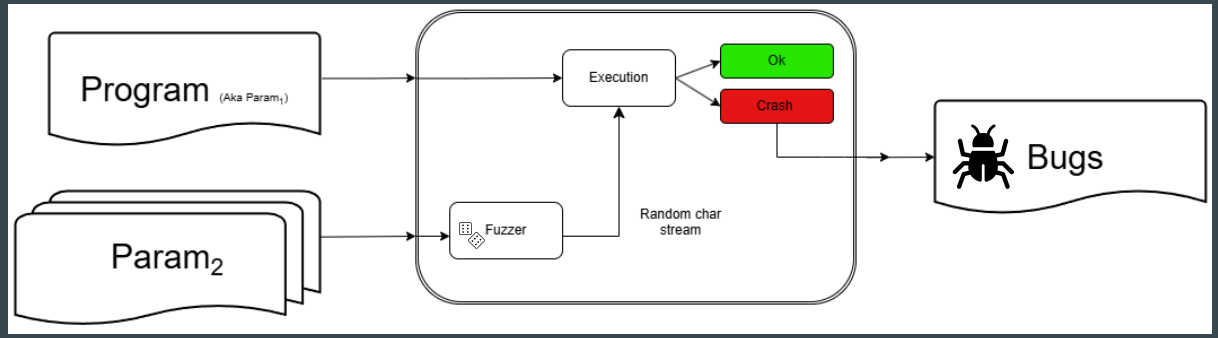
Key Terms

- **Seed Corpus**
  - Set of valid and interesting inputs that serve as starting points.
  - It generates maximal code coverage
  - If it’s not provided, fuzzer would have to guess these inputs from scratch, which can take an indefinite amount of time.

- **Coverage**
  - Gather which code has been executed based on an input.
  - That input is saved when a unique path is observed.
  - Input is used as a basis for future input



Basic Fuzzer Architecture



If Program ($Param_1$) executes, and gives us "Ok result":

- Program is running fine

If Program executes with the fuzzed data and we get a crash:

- Bugs
- An issue with the fuzzing setup

$Param_2$ is basically random based on corpus.

- Sequence of ASCII characters
- Sequence of words, separators, and white space

Fuzzers are Not an “Overkill tool”

- **If there is no crash, fuzzers usually can not find the bug.**
  - Arbitrary file read
  - Privilege Escalation
  - Auth bypass (if it’s not set up for checking permissions for changes)
- **Generating all valid inputs is almost impossible**
- **Most fuzzers requires source code (e.g AFL++, Honggfuzz)**
  - They can perform Blackbox fuzzing using QEMU implement support

Benchmarking Fuzzers

Benchmarking

- Standard test or set of tests used to compare alternatives.
- Consists of a motivating comparison, a task sample, and a set of performance measures

Usage of Benchmark in Fuzzing

Security Researchers View:

- *“How can I compare my homemade fuzzer with popular fuzzers so I can see what I can improve?”*

Academic View:

- *“What’s the difference between your fuzzer and an existing fuzzer?”*
- *How can I prove the usefulness of new technique I made?*

Fuzzer Benchmark Overview

1. Find a baseline fuzzer to compare against (Aka. AFL)
2. A sample of target programs (benchmark suite)
3. Evaluation Metrics
4. Statistical Evaluation
5. A meaningful set of config parameters (Timeouts, Empty Seed?)
6. Enough trials to judge performance

Popular Targets to Fuzz

- **Real programs**
  - nm, objdump, cxxfilt, gif2png, ffmpeg, etc.
- **Google Test Suite**
- **LAVA-M**
- **CGC (Cyber Grand Challenge)**
- **Purposely-Vulnerable Programs**

Evaluation Metrics

Types of Evaluation Metrics

- Measure time to trigger known bugs
- Measure new code coverage
- Measure number of new manually found bugs

Types of Evaluation Metrics

- Measure time to trigger known bugs
- Measure new code coverage
- Measure number of new manually found bugs

What?? I thought Fuzzing was automatic not manual…

Option 1: Measure Number of New Manually Found Bugs

- **Pros**
  - You found a new bug! $$$

- **Cons**
  - Manual labor - Finds a single bug among thousands of unique crashes.
  - Limited effectiveness for well-fuzzed targets
  - Not suitable for academic papers – “So you spent time with your students manually looking for bugs”

Option 2. Measure Time Until Finding Known Bug

- **Pros**
  - Known architecture, and internals
  - Sanity Check

- **Cons**
  - Useless for academics
  - IMO “It doesn’t really highlight that your fuzzer is nothing different than popularized fuzzers”

- **Side Note**

  If you have considered using data sets as corpus, the data sets with known bugs are very limited (LAMA)

Option 3. Measure New Code Coverage ★

- **Pros**
  - Strong correlation with bug finding
  - Fine-grained and easily measurable

- **Cons**
  - Limitations in Blind fuzzing or symbolic approaches
  - Limited differentiation: emulators

Visualization

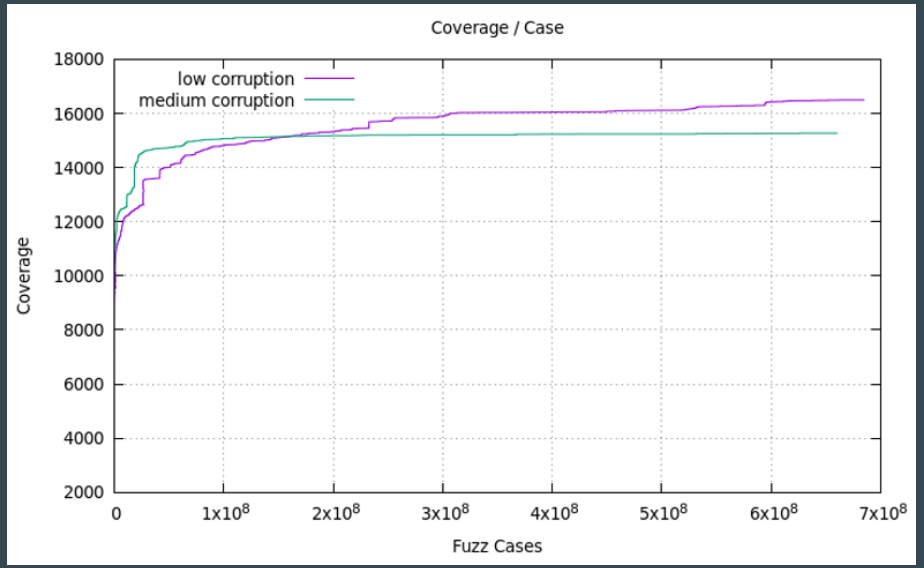
Visualization

- **Any software that can produce graphs should be fine**
  - Graphviz, Gnuplot, ggplot2, Matlab (Hardcore mode)
  - More points, more accurate your graph will be

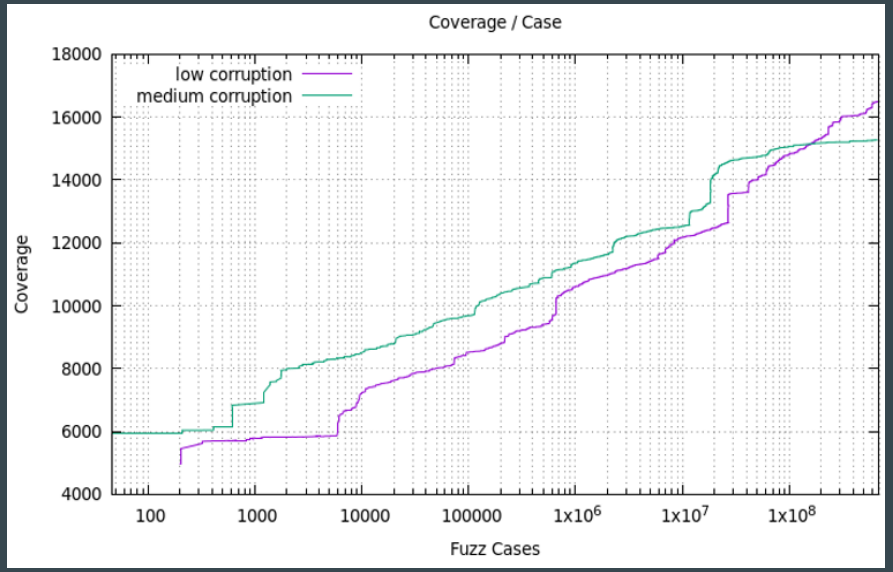
Logscale vs Linear

- Logscale is good when it comes to graphing accurately.
- Assuming the timeout for fuzzer is 24 hours, logscale is superior
- Linear is usually not used in graphing coverage/case, since we are unable to see anything about what happens in the fuzzer in the first ~20 min.
- Linear is usually used in graphing coverage/time, since
- Linear is good when the timeout is less than 24 hours.
    - Small range of values

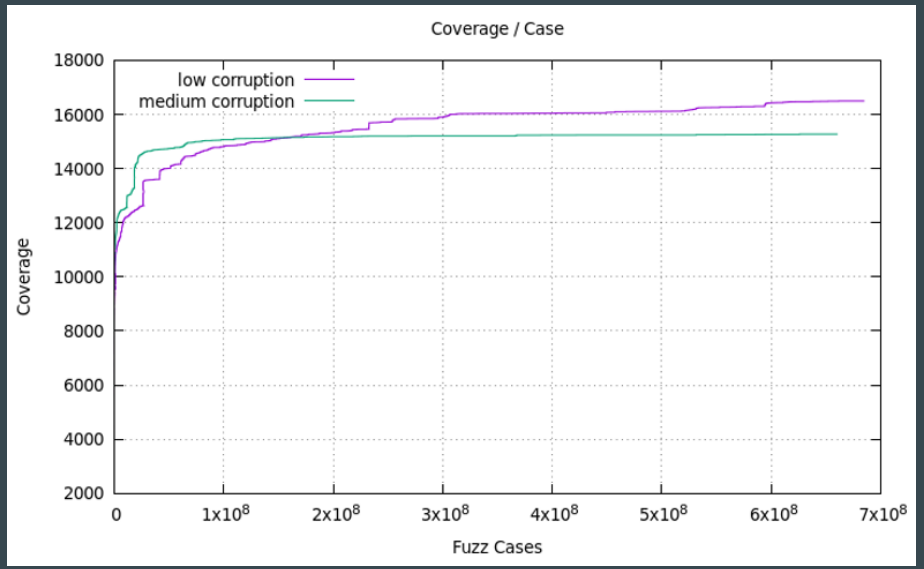
Log Scale graph



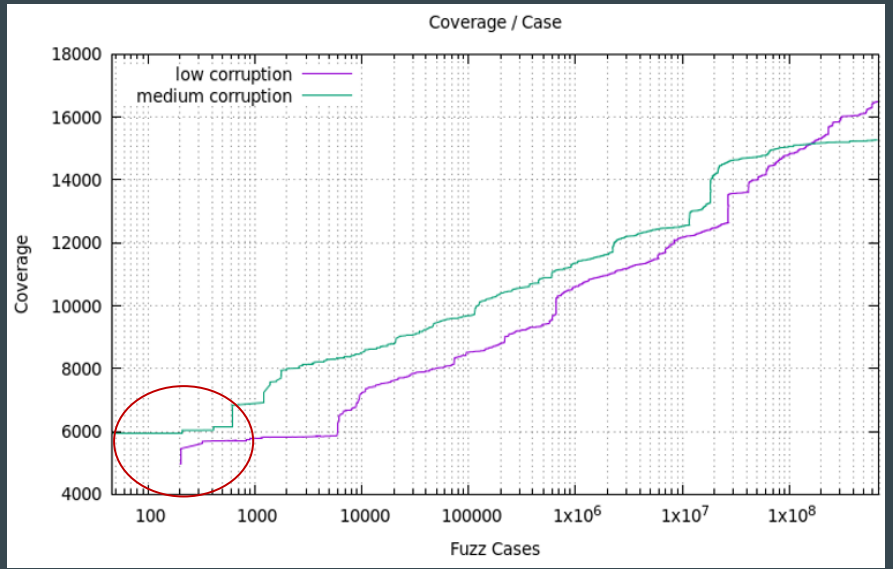
Linear graph



Log Scale graph



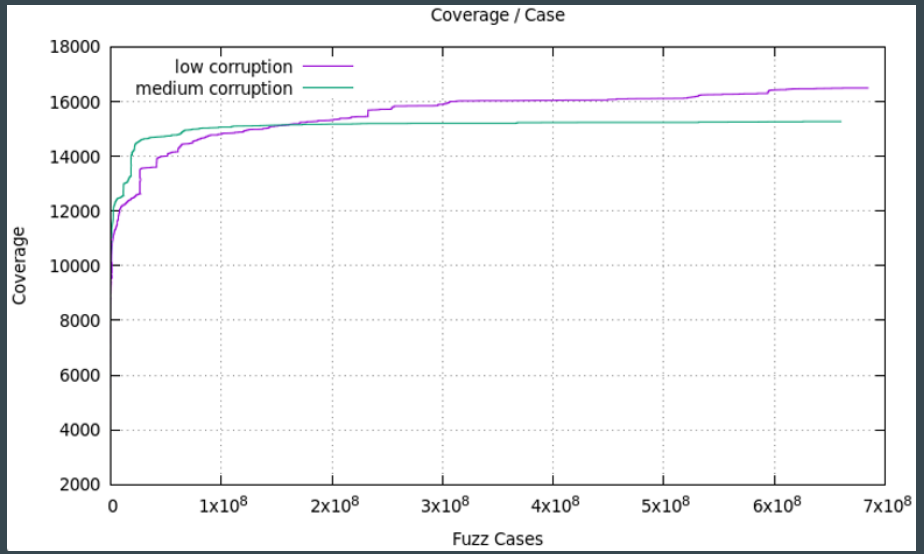
Linear graph



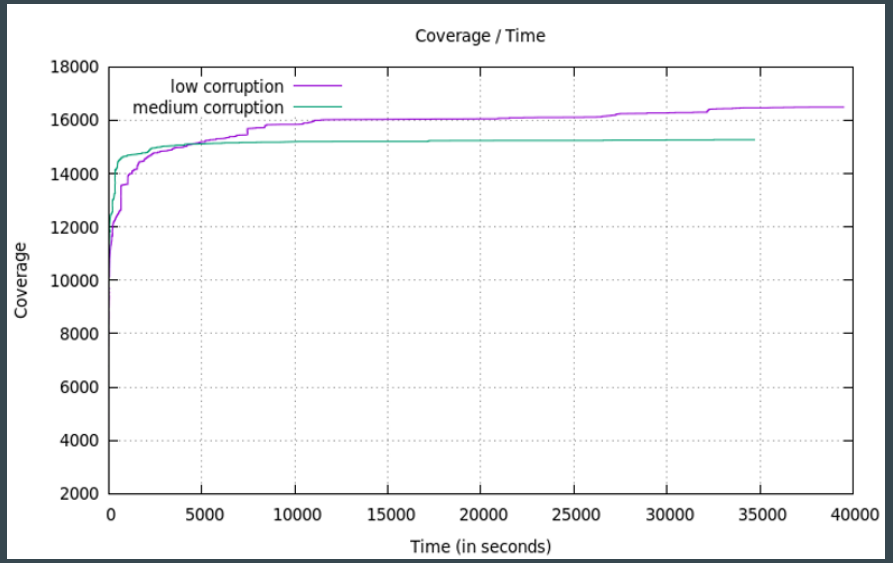
Time-based or Per-Case-Based Representation

- Depends on the use case
- Per-case-based (fuzz case) is better since it doesn't factor in the performance of the fuzzer.
  - Avoids performance bias (resource usage, optimization)
- During development, you can start to inspect the efficiency of the fuzzer in terms of quality of cases produced. (Per-case-based)
  - Code Coverage experiment
- Time-based is good when you are trying to show how fast it's running, and anything that relates time to be honest.

Coverage / Case



Coverage / Time



Statistical Evaluation

Statistical Tests

- Method to decide to accept or reject a hypothesis about a process
- Process is fuzz testing, and the hypothesis is that fuzz tester A is better than B at finding bugs in a particular program
- The confidence of our judgement is captured in the p-value

Types of Statistical Tests

- Mann-Whitney U test
- Bootstrap-based test
- Permutation test

Types of Statistical Tests

- Mann-Whitney U test
- Bootstrap-based test
- Permutation test

} Alternative, Rarely used

Types of Statistical Tests

- Mann-Whitney U test ★
- Bootstrap-based test
- Permutation test

} Alternative, Rarely used

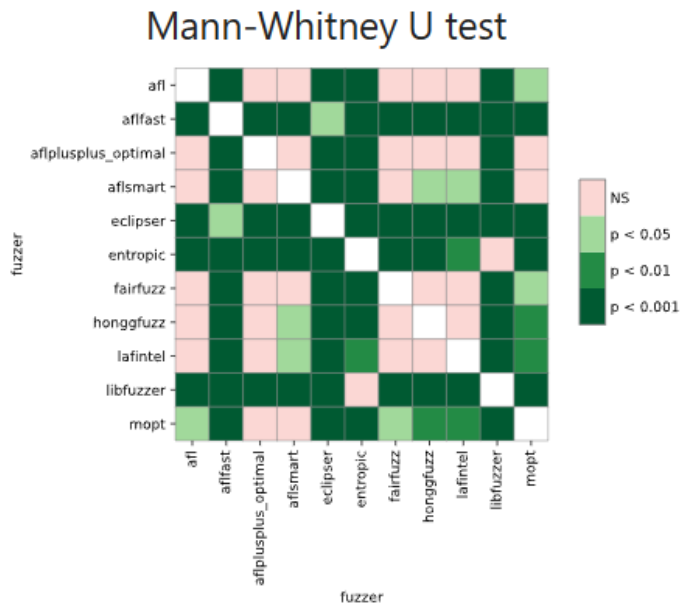
Types of Statistical Tests

- Mann-Whitney U test ★
- Bootstrap-based test
- Permutation test

} Alternative, Rarely used

Mann-Whitney U test on top!!

Mann-Whitney U test



The table summarizes the p values of pairwise Mann-Whitney U tests. Green cells indicate that the reached coverage distribution of a given fuzzer pair is significantly different.

- FuzzBench provides this graph
- This graph is overkill and complicated but If you must make this graph, use FuzzBench.

paper	benchmarks	baseline	trials	variance	crash	coverage	seed	timeout
MAYHEM[7]	R(29)				G	?	V	-
FuzzSim[44]	R(101)	B	100	C	S		R/M	10D
Dowser[18]	R(7)	O	?		O		V	8H
COVERSET[38]	R(10)	O			S, G*	?	R	12H
SYMFUZZ[8]	R(8)	A, B, Z			S		M	1H
MutaGen[23]	R(8)	R, Z			S	L	V	24H
SDF[28]	R(1)	Z, O			O		V	5D
Driller[41]	C(126)	A			G	L, E	V	24H
QuickFuzz-1[16]	R(?)		10		?		G	-
AFLFast[6]	R(6)	A	8		C, G*		E	6H, 24H
SeededFuzz[43]	R(5)	O			M	O	G, R	2H
[46]	R(2)	A, O				L, E	V	2H
AFLGo[5]	R(?)	A, O	20		S	L	V/E	8H, 24H
VUzzer[37]	C(63), L, R(10)	A			G, S, O		V	6H, 24H
SlowFuzz[35]	R(10)	O	100		-		V	
Steelix[26]	C(17), L, R(5)	A, V, O			C, G	L, E, M	V	5H
Skyfire[42]	R(4)	O			?	L, M	R, G	LONG
kAFL[39]	R(3)	O	5		C, G*		V	4D, 12D
DIFUZE[11]	R(7)	O			G*		G	5H
Orthrus[40]	G, R(2)	A, L, O	80	C	S, G*		V	>7D
Chizpurfle[22]	R(1)	O			G*		G	-
VDF[21]	R(18)				C	E	V	30D
QuickFuzz-2[17]	R(?)	O	10		G*		G, M	
IMF[19]	R(105)	O			G*	O	G	24H
[48]	S(?)	O	5		G		G	24H
NEZHA[34]	R(6)	A, L, O			O		R	
[45]	G	A, L					V	5M
S2F[47]	L, R(8)	A, O			G	O	V	5H, 24H
FairFuzz[25]	R(9)	A	20	C	C	E	V/M	24H
Angora[9]	L, R(8)	A, V, O	5		G, C	L, E	V	5H
T-Fuzz[33]	C(296), L, R(4)	A, O			C, G*		V	24H
MEDS[20]	S(2), R(12)	O	10		C		V	6H

Evaluations

- 19/32 papers said nothing about multiple trials
  - Assume 1
- 13/32 papers said multiple trials
  - Varying number; one case not specified
- 3/13 papers characterized variance across runs
- 0 papers performed a statistical test

Things to Consider

Things to Consider

- **Timeout**
  - Timeout ≤ 24 hours
  - Longer timeouts are better

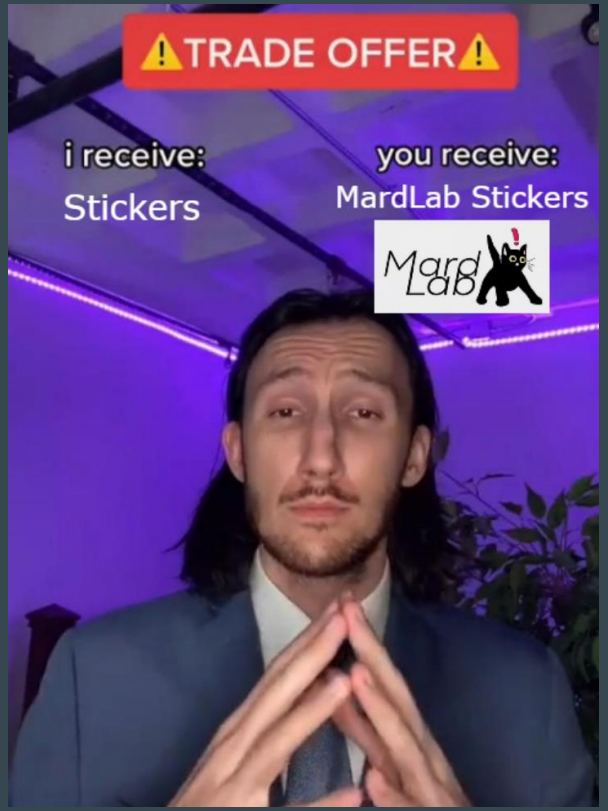
- **Seed Corpus**
  - Performance with different seeds varies dramatically
  - The empty seed can perform well
  - Document seed choices well
  - Evaluate on several seeds to assess performance difference

Things to Consider

- Hardware-based code coverage methods
  - Single Stepping
  - APIC/PIT
  - Intel PT
  - AMD IBS
  - Page fault coverage
- Harness
  - Observes program behavior
  - Crashes
  - Code coverage
  - Error messages

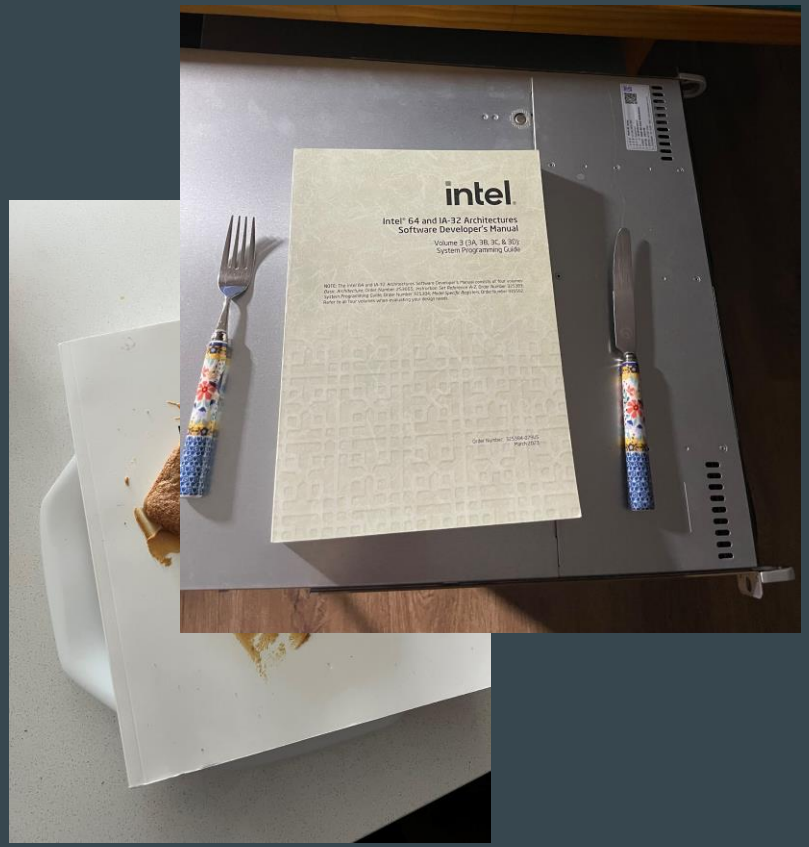
No Optimization?

Sad Mard☹



Thank you!

For letting a 16 year old ramble about fuzzing


intel
Intel® 64 and IA-32 Architectures
Software Developer's Manual
Volume 3 (3A, 3B, 3C, & 3D):
System Programming Guide

Questions?