Sowing the Seeds of Fuzz:

Does the Influence of the Initial Seed Corpus Follow a Universal Law?

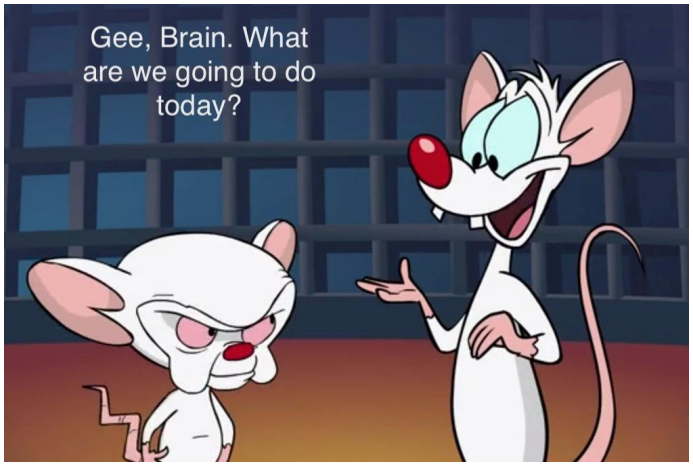
Allison Naaktgeboren

naak@pdx.edu

June 2022

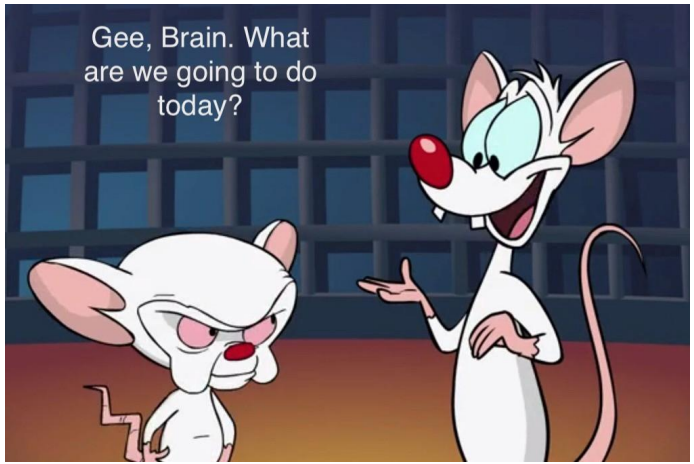
Portland State
Computer Science

Agenda



1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. Research Question
4. Experiment
5. Results
6. Related Work
7. Future Work
8. Conclusion & Acknowledgements
9. Public Questions

Agenda



1. **Motivation & Overview of Fuzzing**
2. Theories about the Influence of the Initial Corpus
3. Experiment
4. Results
5. Related Work
6. Future Work
7. Conclusion & Acknowledgements
8. Public Questions

Motivation

Security Bulletin: IBM Security Guardium is affected by Open Source libxml2 vulnerabilities

Amazon Linux Security Center

Security Bulletin: Vulnerability in libxml2 affects IBM InfoSphere Streams. (CVE-2015-8317)

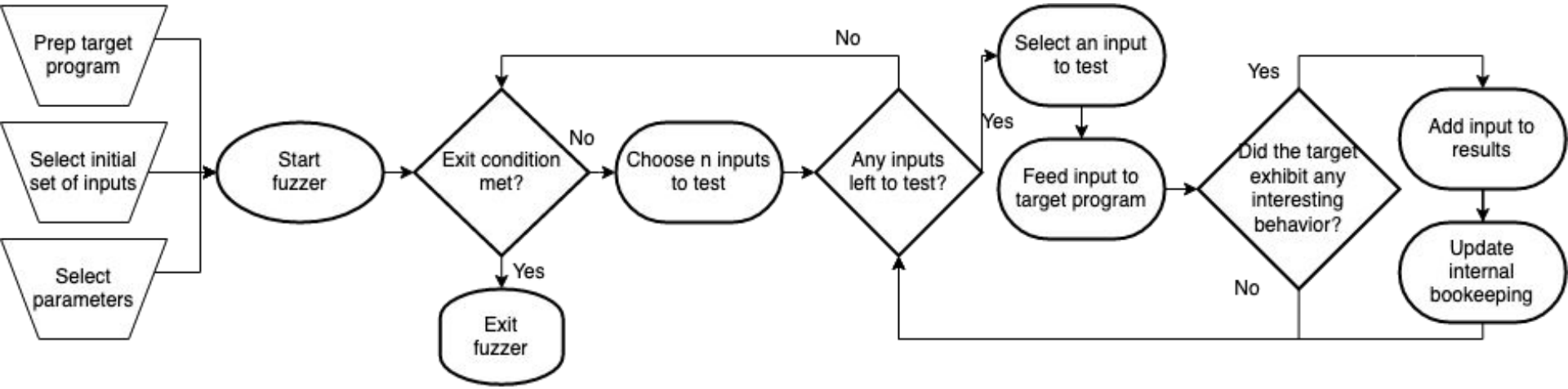
Heartbleed bug 'will cost millions'

How much will it cost to secure open-source software? OpenSSF says $147.9M

Affected Packages

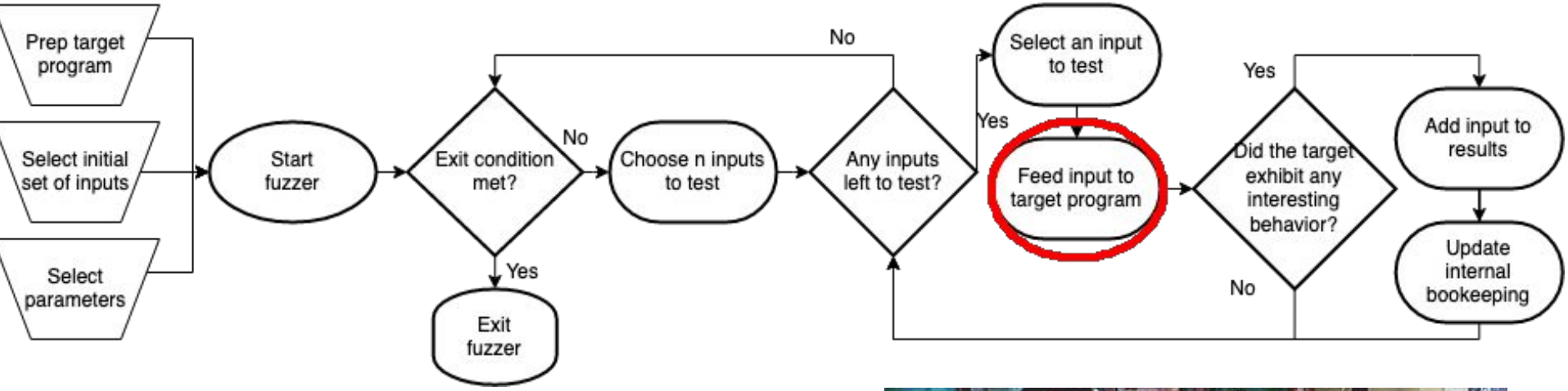
Platform
Amazon Linux 2
Amazon Linux 1

Overview of Fuzzing



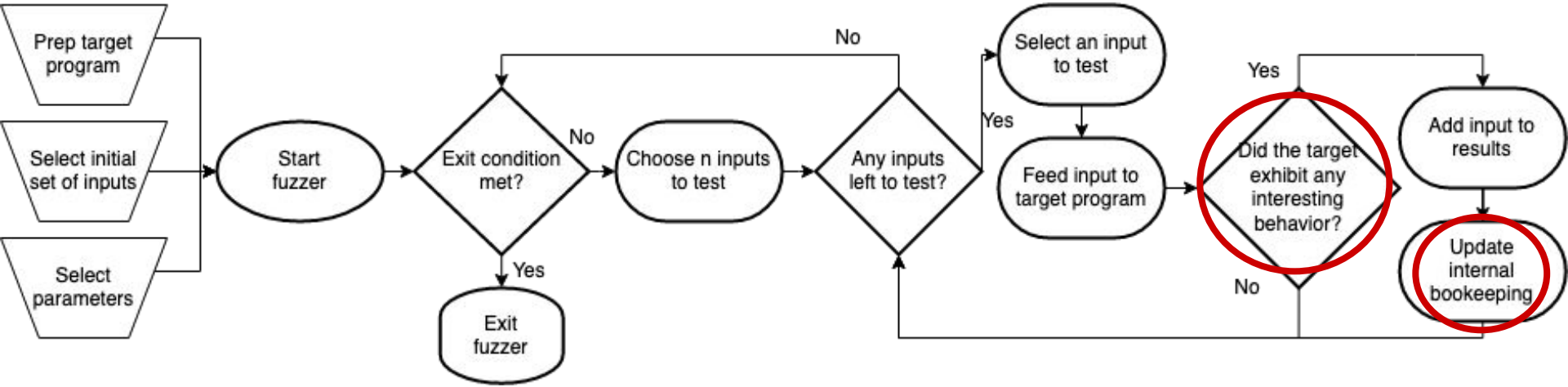
- Aims to thoroughly explore the input space of the fuzzee (victim) looking for inputs (seeds) that cause interesting behavior
- Definition of “Interesting” varies, always includes crashes
- A stochastic (probabilistic) dynamic software-testing technique

Fuzzers, Fuzzees, & Drivers



- Drivers (harnesses) ‘translate’ between fuzzer output and fuzzee input
- Optional
- Influences Magma CPU-hour calculations

What is a Grey-box Fuzzer?

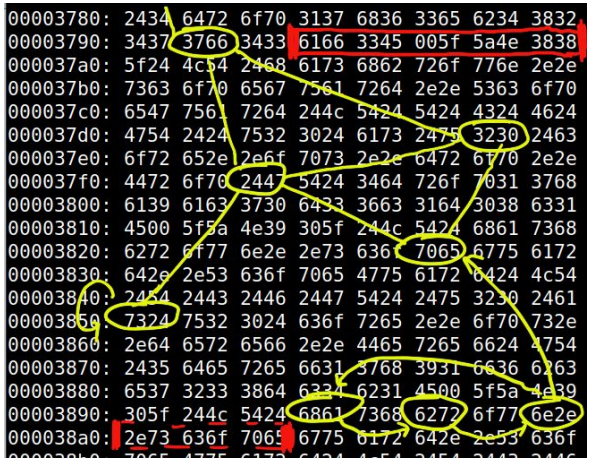


What is a Grey-box Fuzzer?

```
105  // returns a reference
106  ub 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          };
```

```
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
```

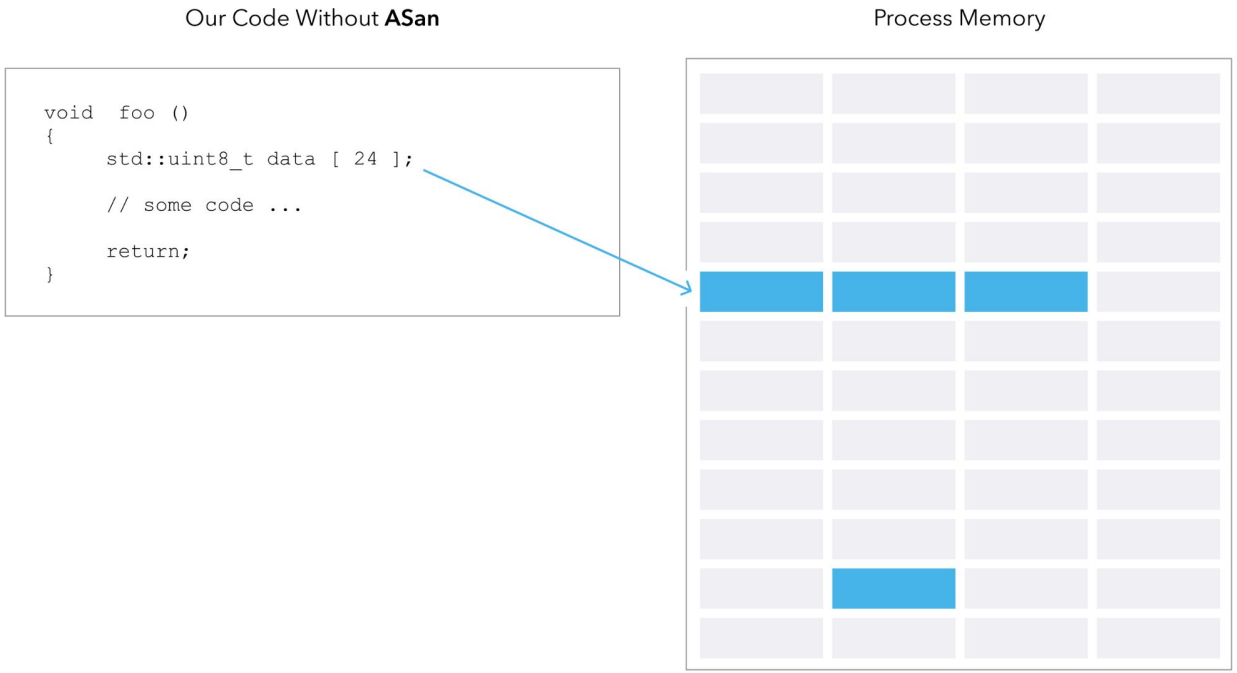
White-box (open-box)



Black-box (closed-box)

Grey-box (leaky-box)

Making More Things Interesting: Address Sanitization



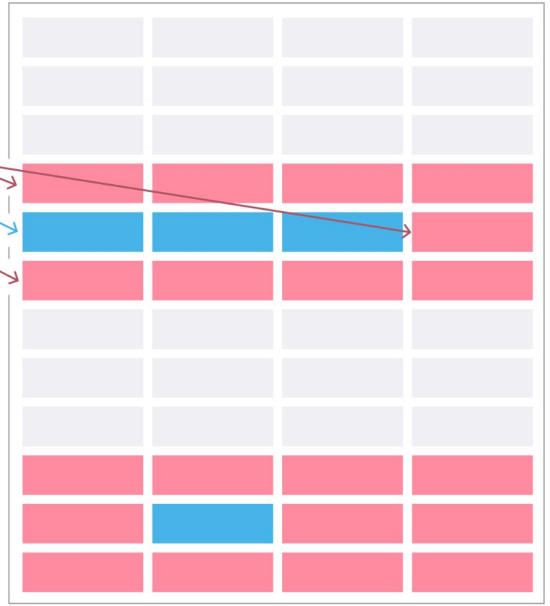
https://m-peko.github.io/craft-cpp/posts/be-wise-sanitize-keeping-your-cpp-code-free-from-bugs/

Making More Things Interesting: Address Sanitization

Our Code With **ASan**

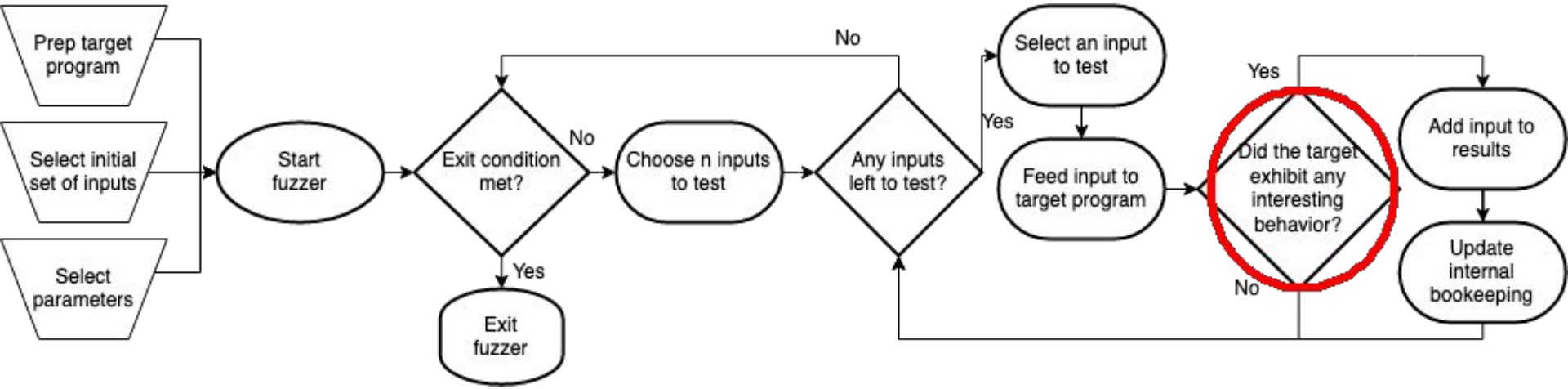
```
void  foo ()
{
      std::uint8_t redzone_1 [ 32 ];
      std::uint8_t data      [ 24 ];
      std::uint8_t redzone_2 [  8 ];
      std::uint8_t redzone_1 [ 32 ];

      // 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



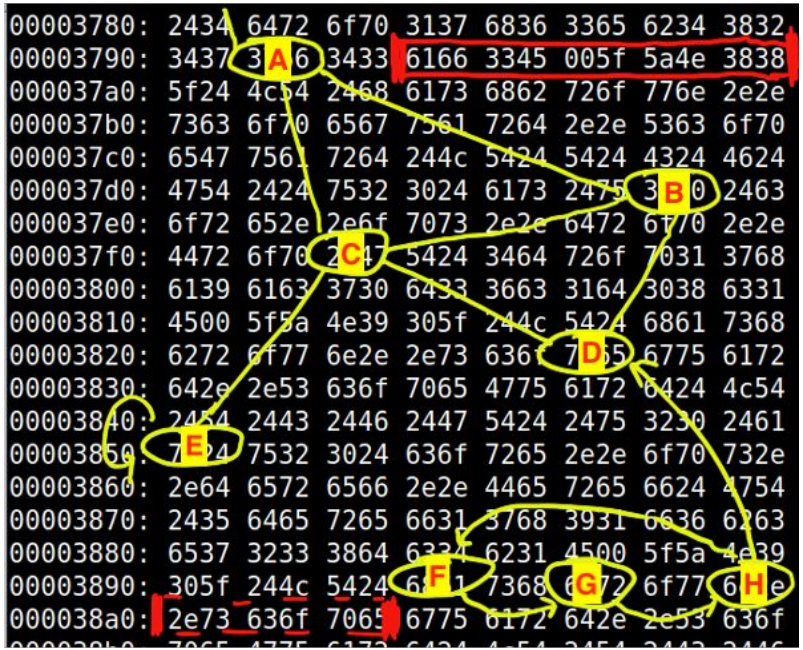
https://m-peko.github.io/craft-cpp/posts/be-wise-sanitize-keeping-your-cpp-code-free-from-bugs/

What is a Coverage-guided Fuzzer?

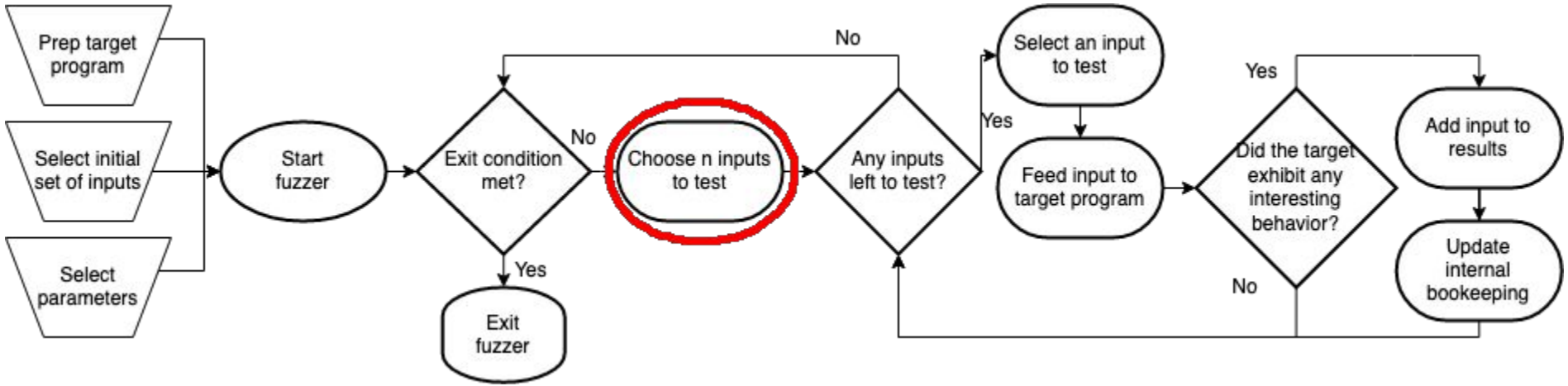


What is a Coverage-guided Fuzzer?

- Uses coverage to determine “interesting” when there isn’t a crash
- AFL style: unique tuples inserted markers into binary
- Example:
  - S1 exercises
  - ABDC -> AB, BD, DC
  - S2 exercises
  - ACEE -> AC, CE, EE

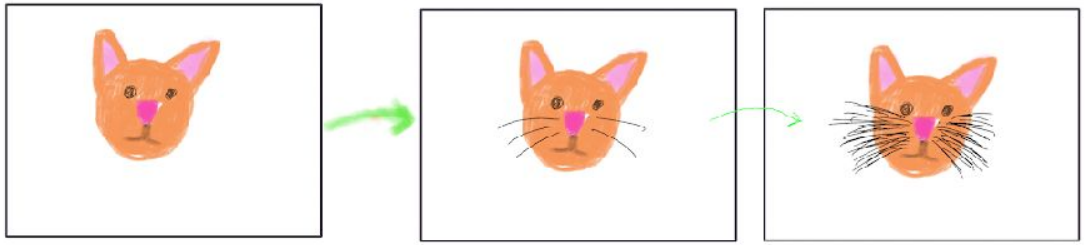


What is a Mutational Fuzzer?



What is a Mutational fuzzer?

Starting Seed

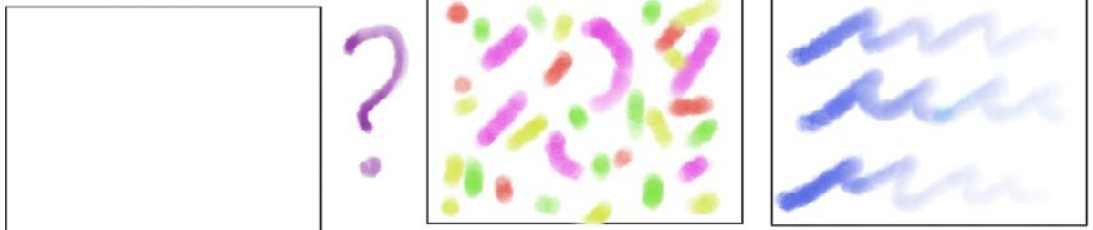


Mutation

Starting Specification

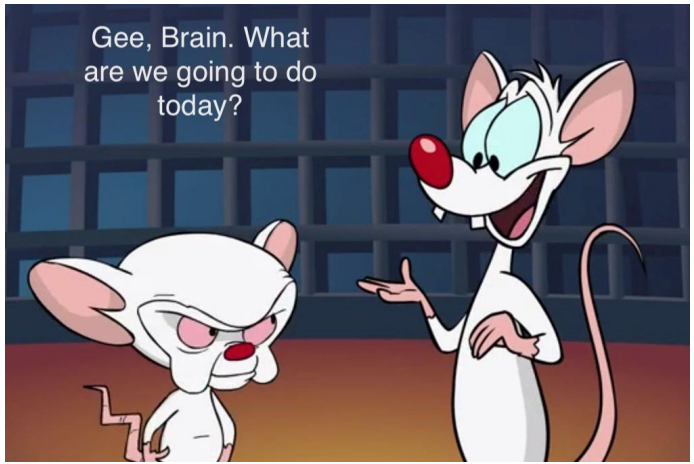
Generation

Starting Source of Randomness



Classic, Naive

Agenda



1. Motivation & Overview of Fuzzing
2. **Theories about the Influence of the Initial Corpus**
3. Research Question
4. Experiment
5. Results
6. Related Work
7. Future Work
8. Conclusion & Acknowledgements
9. Public Questions

Is there a universal rule about the impact of the initial corpus for mutational coverage-guided grey-box fuzzing?

There are two, and they seem to contradict each other

#1: Conventional Theory:

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

Conventional Theory Example

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:

Initial Corpus can easily be replaced. Just fuzz a little longer and the results will be about the same.

Zalewski Theory Example

Building the input corpus

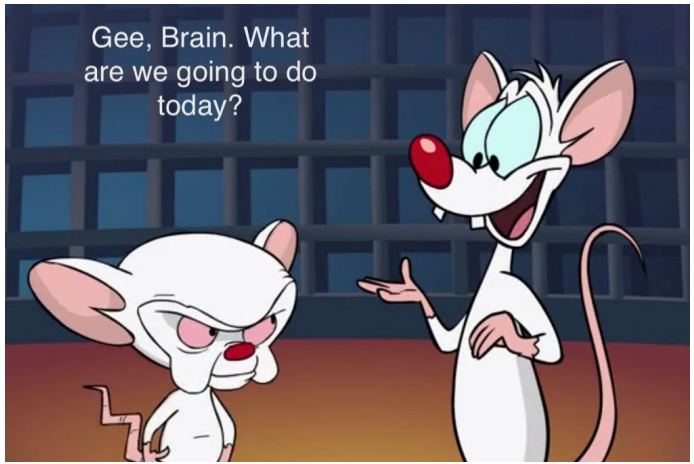
Every fuzzer takes a carefully crafted test cases as input, to bootstrap the first mutations. The test cases should be short, and cover as large part of code as possible. Sadly - I know nothing about netlink. How about we don't prepare the input corpus...

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

Agenda



1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. **Research Question**
4. Experiment
5. Results
6. Related Work
7. Future Work
8. Conclusion & Acknowledgements
9. Public Questions

Is either theory always right about the impact of the initial corpus on performance?

If we can find a counterexample, then it can't be a universal truth

What does ‘Performance’ really mean in Fuzzing?

- Mean bugs discovered
  - T-test 95% interval
- Overall Diversity
  - Rare bugs might be 0-days
  - 0-days worth $$$
  - Uniqueness matters to some audiences

1 Introduction

A General (or professor) walks into a cramped cubicle, telling the lone security analyst (or graduate student) that she has one week to find a zero-day exploit against a certain popular OS distribution, all the while making it sound as if this task is as easy as catching the next bus. Although our analyst

- Per-Flaw Success Probabilities
  - New SOTA
  - Binomial Proportional Confidence Interval (BPCI)
  - 95% CI, Wilson Score
- By the end of the 24-hour runs, were the probabilities about the same?
  - Call this ‘convergence’
  - Low: =< ⅓
  - High: >= ½
- Did Trivial Corpus lose bugs the Nontrivial found in the 24-hour runs?
  - Call this ‘divergence’
  - Low: =< ⅓
  - High: >= ½

Diversity motivation in “Scheduling black-box mutational fuzzing”

Research Question Applied: Is either theory universally right?

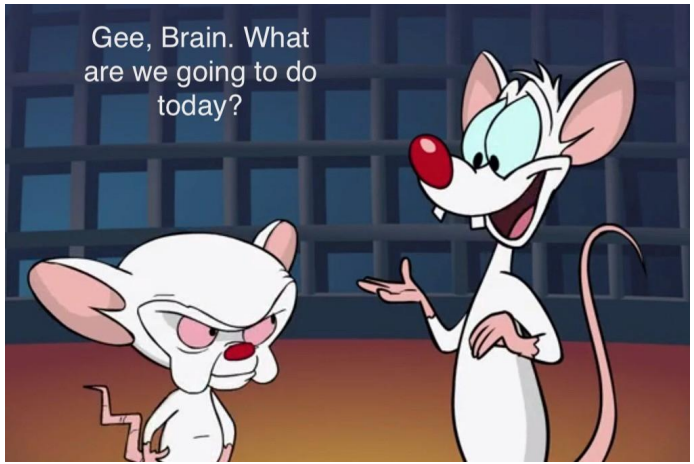
The Conventional Theory predicts Trivial Corpus results will have:

1. Statistically significant lower average bug discovery than the Nontrivial Corpus results
2. Diversity will always be lower than Nontrivial Corpus diversity
3. Per-Flaw Success probabilities will significantly decrease in 24h
4. Low rate of convergence between Trivial & Nontrivial Results in 24h
   a. (=< ⅓ of bugs)
5. High rate of divergence between Trivial & Nontrivial Results in 24h
   a. (>= ½ of bugs)

The Zalewski Theory predicts the Trivial Corpus results will have:

1. Not statistically significant lower average bug discovery than the Nontrivial Corpus results
2. Diversity is not always lower than Nontrivial Corpus diversity
3. Per-Flaw Success probabilities will not significantly decrease in 24h
4. High rate of convergence between Trivial & Nontrivial Results in 24h
   a. (>= ½ of bugs)
5. Low rate of divergence between Trivial & Nontrivial Results in 24h
   a. (=< ⅓ of bugs)

Agenda



1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. Research Question
4. **Experiment**
5. Results
6. Related Work
7. Future Work
8. Conclusion & Acknowledgements
9. Public Questions

Basic Experimental Configuration

- **Magma Fuzzing Benchmark**
  - Know exactly how many bugs present
  - No synthetic bugs or targets
  - Maps to exact bug detected

A ground-truth fuzzing benchmark suite based on real programs with real bugs.

- **Durations:**
  - 5 minutes (CI/CD stand-in, TOFU)
  - 12 hours (Rebert et al.)
  - 24 hours (Klees et al. best practice)
- **Poweredge T440 Intel Xeon Silver 40 server**
  - 16 cores, 32 hyperthreads
  - 15 GB RAM (low)
  - 1 TB drive
  - Ubuntu 20.04.1 LTS

The Two Initial Corpora

- Nontrivial Corpora
  - Control variable
  - Magma default
  - Corpus Minimizer built into Magma
- Trivial Corpora
  - Experimental variable
  - smallest legal file

Fuzzee	Trivial Seed Size	Trivial Corpus Size	Nontrivial Corpus Size
`libxml2`	4 bytes [18]	3	3804
`libtiff`	46 bytes [19]	2	42

Table 1: Trivial Seed, Trivial Corpus Size, and Nontrivial Corpus by Fuzzee.

The Mutational Coverage-Guided Grey-box Fuzzers

AFL

- Common benchmark fuzzer
- Software coverage measurements
- Basic mutations

honggfuzz

- highly tuned fuzzer, with active development
- Hardware & software coverage measurements (e.g. Intel PIN)
- Signal monitoring (POSIX required)
- Advanced mutations

The fuzzees

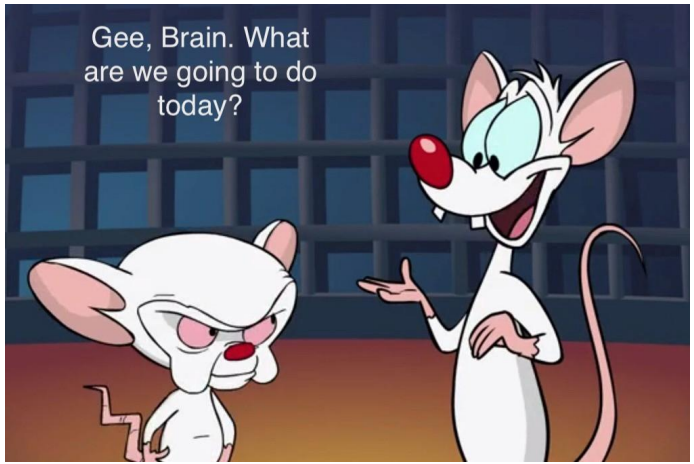
Libxml2

- XML parser
- XML is highly structured
- Linux web servers, Android, Apple iOS, OS X, tvOS, watchOS
- 2 Magma Drivers

Libtiff

- TIFF parser
- TIFF large binary files with header
- Linux web servers, Image Scanning, Fax
- 2 Magma Drivers

Agenda



1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. Research Question
4. Experiment
5. **Results**
6. Related Work
7. Future Work
8. Conclusion & Acknowledgements
9. Public Questions

Overall Means & Diversity: libxml2

Fuzzer	Duration	Significant Difference?	Trivial Corpus		Nontrivial Corpus		Total Flaws
			Mean (std)	Diversity	Mean (std)	Diversity	
AFL	5m	✓	0.9 (.3)	1	2.4 (.49)	3	18
AFL	12h	✓	1.0 (0)	1	2.1 (.3)	3	18
AFL	24h	✓	0.9 (.3)	1	3.0 (0)	3	18
honggfuzz	5m	✓	1.0 (0)	1	3.0 (0)	3	18
honggfuzz	12h	✓	1.0 (0)	1	3.5 (.67)	5	18
honggfuzz	24h	✓	1.2 (.4)	2	4.4 (.49)	6	18

Table 2: Overall Results in `libxml2` Across 10 Trials.

Overall Means & Diversity: libtiff

Fuzzer	Duration	Significant Difference?	Trivial Corpus		Nontrivial Corpus		Total Flaws
			Mean (std)	Diversity	Mean (std)	Diversity	
AFL	5m	✓	1.7 (.46)	2	0.0 (0)	0	14
AFL	12h	✓	3.1 (.54)	5	4.0 (.63)	5	14
AFL	24h	✓	3.1 (.3)	4	4.7 (.64)	8	14
honggfuzz	5m	×	1.2 (.75)	2	1.0 (0)	1	14
honggfuzz	12h	✓	3.5 (.5)	5	4.3 (.46)	7	14
honggfuzz	24h	×	4.4 (.66)	7	4.8 (.87)	7	14

Table 3: Overall Results in Unique Discoveries in `libtiff` Across 10 Trials.

10x Results vs 100x Results

Fuzzer	# Trials	Trivial Corpus		Nontrivial Corpus	
		Mean (std)	Diversity	Mean (std)	Diversity
`libxml2`					
`AFL`	10	0.9 (.3)	1	2.4 (.49)	3
`AFL`	100	1.0 (0)	1	2.0 (0)	2
`honggfuzz`	10	1.0 (0)	1	3.0 (0)	3
`honggfuzz`	100	.99 (.1)	1	2.1 (.27)	3
`libtiff`					
`AFL`	10	1.7 (.46)	2	0.0 (0)	0
`AFL`	100	1.4 (.49)	2	0.3 (.47)	1
`honggfuzz`	10	1.2 (.75)	2	1.0 (0)	1
`honggfuzz`	100	1.6 (.57)	2	1.1 (.27)	4

Table A.1: 100 versus 10 5-minute Trials for `libxml2` and `libtiff`.

Impact on Per-Flaw Success Probabilities: libxml2

Magma ID #	Driver	5m		12h		24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
AFL							
AAH032 CVE-2015-8317 OOB Read	readmem	*(0)*	*17%-69%(4)*	(0)	(0)	**(0)**	**49%-94%(8)**
	xmllint	(0)	(0)	*(0)*	*2%-40%(1)*	**(0)**	**72%-100%(10)**
AAH041 CVE-2016-1762 Heap Over-read	readmem	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**
	xmllint	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**
honggfuzz							
AAH024 CVE-2017-9047 Stack Overflow	readmem	(0)	(0)	*(0)*	*6%-51%(2)*	*(0)*	*6%-51%(2)*
	xmllint	(0)	(0)	*(0)*	*6%-51%(2)*	**(0)**	**49%-94%(8)**
AAH025 CVE-2017-0663 Type Confusion	readmem	(0)	(0)	(0)	(0)	*(0)*	*17%-69%(4)*
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH026 CVE-2017-7375 XML X-Entity	readmem	(0)	(0)	*(0)*	*6%-51%(2)*	*(0)*	*24%-76%(5)*
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH032 CVE-2015-8317 OOB Read	readmem	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**
	xmllint	(0)	(0)	*(0)*	*6%-51%(2)*	**(0)**	**40%-89%(7)**
AAH037 CVE-2016-1838 Heap Over-read	readmem	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	xmllint	*(0)*	*2%-40%(1)*	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**
AAH041 CVE-2016-1762 Heap Over-read	readmem	*(0)*	*2%-40%(1)*	**(0)**	**72%-100%(10)**	**2%-40%(1)**	**72%-100%(10)**
	xmllint	**(0)**	**72%-100%(10)**	**(0)**	**72%-100%(10)**	**2%-40%(1)**	**72%-100%(10)**

Table 4: Degradation Per Flaw of Success Probabilities in `libxml2` from Trivial Initial Corpus. *#%-#%(#)* indicates possible degradation (BPCI intervals overlap). **#%-#%(#)** indicates certain degradation (BPCI intervals do not overlap). (#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

Impact on Per-Flaw Success Probabilities: libtiff

Magma ID #	Driver	5m		12h		24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
AFL							
AAH009 CVE-2016-9535 Heap Overflow	tiffcp	(0)	(0)	(0)	(0)	*(0)*	*2%-40%(1)*
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH010 CVE-2016-5314 Heap Overflow	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	*(0)*	*6%-51%(2)*	*(0)*	*2%-40%(1)*
AAH014 CVE-2016-10269 OOB Read	tiffcp	(0)	(0)	2%-40%(1)	(0)	*2%-40%(1)*	*17%-69%(4)*
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH016 CVE-2015-8784 Heap Overflow	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	(0)	(0)	*(0)*	*2%-40%(1)*
AAH017 CVE-2019-7663 Null Pointer Deref	tiffcp	(0)	(0)	**2%-40%(1)**	**49%-94%(8)**	**(0)**	**72%-100%(10)**
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH020 CVE-2016-3658 Heap Overflow	tiffcp	(0)	(0)	*40%-89%(7)*	*49%-94%(8)*	*49%-94%(8)*	*72%-100%(10)*
	rgba	(0)	(0)	**11%-60%(3)**	**72%-100%(10)**	*24%-76%(5)*	*72%-100%(10)*
honggfuzz							
AAH010 CVE-2016-5314 Heap Overflow	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	*(0)*	*2%-40%(1)*	*2%-40%(1)*	*6%-51%(2)*
AAH013 CVE-2016-10269 OOB Read	tiffcp	(0)	(0)	**11%-60%(3)**	**72%-100%(10)**	*24%-76%(5)*	*72%-100%(10)*
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH015 CVE-2016-10270 OOB Read	tiffcp	**6%-51%(2)**	**72%-100%(10)**	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	*40%-89%(7)*	*72%-100%(10)*	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
AAH017 CVE-2019-7663 Null Pointer Deref	tiffcp	(0)	(0)	*(0)*	*2%-40%(1)*	*(0)*	*11%-60%(3)*
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH020 CVE-2016-3658 Heap Overflow	tiffcp	(0)	(0)	*49%-94%(8)*	*72%-100%(10)*	72%-100%(10)	72%-100%(10)
	rgba	(0)	(0)	*24%-76%(5)*	*72%-100%(10)*	*60%-98%(9)*	*72%-100%(10)*

Table 5: Degradation of Success Probabilities Per Flaw in `libtiff` from Trivial Corpus. *#%-#%(#)* indicates possible degradation (BPCI intervals overlap). **#%-#%(#)** indicates certain degradation (BPCI intervals do not overlap). (#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

Impact on Per-flaw Success Probabilities: libtiff

Magma ID #	Driver	5m		12h		24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
`AFL`							
AAH015 CVE-2016-10270 OOB Read	tiffcp	**72%-100%(10)**	**(0)**	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	**72%-100%(10)**	**(0)**	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
AAH022 CVE-2017-11613 Exhaustion	tiffcp	(0)	(0)	*72%-100%(10)*	*31%-83%(6)*	72%-100%(10)	72%-100%(10)
	rgba	**40%-89%(7)**	**(0)**	*72%-100%(10)*	*60%-98%(9)*	*72%-100%(10)*	*24%-76%(5)*
`honggfuzz`							
AAH014 CVE-2016-10269 OOB Read	tiffcp	(0)	(0)	*6%-51%(2)*	*2%-40%(1)*	*40%-89%(7)*	*11%-60%(3)*
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH022 CVE-2017-11613 Exhaustion	tiffcp	*6%-51%(2)*	*(0)*	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	*24%-76%(5)*	*(0)*	*72%-100%(10)*	*60%-98%(9)*	72%-100%(10)	72%-100%(10)

Unexpected Improvement of Success Probabilities from Trivial Corpus. *#%-#%(#)* indicates possible degradation (BPCI intervals overlap). **#%-#%(#)** indicates certain degradation (BPCI intervals do not overlap). (#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

Convergence: libxml2

Flaw	AFL 24h Convergences	honggfuzz 24h Convergences
AAH024 CVE-2017-9047 Stack Overflow	*Didn't Find*	-
AAH025 CVE-2017-0663 Type Confusion	*Didn't Find*	-
AAH026 CVE-2017-7375 XML External Entity	*Didn't Find*	-
AAH032 CVE-2015-8317 Out of Bounds Read	-	-
AAH037 CVE-2016-1838 Heap Over-read	Converges	Converges
AAH041 CVE-2016-1762 Heap Over-read	-	-

Table 1: Convergences Per-Flaw in `libxml2`. A convergence means the Trivial Result BPCI overlaps $\geq 25\%$ of at least one Nontrivial Result) Cases where all results are 0 don't count as convergences. '*Didn't Find*' means not detected by that fuzzer. 'Converges' means there was at least one convergence. '-' means there were none.

Convergence: libtiff

Flaw	AFL 24h Convergences	honggfuzz 24h Convergences
AAH009 CVE-2016-9535 Heap Overflow	Converges	*Didn't Find*
AAH010 CVE-2016-5314 Heap Overflow	Converges	Converges
AAH013 CVE-2016-10269 Out of Bounds Read	*Didn't Find*	-
AAH014 CVE-2016-10269 Out of Bounds Read	-	-
AAH015 CVE-2016-10270 Out of Bounds Read	Converges	Converges
AAH016 CVE-2015-8784 Heap Overflow	-	Converges
AAH017 CVE-2019-7663 Null Pointer Dereference	-	-
AAH020 CVE-2016-3658 Heap Overflow	-	Converges
AAH022 CVE-2017-11613 Resource Exhaustion	Converges	Converges

Table 3: Convergences Per-Flaw in `libtiff`. A convergence means the Trivial Result BPCI overlaps $\geq 25\%$ of at least one Nontrivial Result) Cases where all results are 0 don't count as convergences. '*Didn't Find*' means not detected by that fuzzer. 'Converges' means there was at least one convergence. '-' means there were none.

Divergence: libxml2

Flaw	AFL 24h Divergences	honggfuzz 24h Divergences
AAH024 CVE-2017-9047 Stack Overflow	*Didn't Find*	Diverges
AAH025 CVE-2017-0663 Type Confusion	*Didn't Find*	Diverges
AAH026 CVE-2017-7375 XML External Entity	*Didn't Find*	Diverges
AAH032 CVE-2015-8317 Out of Bounds Read	Diverges	Diverges
AAH037 CVE-2016-1838 Heap Over-read	-	-
AAH041 CVE-2016-1762 Heap Over-read	Diverges	-

Table 2: Divergences Per-Flaw in `libxml2`. A divergence means the Trivial Result is 0, and at least one Nontrivial Result is at ≥ 3 (BPCI is $\geq 11\%$-60%). '*Didn't Find*' means not detected by that fuzzer. 'Diverges' means there was at least one divergence. '-' means there were none.

Divergence: libtiff

Flaw	AFL 24h Divergences	honggfuzz 24h Divergences
AAH009 CVE-2016-9535 Heap Overflow	-	*Didn't Find*
AAH010 CVE-2016-5314 Heap Overflow	-	-
AAH013 CVE-2016-10269 Out of Bounds Read	*Didn't Find*	-
AAH014 CVE-2016-10269 Out of Bounds Read	-	-
AAH015 CVE-2016-10270 Out of Bounds Read	-	-
AAH016 CVE-2015-8784 Heap Overflow	-	-
AAH017 CVE-2019-7663 Null Pointer Dereference	Diverges*	Diverges
AAH020 CVE-2016-3658 Heap Overflow	-	-
AAH022 CVE-2017-11613 Resource Exhaustion	-	-

Table 4: Divergences Per-Flaw in `libtiff`. A divergence means the Trivial Result is 0, and at least one Nontrivial Result is at ≥ 3 (BPCI is $\geq$ 11%-60%). '*Didn't Find*' means not detected by that fuzzer. 'Diverges' means there was at least one divergence. '-' means there were none.

Research Question + Results Summary

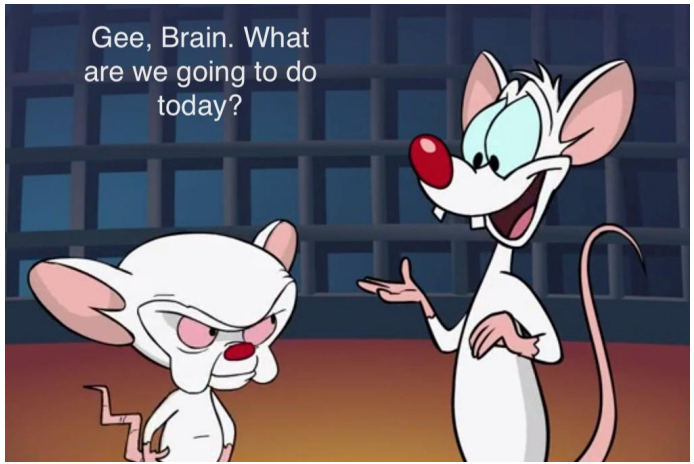
The Conventional Theory predicts Trivial Corpus results will have:

1. Statistically significant lower average bug discovery than the Nontrivial Corpus results
   a. **libxml2: in all experiments, 6/6**
   b. libtiff: not in all experiments, 4/6
2. Diversity will always be lower than Nontrivial Corpus diversity
   a. **libxml2: in all experiments, 6/6**
   b. libtiff: not in all experiments, 2/6
3. Per-Flaw Success probabilities will significantly decrease in 24h
   a. **libxml2: ⅔ c + 0uc, ⅔c + ⅓uc**
   b. libtiff: ⅛ c + ⅝uc, 0c + ½uc
4. Low rate of convergence between Trivial & Nontrivial Results (=< ⅓ of bugs)
   a. **libxml2: ⅓ , ⅓ (low)**
   b. libtiff: ½ , ⅝
5. High rate of divergence between Trivial & Nontrivial Results (>= ½ of bugs)
   a. **Libxml2: ⅔, ⅔ (high)**
   b. libtiff: ⅛*, ⅛

The Zalewski Theory predicts the Trivial Corpus results will have:

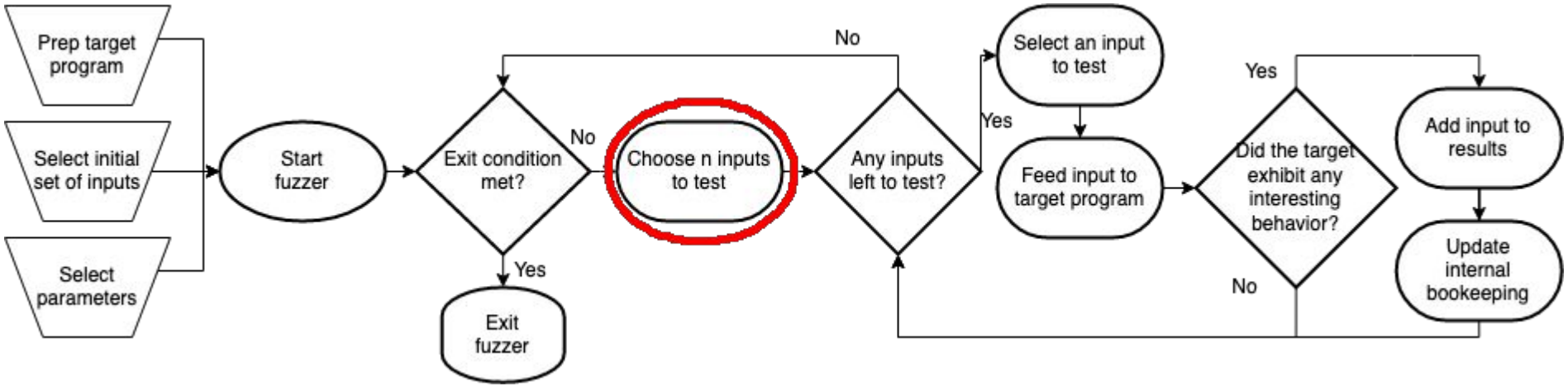
1. Not statistically significant lower average bug discovery than the Nontrivial Corpus results
   a. libxml2: never, 0/6
   b. **libtiff: in some experiments, 2/6**
2. Diversity is not always lower than Nontrivial Corpus diversity
   a. libxml2: never, 0/6
   b. **libtiff: in some experiments, 4/6**
3. Per-Flaw Success probabilities will significantly decrease in 24h
   a. libxml2: ⅔ c + 0uc, ⅔c + ⅓uc
   b. **libtiff: ⅛ c + ⅝uc, 0c + ½uc**
4. High rate of convergence between Trivial & Nontrivial Results (>= ½ of bugs)
   a. libxml2: ⅓ , ⅓
   b. **libtiff: ½ , ⅝ (high)**
5. Low rate of divergence between Trivial & Nontrivial Results (=< ⅓ of bugs)
   a. libxml2: ⅔, ⅔
   b. **libtiff: ⅛*, ⅛ (low)**

Agenda



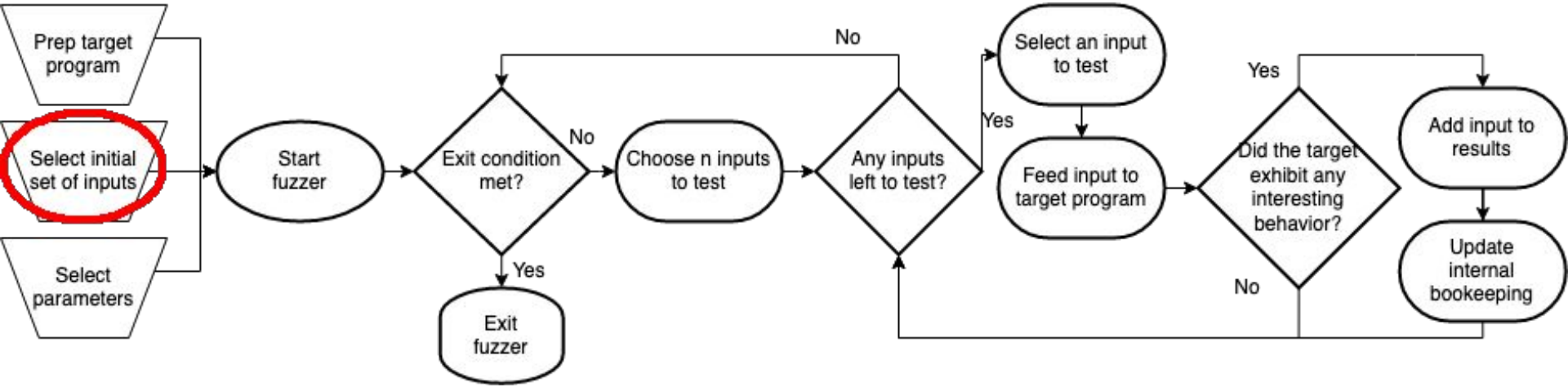
1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. Research Question
4. Experiment
5. Results
6. **Related Work**
7. Future Work
8. Conclusion & Acknowledgements
9. Public Questions

Related Work: Seed Selection (Corpus Evolution)



- Concerned with picking the best seeds (inputs) at the n^{th} iteration
- But the 0^{th} iteration seeds (inputs) aren't studied?
- Examples
  - Optimizing Seed Selection for Fuzzing
  - SeededFuzz: Selecting and Generating Seeds for Directed Fuzzing

Related Work: Corpus Minimization (Distillation)



- Redundant seeds (inputs) are (generally) bad!
- Ok, so how good can they be? Not studied.
- Examples
  - MoonShine: Optimizing OS Fuzzer Seed Selection with Trace Distillation
  - Corpus Distillation for Effective Fuzzing , Seed selection for successful fuzzing

Related Work: Klees et al. demonstrates corpus influence

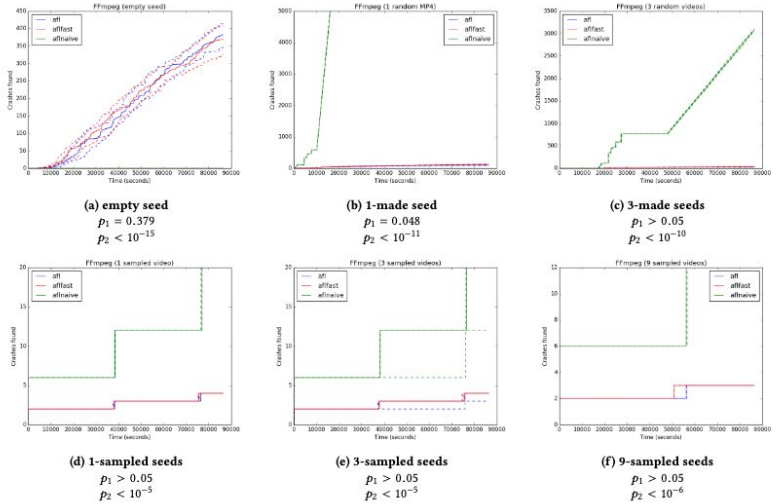
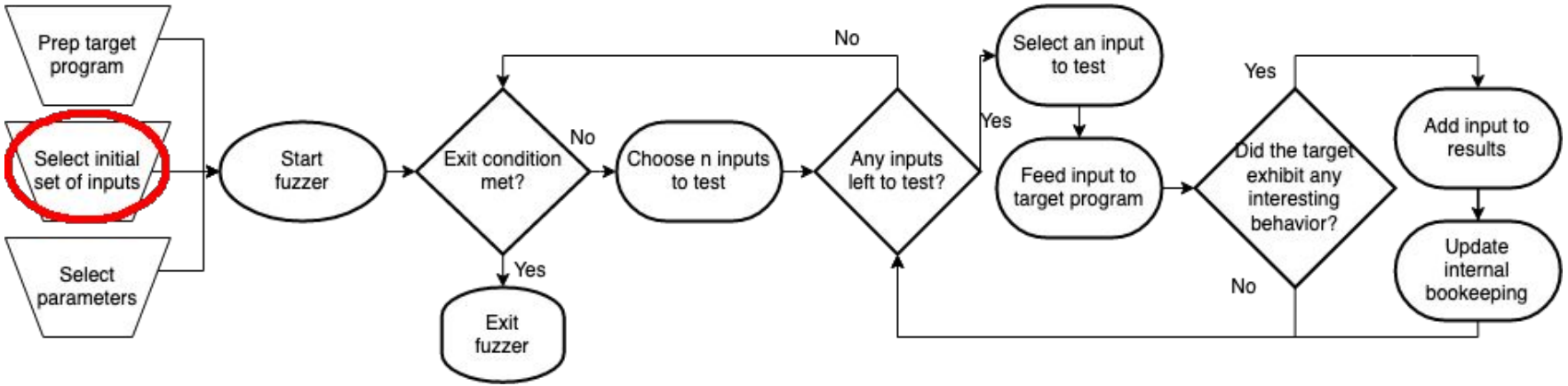
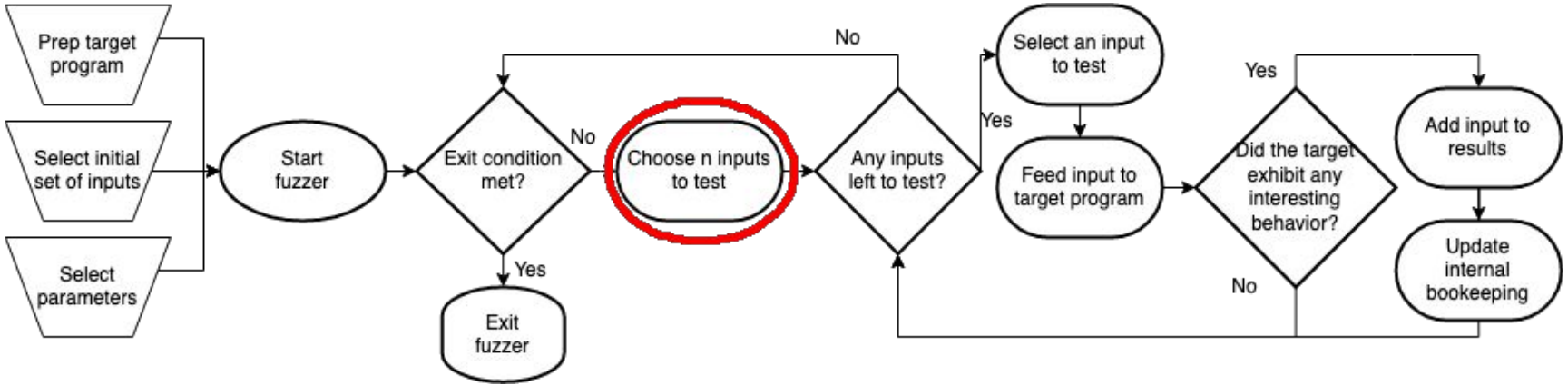


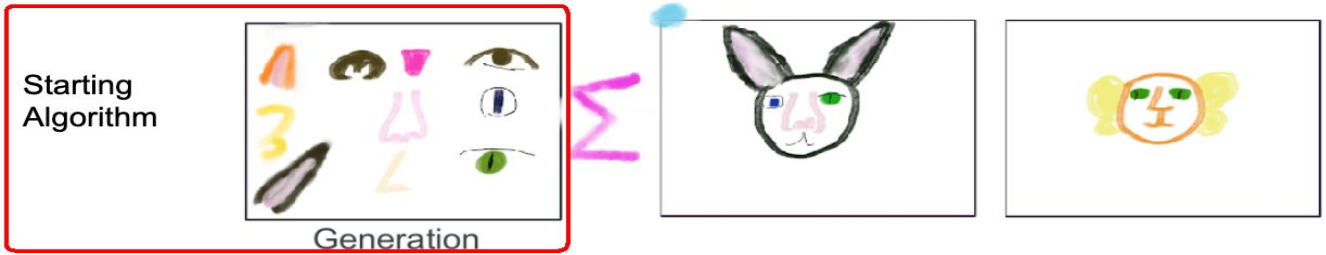
Figure 3: *FFmpeg* results with different seeds. Solid line is median result; dashed lines are confidence intervals. p_1 and p_2 are the p-values for the statistical tests of AFL vs. AFLFast and AFL vs. AFLNaive, respectively.



https://dl.acm.org/doi/pdf/10.1145/3243734.3243804

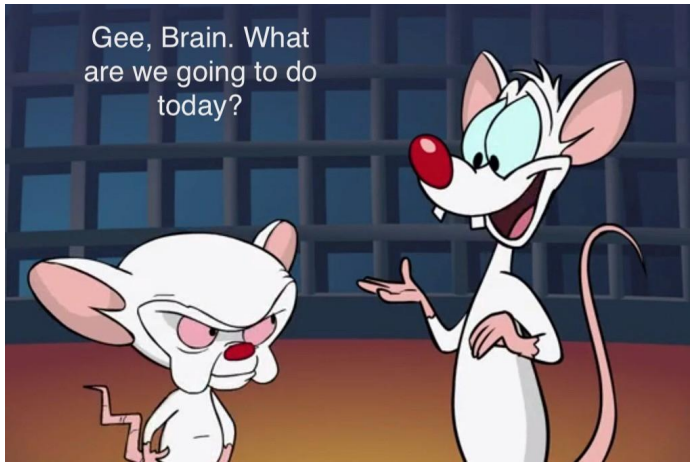
Related Work: Seed Generation





- (Pure) Generational fuzzers seek to construct seeds rather than mutate old ones
- Don't really use the initial seeds (initial set of inputs) or apply to mutational fuzzing
- Examples
  - SmartSeed: Smart Seed Generation for Efficient Fuzzing
  - Skyfire: Data-Driven Seed Generation for Fuzzing

Agenda



1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. Research Question
4. Experiment
5. Results
6. Related Work
7. **Future Work**
8. **Conclusion & Acknowledgements**
9. Public Questions

Future Work

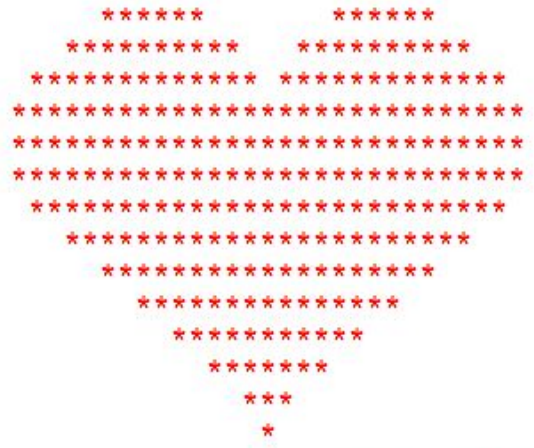
1. What causes a fuzzee to behave like one theory or the other?
    - Highly structured vs binary data?
    - Large attack surface vs small?
    - Could a topology of the target predict which theory it follows?
2. Are other fuzzer types (directed, protocol, hybrid, etc) more or less sensitive to the degradation?
3. Why are some drivers more sensitive to the initial corpus than others?
    - Can this be modeled as a topology to predict which drivers might be sensitive?
4. How to build optimal driver programs?
    - And how might that affect the initial corpus sensitivity?

Conclusion

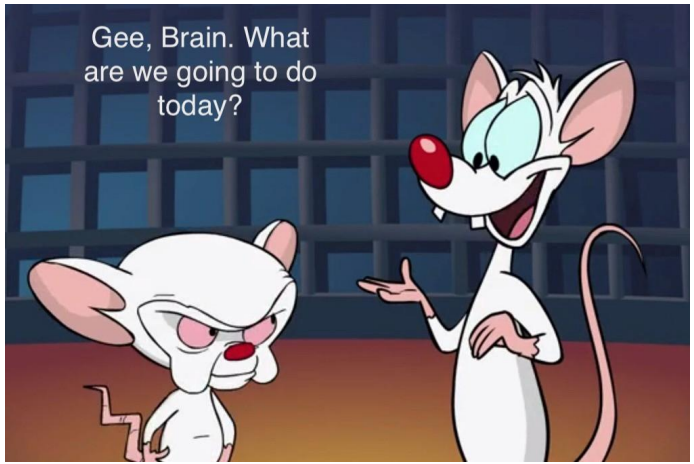
- Disproof by counterexample
  - Libxml2 behaves like the conventional theory predictions
  - Libtiff behaves more like the Zalewski theory predictions
- We don’t yet know what causes a fuzzee to behave like one or the other
- Per-Flaw Binomial Proportion Confidence Interval (BPCI) is interesting as a metric, but may not be practical in future work
  - requires very large sample sizes
  - Unwieldy to work with data, not easily combined or averaged

Acknowledgements

- Research Proficiency Exam Committee
- NSF grant, #1821841
- Andrew Ruef, Adrian Herrera
- Yasodha Suriyakumar, Jeremy Vonderfecht
- Kelly Duncan
- Ted Cooper



Agenda



1. Motivation & Overview of Fuzzing
2. Theories about the Influence of the Initial Corpus
3. Research Question
4. Experiment
5. Results
6. Related Work
7. Future Work
8. Conclusion & Acknowledgements
9. **Public Questions**

Optional Slides

Libxml2 Driver Influence on Overall Averages

Fuzzer, Duration	Driver	Trivial Corpus		Nontrivial Corpus		Total Flaws
		Mean (std)	Diversity	Mean (std)	Diversity	
AFL 5m	readmem	0.9 (.3)	1	2.4 (.49)	1	18
	xmllint	0.0 (0)	0	1.0 (0)	1	18
AFL 12h	readmem	1.0 (0)	1	2.0 (0)	2	18
	xmllint	0.0 (0)	0	1.1 (.3)	2	18
AFL 24h	readmem	0.9 (.3)	1	2.8 (.4)	3	18
	xmllint	0.0 (0)	0	2.0 (0)	2	18
honggfuzz 5m	readmem	1.0 (0)	1	1.1 (.3)	2	18
	xmllint	0.0 (0)	0	2.0 (.3)	3	18
honggfuzz 12h	readmem	1.0 (0)	1	2.6 (.49)	5	18
	xmllint	0.0 (0)	0	3.2 (.4)	4	18
honggfuzz 24h	readmem	1.1 (.3)	2	3.6 (.8)	6	18
	xmllint	0.1 (.3)	1	3.8 (.4)	4	18

Table B.5: Overall Results by Driver in `libxml2` Across 10 Trials.

Libtiff Driver Influence on Overall Averages

Fuzzer, Duration	Driver	Trivial Corpus		Nontrivial Corpus		Total Flaws
		Mean (std)	Diversity	Mean (std)	Diversity	
AFL 5m	tiffcp	1.0 (0)	1	0.0 (0)	0	14
	rgba	1.7 (.45)	2	0.0 (0)	0	14
AFL 12h	tiffcp	2.9 (.7)	5	3.2 (1.2)	4	14
	rgba	2.3 (.46)	3	3.1 (.3)	4	14
AFL 24h	tiffcp	2.9 (.54)	4	4.4 (.49)	5	14
	rgba	2.5 (.5)	3	2.8 (.75)	6	14
honggfuzz 5m	tiffcp	0.4 (.66)	2	1.0 (0)	1	14
	rgba	1.2 (.75)	2	1.0 (0)	1	14
honggfuzz 12h	tiffcp	3.3 (.78)	5	4.2 (.4)	6	14
	rgba	2.5 (.5)	3	3.0 (.0)	4	14
honggfuzz 24h	tiffcp	4.2 (.75)	5	4.6 (.66)	6	14
	rgba	3.1 (.54)	5	3.2 (.4)	4	14

Table B.6: Overall Results by Driver in `libtiff` across 10 Trials.

Libxml2 AFL Driver Success Probabilities

Magma ID #	Driver	AFL 5m		AFL 12h		AFL 24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
AAH024	readmem	(0)	(0)	(0)	(0)	(0)	(0)
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH025	readmem	(0)	(0)	(0)	(0)	(0)	(0)
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH026	readmem	(0)	(0)	(0)	(0)	(0)	(0)
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH032	readmem	(0)	17%-69%(4)	(0)	(0)	(0)	49%-94%(8)
	xmllint	(0)	(0)	(0)	2%-40%(1)	(0)	72%-100%(10)
AAH037	readmem	60%-98%(9)	72%-100%(10)	72%-100%(10)	72%-100%(10)	60%-98%(9)	72%-100%(10)
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH041	readmem	(0)	72%-100%(10)	(0)	72%-100%(10)	(0)	72%-100%(10)
	xmllint	(0)	72%-100%(10)	(0)	72%-100%(10)	(0)	72%-100%(10)

Table B.1: Influence of Corpus on `AFL`'s Success Probabilities `libxml2` Across 10 Trials.(#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

Libxml2 honggfuzz Driver Success Influence

Magma ID #	Driver	honggfuzz 5m		honggfuzz 12h		honggfuzz 24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
AAH024	readmem	(0)	(0)	(0)	6%-51%(2)	(0)	6%-51%(2)
	xmllint	(0)	(0)	(0)	6%-51%(2)	(0)	49%-94%(8)
AAH025	readmem	(0)	(0)	(0)	(0)	(0)	17%-69%(4)
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH026	readmem	(0)	(0)	(0)	6%-51%(2)	(0)	24%-76%(5)
	xmllint	(0)	(0)	(0)	(0)	(0)	(0)
AAH032	readmem	(0)	72%-100%(10)	(0)	72%-100%(10)	(0)	72%-100%(10)
	xmllint	(0)	(0)	(0)	0-0%(2)	(0)	40%-89%(7)
AAH037	readmem	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	xmllint	(0)	2%-40%(1)	(0)	72%-100%(10)	(0)	72%-100%(10)
AAH041	readmem	(0)	2%-40%(1)	(0)	72%-100%(10)	2%-40%(1)	72%-100%(10)
	xmllint	(0)	72%-100%(10)	(0)	72%-100%(10)	2%-40%(1)	72%-100%(10)

Table B.2: Influence of Corpus on `honggfuzz`'s Success Probabilities `libxml2` Across 10 Trials. (#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

Libtiff AFL Driver Success Influence

Magma ID #	Driver	AFL 5m		AFL 12h		AFL 24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
AAH009	tiffcp	(0)	(0)	(0)	(0)	(0)	2%-40%(1)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH010	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	(0)	6%-51%(2)	(0)	2%-40%(1)
AAH013	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH014	tiffcp	(0)	(0)	2%-40%(1)	(0)	2%-40%(1)	17%-69%(4)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH015	tiffcp	72%-100%(10)	(0)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	72%-100%(10)	(0)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
AAH016	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	(0)	(0)	(0)	2%-40%(1)
AAH017	tiffcp	(0)	(0)	2%-40%(1)	49%-94%(8)	(0)	72%-100%(10)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH020	tiffcp	(0)	(0)	40%-89%(7)	49%-94%(8)	49%-94%(8)	72%-100%(10)
	rgba	(0)	(0)	11%-60%(3)	72%-100%(10)	24%-76%(5)	72%-100%(10)
AAH022	tiffcp	(0)	(0)	72%-100%(10)	31%-83%(6)	72%-100%(10)	72%-100%(10)
	rgba	40%-89%(7)	(0)	72%-100%(10)	60%-98%(9)	72%-100%(10)	24%-76%(5)

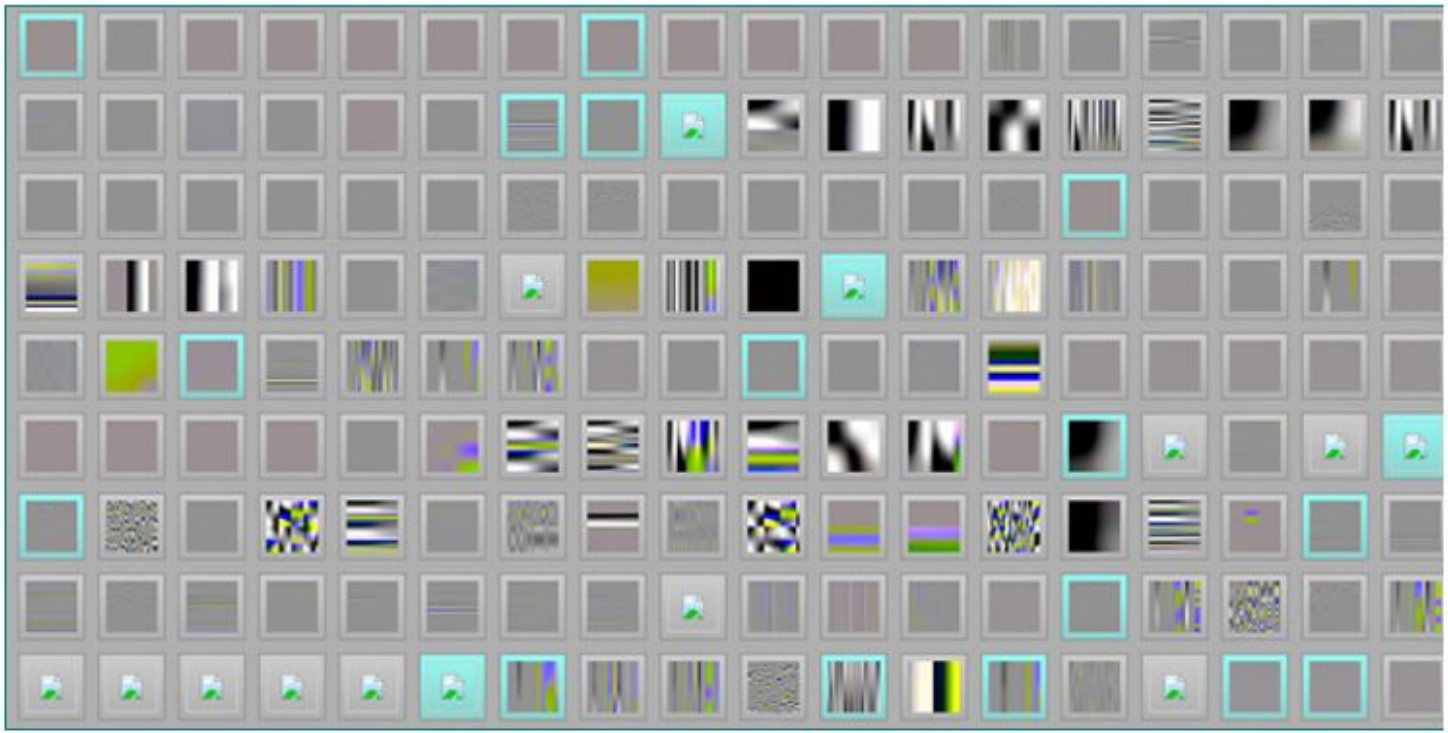
Table B.3: Influence of Corpus on AFL's on Success Probabilities `libtiff` Across 10 Trials. (#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

Libtiff honggfuzz Driver Success Influence

Magma ID #	Driver	honggfuzz 5m		honggfuzz 12h		honggfuzz 24h	
		Trivial	Nontrivial	Trivial	Nontrivial	Trivial	Nontrivial
AAH009	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH010	tiffcp	(0)	(0)	(0)	(0)	(0)	(0)
	rgba	(0)	(0)	(0)	2%-40%(1)	2%-40%(1)	6%-51%(2)
AAH013	tiffcp	(0)	(0)	11%-60%(3)	72%-100%(10)	24%-76%(5)	72%-100%(10)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH014	tiffcp	(0)	(0)	6%-51%(2)	2%-40%(1)	40%-89%(7)	11%-60%(3)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH015	tiffcp	6%-51%(2)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	40%-89%(7)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
AAH016	tiffcp	(0)	(0)	(0)	(0)	2%-40%(1)	(0)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH017	tiffcp	(0)	(0)	(0)	2%-40%(1)	(0)	11%-60%(3)
	rgba	(0)	(0)	(0)	(0)	(0)	(0)
AAH020	tiffcp	(0)	(0)	49%-94%(8)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	(0)	(0)	24%-76%(5)	72%-100%(10)	60%-98%(9)	72%-100%(10)
AAH022	tiffcp	6%-51%(2)	(0)	72%-100%(10)	72%-100%(10)	72%-100%(10)	72%-100%(10)
	rgba	24%-76%(5)	(0)	72%-100%(10)	60%-98%(9)	72%-100%(10)	72%-100%(10)

Table B.4: Influence of Corpus on `honggfuzz`'s on Success Probabilities `libtiff` Across 10 Trials. (#) Indicates # of successes in 10 trials. (0) indicates a complete failure, where the BPCI is 0%-28%.

The first image, hit after about six hours on an 8-core system, looks very unassuming: it's a blank grayscale image, 3 pixels wide and 784 pixels tall. But the moment it is discovered, the fuzzer starts using the image as a seed - rapidly producing a wide array of more interesting pics for every new execution path:



https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html