CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

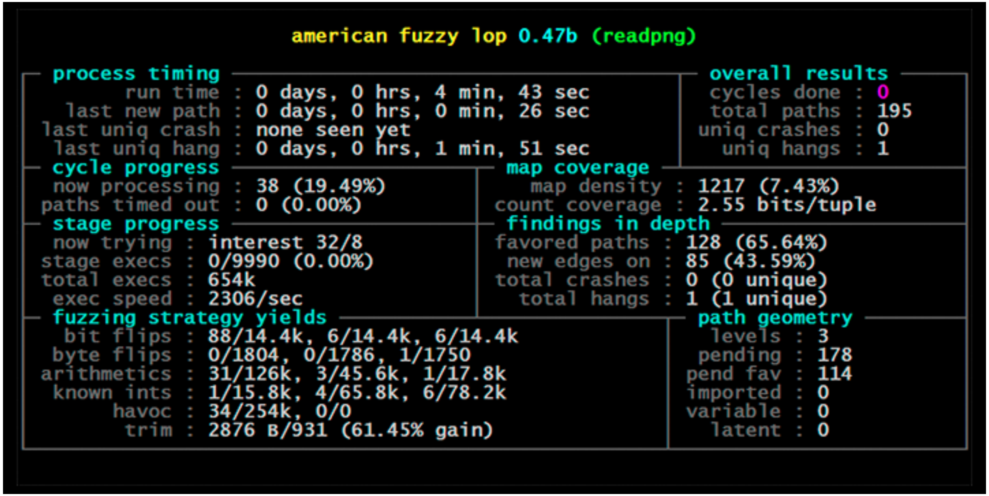
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Systematic Assessment of Fuzzers using Mutation Analysis

Philipp Görz[1]	@phigo@mastodon.social
Björn Mathis[1]	@bjrnmath
Keno Hassler[1]	
Emre Güler[2]	@emrexgueler
Thorsten Holz[1]	@thorstenholz
Andreas Zeller[1]	@andreaszeller
Rahul Gopinath[3]	@rahul@gopinath.org

[1] CISPA Helmholtz Center for Information Security, Germany [2] Ruhr-University Bochum, Germany [3] University of Sydney, Australia

Fuzz Testing / Fuzzing



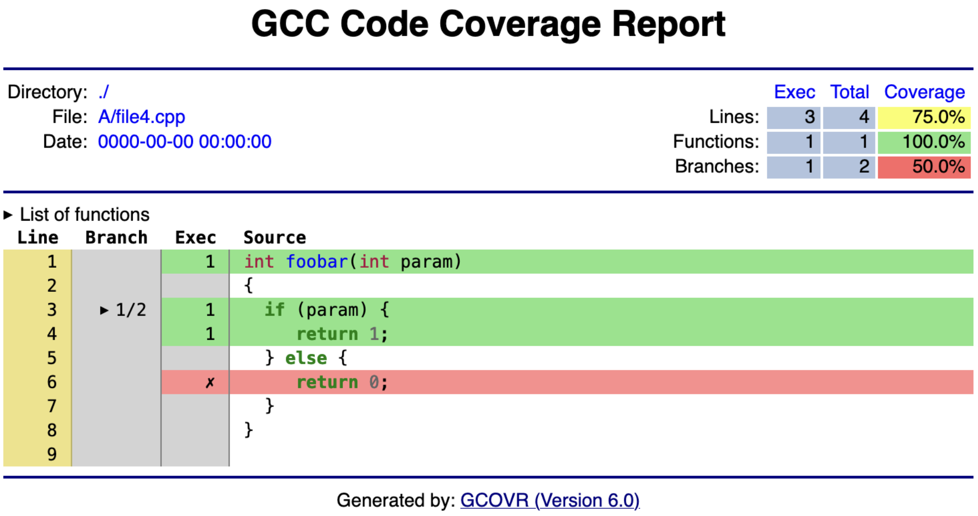
```
                    american fuzzy lop 0.47b (readpng)

┌─ process timing ─────────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 4 min, 43 sec        │  cycles done : 0  │
│   last new path : 0 days, 0 hrs, 0 min, 26 sec        │  total paths : 195│
│ last uniq crash : none seen yet                        │ uniq crashes : 0  │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec        │   uniq hangs : 1  │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────┤
│  now processing : 38 (19.49%)       │    map density : 1217 (7.43%)       │
│ paths timed out : 0 (0.00%)         │ count coverage : 2.55 bits/tuple    │
├─ stage progress ────────────────────┼─ findings in depth ─────────────────┤
│  now trying : interest 32/8         │ favored paths : 128 (65.64%)        │
│ stage execs : 0/9990 (0.00%)        │  new edges on : 85 (43.59%)         │
│ total execs : 654k                  │ total crashes : 0 (0 unique)        │
│  exec speed : 2306/sec              │   total hangs : 1 (1 unique)        │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────┤
│   bit flips : 88/14.4k, 6/14.4k, 6/14.4k            │    levels : 3      │
│  byte flips : 0/1804, 0/1786, 1/1750                │   pending : 178    │
│ arithmetics : 31/126k, 3/45.6k, 1/17.8k             │  pend fav : 114    │
│  known ints : 1/15.8k, 4/65.8k, 6/78.2k             │  imported : 0      │
│       havoc : 34/254k, 0/0                          │  variable : 0      │
│        trim : 2876 B/931 (61.45% gain)              │    latent : 0      │
└─────────────────────────────────────────────────────┴────────────────────┘
```

https://lcamtuf.coredump.cx/afl/

Evaluating Fuzzers

	Comparable	Unbiased	Custom Subjects	Guaranteed Faults
<Approach>				

Evaluating Fuzzers - Coverage?



GCC Code Coverage Report

Directory: ./
File: A/file4.cpp
Date: 0000-00-00 00:00:00

	Exec	Total	Coverage
Lines:	3	4	75.0%
Functions:	1	1	100.0%
Branches:	1	2	50.0%

▸ List of functions

Line	Branch	Exec	Source
1		1	`int foobar(int param)`
2			`{`
3	▸ 1/2	1	`  if (param) {`
4		1	`    return 1;`
5			`  } else {`
6		✗	`    return 0;`
7			`  }`
8			`}`
9			

Generated by: GCOVR (Version 6.0)

https://github.com/gcovr/gcovr

Evaluating Fuzzers

	Comparable	Unbiased	Custom Subjects	Guaranteed Faults
Coverage	✓	—	—	—

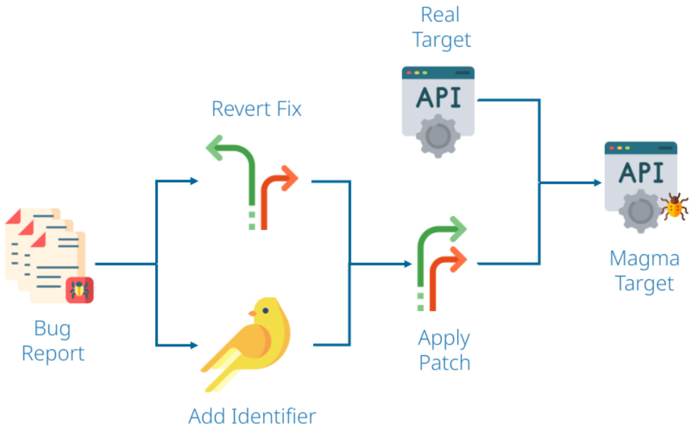
Evaluating Fuzzers - Finding New Bugs?

https://www.cve.org/

Evaluating Fuzzers

	Comparable	Unbiased	Custom Subjects	Guaranteed Faults
Coverage	✓	—	—	—
New Bugs	✗	✗	✓	—

Evaluating Fuzzers - Refinding Known Bugs?



https://hexhive.epfl.ch/magma/

Evaluating Fuzzers

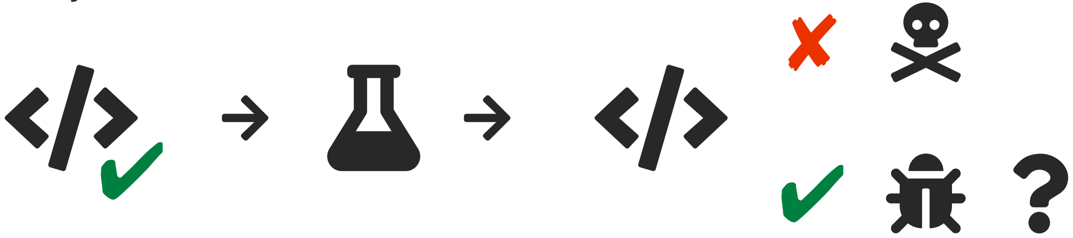
	Comparable	Unbiased	Custom Subjects	Guaranteed Faults
Coverage	✔	—	—	—
New Bugs	✘	✘	✔	—
Known Bugs	✔	✘	✘	✔

Mutation Testing / Mutation Analysis

Fuzzing Your Test Suite

Mutation Testing / Mutation Analysis

```
① ~~unsigned~~ int len = message_length(msg);
if (len ② ~~<~~ >= MAX_BUF_LEN ③ + 16) {
    copy_message(msg);
} else {
    // Invalid length, handle error
}
```



Evaluating Fuzzers

	Comparable	Unbiased	Custom Subjects	Guaranteed Faults
Coverage	✔	—	—	—
New Bugs	✘	✘	✔	—
Known Bugs	✔	✘	✘	✔
Mutation Testing	✔	✔	✔	✘

What’s the Problem?

- Computationally Expensive!
  - Mutation Testing: Execute Test Generator (Fuzzer) for each Mutation
  - Fuzzing: The More Executions the Better

Contributions

- Reduce Computational Costs
  - Split Phases
    - Coverage Fuzzing
    - Mutation Fuzzing
  - Supermutants
    - Evaluate Multiple Mutations with one Fuzzing Run

- Mutation Operators
  - Traditional Operators
  - Security Specific Operators

Results

- Coverage Accounts for most Mutants Detected
- ASAN Moderately Increases Number of Killed Mutants
- Mutations are Coupled to Real Faults

Code

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

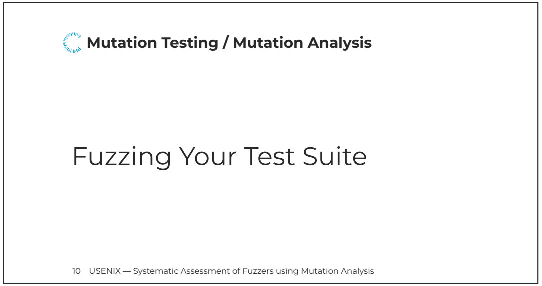
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Code is Publicly Available!

Interested? Talk to Us!

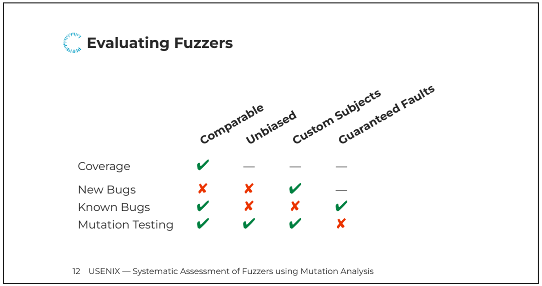
SBFT'24?!

github.com/CISPA-SysSec/mua_fuzzer_bench



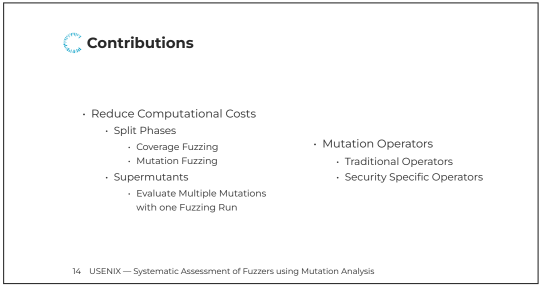
Mutation Testing / Mutation Analysis

Fuzzing Your Test Suite



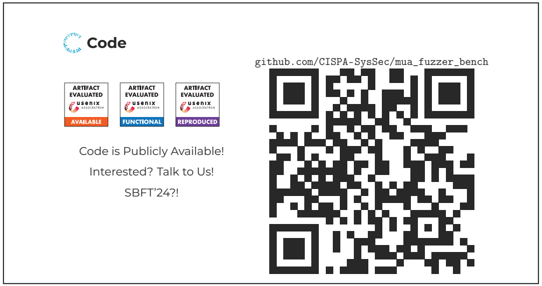
Evaluating Fuzzers

	Comparable	Unbiased	Custom Subjects	Guaranteed Faults
Coverage	✓	—	—	—
New Bugs	✗	✗	✓	—
Known Bugs	✓	✗	✗	✓
Mutation Testing	✓	✓	✓	✗



Contributions

- Reduce Computational Costs
  - Split Phases
    - Coverage Fuzzing
    - Mutation Fuzzing
  - Supermutants
    - Evaluate Multiple Mutations with one Fuzzing Run
- Mutation Operators
  - Traditional Operators
  - Security Specific Operators



Code

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

`github.com/CISPA-SysSec/mua_fuzzer_bench`

Code is Publicly Available!

Interested? Talk to Us!

SBFT'24?!

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

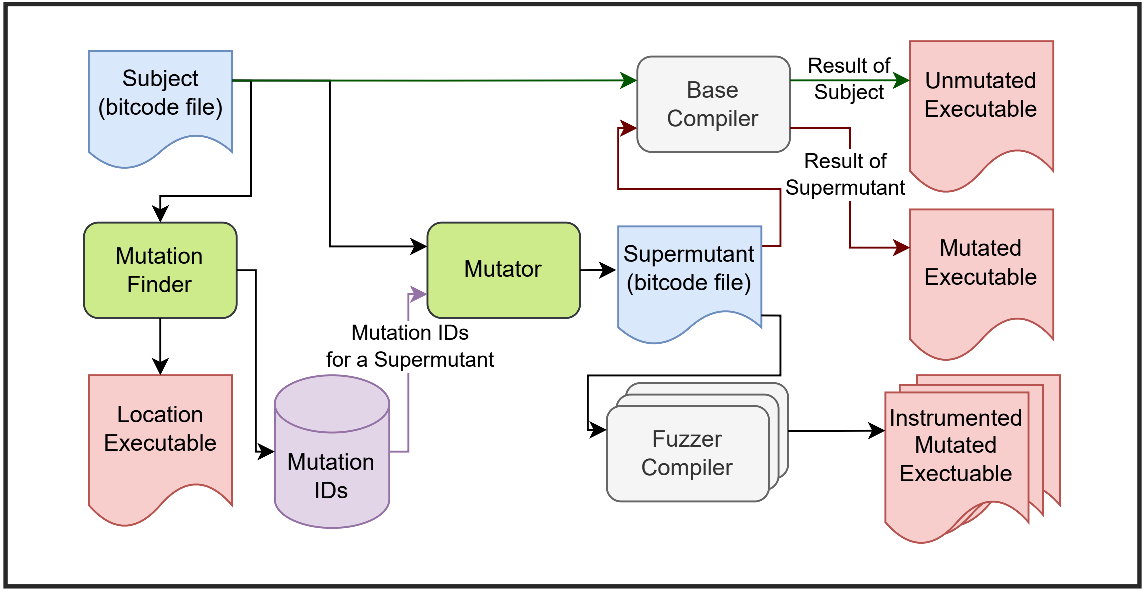
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Systematic Assessment of Fuzzers using Mutation Analysis

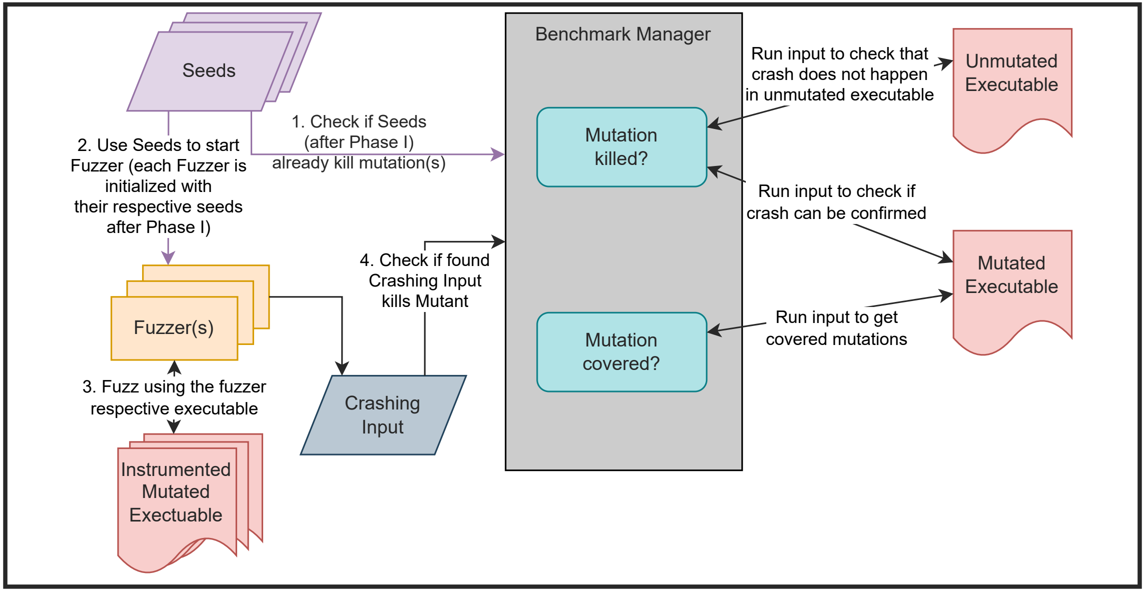
Philipp Görz[1]	@phigo@mastodon.social
Björn Mathis[1]	𝕏 @bjrnmath
Keno Hassler[1]	
Emre Güler[2]	𝕏 @emrexgueler
Thorsten Holz[1]	𝕏 @thorstenholz
Andreas Zeller[1]	𝕏 @andreaszeller
Rahul Gopinath[3]	@rahul@gopinath.org

[1] CISPA Helmholtz Center for Information Security, Germany [2] Ruhr-University Bochum, Germany [3] University of Sydney, Australia

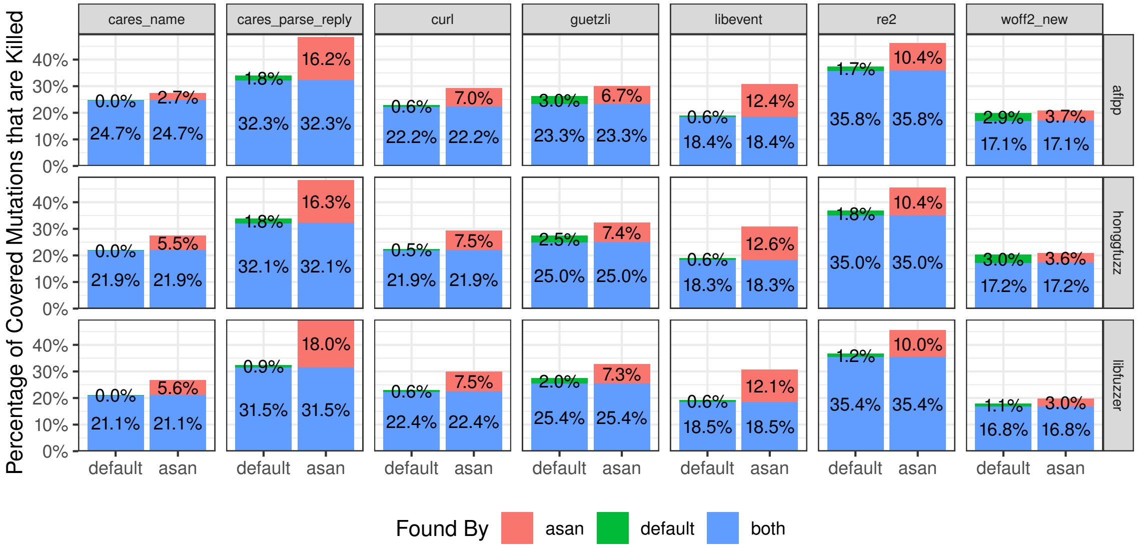
Compilation Procedure



Checking Procedure


Seeds
1. Check if Seeds (after Phase I) already kill mutation(s)
2. Use Seeds to start Fuzzer (each Fuzzer is initialized with their respective seeds after Phase I)
Fuzzer(s)
3. Fuzz using the fuzzer respective executable
Instrumented Mutated Exectuable
Crashing Input
4. Check if found Crashing Input kills Mutant
Benchmark Manager
Mutation killed?
Mutation covered?
Run input to check that crash does not happen in unmutated executable
Unmutated Executable
Run input to check if crash can be confirmed
Mutated Executable
Run input to get covered mutations

ASan Percentages



Supermutants Computational Reduction

Subject	#Mutants	#Supermutants	Factor
Curl	29,118	5,804	5.02
Guetzli	22,961	13,040	1.76
Woff2 (New)	40,914	5,930	6.90
Cares (Name)	4,822	550	8.77
Cares (Parse Reply)	4,822	1,288	3.74
libevent	17,234	864	19.95
re2	21,407	9,670	2.21
Sum	141,278	37,146	**3.80**

Wallclock Time

	CPU (Years)	4 Servers (Days)
Seed Collection	1.99	3.50
Default	14.37	25.22
Seed + Default	16.36	28.72
ASAN	15.16	26.61
24 Hours Runs	7.42	13.02
Sum	38.95 Years	68.34 Days

Four servers: Intel Xeon Gold 6230R CPU (52 cores) and 188 GB RAM.
Note that evaluating a single fuzzer takes 4.09 CPU years with our chosen subjects ("Seed + Default" / #Fuzzers).

24 Hour Runs

Prog	Total	AFL	AFL++	libFuzzer	Honggfuzz
re2	104	0	0	0	0
Woff2 (New)	104	0	0	0	1
Curl	104	0	0	1	0
Guetzli	104	0	0	0	1
Libevent	104	0	0	0	0
Cares (Name)	66	0	0	0	0
Cares (Parse Reply)	104	0	0	0	0

Mutants killed during 24 hour runs on 104 stubborn mutants for each subject using ASAN.

Not Independent Mutants

Program	afl	aflpp	honggfuzz	libfuzzer
Curl	4,850	5,836	4,851	3,852
Guetzli	10	24	16	0
Libevent	0	2	0	0
re2	39	66	37	47
Woff2 (New)	26	46	56	48
Cares (Name)	4	0	0	0
Cares (Parse Reply)	2	4	4	0

Number of mutants that were covered together with other mutants (i.e., mutants wrongly thought independent).