Not All Coverage Measurements Are Equal

Fuzzing by Coverage Accounting for Input Prioritization

NDSS Symposium 2020

Yanhao Wang, Xiangkun Jia, Yuwei Liu, Kyle Zeng,
Tiffany Bao, Dinghao Wu, and Purui Su

AFL Family and Coverage-based Fuzzing

AFL

AFLFast

FairFuzz

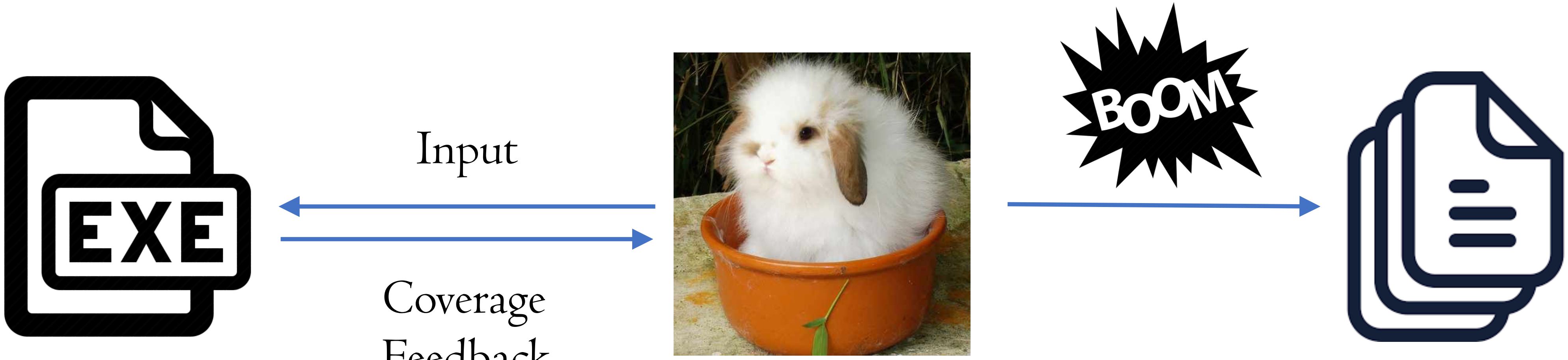
CollAFL

QSYM

AFL-Sensitive

Driller

AFL Family and Coverage-based Fuzzing



Input

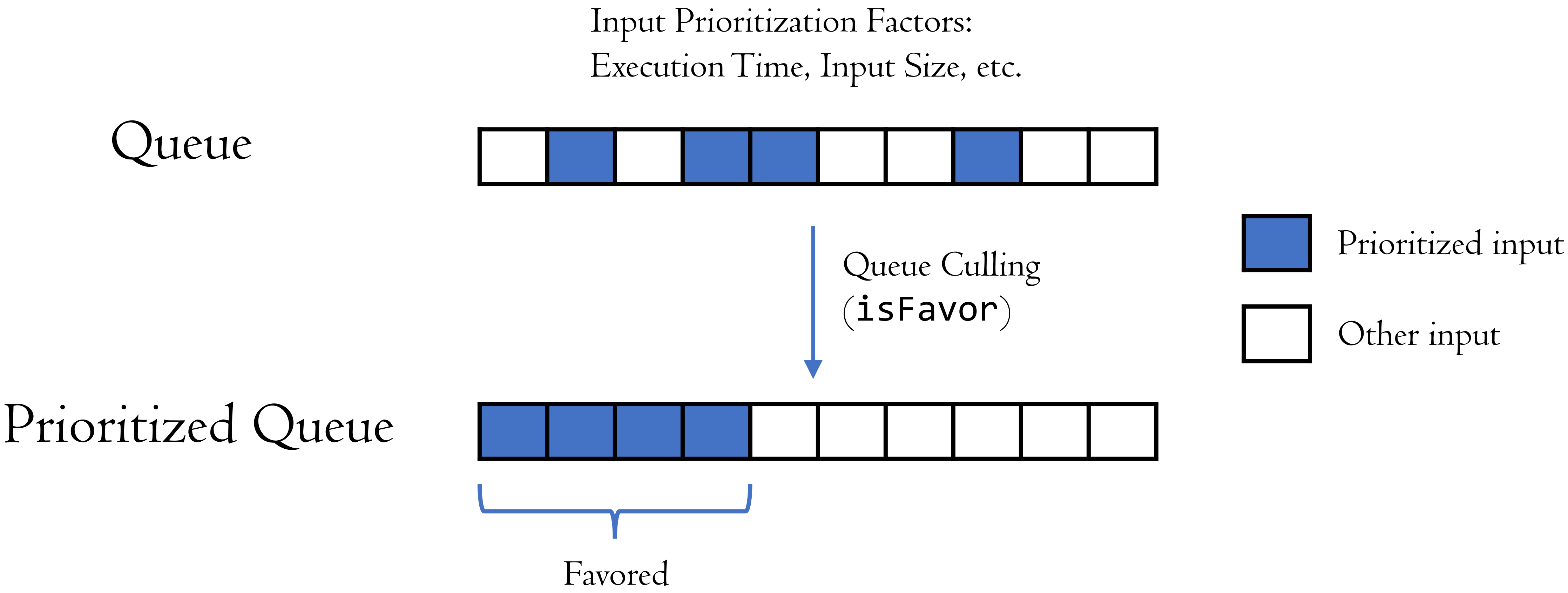
Coverage Feedback

Program

Fuzzer

Crash Inputs

Coverage-based Fuzzing: The Internals



Input Prioritization Factors:
Execution Time, Input Size, etc.

Queue

Queue Culling
(`isFavor`)

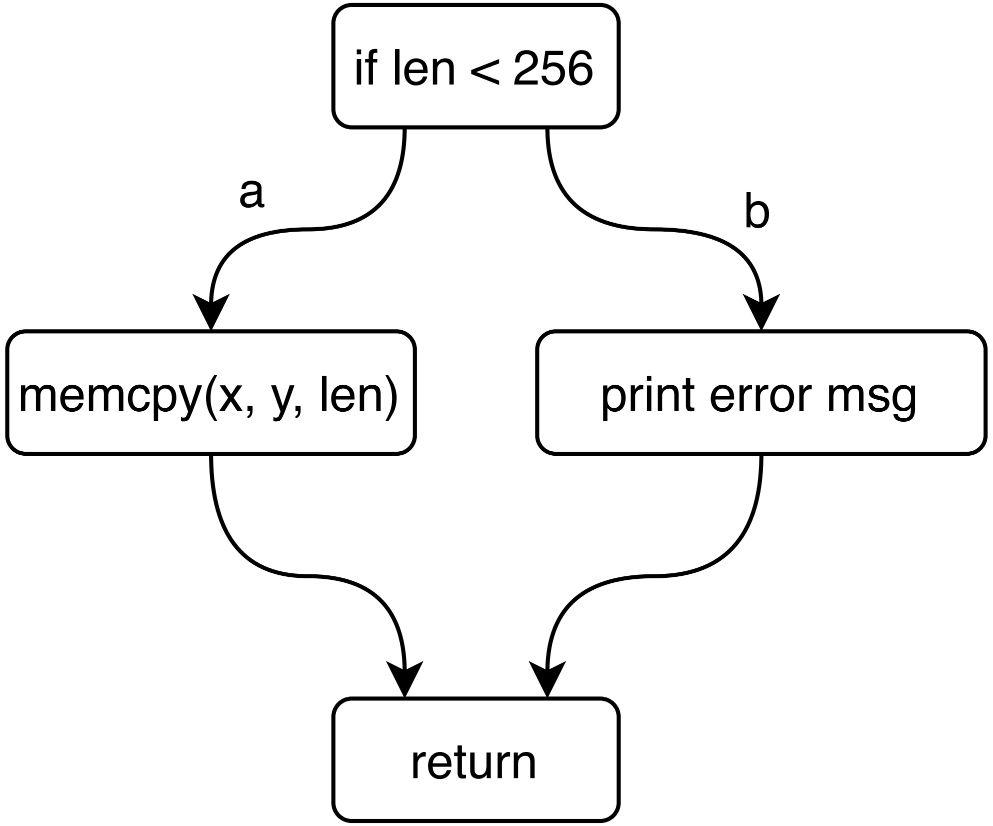
Prioritized Queue

Favored

🟦 Prioritized input

□ Other input

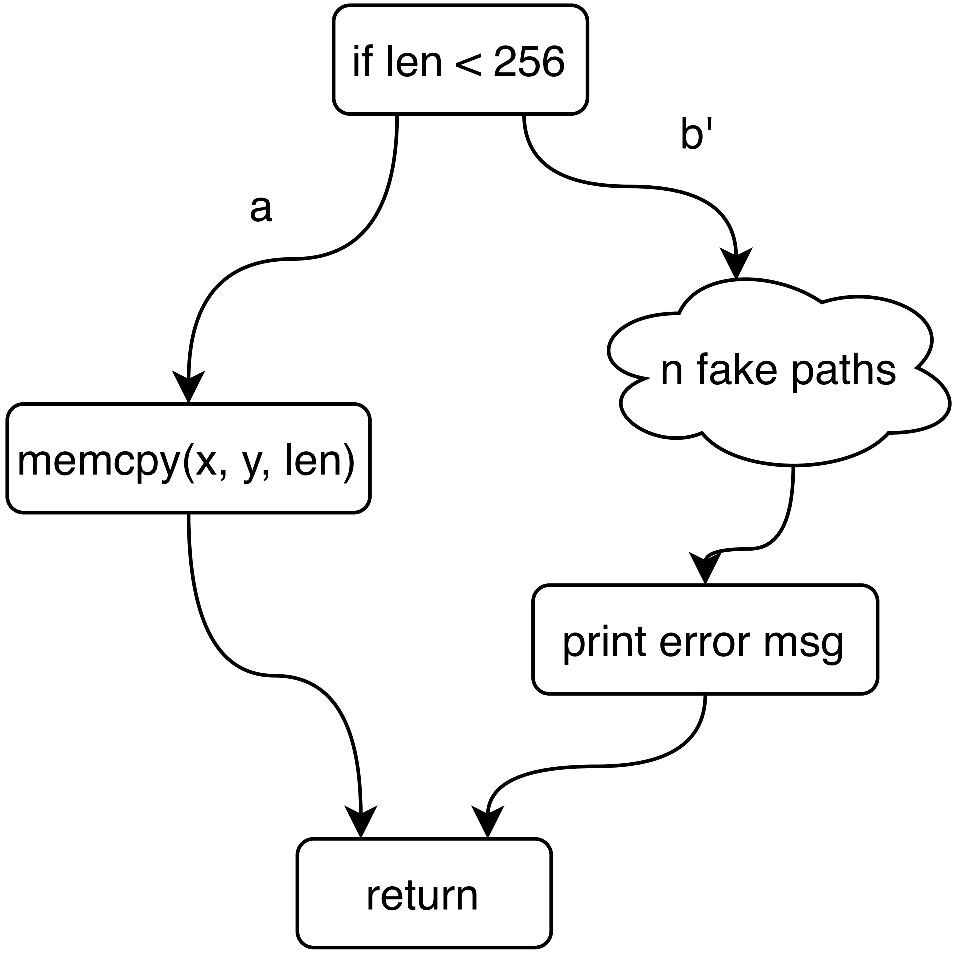
Coverage Measurements are Treated Equally



Spend equal time on security-sensitive paths and security-insensitive paths

Delay finding vulnerabilities

Anti-Fuzzing



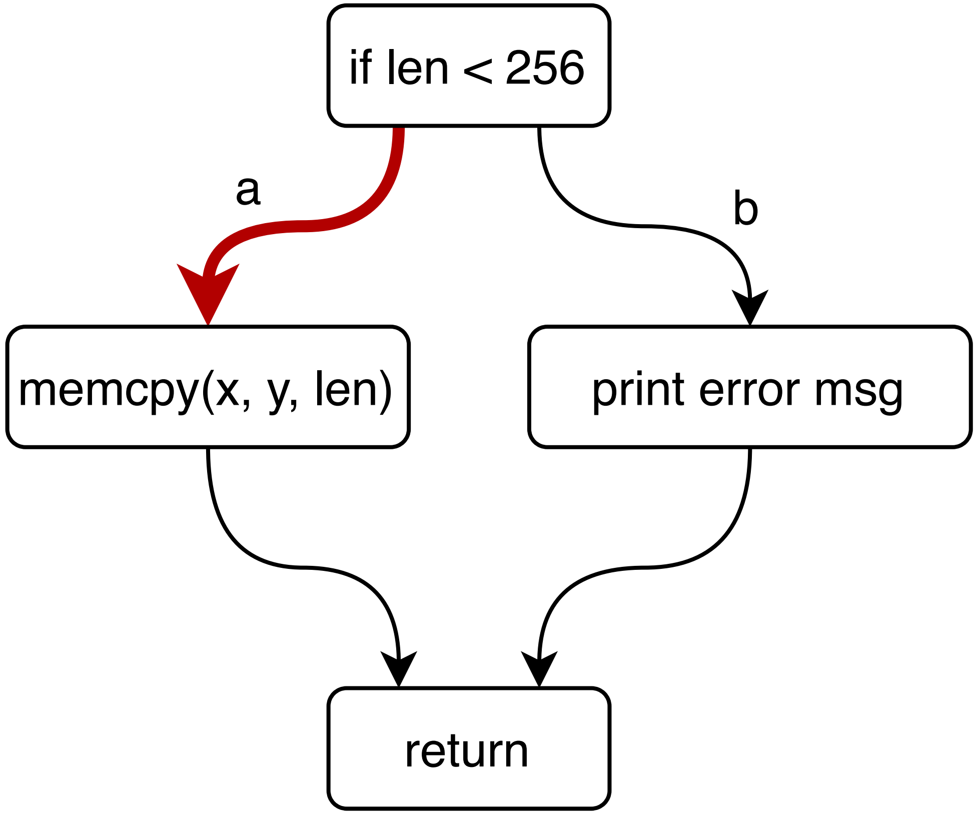
Inject fake coverage measurements to mislead coverage-based fuzzers

What then?

do not

We treat coverage measurements equally

Coverage Accounting



The prioritization of input reflects security sensitivity

Coverage Accounting

What should be the indicators?

function level loop level basic block level

Design a new queue culling scheme based on coverage accounting metrics

Function Level

malloc

memcpy free ...... memmove

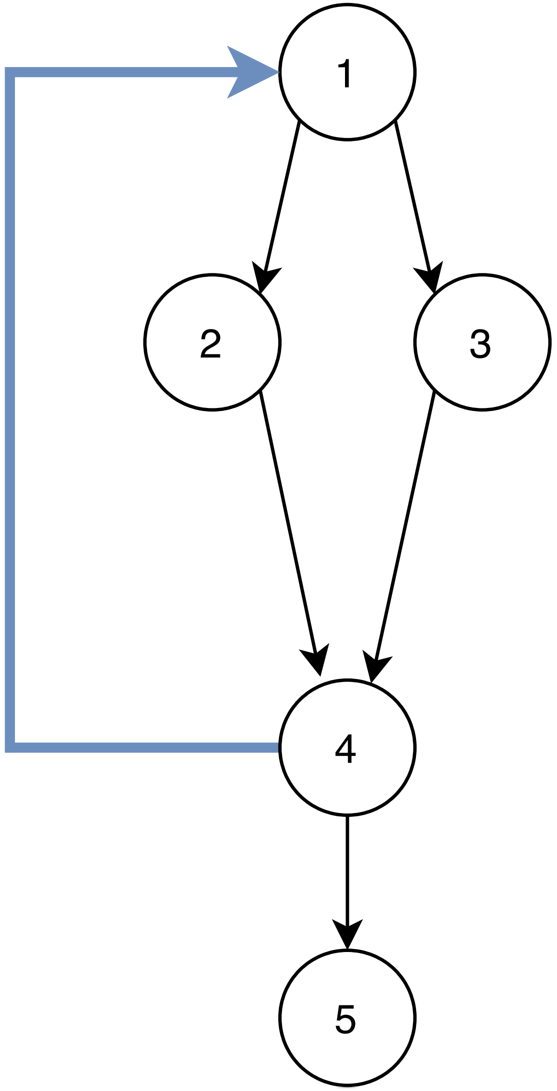
memset

Some functions are inherently likely to be involved in memory corruptions

We crawled call-stacks from webpages of all CVEs in the latest 4 years

Function	Number	Function	Number
memcpy	80	free	12
strlen	35	memset	12
ReadImage	17	delete	11
malloc	15	memcmp	10
memmove	12	getString	9

Loop Level



Incorrect looping condition is often the root cause of memory corruption vulnerabilities

Basic Block Level

```
shl     [rbp+var1], 4
mov     edx, [rbp+var1]
mov     eax, edx
shl     eax, 4
add     eax, edx
mov     [rbp+var1], eax
mov     rdx, [rbp+var2]
mov     rax, [rbp+i]
add     rax, rdx
movzx   edx, byte ptr [rax]
movzx   eax, [rbp+var3]
xor     eax, edx
movzx   eax, al
add     [rbp+var1], eax
movzx   edx, [rbp+var3]
mov     eax, edx
shl     eax, 3
```

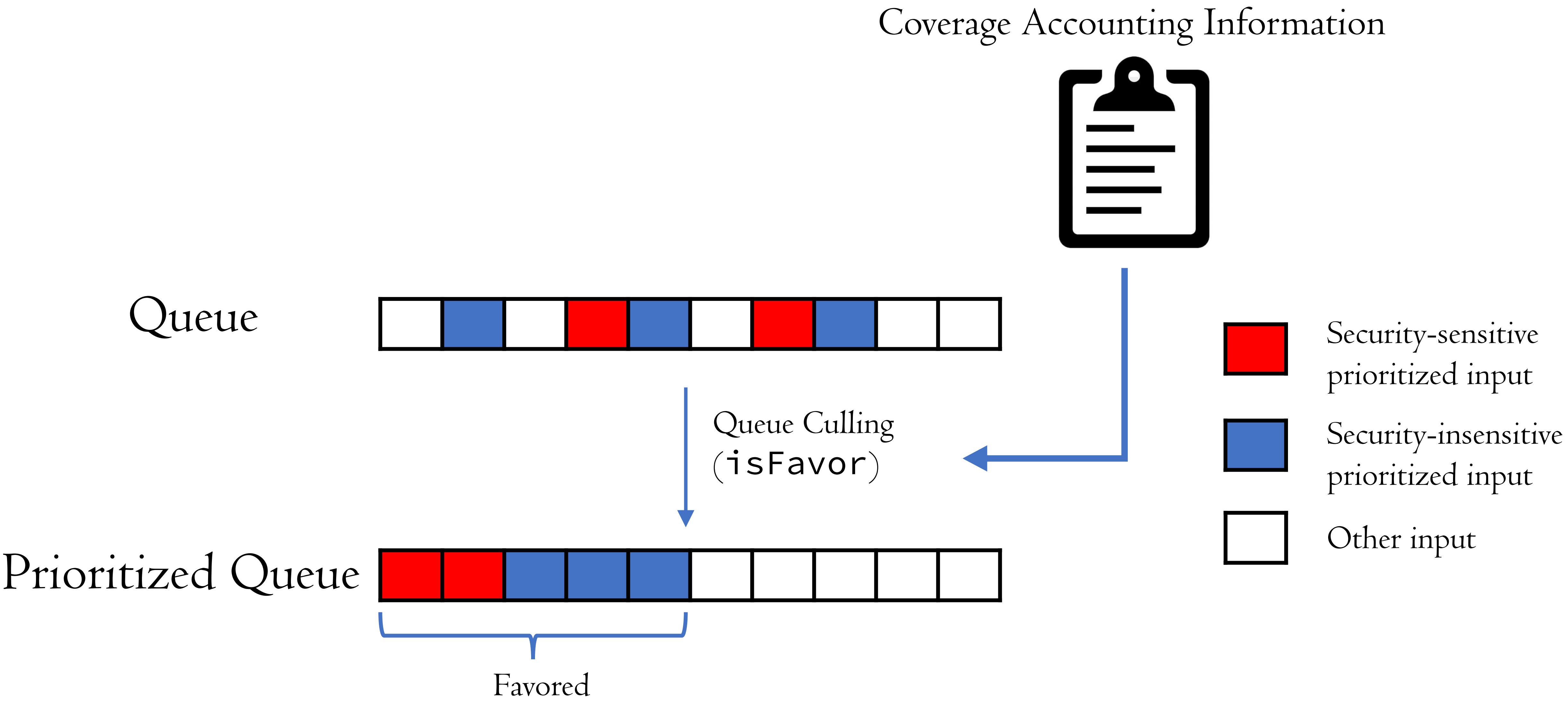
→

🟦	1	shl	[rbp+var1], 4
🟩	2	mov	edx, [rbp+var1]
	3	mov	eax, edx
	4	shl	eax, 4
	5	add	eax, edx
🟦	6	mov	[rbp+var1], eax
🟩	7	mov	rdx, [rbp+var2]
🟩	8	mov	rax, [rbp+i]
	9	add	rax, rdx
🟩	10	movzx	edx, byte ptr [rax]
🟩	11	movzx	eax, [rbp+var3]
	12	xor	eax, edx
	13	movzx	eax, al
🟦	14	add	[rbp+var1], eax
🟩	15	movzx	edx, [rbp+var3]
	16	mov	eax, edx
	17	shl	eax, 3

🟩 read

🟦 write

Design



Coverage Accounting Information

Queue

Queue Culling (`isFavor`)

Prioritized Queue

Favored

🟥 Security-sensitive prioritized input

🟦 Security-insensitive prioritized input

□ Other input

TortoiseFuzz: Coverage-based Fuzzer with Coverage Accounting

AFL

AFLFast

FairFuzz

CollAFL

QSYM

AFL-Sensitive

Driller

TortoiseFuzz

TortoiseFuzz: Coverage-based Fuzzer with Coverage Accounting

The Hare and The Tortoise Story, Bedtime Story by Kids Hut
https://www.youtube.com/watch?v=eMXmMHVNx4U

Implementation

We implement coverage accounting on AFL as TortoiseFuzz

We implement TortoiseFuzz for both source code and binaries

Experiment Setup

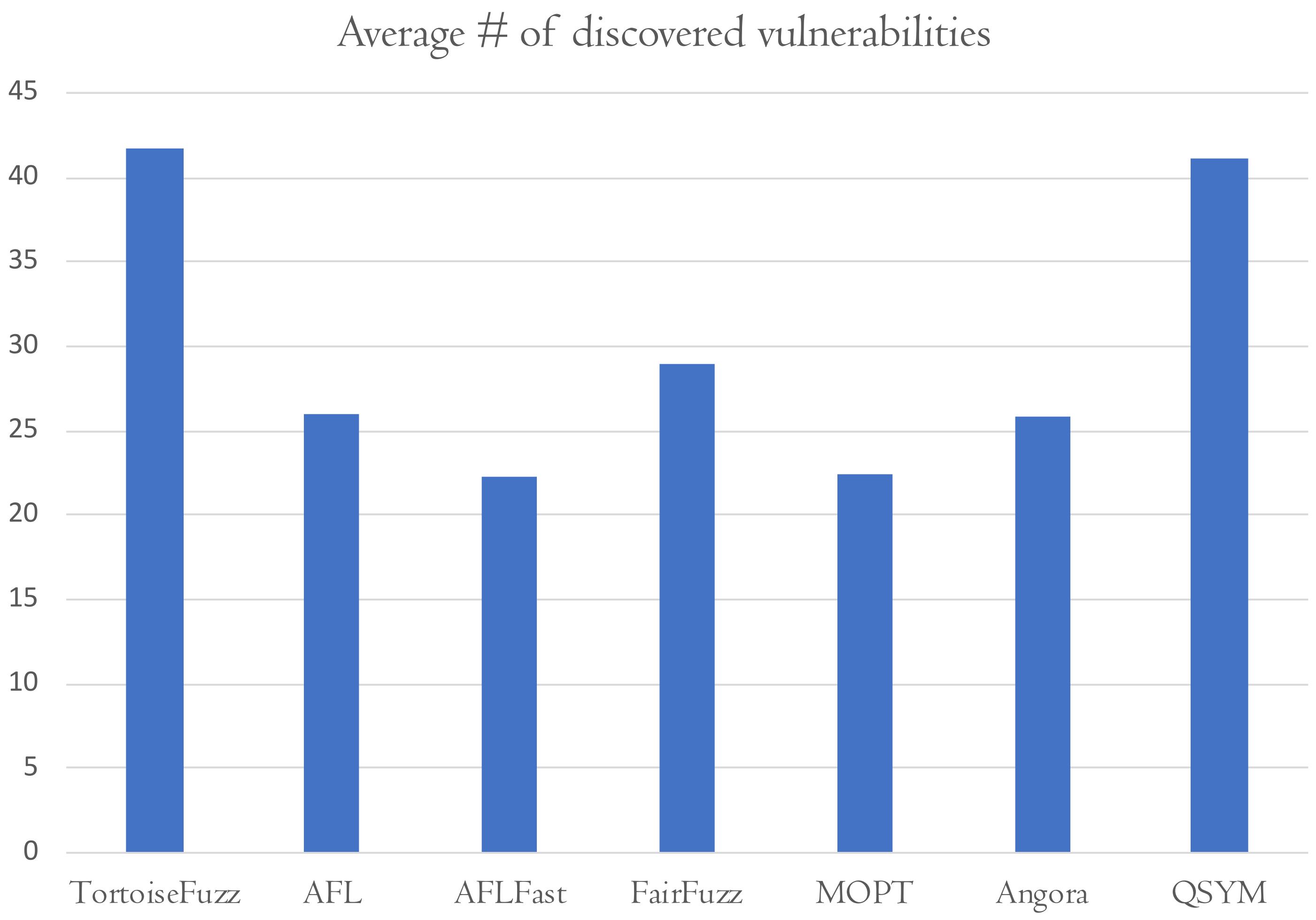
We ran TortoiseFuzz on 30 real-world programs

Each experiment lasted for 140 hours

Each experiment was done 10 times

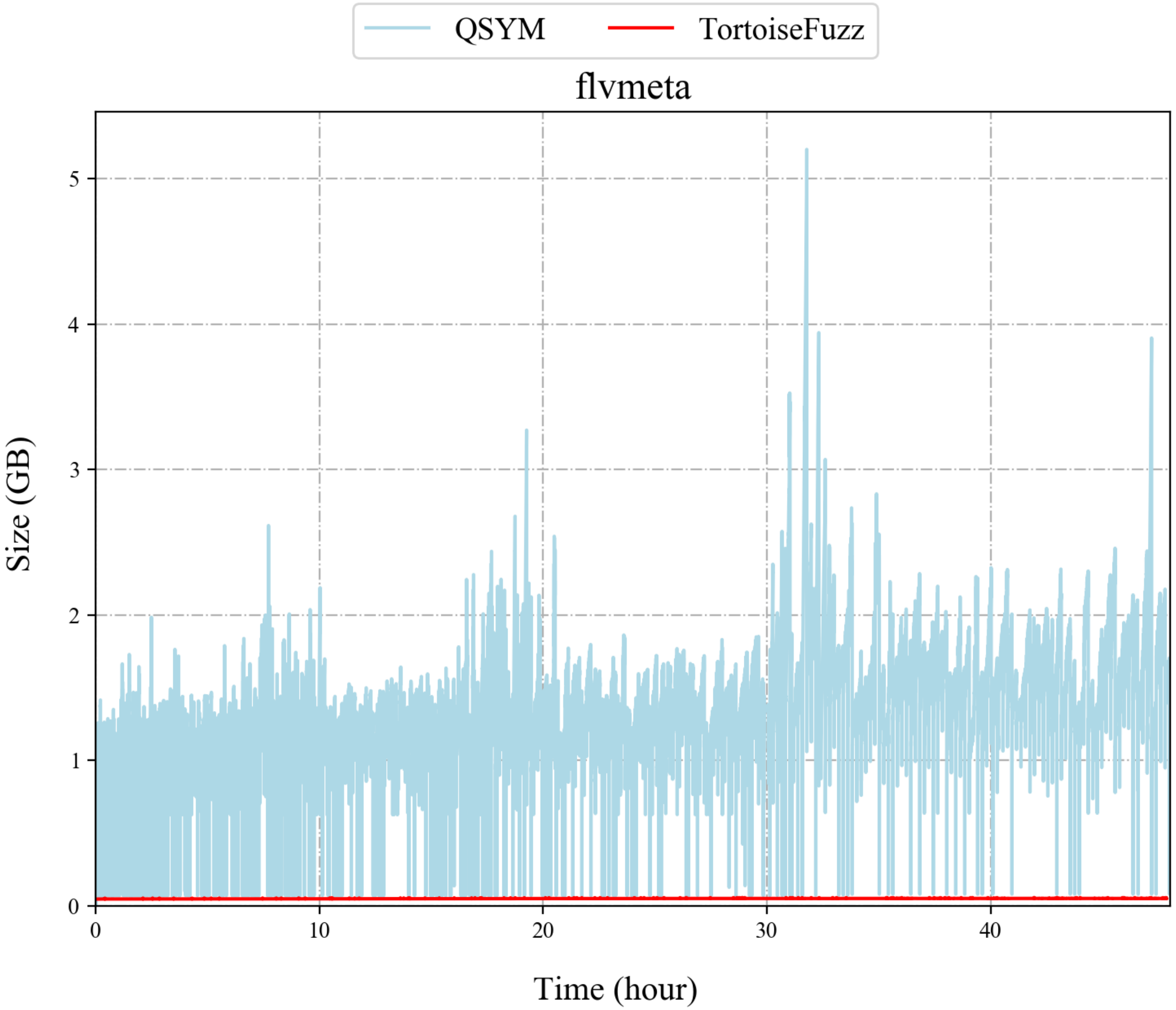
We performed Mann-Whitney U test to measure statistical significance

Vulnerability Discovery



TortoiseFuzz outperforms 5 state-of-the-art fuzzers and achieves comparable results with QSYM

Comparison with QSYM



TortoiseFuzz uses 2% of QSYM's memory usage on average

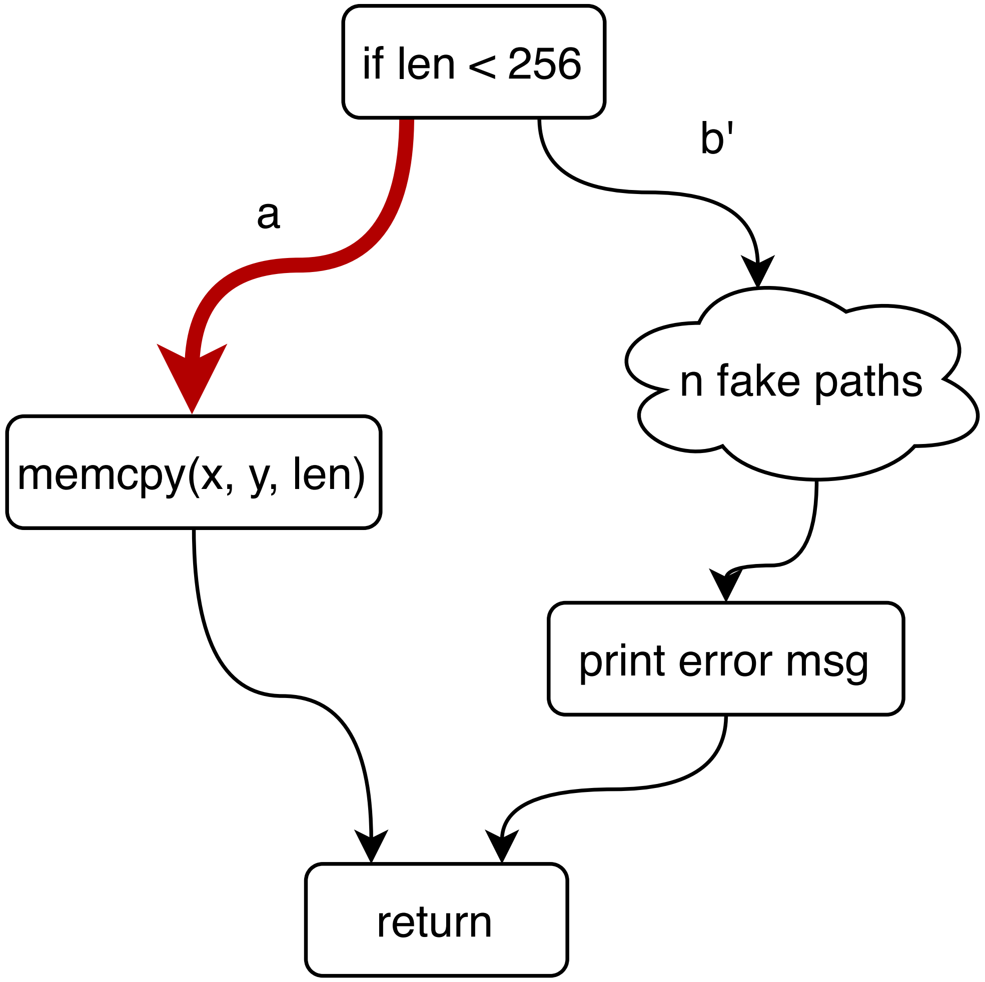
Complementary to Other Fuzzers

Coverage accounting helps improve QSYM in discovering vulnerabilities

Average # of discovered vulnerabilities	
QSYM	QSYM + coverage accounting
39.8	51.2

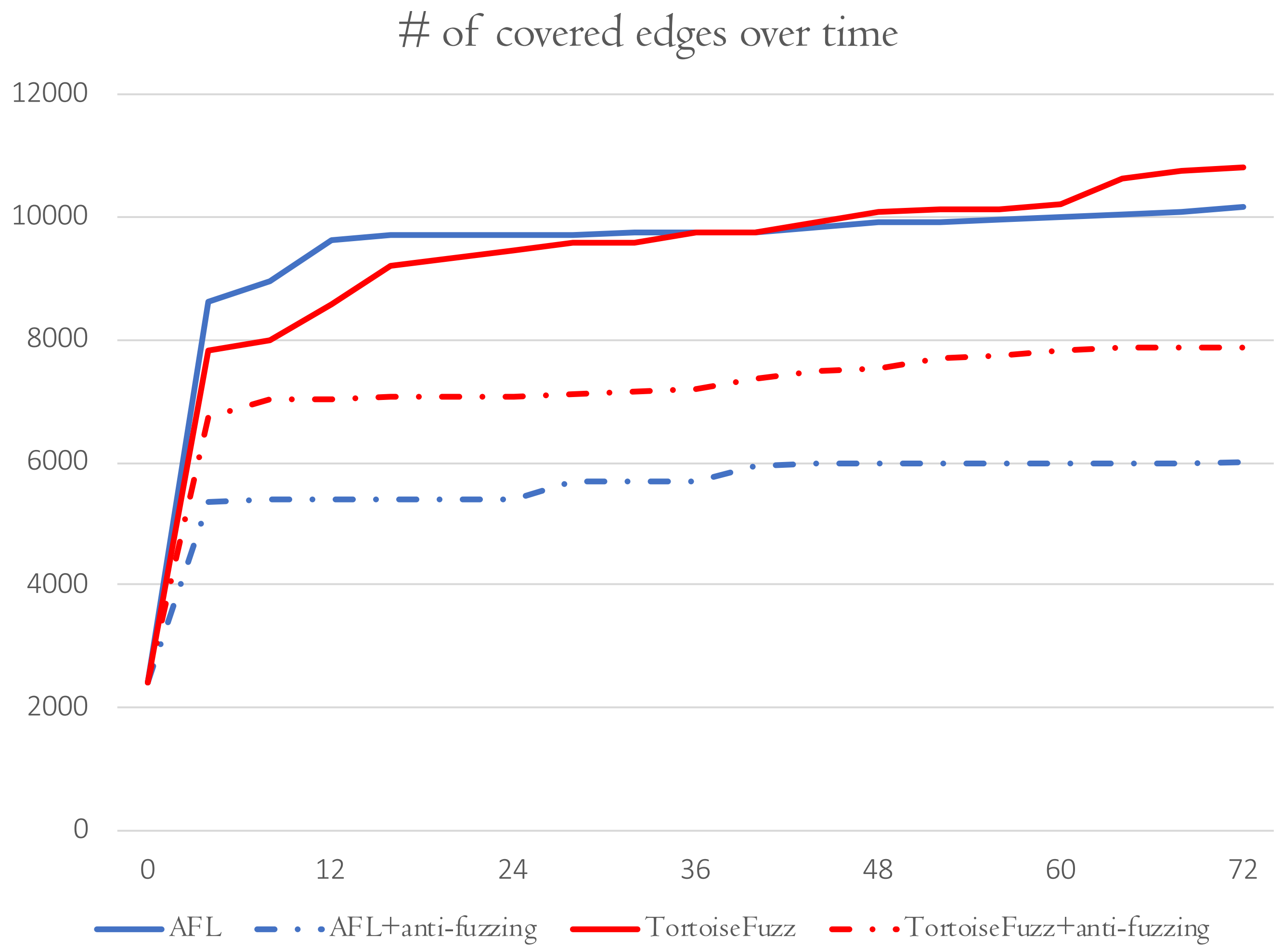
28.6% improvement

Robustness to Anti-fuzzing



Fake paths do not contain many coverage accounting info

Robustness to Anti-fuzzing



Coverage accounting metrics are more robust to anti-fuzzing

Conclusion

We propose coverage accounting which is complementary to other coverage-based fuzzers

We design and implement TortoiseFuzz, and we are going to release it at https://github.com/TortoiseFuzz/TortoiseFuzz

We evaluate TortoiseFuzz on 30 real-world programs and find 20 zero-day vulnerabilities

TortoiseFuzz outperforms 5 state-of-the-art fuzzers and achieves comparable results with QSYM with 2% of its memory usage

Not All Coverage Measurements Are Equal

Fuzzing by Coverage Accounting for Input Prioritization

Thank you!

Q & A

Kyle Zeng zengyhkyle@asu.edu