No Linux, No Problem:

Fast and Correct Windows Binary Fuzzing via Target-embedded Snapshotting

Leo Stone
leo@vt.edu

Rishi Ranjan
rishiranjan@vt.edu

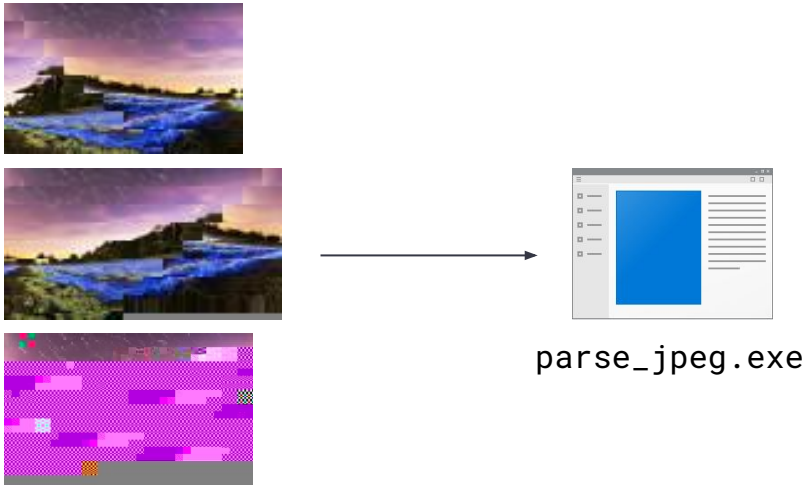
Stefan Nagy
snagy@cs.utah.edu

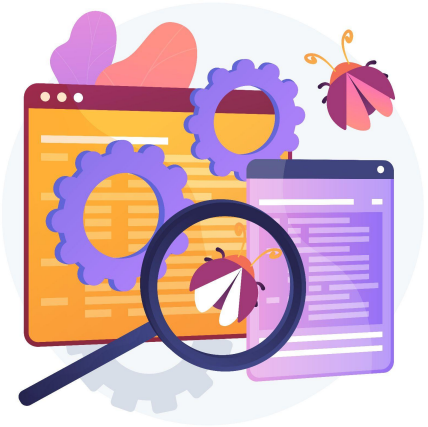
Matthew Hicks
mdhicks2@vt.edu

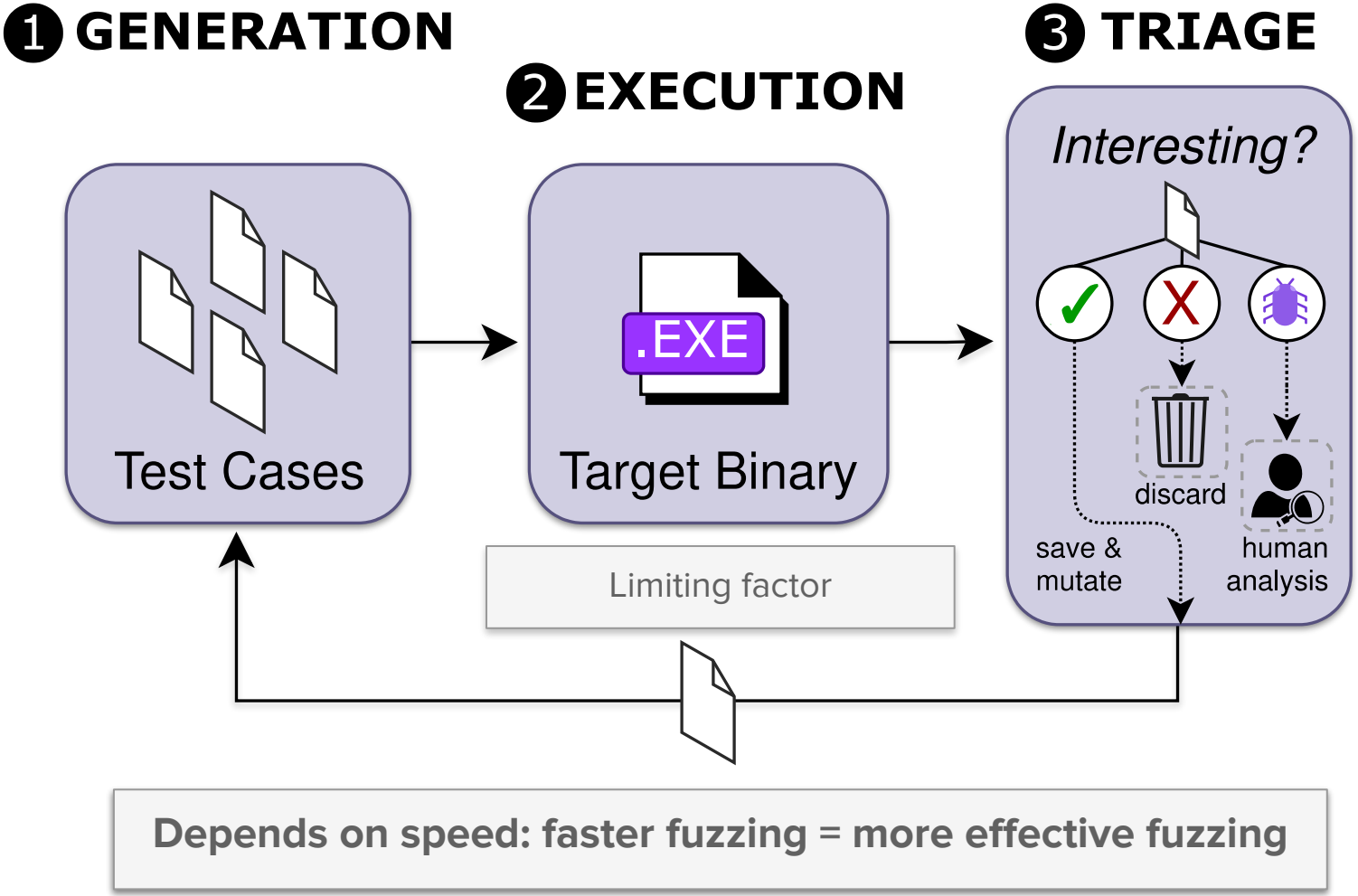
COMPUTER SCIENCE
VIRGINIA TECH

Introduction

- **Fuzzing:** most successful method for automated software testing
- Attempt to break program using **many** randomized inputs





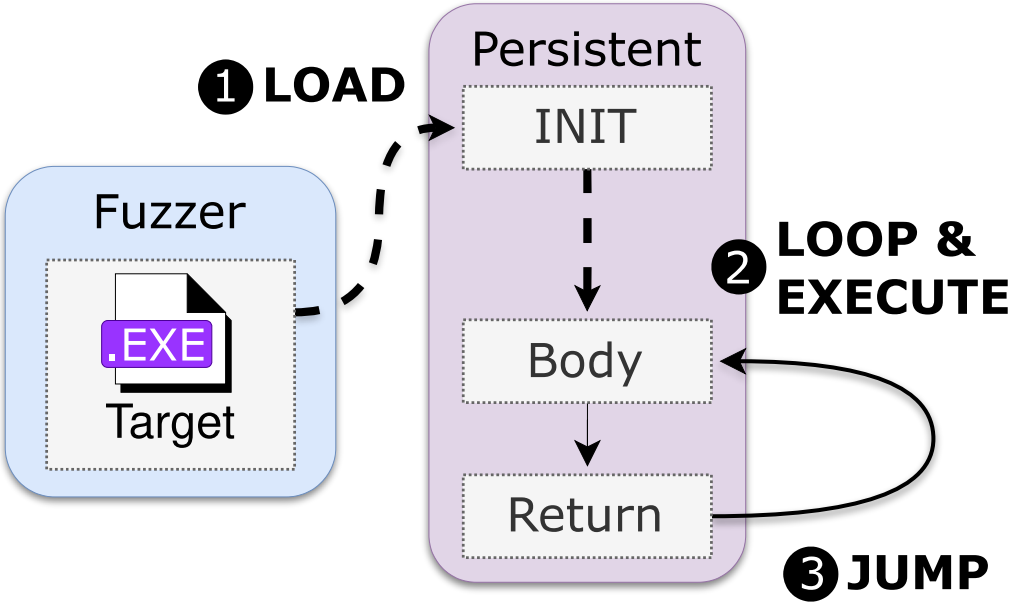

1 GENERATION
2 EXECUTION
3 TRIAGE
Interesting?
Test Cases
.EXE
Target Binary
discard
save &
mutate
human
analysis
Limiting factor
Depends on speed: faster fuzzing = more effective fuzzing

Motivation

- On Linux, execution is very fast ➜ fuzzing is effective
  - `fork()`
  - Kernel modifications
- No equivalents on Windows…
  - Windows software ecosystem is larger, but fuzzing is orders of magnitude less effective than on Linux!

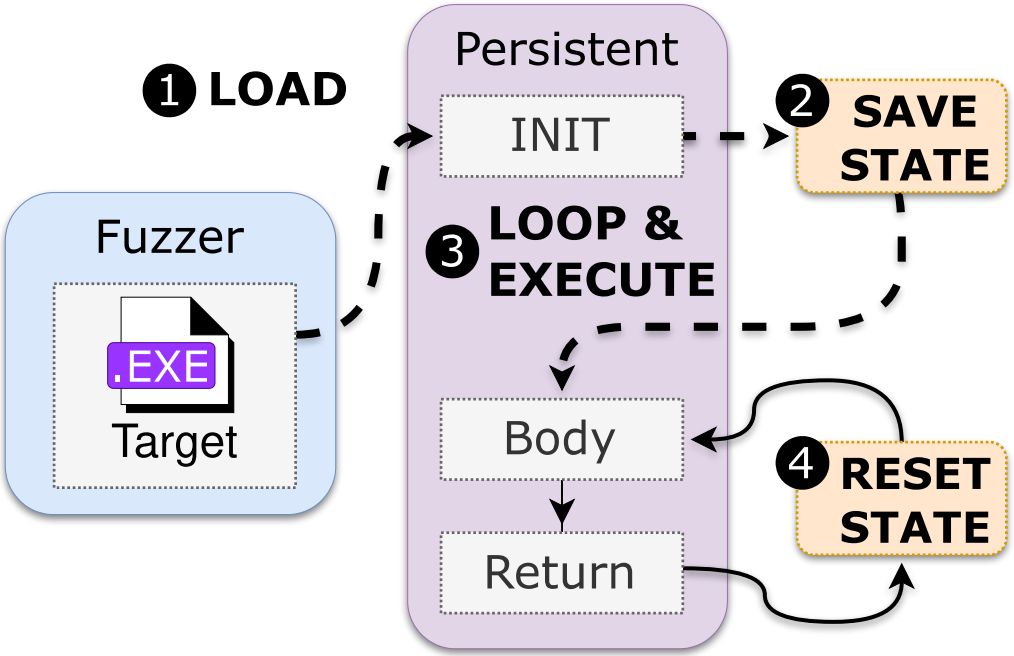
- **Challenge: without kernel support, how can we build an efficient Windows fuzzer?**

One Solution: Persistent Mode



Fast, but **incorrect**
Execution changes state…
Many binaries crash!

Our Approach



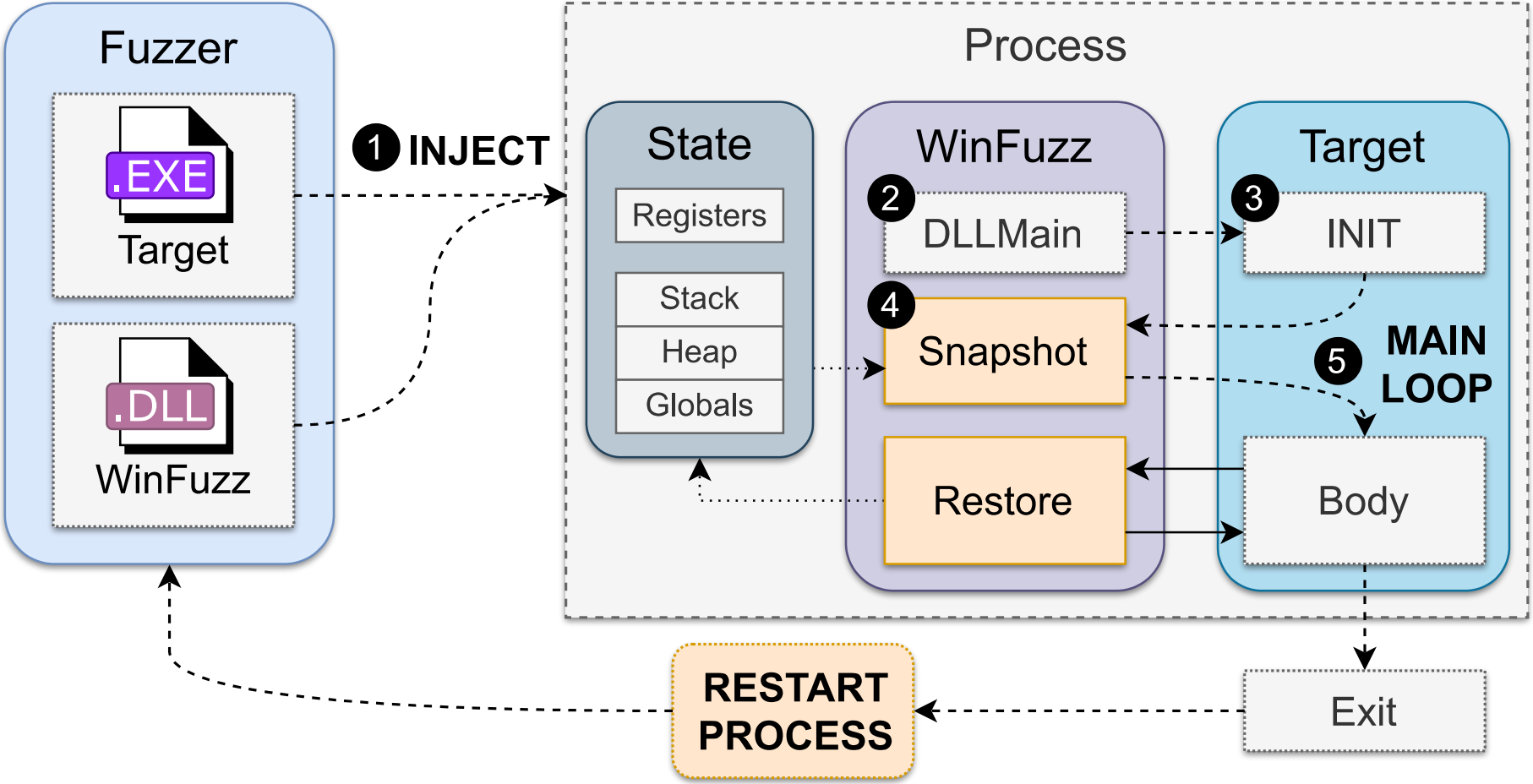
Copy state to a **snapshot**
Process restores own state
Fast like persistent mode, but correct!

Key Insight

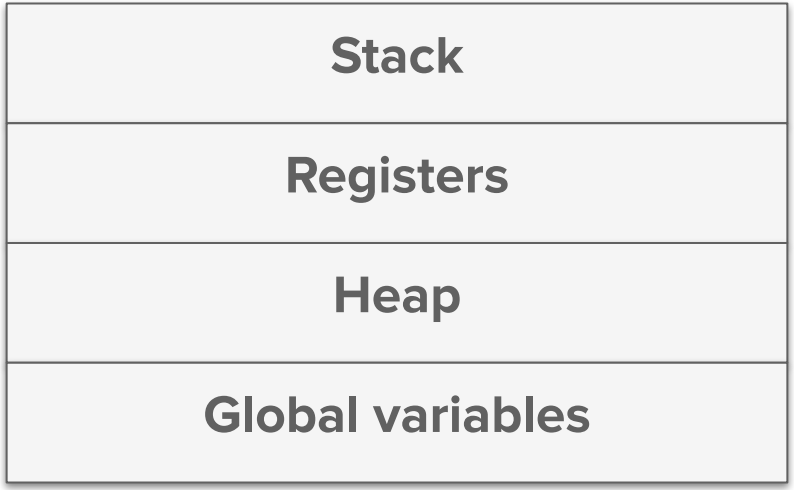
All relevant state is controllable with language-level constructs
No kernel support necessary

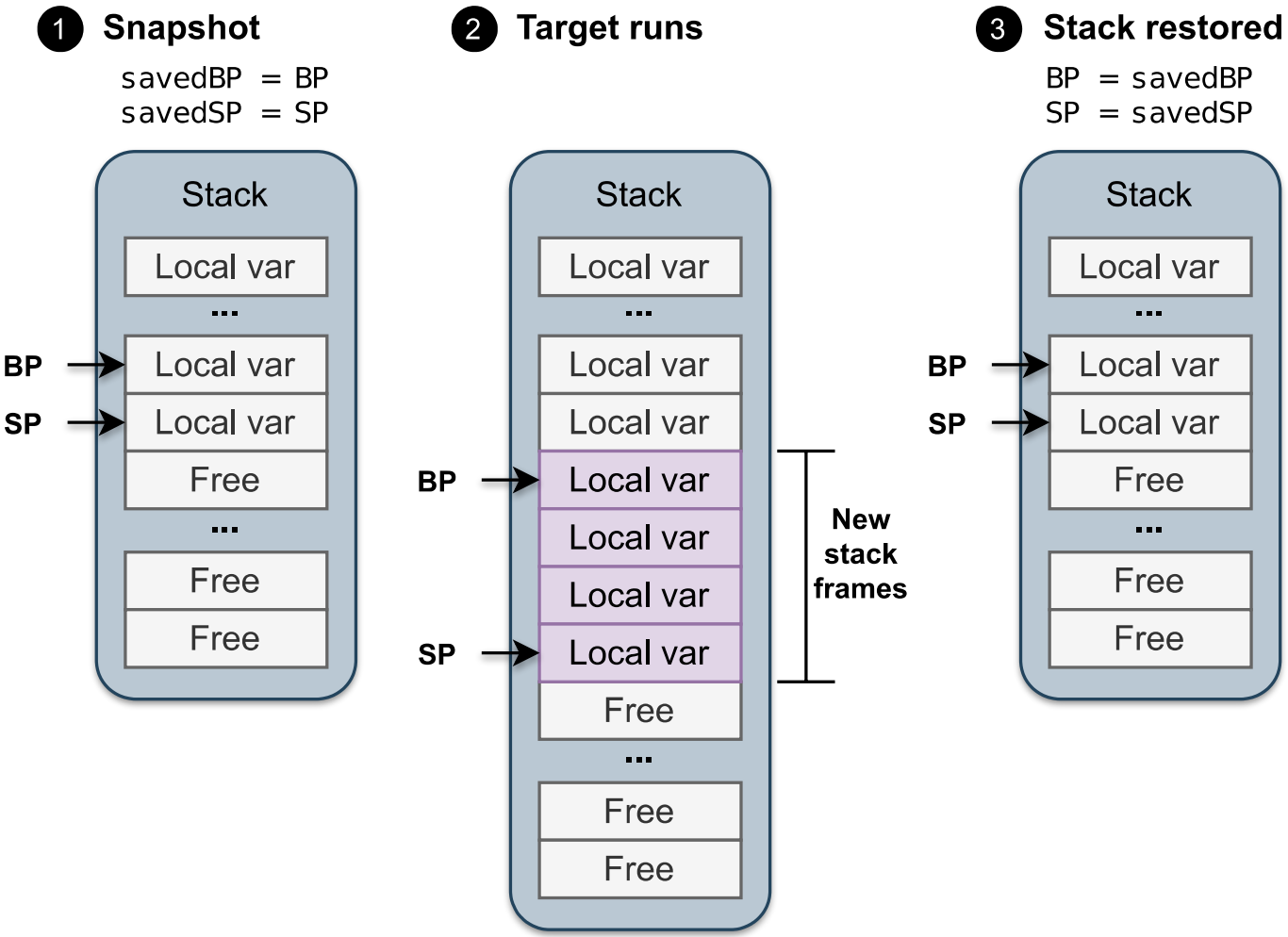
Implementation - WinFuzz

- Based on Winnie, an existing Windows fuzzer
- Main steps:
  1. DLL injection
  2. `DLLMain()` - hook setup
  3. Initializing target
  4. Taking state snapshot
  5. Main fuzzing loop: run and restore state


Fuzzer
.EXE
Target
.DLL
WinFuzz
1 INJECT
Process
State
Registers
Stack
Heap
Globals
WinFuzz
2 DLLMain
4 Snapshot
Restore
Target
3 INIT
5 MAIN LOOP
Body
Exit
RESTART PROCESS

Elements of Program State

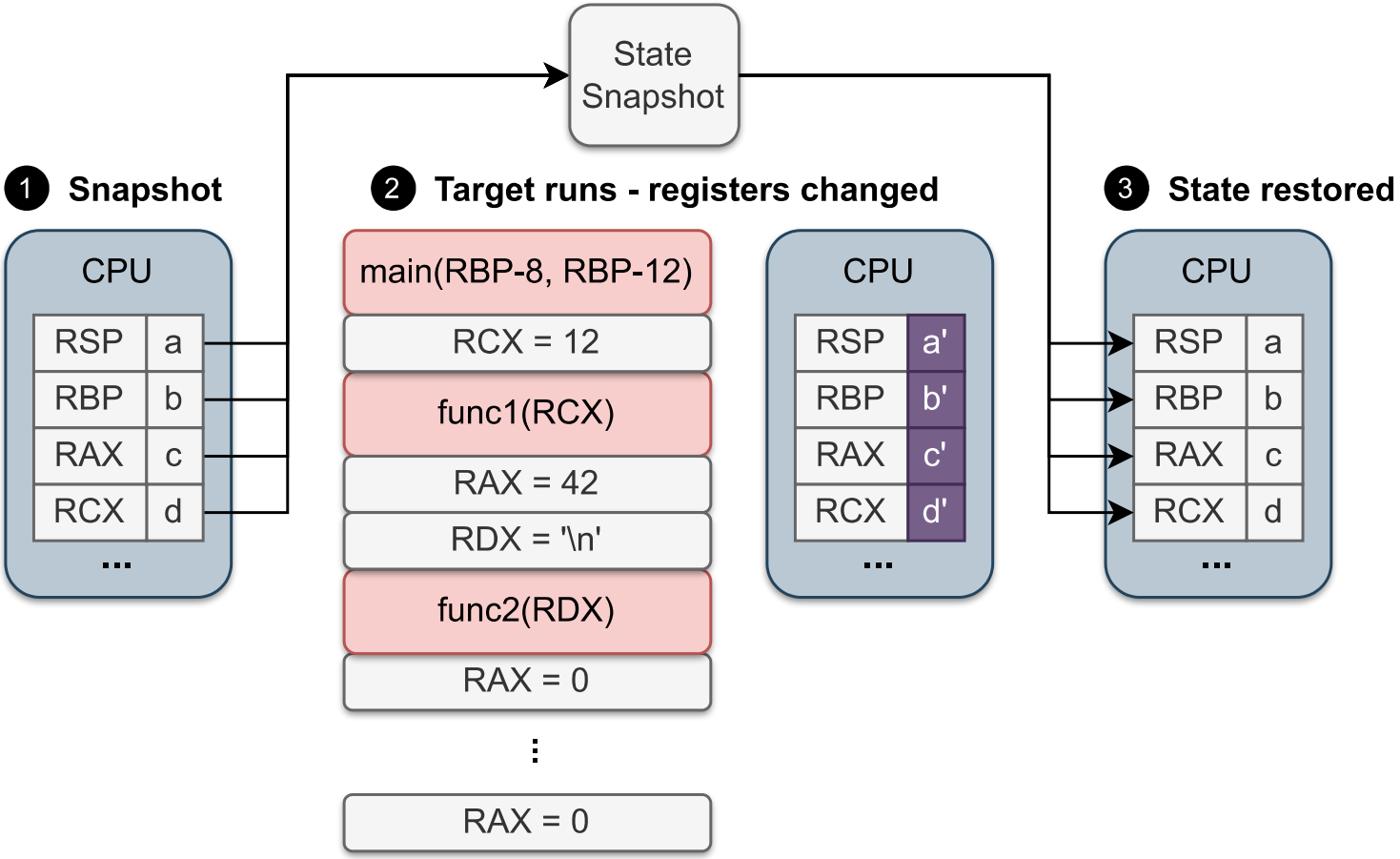



1 Snapshot
savedBP = BP
savedSP = SP
Stack
Local var
...
BP
Local var
SP
Local var
Free
...
Free
Free
2 Target runs
Stack
Local var
...
Local var
Local var
BP
Local var
Local var
Local var
SP
Local var
Free
...
Free
Free
New stack frames
3 Stack restored
BP = savedBP
SP = savedSP
Stack
Local var
...
BP
Local var
SP
Local var
Free
...
Free
Free

Resetting Registers

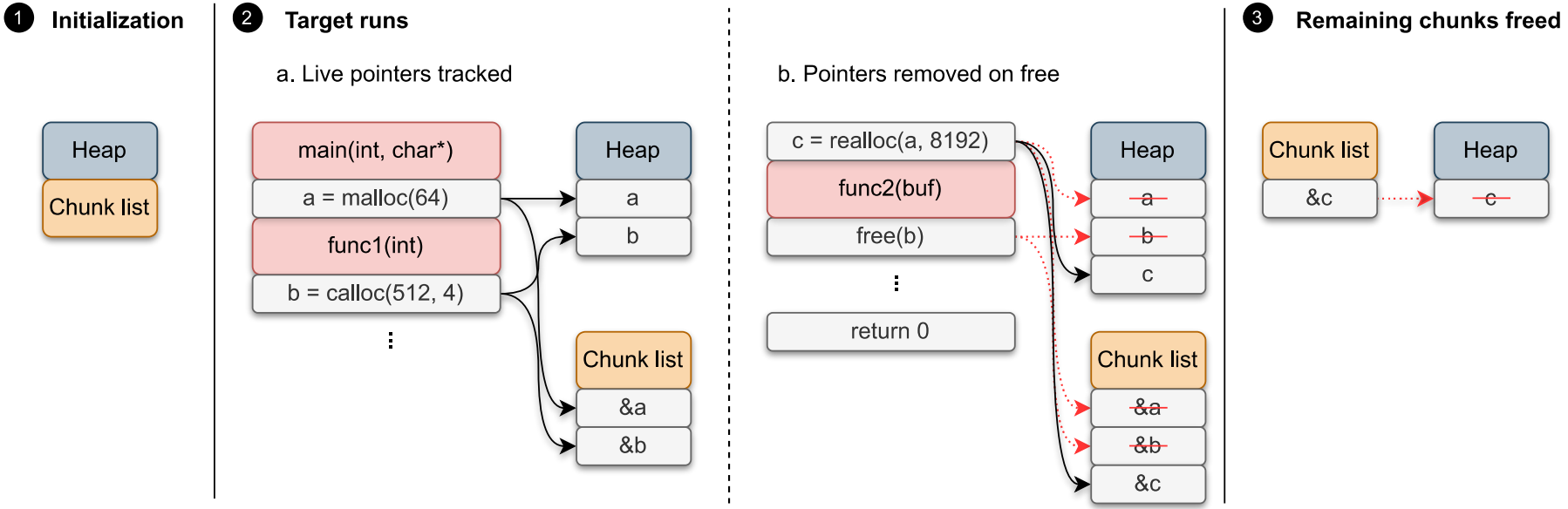
- 2 types of parameters:
  - Simple values
  - Pointers
- We reset each register to its original value

- We avoid returning from the target function and destroying any stack frames that could hold target parameters


State Snapshot
1 Snapshot
2 Target runs - registers changed
3 State restored
CPU
RSP a
RBP b
RAX c
RCX d
...
main(RBP-8, RBP-12)
RCX = 12
func1(RCX)
RAX = 42
RDX = '\n'
func2(RDX)
RAX = 0
RAX = 0
CPU
RSP a'
RBP b'
RAX c'
RCX d'
...
CPU
RSP a
RBP b
RAX c
RCX d
...

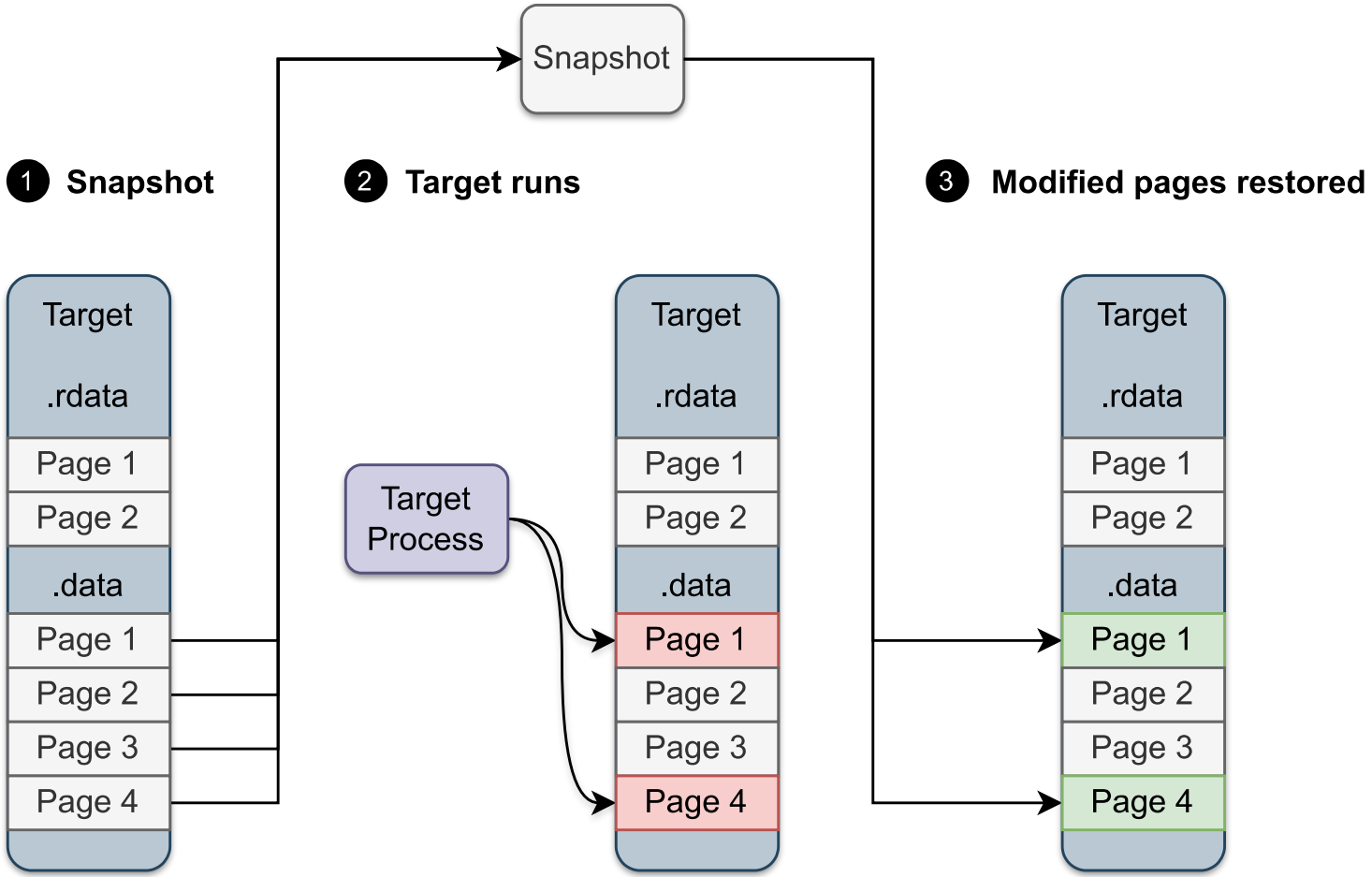
Resetting the Heap

- Memory leaks are expected - we’re trying to break the program!
- Small memory leak can cause a fuzzer crash
- We use heap API hooks to prevent memory leaks


1 Initialization
Heap
Chunk list
2 Target runs
a. Live pointers tracked
main(int, char*)
a = malloc(64)
func1(int)
b = calloc(512, 4)
Heap
a
b
Chunk list
&a
&b
b. Pointers removed on free
c = realloc(a, 8192)
func2(buf)
free(b)
return 0
Heap
a
b
c
Chunk list
&a
&b
&c
3 Remaining chunks freed
Chunk list
&c
Heap
c

Resetting Global Memory

- Copy correct starting state of all mutable global sections
- Use guard pages to track modifications
- Only restore *modified* pages


Snapshot
1 Snapshot
2 Target runs
3 Modified pages restored
Target
.rdata
Page 1
Page 2
.data
Page 1
Page 2
Page 3
Page 4
Target Process
Target
.rdata
Page 1
Page 2
.data
Page 1
Page 2
Page 3
Page 4
Target
.rdata
Page 1
Page 2
.data
Page 1
Page 2
Page 3
Page 4

Evaluation

- Criteria: correctness, performance, bug discovery
  - Versus state-of-the art: Winnie (custom forkserver) and WinAFL (process creation/persistent mode)

- Setup: Azure instances running Windows 10 Pro with single-core 2.1 GHz Intel Xeon CPUs, 3.5 GB RAM

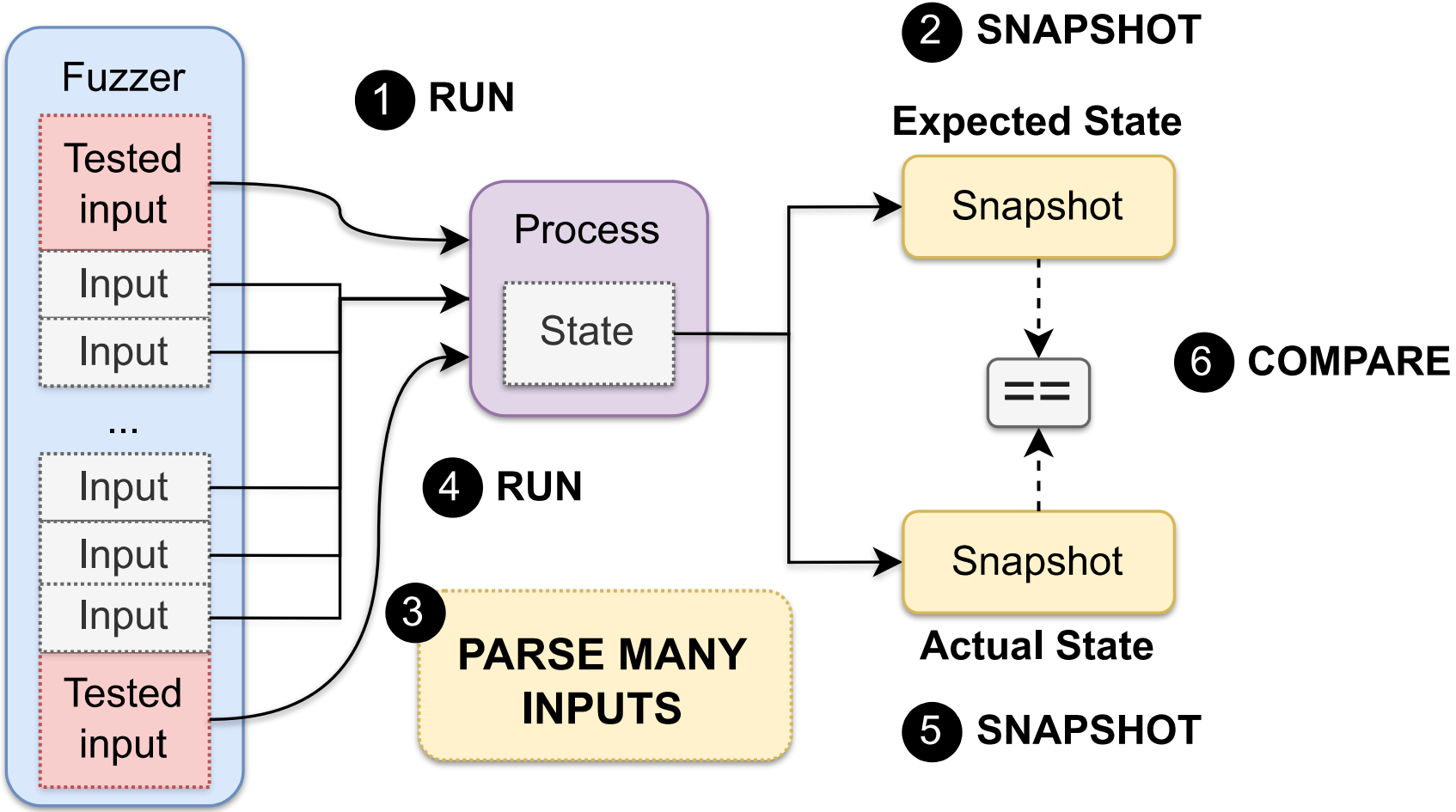
- Each fuzzing trial ran at least 5 times to collect statistically significant results
  - Mann-Whitney u-tests used to determine significance

Benchmarks

Program	WinFuzz	Winnie	WinAFL	Source	File Format	Size (KB)	Basic Blocks
tar	✔	✔	✗	Proprietary	.tar	606	30758
nconvert	✔	✔	✗	Proprietary	.png	2458	91550
freetype	✔	✔	✔	Open Source	.ttf	482	20891
audiofile	✔	✔	✔	Open Source	.wav	45	1504
flac	✔	✔	✔	Open Source	.flac	686	19292
nanosvg	✔	✔	✔	Open Source	.svg	47	1966
sqlite	✔	✔	✔	Open Source	.db	802	46758
smpdf	✔	No cov	✗	Proprietary	.pdf	3379	39816
irfanview	✔	No cov	✔	Proprietary	.png	1946	55187
jq	✔	No cov	✗	Open Source	.json	2662	13965

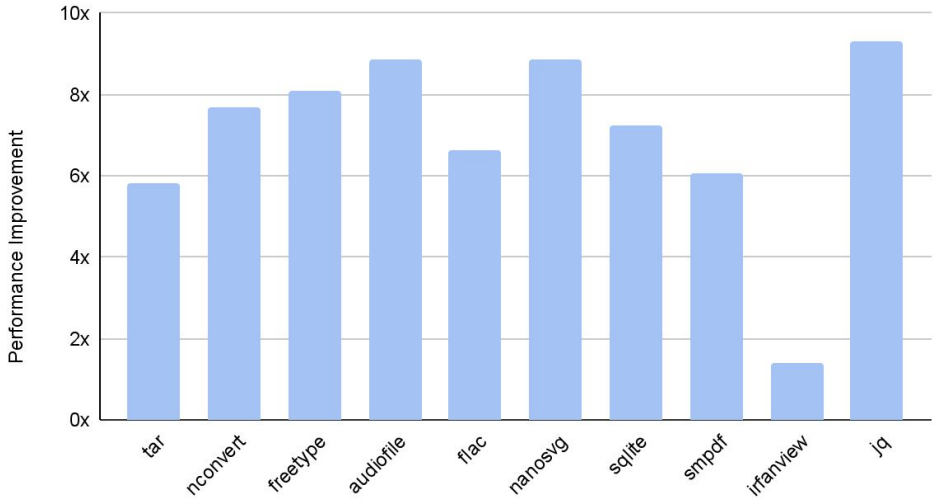
Correctness Test

- New fuzzer mode that checks for state corruption by comparing program states
- Used to test all benchmarks in corpus
- Available as part of our open source implementation
  - New users can test their own targets and saved inputs

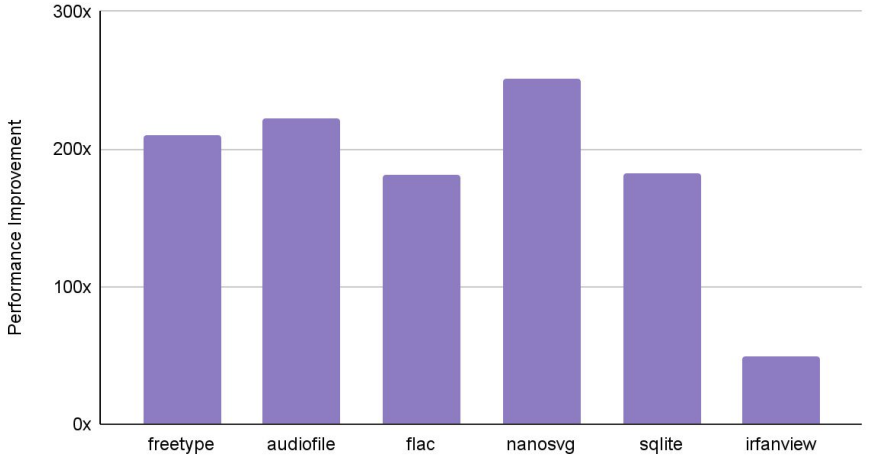

Fuzzer
Tested input
Input
Input
...
Input
Input
Input
Tested input
1 RUN
Process
State
2 SNAPSHOT
Expected State
Snapshot
==
6 COMPARE
Snapshot
Actual State
4 RUN
3 PARSE MANY INPUTS
5 SNAPSHOT

Results: Throughput

Versus Winnie



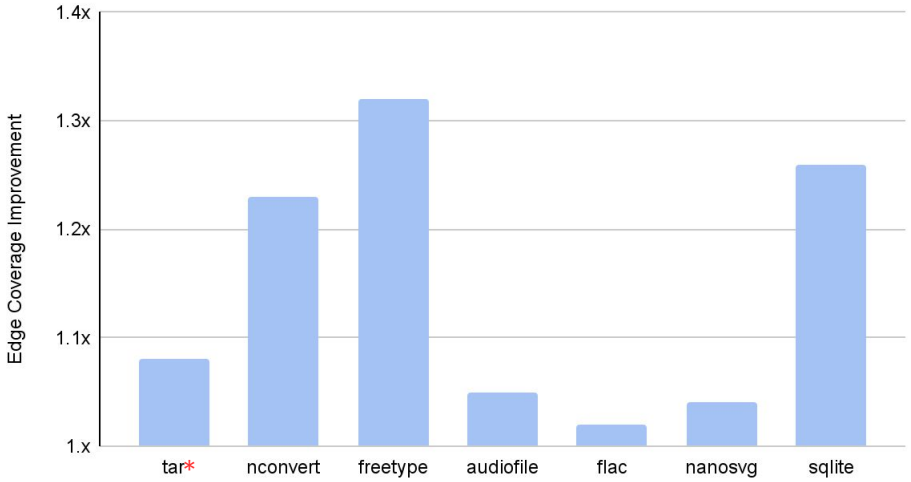
Versus WinAFL



Average improvement: 7x vs. Winnie and 182x vs. WinAFL

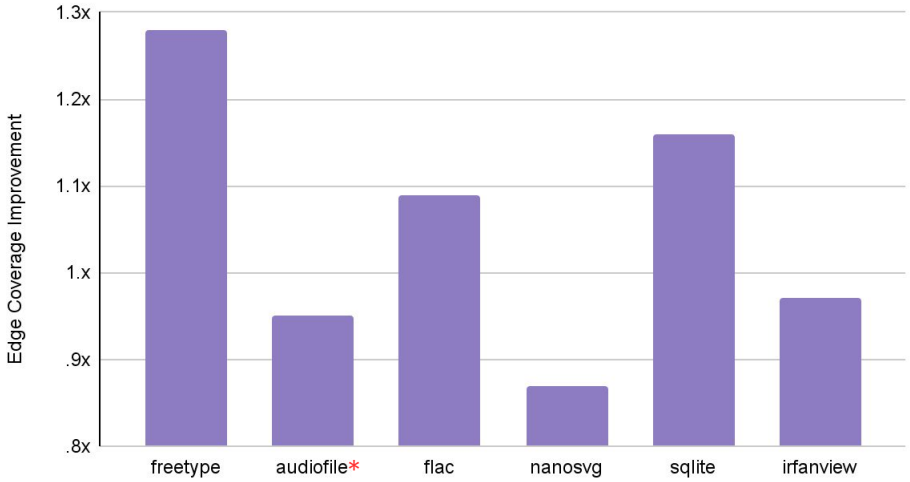
Results: Edge Coverage

Versus Winnie



*ρ=0.148

Versus WinAFL



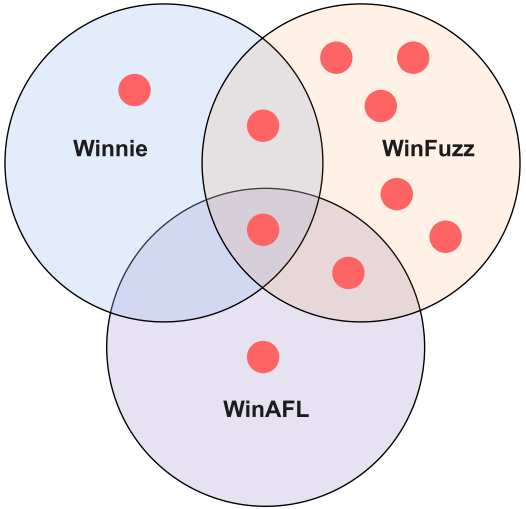
*ρ=0.216

Average improvement: 15% vs. Winnie and 5% vs. WinAFL

Bug Discovery Time

- We compared the average time taken to find specific bugs
- Some benchmarks (flac, nanosvg, audiofile) used older versions to increase bug count

- 10 unique bugs found across all fuzzers/trials
  - Winnie: 3/10
  - WinAFL: 3/10
  - WinFuzz: 8/10



Binary	Category	WinFuzz	Winnie	WinAFL
flac	Null ptr deref	12.25 s	15.6 s	243.8 s
nconvert	Illegal address	2.1 hrs	✗	✗
nconvert	Invalid free	3.6 hrs	✗	✗
nconvert	Invalid ptr deref	7.2 hrs	✗	✗
nconvert	Heap overflow	8.5 hrs	15.8 hrs	✗
nconvert	Illegal address	✗	4.5 hrs	✗
nanosvg	Stack overflow	1.4 min	✗	✗
nanosvg	Null ptr deref	1.8 min	✗	✗
nanosvg	Null ptr deref	✗	✗	21.9 hrs
audiofile	Illegal Address	12 min	✗	5.2 hrs
	WinFuzz's speedup		1.56x	23x

Undiscovered Bugs

- We ran additional experiments with WinFuzz to find 0-day bugs
- All bugs were reported to authors

Binary	Description
nconvert	2 invalid ptr reads, 3 invalid ptr writes
audiofile	Infinite loop
jhead	Invalid ptr read
flvmeta	2 invalid ptr reads
gpmf-parser	1 invalid ptr read, 1 invalid ptr write
gpmf-parser	Invalid ptr write
pdf2json	Stack buffer overrun (ntdll.dll)
pdf2json	Stack buffer overrun (pdf2json.exe)
pdf2json	Stack overflow
Total 0-day bugs:	9

Thank you!

Q&A

(Open source release pending)
github.com/FoRTE-Research

“Bugs detection” graphic by vectorjuice on Freepik