SOFTWARE RELIABILITY GROUP

Imperial College London

Sparse Symbolic Loop Execution

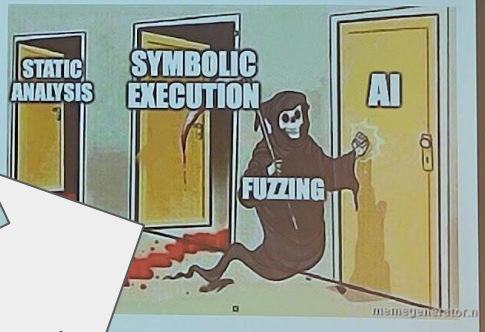
Frank Busse · Martin Nowack · Cristian Cadar

3rd International Fuzzing Workshop
16 September 2024, Vienna

Faster vs Smarter

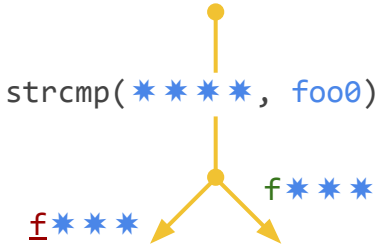
NYU

- The history of fuzzing research is littered with the wreckage of systems that thought they could beat “dumb” fuzzers
- Core problem: you have to be very, **very** smart to beat millions of exec/s
- And scaling up execs/s is a lot easier than making an analysis smarter!



ful for Fuzzing?

Symbolic Execution



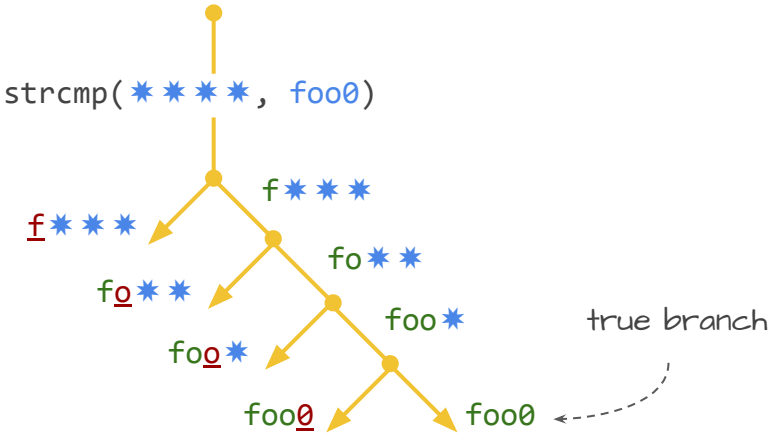
- starts with **symbolic inputs**
- aims to explore as many feasible paths as possible
- uses **SMT solver** to check path feasibility + error conditions, and create concrete inputs for selected paths

```
int strcmp(const char *l, const char *r) {
      for (; *l==*r && *l; l++, r++);
      return *(unsigned char *)l - *(unsigned char *)r;
}
```

```
if (!strcmp(s, ”foo”))
      …
```

✱ unconstrained byte
f byte is not ‘f’
f byte is ‘f’

Symbolic Execution



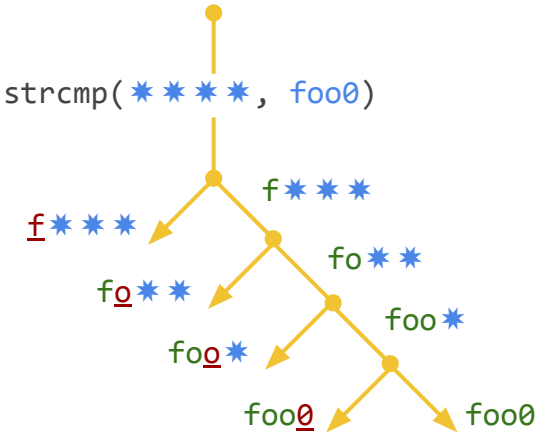
- 1 + 4 branches to explore
- loops contribute to **path explosion**

```
int strcmp(const char *l, const char *r) {
      for (; *l==*r && *l; l++, r++);
      return *(unsigned char *)l - *(unsigned char *)r;
}
```

```
if (!strcmp(s, ”foo”))
      …
```

✱ unconstrained byte
f byte is not ‘f’
f byte is ‘f’

Sparse Symbolic Loop Execution



Idea: pick “**interesting**” **paths** and ignore the rest.

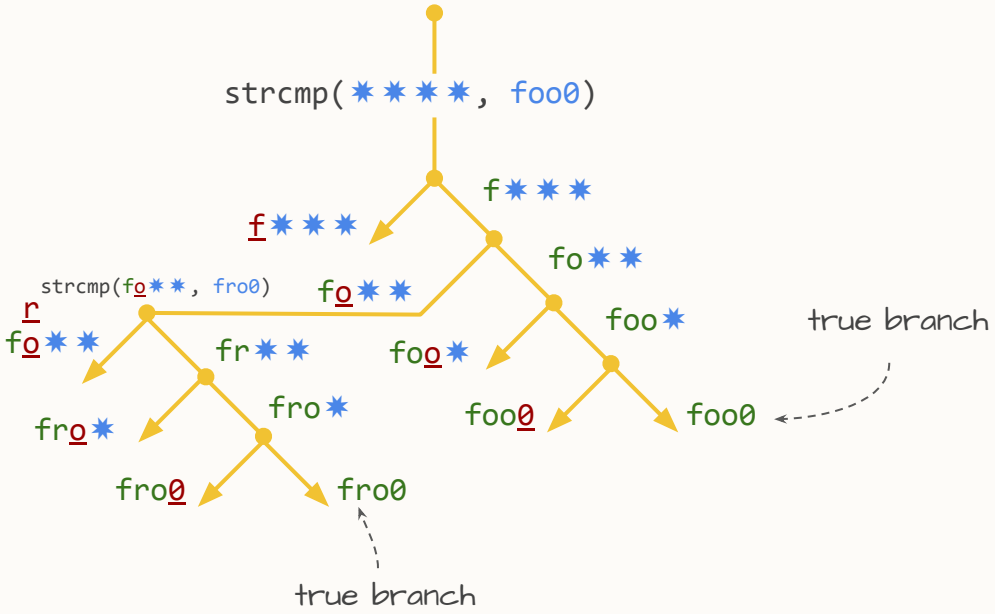
Goal: lose as little coverage as possible.

```
int strcmp(const char *l, const char *r) {
      for (; *l==*r && *l; l++, r++);
      return *(unsigned char *)l - *(unsigned char *)r;
}
```

```
if (!strcmp(s, ”foo”))
      …
```

✱ unconstrained byte
f byte is not ‘f’
f byte is ‘f’

Sparse Symbolic Loop Execution



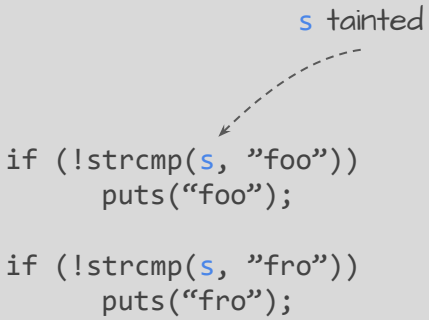

```
if (!strcmp(s, ”foo”))
      puts(“foo”);

if (!strcmp(s, ”fro”))
      puts(“fro”);
```

✱ unconstrained byte
f byte is not ‘f’
f byte is ‘f’

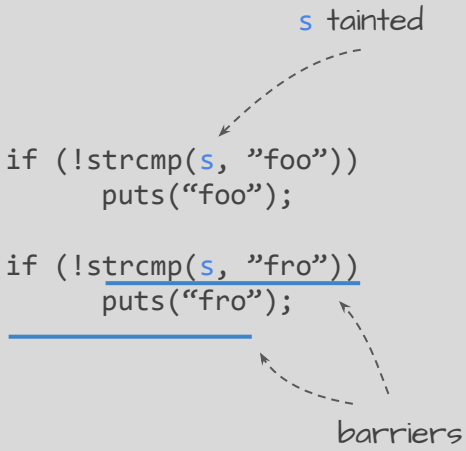
Sparse Symbolic Loop Execution

- statically "**taints**" all values that could be affected by a loop



Sparse Symbolic Loop Execution

- statically "**taints**" all values that could be affected by a loop
- computes **loop-impact barriers**, where no relevant decision points can be reached anymore

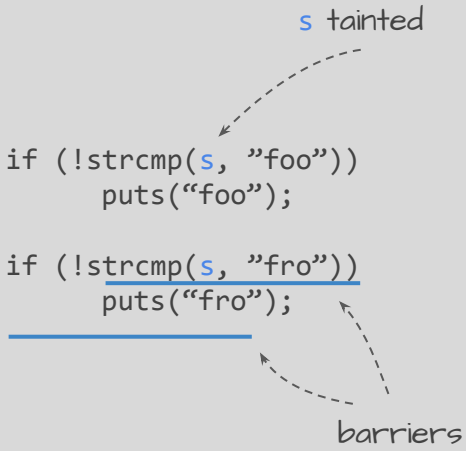


Sparse Symbolic Loop Execution

- **tracks** behaviour (branch) at decision points up to barrier
- **filters** states (paths) at barriers according to the uniqueness of their behaviour at relevant decision points

Path	if_{foo}	if_{fro}
foo0	true	false
fro0	false	true
f✱✱✱, ..., foo0	false	false

keep only **one** or **sample** e.g. 1st, 2nd, 4th, ...



Research Questions

RQ1: Is SSLE an **effective** approach to postpone or filter states, thereby reducing path explosion?

RQ2: How does SSLE compare to **less complex approaches**?

Planned Evaluation

Prototype

- **SparKLE** implemented on top of KLEE (https://klee-se.org/)

Benchmarks

- ~50 benchmarks (Binutils, Coreutils, diff, gawk, gcal, gmake, gzip, libsndfile, libtiff, libxml)
- 1hr, 4GiB memory limit for symbolic executor

Planned Evaluation - Efficacy

RQ1

- comparison (coverage) against KLEE
- **DFS** search heuristic, several combinations of configuration flags

  tainting thresholds ✕ filter strategies ✕ state revival rate

  tainting threshold taint *x* functions along call stack and *y* functions down call graph
  filter strategy either keep only one witness or use bucketing approach (1, 2, 4, 8, ...)
  revival rate select *n%* of states from postponed set

- **rndcov** search heuristic, 2 “best” DFS configurations, random subset of 10 benchmarks, 100 repetitions

Planned Evaluation - Lightweight Approaches

RQ2

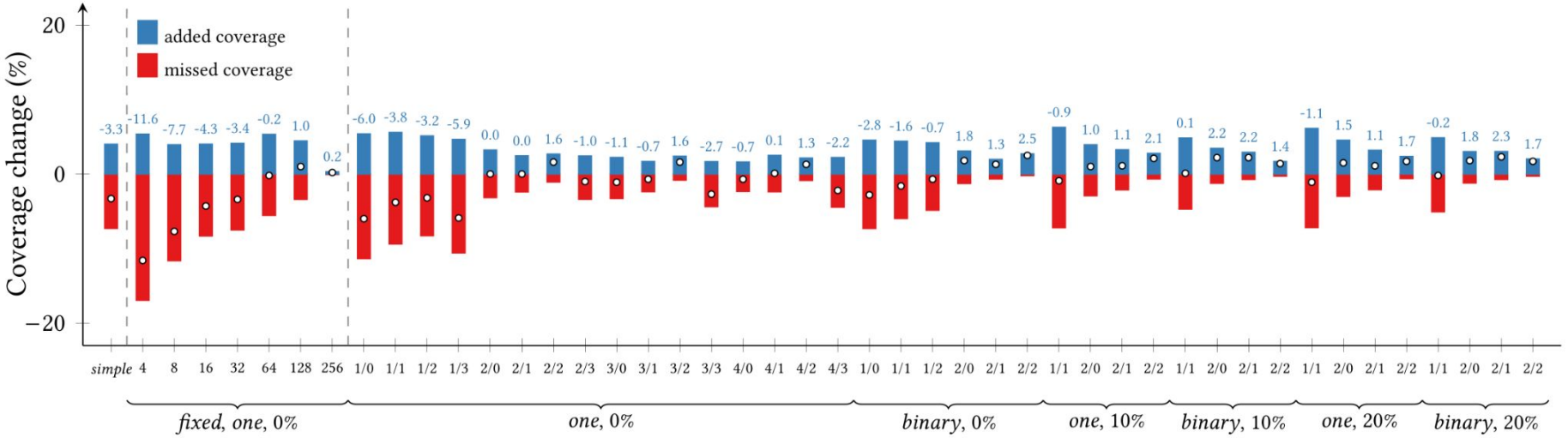
Fixed Decision Points

- **no taint analysis**, just (configurable) fixed number of observed decision points
- proposed decision points limits: 1, 2, 4, 8, 16, 32, 64, 128, 256
- DFS search heuristic

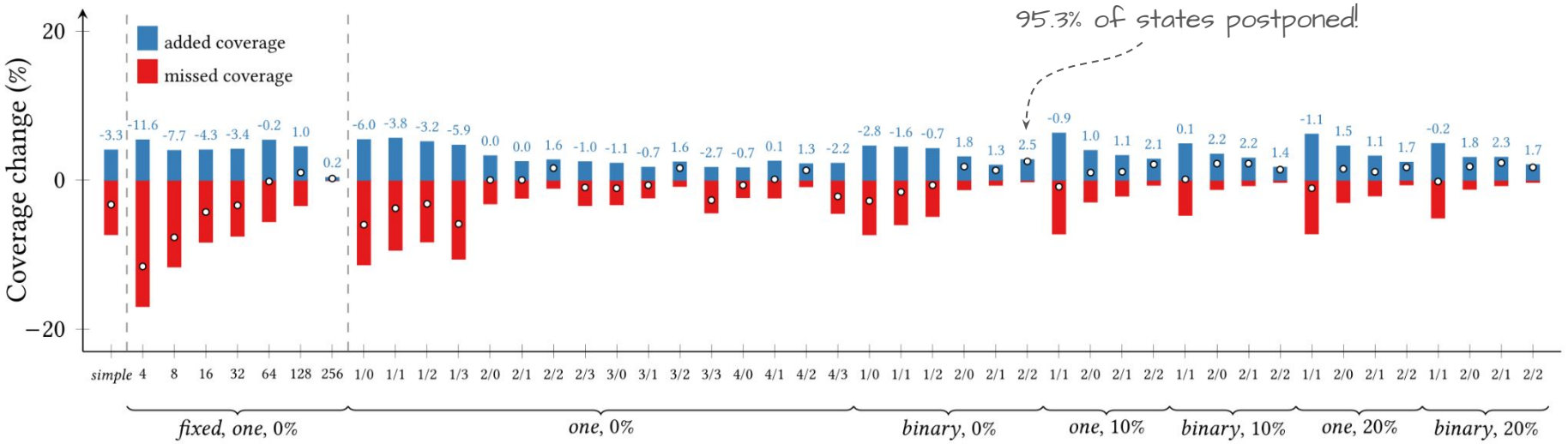
Simple

- **no taint analysis, no decision point tracking**
- only sample paths based on iteration count (0, 1, 2, 3, 4, 8, 16, ..., n)
- DFS search heuristic

Preliminary Results - Relative Coverage (DFS)



Preliminary Results - Relative Coverage (DFS)

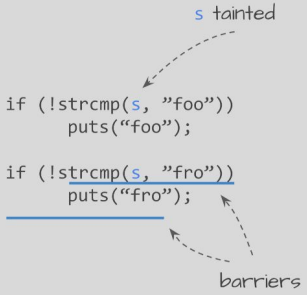


Sparse Symbolic Loop Execution

- **tracks** behaviour (branch) at decision points up to barrier
- **filters** states (paths) at barriers according to the uniqueness of their behaviour at relevant decision points

Path	if_{foo}	if_{fro}
foo0	true	false
fro0	false	true
f***, ..., foo0	false	false

keep only one or sample e.g. 1st, 2nd, 4th, ...



klee-se.org

	Loops			Symbolic branches
	detected	reached	symbolic	
min	102	38	4	49
mean	373	76	14	665,711
max	1,517	238	27	7,097,544