FINE-GRAINED MEMORY OBJECT REPRESENTATION IN SYMBOLIC EXECUTION

MARTIN NOWACK
M.NOWACK@IMPERIAL.AC.UK
char * a = malloc(1024);
int32 i = 10;
a[i]++;
if (i != 12345)
{
    a[i-2] = a[i] * 2;
} else {
    a[i+2] = a[i] - 2;
}
char * a = malloc(1024);
int32 i = 10;

a[i]++;
if (i != 12345) {
    a[i-2] = a[i] * 2;
} else {
    a[i+2] = a[i] - 2;
}
char * a = malloc(1024);
int32 i = 10;

a[i]++;
if (i != 12345) {
    a[i-2] = a[i] * 2;
} else {
    a[i+2] = a[i] - 2;
}
char * a = malloc(1024);
int32 i = symbolic;
a[i]++;
if (i != 12345)
{
    a[i-2] = a[i] * 2;
} else 
{
    a[i+2] = a[i] - 2;
}
STATE – A SIMPLIFIED VIEW

- Path Constraints
- Registers (i.e., program counter)
- Allocated Memory
  - Stack-local
- Heap
THE MANY STATES...
GOAL

- Scale symbolic execution
- Avoid premature termination of states
- Sort/Reason about states
Copy on Write (CoW)
MALLOC(1024)

store(2, 7)
HANDLING SYMBOLICS

store(sym_7, 7)

load(5)
FINE-GRAINED MEMORY OBJECT REPRESENTATION
INSIGHT I:
CHANGES ARE (OFTEN) SMALL;
SHARE COMMON PARTS
INSIGHT II: CHANGES ARE (OFTEN) LOCAL AND OF SIMILAR TYPE
EVERYTHING IS A LAYER

BASICS
OPTIMISATIONS
INDEX-BASED ACCESS

MALLOC(1024)

Oldest

Most recent

load(2) \rightarrow A
load(1) \rightarrow 0
IN-PLACE UPDATE

MALLOCD(1024)

write(2,B)
write(1, 0)
LAYER INVALIDATION

MALLOCS(1024)

write(2,0)

S1

S3
HANDLING SYMBOLIC INDICES

(Symbol1, 5); (Symbol2; A)

Symbolic index layer
**OPTIMISATION**

**LAYER TYPES**

![Diagram showing layer types with MALLOC(1024)]

---

**Allocated Space**

- ~10 byte
- Initialised bytes: `sizeof() * 1bit`
- Map: index -> value
EVALUATION
BENCHMARKS

GNU Coreutils vs. MEMORY

Depth-First

Search Strategies

Random + Target Uncovered

Breadth-First
RQ1: Changes in Execution Time
WALLTIME - DEPTH FIRST SEARCH

Walltime (min)

Application

KLEE

Memory
WALLTIME - BREADTH FIRST SEARCH

Walltime (min)

Application

KLEE

Memory
RQ2: CHANGES IN MEMORY CONSUMPTION
MEMORY USAGE - BREADTH FIRST SEARCH

Memory Usage (MB)

Application

KLEE

Memory

3750
SUMMARY

THIS RESEARCH HAS BEEN SUPPORTED BY: UK EPSRC VIA GRANT EP/N007166/1, EP/R011605/1
OBJECT STATE HASHING

\[ HS := I_1 \oplus V_1 \oplus \ldots \oplus I_n \oplus V_n \]

\[ HS_{\text{prev}} := 0, 0, 0, 0 \]

\[ HS := HS_{\text{prev}} \oplus 0 \oplus A \]

\[ HS_2 := \ldots \]

\[ 0, 0, 0, 0 \]

\[ A, A, A, A \]

\[ 0, 0, 0, 0 \]