ISP RAS

Towards Symbolic Pointers Reasoning in Dynamic Symbolic Execution

Daniil Kuts

September 24, 2021

ISP RAS

arxiv.org/abs/2109.03698

Motivation

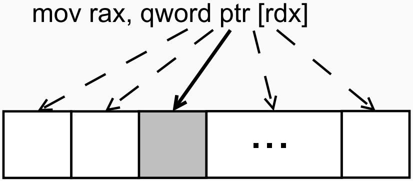
Example of the program branch with symbolic address dependency:

```c
int table[6] = {3, 7, 14,
                0, 5, 11};
int foo(int a)
{
  int res = table[a];
  if (res == 5)
  {
    abort();
  }
  return res;
}
```

```
780: lea    rax,[rip+0x200899]
787: movsxd rdi,edi
78a: mov    eax,DWORD PTR [rax+rdi*4]
78d: cmp    eax,0x5
790: je     794 <foo+0x14>
792: repz ret
794: sub    rsp,0x8
798: call   5b0 <abort@plt>
```

Symbolic Addresses Processing

- Process only symbolic loads
- Determine approximate symbolic address bounds
- Model memory access with SMT-formula and assign it to the result of load operation



Address Bounds Reasoning

- Select a memory region of a constant size around current symbolic address value
  - easiest way, low accuracy
- Binary search with SMT-solver for lower and upper bounds
  - the most accurate method, but multiple solver invocations affect performance
- Lower bound can be retrieved from the symbolic address AST
  - fast and accurate method, but sometimes heuristic may fail
- Upper bound can be retrieved from the index validation in previous basic block
  - index validation is uncommon for the memory table accesses

Address Bounds Reasoning

Analysing symbolic address AST to determine lower bound:

- Symbolized addresses are consist of concrete and symbolic parts
- Concrete part is the base address of the table in memory, i.e. the lower bound of memory access
- Symbolic part is an offset in the table, that depends on user input
- Sometimes base address may be computed from several concrete values

```
1733f movsxd rax, dword ptr [rdx + rax*4]
AST = (bvadd ref!1519 (bvmul ref!1506 (_ bv4 64)))
    ref!1519 -> (bvadd (_ bv895833 64) (bvadd ref!1518 (_ bv7 64)))
    ref!1518 -> (ite (= ref!1516 (_ bv1 1))
                    (_ bv140737283085843 64)
                    (_ bv140737283085112 64))
        ref!1518 evaluated as (_ bv140737283085843 64)
base address = 895833 + 140737283085843 + 7 = 0x7ffff3d18173
```

Modeling Memory Access: Nested ITE Tree

Build SMT-formula that establish dependencies between the possible symbolic address values and memory values.

- Iterate over all assumed memory region and build nested if-then-else (ITE) tree
- Merge nodes which lead to the same memory value
- Use current concrete memory value if solver could pick symbolic address outside the reasoned memory bounds

$sym \leftarrow symbolic\_address$
if $sym == a1$ **then** $value\_1$
else
  if $sym == a2 \vee sym == a3$ **then** $value\_2$
  else
    if $sym == a4$ **then** $value\_4$
    else $current\_value$
  end
end

Modeling Memory Access: Binary Search Tree

```
sym ← symbolic_address
if sym < 0x300 then
    if sym < 0x100 then
        current_value
    else
        if sym == 0x100 then
            value_1
        else  value_2
    end
end
else
    if sym < 0x400 then  value_2
    else
        if sym == 0x400 then
            value_3
        else  current_value
    end
end
```

- Build a binary search tree over all possible symbolic address values
- For the addresses outsde the assumed memory bounds use current memory value

Modeling Memory Access: Linearization

Originally proposed for modeling memory accesses by Mayhem

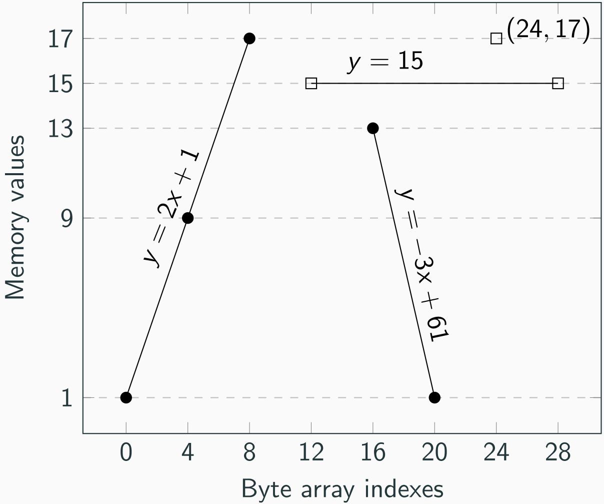
- Optimize binary search tree by merging several nodes with linear function
- Represent memory region as a set of points on index/value plot and draw lines through the consequent points
- Process symbolized memory values separately: build nested ITE tree for them and prepend it before the linearized BST

`Index = concrete address - lower bound`

index	0	4	8	12	16	20	24	28
memory value	1	9	17	15	13	1	17	15

S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley. Unleashing Mayhem on binary code

Linearization



Linearization

- Build a binary search tree over set of points and lines
- Prepend BST with nested ITE tree built for:
  - symbolized memory cells
  - horizontal lines for some points

```
sym ← symbolic_address − lower_bound
if sym == 12 ∨ sym == 28 then 15
else
    if sym < 24 then
        if sym < 16 then 2 ∗ sym + 1
        else −3 ∗ sym + 61
    end
    else
        if sym < 32 then 17
        else current_value
    end
end
```

Memory Access Modeling Methods Comparison

Application	Z3			Yices2			Bitwuzla		
	LIN	ITE	BST	LIN	ITE	BST	LIN	ITE	BST
cjpeg	**1m7s**	1m16s	1m14s	**1.2s**	2s	2.6s	30.8s	**7s**	20.6s
eperl	**11.4s**	12.4s	20.3s	4.8s	**2.4s**	2.9s	17s	19.5s	**13.5s**
hdp	**18s**	27.5s	19.9s	**1.7s**	2.1s	2.5s	**8.2s**	17s	13s
jasper	**8.2**s	8.8s	9.8s	**1.4s**	**1.5s**	**1.5s**	**5.1s**	5.5s	5.7s
libcbor	**11.5s**	T/O	17.4s	**0.8s**	1s	1.3s	**2.8s**	6.4s	4.3s
libxml2	**4.2s**	**4.4s**	5.1s	**0.9s**	1.3s	1.5s	**3s**	8.8s	7.3s
minigzip	**0.9s**	4.4s	8.6s	**0.2s**	1.4s	1.9s	**0.5s**	10.1s	9s
muraster	**4.2s**	4.6s	5.8s	1.2s	**1s**	1.2s	**4.7s**	5.3s	5s
openssl	**2.4**s	6s	7.7s	**3s**	4.2s	5s	24s	**22s**	31s
re2	**6.4s**	7s	7.2s	2.8s	**0.9s**	1.3s	**2.8s**	3.6s	5s
readelf	**2.3**s	3.2s	3.6s	**1s**	**1.1s**	1.2s	13.4s	**11.5s**	**11.6s**
sqlite3	**41s**	50s	47.6s	**3.7s**	5.5s	5.8s	19s	37s	**14.5s**
suricata	**3.2s**	**3.2s**	**3.5s**	**0.6s**	**0.6s**	0.8s	**3.7s**	4s	4.4s
yodl	**4.7s**	6.9s	10s	**1.4s**	2.2s	2.4s	**3.9s**	9.5s	9.5s
tiff2pdf	1m33s	**39s**	1m5s	8.6s	7.2s	**6.9s**	37.5s	**18s**	20s

Performance Evaluation

Application	Path predicate time		Total time		Queries / min	
	default	symaddr	default	symaddr	default	symaddr
cjpeg	18s	1m31s	60m	60m	5.3	5.1
libxml2	15s	16s	9m59	60m	924.1	122.4
readelf	27s	36s	60m	60m	85.7	13.1
libcbor	1.8s	2.1s	12s	1m58s	2176.5	210.2
openssl	1m19s	1m38s	60m	60m	44.7	18.5
sqlite3	9.1s	10.7s	12m49s	14m56s	2871.5	2608.1
minigzip	59s	3m48s	16m23s	60m	582.9	7.6
hdp	23s	31s	60m	60m	156.2	31.9
yices-smt2	10s	24s	22m22s	60m	494.1	50.6
yodl	6s	7s	9m2s	20m8s	852.3	396.1
jasper	10m12s	16m16s	60m	60m	203	115.3

Efficiency Evaluation

Application	SAT		Accuracy	
	default	symaddr	default	symaddr
cjpeg	56	54	89.3%	92.6%
libxml2	1247	1244	82.4%	90.1%
readelf	2029	287	86.9%	81.2%
libcbor	275	295	100%	40.6%
openssl	1000	234	75.7%	70.5%
sqlite3	8414	10340	99.9%	100%
minigzi	7569	238	51.5%	100%
hdp	4417	962	73.7%	68.3%
yices-smt2	5536	621	70.2%	89%
yodl	1150	1421	98.3%	98.3%
jasper	4164	3336	82.6%	81.4%

The Number of Discovered Symbolic Branches

Application	Total		Unique		New and unique	
	default	symaddr	default	symaddr	default	symaddr
cjpeg	6992	30098	150	233	3	86
libxml2	9840	16423	452	531	0	79
readelf	19790	23009	924	937	0	13
libcbor	122	158	31	34	0	3
openssl	7561	7804	200	220	0	20
sqlite3	6979	9001	55	67	0	12
minigzip	8977	52861	23	68	0	45
hdp	28227	30620	431	460	2	31
yices-smt2	10462	23497	94	555	0	461
yodl	6676	6992	65	79	0	14
jasper	771811	1093902	97	107	0	10

Explored Program Coverage

Application	Code coverage (%)		Coverage diff (%)	
	default	symaddr	default\symaddr	symaddr\default
cjpeg	19.58	20.82	0	1.25
libxml2	7.8	9.6	0	1.8
readelf	16	15.8	0.8	0.6
libcbor	70.43	59.17	14.4	3.14
openssl	5.19	5.25	0.02	0.08
sqlite3	5.5	5.6	0	0.1
minigzip	29.69	31.14	0	1.45
hdp	9.5	9.2	0.34	0.04
hdp(libmfhdf)	13.95	14.83	0.45	1.33
hdp(libdf)	9.18	8.65	0.65	0.12
yices-smt2	2.23	2.33	0	0.1
yodl	28.25	29.17	0	0.92
jasper	9.94	10.07	0	0.13

Questions?