Symbolic Execution with Angr
RPISEC

Avi Weinstock (aweinstock), Luke Biery (tiecoon)

December 6, 2019

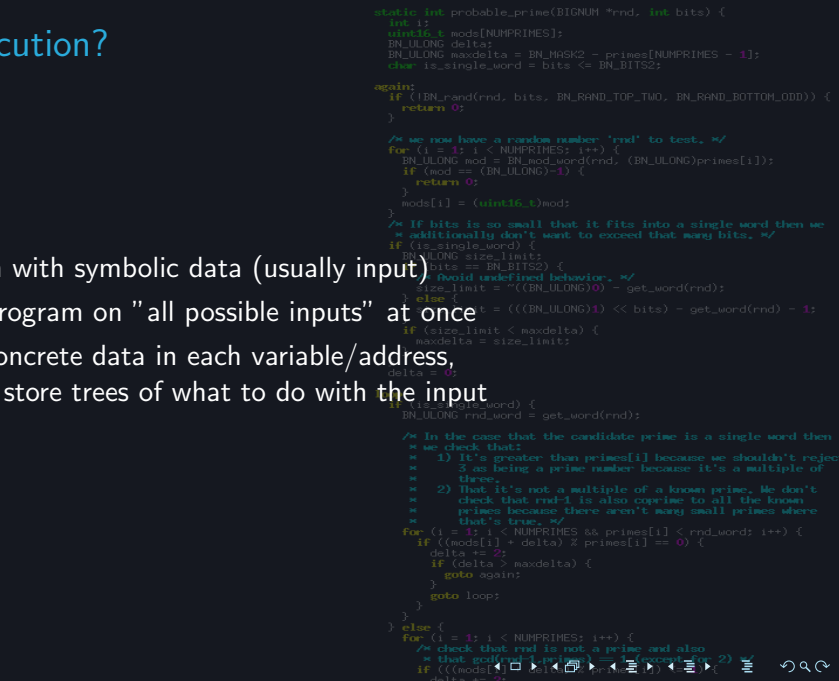
Overview

- What is Symbolic Execution? What techniques does it compete with?
- How symbolic execution works (theory)
- How symbolic execution works (Angr commands)
- Solving MBE lab1A with Angr

Background - What it is and what is the problem space?

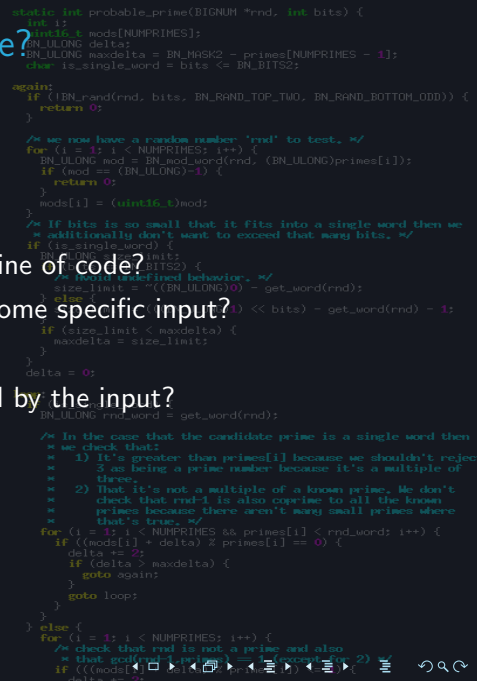
What is Symbolic Execution?

- Executes a program with symbolic data (usually input)
- Essentially runs a program on "all possible inputs" at once
- Instead of having concrete data in each variable/address, variables/addresses store trees of what to do with the input



What problems does Symbolic Execution solve?

- What input to provide to reach/avoid a specific line of code?
- How is a value deep in the program affected by some specific input?
- Do any inputs lead to any crash?
- On a crashing input, what registers are controlled by the input?



Symbolic Execution vs Fuzzing

Symbolic Execution	Fuzzing
+ Explores all inputs	- Only explores random inputs
+ Very detailed output	- Only learn crash vs non-crash
- Uses more memory/time	+ Uses around as much memory/time as target program

TODO: more comparisons/columns? emphasize that "all inputs" means that symexec can find constant-time comparisons against a giant constant, unlike coverage-guided?

How symbolic execution works in general

Setting up a state for symbolic execution

```
import z3
registers = ['eax', 'ebx', 'ecx', 'edx', 'ebp', 'esp'] # and so on
symstate = {reg: z3.BitVec(reg, 32) for reg in registers}
symstate['memory'] = z3.Array('memory', z3.BitVecSort(32), z3.BitVecSort(8))
```

- Note that the z3 variable eax in the model will be the starting value of eax
- `symstate['eax']` will be mutated throughout the computation, and will contain an expression corresponding to the ending value of eax

Symbolically executing branch-free code

- Translate arithmetic, indexing, etc into SMT constraints

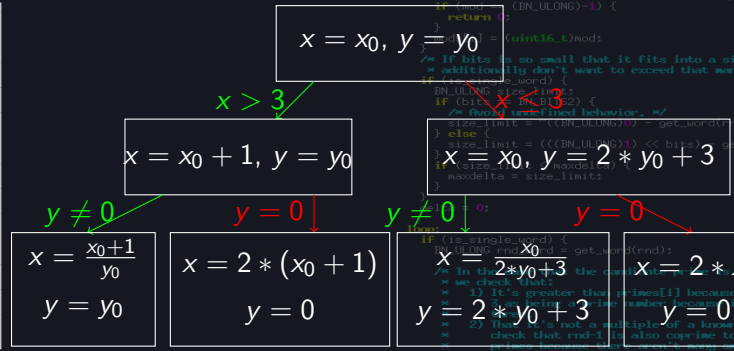
`mov eax, ebx`	`symstate['eax'] = symstate['ebx']`
`add ecx, edx`	`symstate['ecx'] += symstate['edx']`

`mov byte [esp+0x10], al`	`esp_10 = symstate['esp']+0x10`
	`al = z3.Extract(7, 0, symstate['eax'])`
	`symstate['memory'] = z3.Store(symstate['memory'], esp_10, al)`

`movsx eax, byte [eax]`	`star_eax = z3.Select(symstate['memory'], eax)` `symstate['eax'] = z3.SignExt(24, star_eax)`

Symbolically executing branchs

```
int f(int x, int y) {
    if (x > 3) {
        x += 1;
    } else {
        y = 2*y+3;
    }
    if(y != 0) {
        x /= y;
    } else {
        x *= 2;
    }
    return x + y;
}
```



TODO: Avi

- symbolic loops
- symbolizing tainted memory (e.g. `sym_memory[(EBP+username+i)]` vs `sym_username[i]`)
- state explosion when looping on symbolic data

How to use Angr for symbolic execution

TODO: Luke

- loading binariess
- marking input as symbolic
- initiating the search/pruning the search space
- simprocedures for shortcutting syscalls?

Example: MBE lab1A with Angr

Resources

- https://github.com/angr/
- https://github.com/Z3Prover/z3/
- https://github.com/RPISEC/MBE