QSYM: A PRACTICAL CONCOLIC EXECUTION ENGINE TAILORED FOR HYBRID FUZZING

Insu Yun, Sangho Lee, Meng Xu, Yeongjin Jang and Taesoo Kim,
FINDING SECURITY BUGS

- Fuzzing
  - Automated test to monitor exceptions (crashes & memory leaks)

- Pro: general inputs (loose branch condition: x<1000)
- Con: specific inputs
FINDING SECURITY BUGS

- Concolic Execution - concrete execution drive the symbolic execution through specific path
  - Symbolic Execution
    - Execution through all paths
  - Concrete Execution
    - Executing with values

- Pro: specific inputs (narrow conditions: \( x == 0xfdsgs \))
- Con: path explosion - feasible paths in a program grows exponentially with an increase in program size
y = read();
z = y * 2;
if (z == 12) {
    fail();
} else {
    printf(" - ");
}
HYBRID FUZZING

- Combination of techniques
  - Fuzzing – explore trivial input spaces
  - Concolic – solve complex branches

- Forking when needed

- Proven to work by Driller
  - 6 new crashing inputs not found by using individually
HYBRID FUZZING

Program

Basic block

Intermediate Representations

Coverage

State forking

Constraints

Test cases

- push ebp
- mov ebp, esp
- ...

- t0 = GET:i32(ebp)
- t1 = GET:i32(esp)
- t2 = Sub32(t1,0x00000004)
- ...

A[0] == ‘A’
...

Fuzzing
HYBRID FUZZING - PROBLEMS

• Slow to generate constraints

• No support for complete system calls

• Bad at generating test cases
QYSM

- Remove IR translation layer to reduce overhead (minimal symbolic emulation)
- Concrete execution to model external environment – support to system calls (models minimal system calls)
  
  \[
  \text{mprotect}(\text{addr}, \text{sym\_size}, \text{PROT\_R})
  \]
  
  \[
  \text{mprotect}(\text{addr}, \text{conc\_size}, \text{PROT\_R})
  \]

- Smart constraint solving
  - Incomplete constraints (efficiency) – Unrelated concrete elimination
    - Only solve constraint associated to branch
    - Overly constrained path (solve portion)
QSYM - INCOMPLETE CONSTRAINTS

\[ x = \text{int}(\text{input}()) \]
\[ y = \text{int}(\text{input}()) \]

# Incomplete constraints
mprotect(addr, x, PROT_R)

**if** \( y \times y = 1337 \times 1337 \): bug()

Constraints for \( x \) (Incomplete)
&& \( y \times y = 1337 \times 1337 \)

Path constraints

\( y \times y = 1337 \times 1337 \)

Branch dependent constraints

x = Use concrete value
y = 1337
QSYM – OVERLY CONSTRAINED PATHS

type = int(input())

if type == TYPE1:
    parse_TYPE1()

...

if type == TYPE2:
    parse_TYPE2()

circle type = int(input())

if type == TYPE1:
    ...
    + long time

if type == TYPE2:
    Unsatisfiable: No test case
1. Instruction-level execution

Program

```
push ebp
mov ebp, esp
...
```

Basic block

```
A[0] == 'A'
...
```

Constraints

Coverage

Test cases

Fuzzing
Detect repetitive basic blocks and prunes them for symbolic execution with subset of constraints.
Counts frequency of basic blocks and at runtime selects the repetitive blocks to prune.
If a basic block is executed frequently then it will stop generating constraints for it.
Over-pruning basic block – miss solvable path.
- Grouping multiple executions
- Context sensitivity – If block are in different branches
IMPLEMENTATION

- Intel Pin used for emulation
  - API that allows context information such as register contents to be passed to the injected code as parameters

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concolic execution core</td>
<td>12,528 LoC of C++</td>
</tr>
<tr>
<td>Expression generation</td>
<td>1,913 LoC of C++</td>
</tr>
<tr>
<td>System call abstraction</td>
<td>1,577 LoC of C++</td>
</tr>
<tr>
<td>Hybrid fuzzing</td>
<td>565 LoC of Python</td>
</tr>
</tbody>
</table>
**QSYM – REAL WORLD SCALABLE**

- Apply QSYM to programs large in size and previously fuzzed
- 13 new unknown bugs found in software
- Google’s OSS-Fuzz generated 10 trillion test inputs a day for a few months to fuzz these applications
  - QSYM ran them for three hours using a single workstation
- Driller – Hybrid Fuzzer (test cases)
• OSS – Fuzz (2 years)
• QSYM generates test case to reach this bug
LIMITATIONS

- Specialized to test on x86 architecture
- Other executors using IR can be ran on other architectures
CONCLUSION

- QSYM is a hybrid fuzzing model that is scalable to real world applications

- Outperforms current models for bug finding