Formal Verification of JIT by Symbolic Execution

Boris Shingarov

High-Level Goal

- Verification of the VM using formal methods

Previously-tried other approaches

- Correct-by-construction through synthesis
  - B.Shingarov: "*Synthesis of Target-Agnostic JIT*" — Córdoba, 2014; Cambridge, 2014; Brescia, 2015
- Using full proofs in a dependent-type system
  - B.Shingarov: "*Writing a Smalltalk VM in Coq*" — Maribor, 2017

Prohibitive R&D Cost

- VM engineers run away in panic
- Nobody follows

Symbolic Execution

- Well-established techniques, widely used for:
  - Compiler correctness proofs
  - Security vulnerability discovery
  - etc…
- Virtual CPU, works by using an SMT solver for constraint propagation between states

Available Analysis Tools

- Alive
  - Lopez et al: “*Provably Correct Peephole Optimizations with Alive*” — PLDI, 2015
    - LLVM optimizations
- angr
  - Shoshitaishvili et al: “*The Art of War*” — IEEE 2016
    - Wide range of binary analysis techniques

LABWARE Results Count

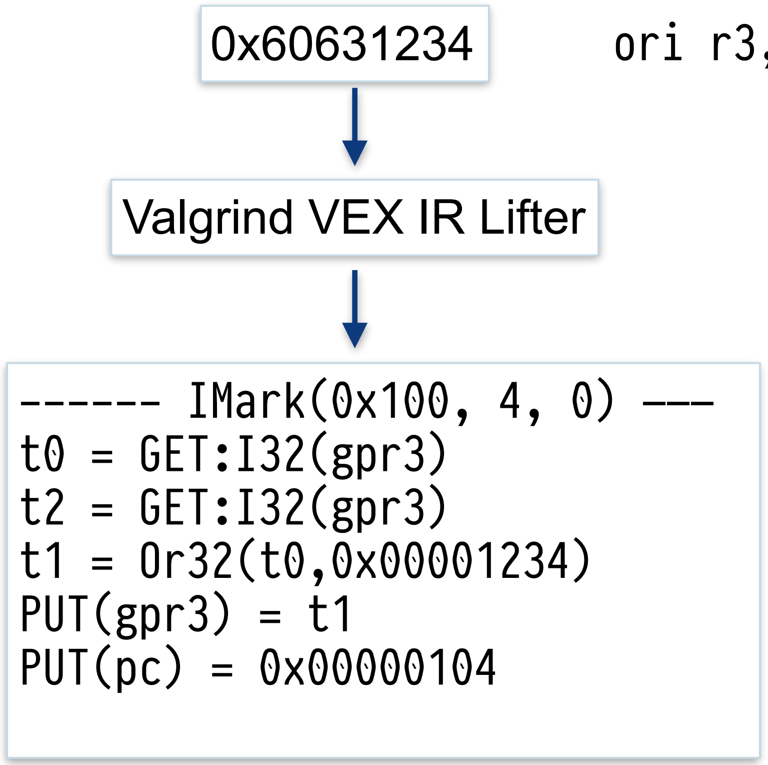
Problem: Self-Modifying Code

- Let’s start with the unproblematic case first

Symbolic Testing

$EAX = 5$	$EAX = \chi$
0x40 inc EAX	0x40 inc EAX
$EAX = 6$	$EAX = \chi+1$

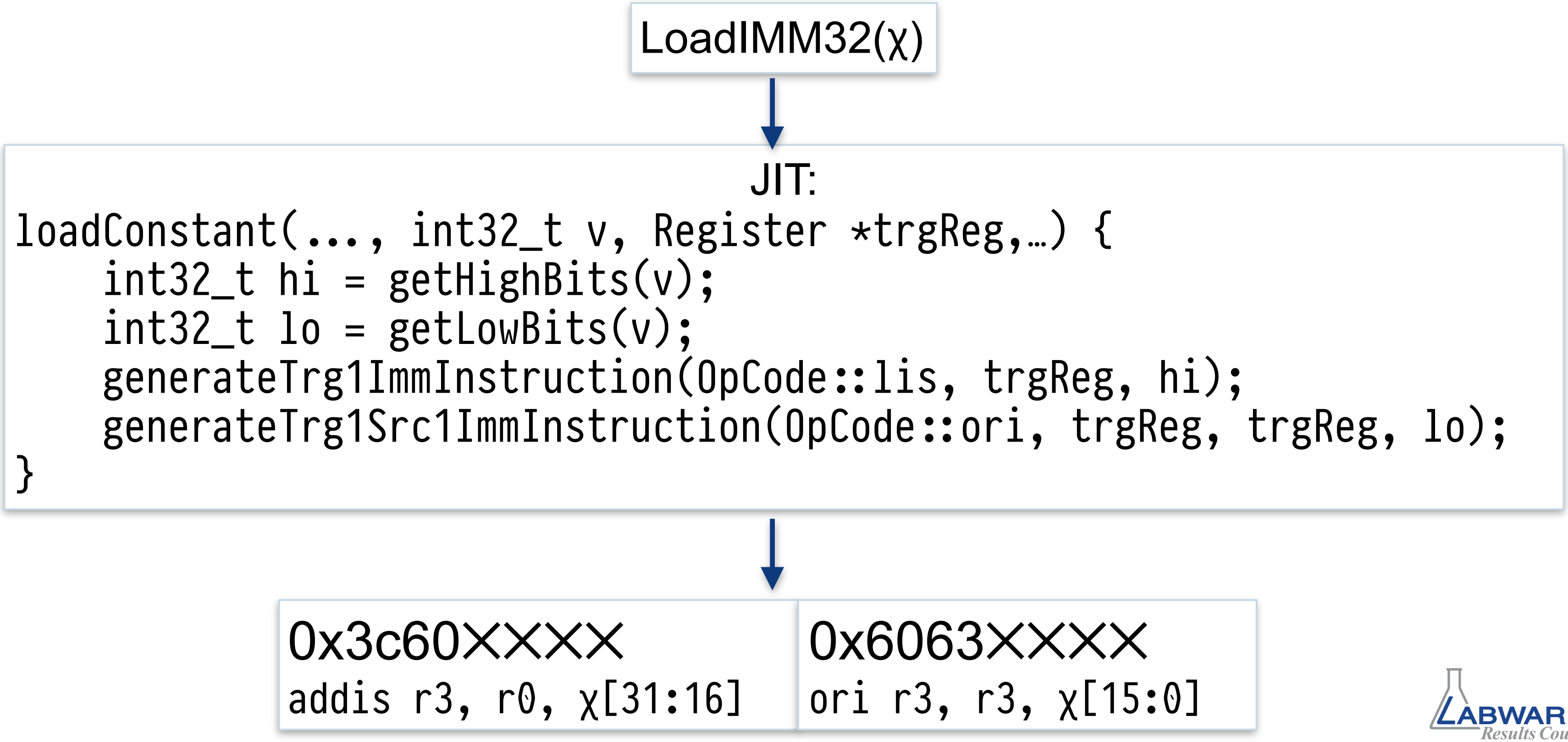
The Symbolic Execution Engine



```
ori r3, r3, 0x1234
```

```
------ IMark(0x100, 4, 0) ---
t0 = GET:I32(gpr3)
t2 = GET:I32(gpr3)
t1 = Or32(t0,0x00001234)
PUT(gpr3) = t1
PUT(pc) = 0x00000104
```

Problem: No Symbolic Binaries!



LoadIMM32(χ)

JIT:

```
loadConstant(..., int32_t v, Register *trgReg,…) {
    int32_t hi = getHighBits(v);
    int32_t lo = getLowBits(v);
    generateTrg1ImmInstruction(OpCode::lis, trgReg, hi);
    generateTrg1Src1ImmInstruction(OpCode::ori, trgReg, trgReg, lo);
}
```

0x3c60XXXX	0x6063XXXX
`addis r3, r0, χ[31:16]`	`ori r3, r3, χ[15:0]`

Problem: No Symbolic Binaries!

0x3c60XXXX	0x6063XXXX
`addis r3, r0, χ[31:16]`	`ori r3, r3, χ[15:0]`

↓

Valgrind VEX IR Lifter

↓

Solution: Write new Lifter

(Digression: my binutils-in-Smalltalk are synthesized from a formal processor description given in a PDL)

- Augment the PDL
    - IR templates
    - Just like Valgrind IR, but can contain computer algebra

DEMO

Useful Results

- Work-in-Progress
- Working proof of concept
- Simple PowerPC and RISC-V code
- Practical reasons: no support for RISC-V in angr
- Shingarov—Vraný: “*Status of Dynamic Language Runtimes on RISC-V*” — RISC-V Workshop
  - Useful in determining correctness of OMR RISC-V backend

Future work

- Synthesize the DSL
  - by symbolic execution of RISC-V Spike
- Superoptimize
- What to vet a realistic JIT against?
- Polymorphic Inline Cache
- Equivalence vs Refinement
- Seed a Verified VM Community

https://github.com/shingarov/petrich/tree/machine-arithmetic

LABWARE
Results Count