Warpcore

Nick Gregory, Josh Hofing

Overview

- Eventual goal: to be a Cyber Reasoning System (CRS)
  - Automatically find bugs in programs
  - Maybe even exploit them
- For this semester
  - Lay the groundwork (wrappers, core abstractions, etc.)
  - Create a concrete executor (an emulator) for a small ARM program

Parts

- ELF parser
- Intermediate Language (IL)
- ARM assembly -> IL transpiler
- SMT abstraction
- Executor

ELF Parser/Loader

- Now the second ELF loader I’ve written…
- Just needs to be able to extract data into the correct vaddrs
- Don’t need linking, relocations, etc. (yet)
- Not very complex

IL

- Custom levelled, tree-based IL (think Binja Lifted IL)
- We don’t want to have to lift the entire binary at startup
  - Addresses are a pair (level, addr), so you can jump between levels, letting you just insert jumps to the unlifted (“native”) level for unlifted code
  - Binaries are large and we’re white-box fuzzing, so we won’t hit most of the code most of the time
- This adds a lot of required heavy-lifting to the execution engine
  - It needs to understand how to lift new code as we run, not screw it up, etc
  - But we think that it will be a win
  - We view the lifted code as a cache which should be basically freely dumbable
- A huge part of the idea is that we can “JIT” large parts of IL to make execution faster

ARM -> IL Transpiler

- ARM released a machine-readable specification describing instruction semantics
- We transpile the Architecture Specification Language (ASL) code into C++ IL operations
- Unnecessary language constructs complicate the transpiler...
- ~90% done though!

ARM -> IL Transpiler - Future Work

- Move to Sail: https://github.com/rems-project/sail
- Much saner language
- Also already has definitions of MIPS, ARM and (most) x86(-64) instructions!

SMT Abstractions

- Want the ability to use different symbolic execution engines
  - Maybe even multiple in a single run…
- C++ wrapper around SMT-LIB abstractions
- And code to actually run the SMT solver and get results back

Executor

- As mentioned, goal for this semester is concrete execution of a small ARM binary
- No need for symbolic memory yet
- Concrete executor supports almost all of the IL
- And just enough aarch64 Linux syscalls to do “Hello, World!”
- Too bad nobody has written a machine-readable Linux syscall specification

Questions?