

Virtual Machines

Binary Translation

Binary Translation

- Emulation:
 - Guest code is traversed and instruction classes are mapped to routines that emulate them on the target architecture.
- Binary translation:
 - The entire program is translated into a binary of another architecture.
 - Each binary source instruction is emulated by some binary target instructions.

Challenges

- Can we really just read the source binary and translate it statically one instruction at a time to a target binary?
 - What are some difficulties?

Challenges

- Code discovery and dynamic translation
 - How to tell whether something is code or data?
 - Consider a jump instruction: Is the part that follows it code or data?
- Code location problem
 - How to map source program counter to target program counter (without having a table as long as the program for instruction-by-instruction mapping)?

Things to Notice

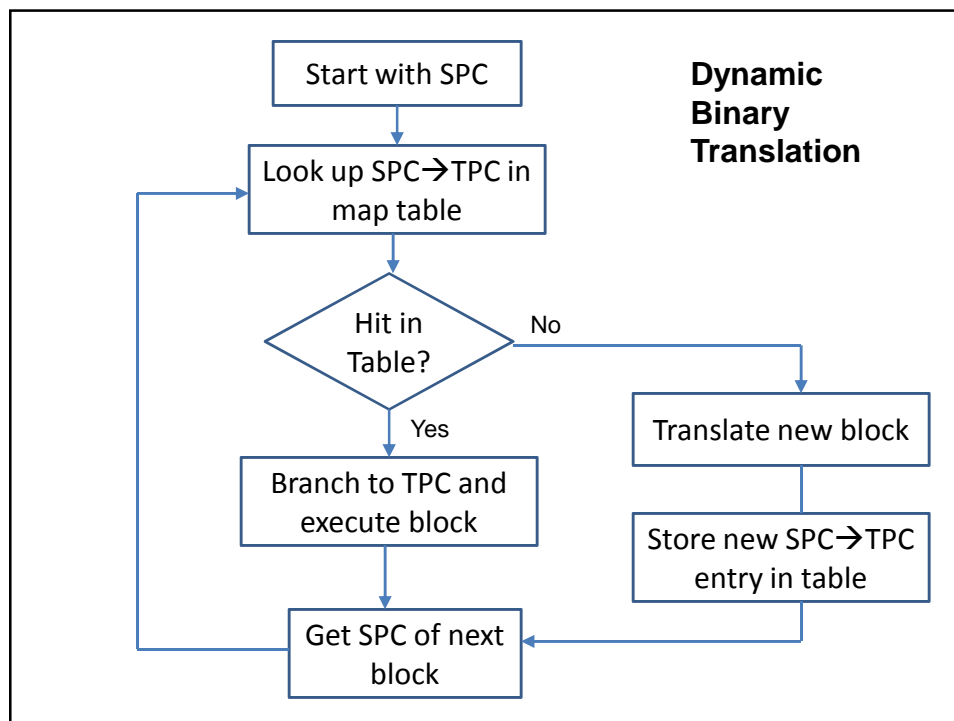
- You only need source-to-target program counter mapping for locations that are *targets of jumps*. Hence, only map those locations.
- You always know that something is an instruction (not data) in the source binary if the source program counter eventually ends up pointing to it.
- The problem is: You do not know targets of jumps (and what the program counter will end up pointing to) at static analysis time!
 - Why?

Solution

- Incremental Pre-decoding and Translation
 - As you execute a source binary block, translate it into a target binary block (this way you know you are translating valid instructions)
 - Whenever you jump:
 - If you jump to a new location: start a new target binary block, record the mapping between source program counter and target program counter in map table.
 - If you jump to a location already in the map table, get the target program counter from the table
 - Jumps must go through an emulation manager. Blocks are translated (the first time only) then executed directly thereafter

Dynamic Basic Blocks

- Program is translated into chunks called “dynamic basic blocks”, each composed of straight machine code of the target architecture
 - Block starts immediately after a jump instruction in the source binary
 - Block ends when a jump occurs
- At the end of each block (i.e., at jumps), emulation manager is called to inspect jump destination and transfer control to the right block with help of map table (or create a new block and map table entry, if map miss)



Same ISA Emulation

- Why?

Same ISA Emulation

- OS Emulation
 - Same machine architecture, different OS
 - Copy code as is, translate system calls.
- Replication
 - Copy code as is, except special instructions that deal with virtualized resources such as I/O.
 - Code running on Guest VMs (e.g., Guest OS code) cannot directly access these I/O resource.
- Program shepherding
 - Check that code does not execute something it is not supposed to (e.g., exploit security holes).

Optimizations

- Translation chaining
 - The counterpart of threading in interpreters
 - The first time a jump is taken to a new destination, go through the emulation manager as usual
 - Subsequently, rather than going through the emulation manager at that jump (i.e., once destination block is known), just go to the right place.
 - What type of jumps can we do this with?

Optimizations

- Translation chaining
 - The counterpart of threading in interpreters
 - The first time a jump is taken to a new destination, go through the emulation manager as usual
 - Subsequently, rather than going through the emulation manager at that jump (i.e., once destination block is known), just go to the right place.
 - What type of jumps can we do this with?
 - Fixed destination jumps!

What about Register Indirect Jumps?

- Jump destination depends on value in register.
- Must search map table for destination value (expensive operation)
- Solution?

What about Register Indirect Jumps?

- Jump destination depends on value in register.
- Must search map table for destination value (expensive operation)
- Solution
 - Caching: add a series of if statements, comparing register content to common jump source program counter values from past execution (most common first).
 - If there is a match, jump to corresponding target program counter location.
 - Else, go to emulation manager.