Goals for Today

• **Learning Objective:**
  - Understand challenges in Static/Dynamic Binary Translation

• **Announcements, etc:**
  - Midterm debrief forthcoming *on Friday*
  - MP2 extension: **now due on March 23rd**

**Reminder:** Please put away devices at the start of class
CS 423
Operating System Design: Binary Translation

Professor Adam Bates
Spring 2017
• **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.
• 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 problem
  – 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?
  – Can we do this without having a table as long as the program for instruction-by-instruction mapping?
• **Observation #1:** You always know that something is an instruction (not data) if the source program counter eventually ends up pointing to it.

• **Observation #2:** You only need source-to-target program counter mapping for locations that are *targets of jumps*. Hence, only map those locations.

• **Observation #3:** You do not know targets of jumps (and what the program counter will end up pointing to) at static analysis time!
  – Why?
Solution: Dynamic Translation

• 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)
Dynamic Binary Translation

1. Start with SPC
2. Look up SPC → TPC in map table
3. Hit in Table?
   - Yes: Branch to TPC and execute block
   - No: Translate new block
4. Get SPC of next block
5. Store new SPC → TPC entry in table
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?
• 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 Only!!!
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.