Dynamic Binary Instrumentation & Dynamic Taint Analysis

CS390R - UMass Amherst

Course Information

- Project 5 assigned & due May 4
- Presentation Checkpoint 2 due in about 2 weeks
- Homework on Program Analysis due in 1 week

Today’s Content

- Intel Pin Intro
- Use-Cases
- Other DBI Frameworks
- JIT Compilation
- Example and Demos
- Virtualization Deobfuscation
- Dynamic Taint Analysis
- Design Considerations
- Practical DTA

What is Intel Pin

- Dynamic Binary Instrumentation framework
- It enables you to write modules that execute some program and insert code into it at execution time to track things you are interested in
- A big advantage over eg. LLVM is that source code is not required
- Can easily hook any function in the program
- Unlike LLVM, Pin is not an analysis framework
  - It is meant to be used to instrument a binary and collect runtime data
  - Simple analysis can be performed at runtime, but it's apis are much weaker than eg. LLVM's so generally you use Pin to collect information and then use something else to perform the analysis
  - In your project, you will combine Pin's instrumentation with Ghidra's analysis

Use-cases

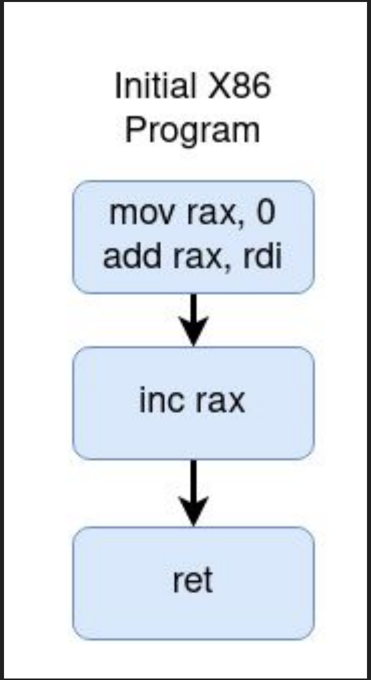
- Program analysis
  - Performance profiling (Intel uses this for its VTune profiler)
  - Valgrind style memory profiling
  - Trace collections
  - Taint Analysis
- Computer Architecture study
  - Processor/cache simulations (MIT uses this in their PHD Comp-Arch course, and intel uses this to eg. model branch predictions and cache setups)
- Binary Translation
  - Straightup modify program behavior or emulate unsupported instructions
- Dynamic malware deobfuscation

Alternative DBI Frameworks

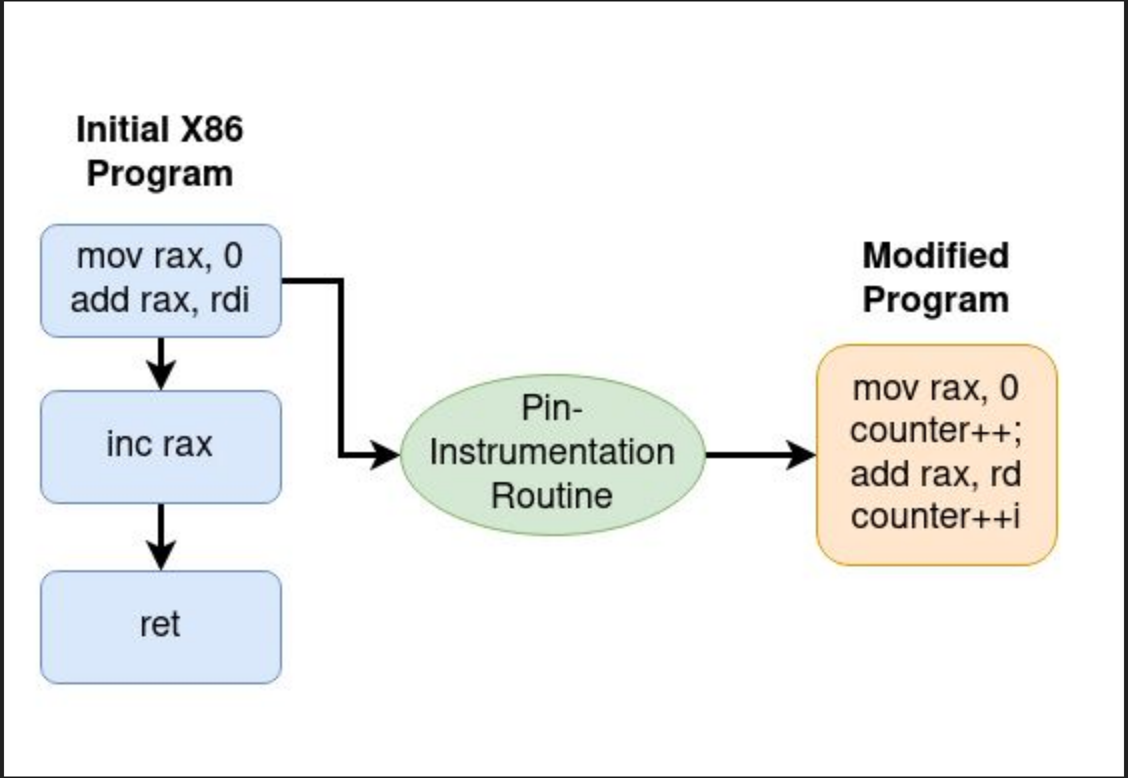
- Dynamorio
  - More powerful and generally faster than pin
  - Feels less supported and can be hard to get setup when compared to pin
  - Entirely open source
- Valgrind
  - Basically superseded by Dynamorio
- Frida
  - A lot less powerful and slower
  - Easier to use in a lot of ways though, very straightforward js api
  - Frequently used when working with eg. mobile apps to perform various forms of hooks
- GDB
  - Very weak api since it is not really made for DBI
  - Very slow

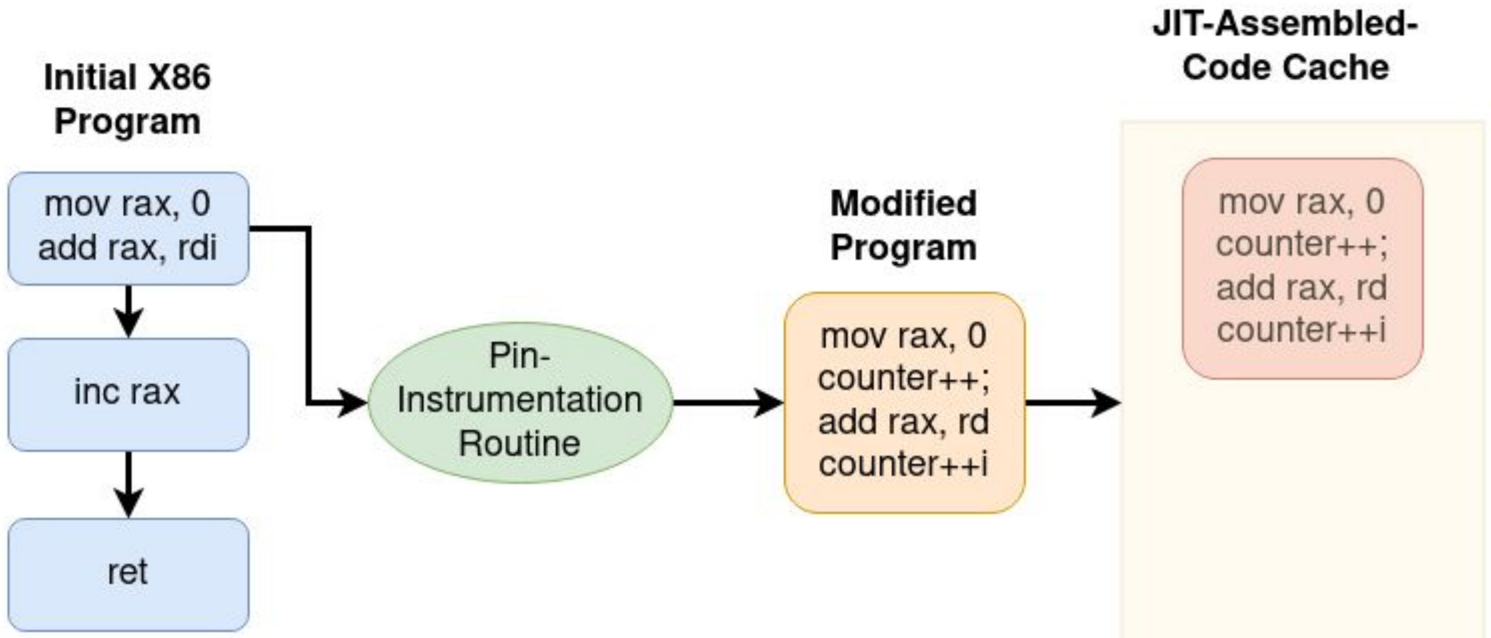
JIT - Just in Time Compilation

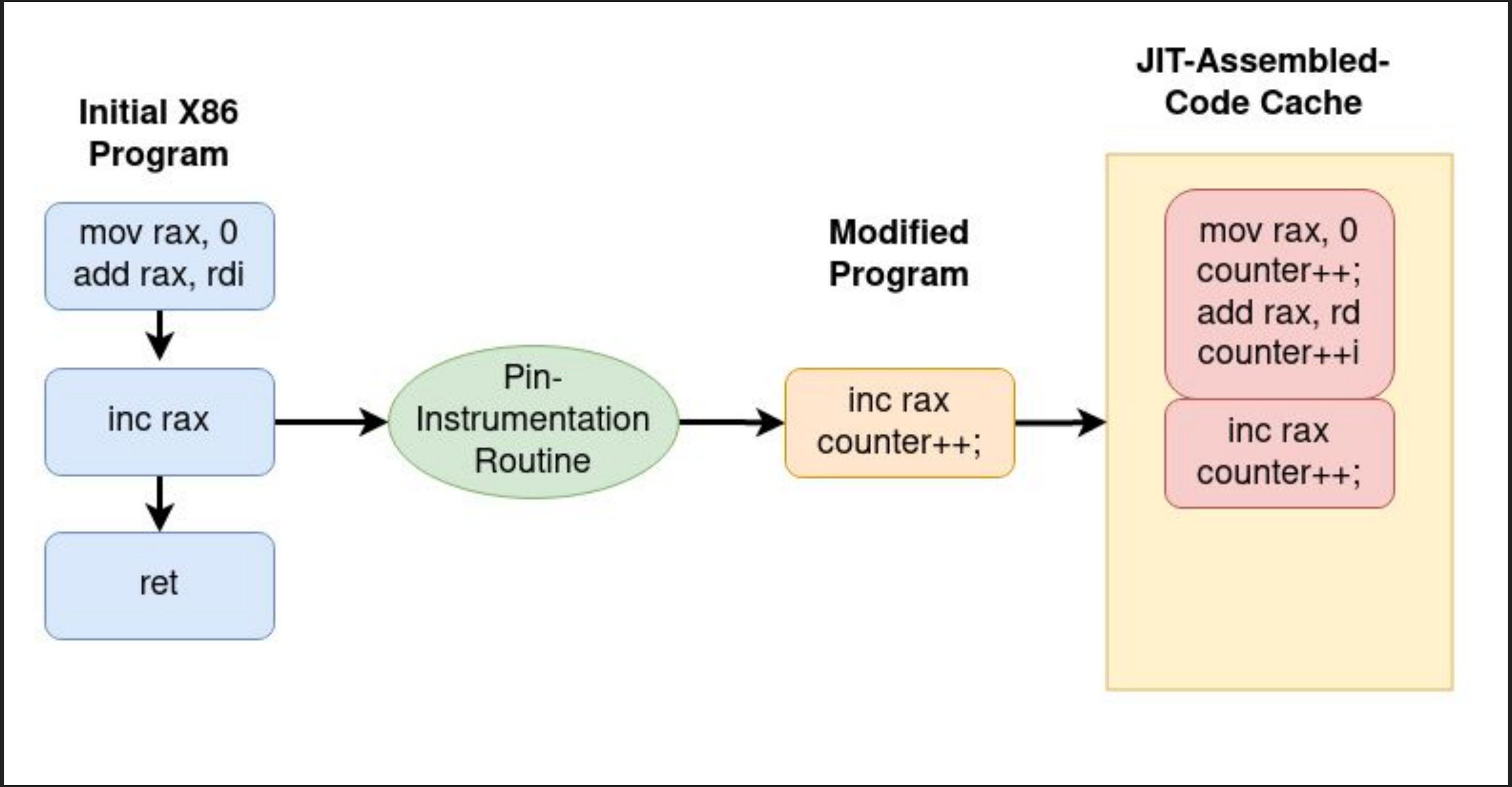
- Compile code at runtime, one block/function at a time
- This is done by a lot of otherwise interpreted code like Java in the JVM, or Javascript in most Browsers
- In the case of pin, its input is not bytecode, but a binary
- It goes to the starting point and starts execution from there just like an OS-loader would, but instead of actually executing the code, if it has not yet encountered the block of instructions, they are lifted into an IR
- Pintools can then operate on this IR and make modifications
- Finally this IR is JIT compiled into a memory cache and executed from there until another new block comes up

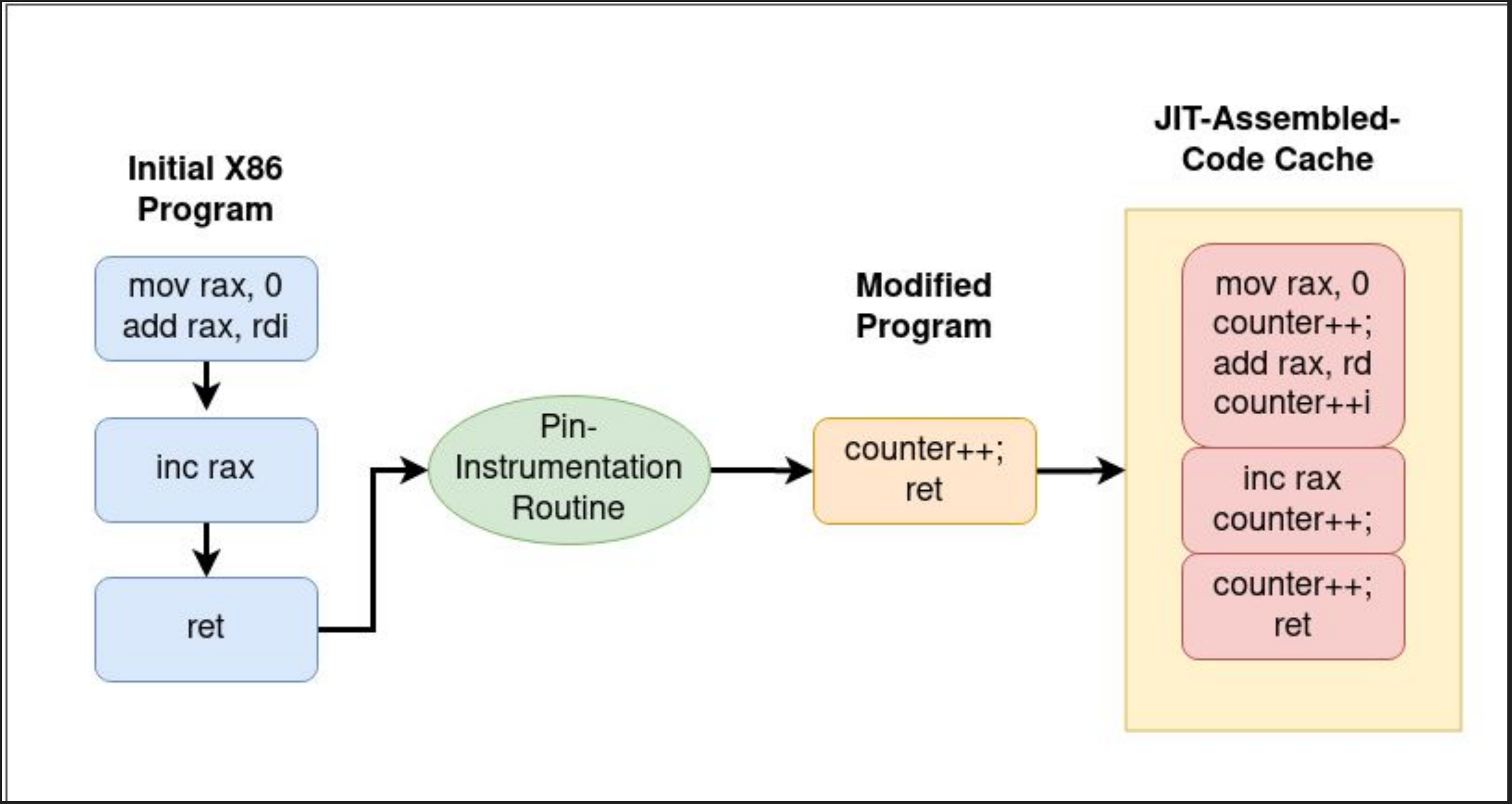


```
mov rax, 0
add rax, rdi
```


Initial X86 Program
mov rax, 0
add rax, rdi
inc rax
ret
Pin-Instrumentation Routine
Modified Program
mov rax, 0
counter++;
add rax, rd
counter++i


Initial X86 Program
mov rax, 0
add rax, rdi
inc rax
ret
Pin-Instrumentation Routine
Modified Program
mov rax, 0
counter++;
add rax, rd
counter++i
JIT-Assembled-Code Cache
mov rax, 0
counter++;
add rax, rd
counter++i


Initial X86 Program
mov rax, 0
add rax, rdi
inc rax
ret
Pin-Instrumentation Routine
Modified Program
inc rax
counter++;
JIT-Assembled-Code Cache
mov rax, 0
counter++;
add rax, rd
counter++i
inc rax
counter++;


Initial X86 Program
mov rax, 0
add rax, rdi
inc rax
ret
Pin-Instrumentation Routine
Modified Program
counter++;
ret
JIT-Assembled-Code Cache
mov rax, 0
counter++;
add rax, rd
counter++i
inc rax
counter++;
counter++;
ret

Demo - Instruction Counting

```
cd ~/install/pin-3.20-98437-gf02b61307-gcc-linux/source/tools/ManualExamples

../../../pin -t obj-intel64/inscount0.so -o inscount0.log -- /bin/ls
```

Demo - Address Trace

```
cd ~/install/pin-3.20-98437-gf02b61307-gcc-linux/source/tools/ManualExamples

../../../pin -t obj-intel64/itrace.so -- /bin/ls
```

Dynamic Taint Analysis

- Technique to determine how data flows through a given program and influences its state (Dataflow Analysis, but dynamic)
- This is generally done by marking certain bytes of memory and tracking how it flows through program execution
- This can be compared to coloring the water in a river and then seeing if the colored water shows up in any nearby lakes
- This is often implemented on top of other DBI engines like Pin
- The taint data (markings for data) are stored in a shadow-memory region. Separate region of memory allocated by the DTA system to keep track of the taint status of the rest of memory
    - This is generally a massive array/bitmap
    - 32-bit memory-region, 4GB, even with a single bit/byte -> 4GB/8 = 512MB for shadow memory
    - 64-bit systems requires some more complex data-structures (usually similar to page tables)

DTA in 3 Steps

- Define Taint Sources
  - This defines the data we want to track (syscalls, function entry-points, instructions, etc)
  - Eg. To track data coming from network-related activities, define recv/recvfrom syscalls as taint sources and mark all data produced by them
- Define Taint Sinks
  - Program locations for which we want to check whether they can be influenced by data from taint sources
- Tracking Input Propagation
  - All instructions handling taint data need to be instrumented
  - If the input to the instruction is tainted, the output needs to be tainted too
  - This way taint can be propagated all the way from initial data to the data that reaches the final sinks

Design Considerations

- Taint Granularity
  - Do we want to track this at bits, byte, page-level? More fine-grained is more accurate but also slower
- Taint Colors
  - Different colors can be used to determine not only if it is tainted, but also from which (out of potentially several) inputs
  - 1-color taint-propagation on a byte-level granularity can be tracked using a single bit per byte, if we want to eg. track 8 colors instead, we now incur 8x the memory overhead
- Taint Policies
  - This describes how the DTA system propagates taint and how different colors are merged
  - Commonly done by just taking unions of different colors, but might require changes depending on target

Practical Taint Analysis

We unfortunately don’t have time to cover dynamic taint analysis in any more depth than this. If you to see some examples of how to apply these concepts using the libdft library and pin, check out the book below.

- https://practicalbinaryanalysis.com/