Week XX
Dynamic Instrumentation

Presenter Name

sigpwny{}

funny image here

Announcements

- Announcement 1

- Announcement 2

- Announcement 3

Background

- Binary reverse engineering
- Two ways to find out what it does
    - Static analysis: looking at the binary without running it
    - **Dynamic analysis**: collecting information while running it
- Some common dynamic analysis tools:
    - gdb: classic debugger
    - angr: symbolic analysis

Motivation

- What if you wanted to:
    - Print the arguments to every `strcmp` call?
    - Count the number of function calls/code lines/instructions?
    - Log every memory write?

What is it

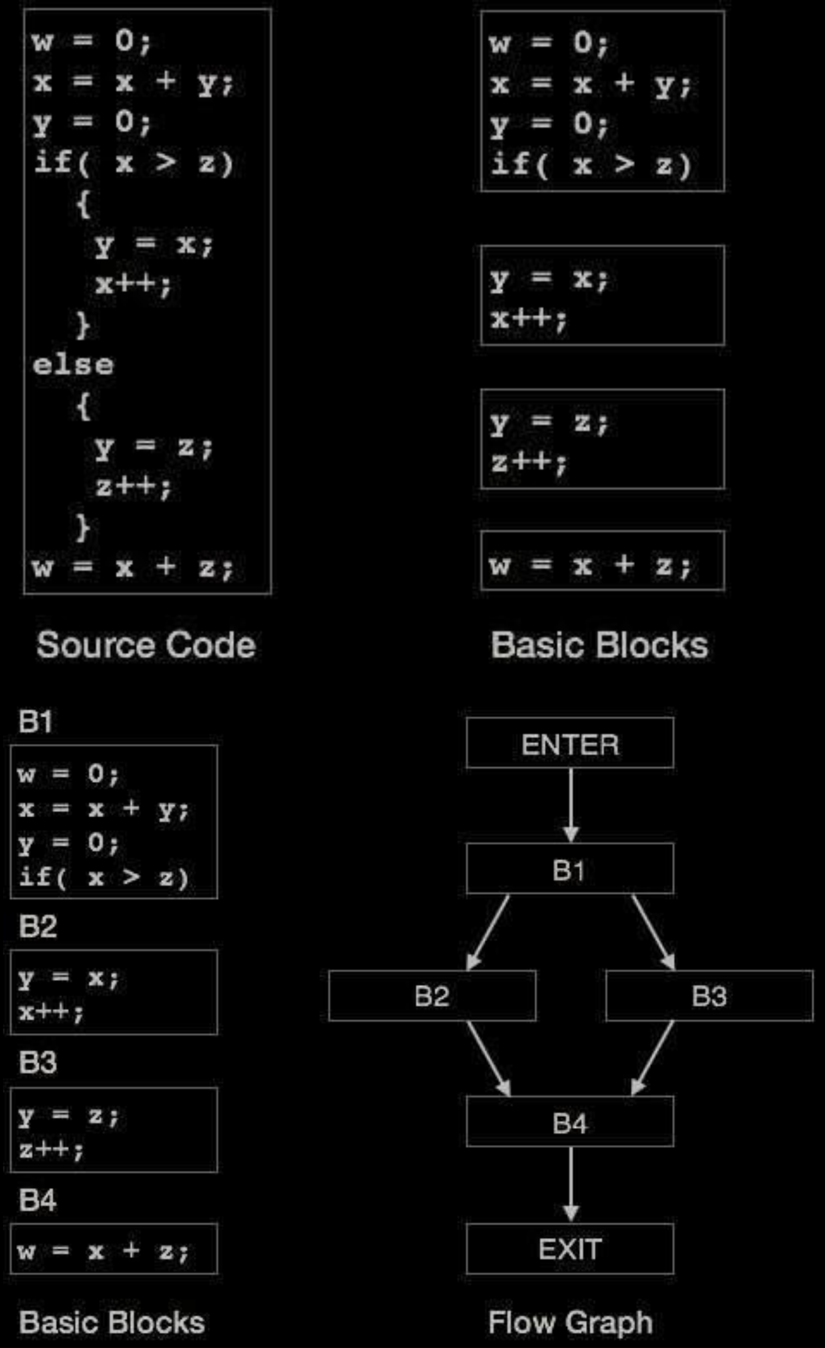
- Modifying binaries on-the-fly
- Add our own code ("instruments")
- Control flow recovery
- Added code does not affect the binary

Usage Cases

- Instruction counting
- Function call statistics
- VM instruction tracing
- Memory watching
- Syscall tracing

How it works

- Disassemble the binary (recursive, linear)
  - Surprisingly non-trivial, esp. w/ variable-length instruction ISAs
- Analyze the disassembly and get "basic blocks"
  - Boundary at jumps/calls/rets
- Each basic block is individually analyzed



```
w = 0;
x = x + y;
y = 0;
if( x > z)
  {
    y = x;
    x++;
  }
else
  {
    y = z;
    z++;
  }
w = x + z;
```

Source Code

```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

```
y = x;
x++;
```

```
y = z;
z++;
```

```
w = x + z;
```

Basic Blocks

B1

```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

B2

```
y = x;
x++;
```

B3

```
y = z;
z++;
```

B4

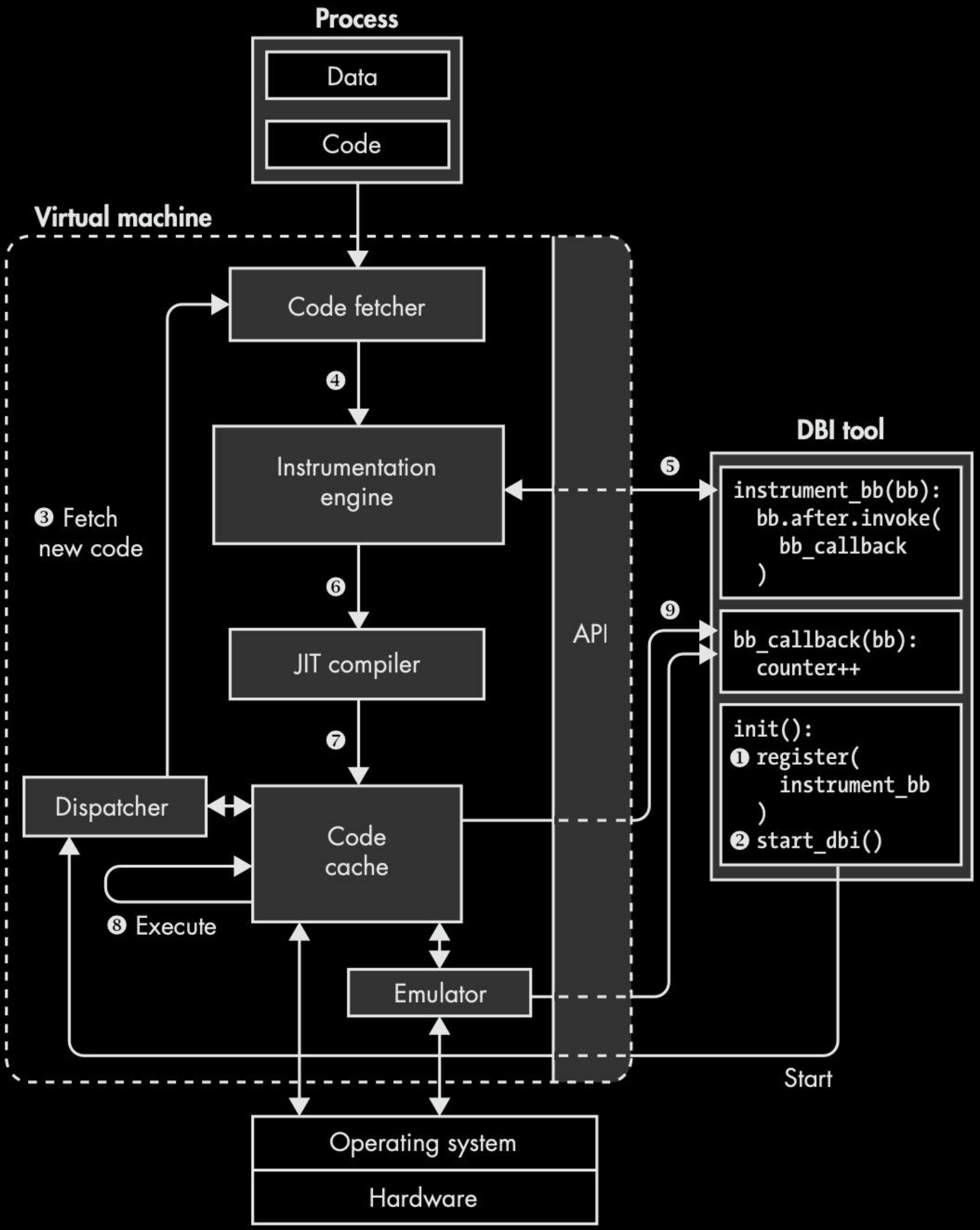
```
w = x + z;
```

Basic Blocks

Flow Graph

Basic overview

1. Load the binary
2. Disassemble, recover basic blocks
3. Add instrumentation
4. Add to cache
5. Run the binary
   a. Lazy; instrument more block only if necessary



```
instrument_bb(bb):
  bb.after.invoke(
    bb_callback
  )
```

```
bb_callback(bb):
  counter++
```

```
init():
1 register(
    instrument_bb
  )
2 start_dbi()
```

Intel® Pin

Basic instruction count example

Harder VM example

Similar tools

- Valgrind, DynamoRIO
    - Inject instrumentation at runtime
    - High-quality premade tools
- Clang/LLVM ASAN/MSAN/UBSAN
    - Compile-time instrumentation
- e9patch
    - Static binary rewriting without control flow recovery

Further reading

- *Practical Binary Analysis* by Dennis Andriesse

Next Meetings

Sunday Seminar: YYYY-MM-DD

-
-
-

Next Thursday: YYYY-MM-DD

-
-
-