# Introduction to Reverse Engineering, part 2

Zack Orndorff

# Quick review

x86 instructions look like oper dest, source

Flags exist, and they work along with cmp, test, and conditional jumps

The stack exists. You can push and pop things. Local variables are also here.

Calling conventions exist. For x86_64, params are passed in registers. Return value in rax.

| | | |
|---|---|---|
| “bottom of stack” | 0xFFF0 | “bottom \0” |
| | 0xFFE8 | 0xDEADBEEF |
| | 0xFFE0 | 0xF00DBEEF |
| | 0xFFD8 | “hithere\0” |
| | 0xFFD0 | 0x1337 |
| rsp -> “top of stack” | 0xFFC8 | 4 |

Registers: rdi, rsi, rdx, rcx, r8, r9

```
test rax, rax
jz somewhere
```

```
cmp rax, 4
jg somewhere
```

# But looking at asm was tedious?

Again, to quote Chris, “RE is the art of NOT reading assembly”

But we read a lot of it last week

How can we avoid reading assembly?

```
0000ae1a  vmovsd  xmm2, qword [r12]
0000ae20  vmovsd  xmm1, qword [r12+0x18]
0000ae27  vmovsd  xmm5, qword [r12+0x30]
0000ae2e  vmovsd  xmm3, qword [r12+0x8]
0000ae35  vmovsd  xmm6, qword [r12+0x10]
0000ae3c  vmovsd  xmm4, qword [r12+0x38]
0000ae43  vsubsd  xmm1, xmm1, xmm2
0000ae47  vsubsd  xmm5, xmm5, xmm2
0000ae4b  vmovsd  xmm2, qword [r12+0x28]
0000ae52  vmovsd  xmm0, qword [r12+0x20]
0000ae59  vsubsd  xmm4, xmm4, xmm3
0000ae5d  vsubsd  xmm2, xmm2, xmm6
0000ae61  vsubsd  xmm0, xmm0, xmm3
0000ae65  vmovsd  xmm3, qword [r12+0x40]
0000ae6c  vsubsd  xmm3, xmm3, xmm6
0000ae70  vmulsd  xmm6, xmm4, xmm2
0000ae74  vfmsub231sd xmm6, xmm0, xmm3
0000ae79  vmulsd  xmm3, xmm1, xmm3
0000ae7d  vmulsd  xmm0, xmm0, xmm5
0000ae81  vmovsd  qword [rbp-0x50], xmm6 {var_60}
0000ae86  vfmsub132sd xmm2, xmm3, xmm5
0000ae8b  vfmsub132sd xmm1, xmm0, xmm4
0000ae90  vmovsd  qword [rbp-0x48], xmm2 {var_58}
0000ae95  vmovsd  qword [rbp-0x40], xmm1 {var_50}
0000ae9a  vmovupd xmm2, xmmword [r10] {var_60}
0000ae9f  vmulpd  xmm0, xmm2, xmm2
0000aea3  vunpckhpd xmm3, xmm0, xmm0
0000aea7  vaddsd  xmm0, xmm0, xmm3
0000aeab  vxorpd  xmm3, xmm3, xmm3
0000aeaf  vfmadd231sd xmm0, xmm1, xmm1
```

# Technique #1: Strings!


IDA: View > Open Subviews > Strings or Shift-F12

Binja: Lower 2nd to right menu, select Strings

Ghidra: Window > Defined Strings

Find strings that are interesting (say errors or UI messages) and look at their XREFs

*Demo*

# Technique #2: Imports

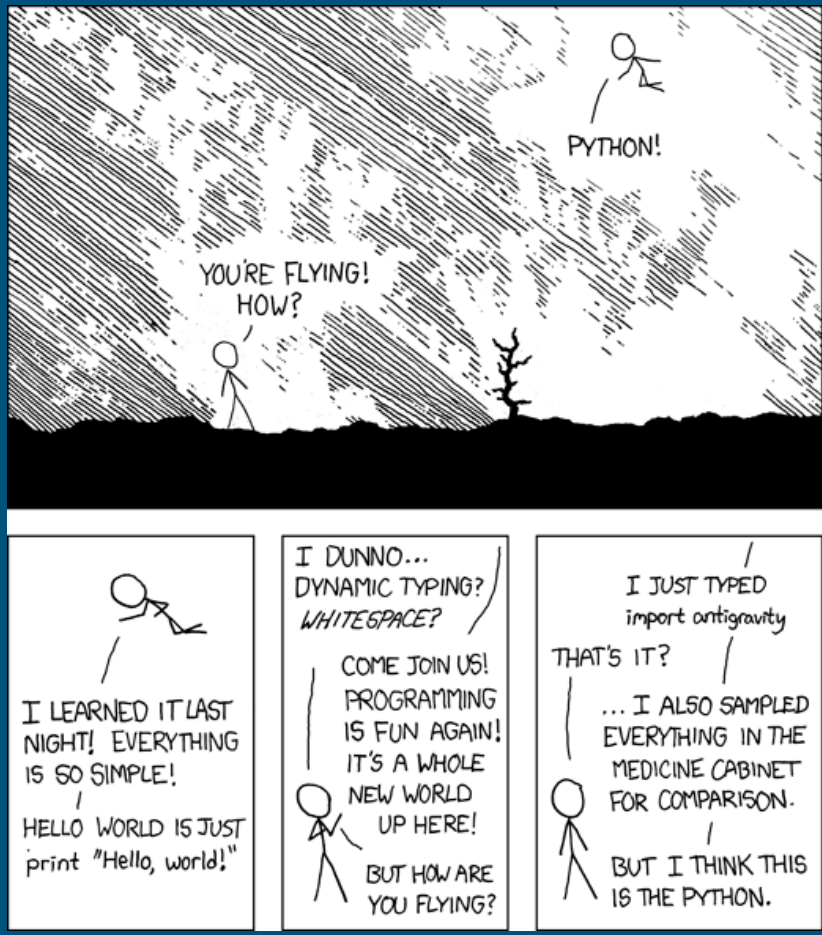
IDA: View > Open Subviews > Imports

Binja: (WIP AFAIK)

Ghidra: Window > Symbol Tree > Imports

Look at interesting imports, say, IO, or interesting libraries

*Demo*



# Technique #3: Debuggers

Break at different places until you find something interesting, then analyze it?

Verify your static analysis results.

Windows: x64dbg

Linux: Use gdb with someone else's config. I recommend one of:

- https://github.com/longld/peda
- https://github.com/gdbinit/Gdbinit

# Ghidra


RE tool developed by NSA

Written in Java, kinda slow, but works

Free, includes decompilation for all kinds of architectures

Keybinds are all different, L to rename, must right click or find in listing for XREFs

Views are under Window

*Demo*

# Decompilers?!

You mean we wasted all that manual effort last week? Well, not so fast…

- Decompilers are leaky abstractions. You need to know what they're doing for you.
- Decompilers can be wrong. Whether just plain wrong or messy, you need a backup plan.
- Decompilers can fail on erratic input, and you still need to be able to reverse it.