Week 04
Reverse Engineering I

Nathan

Announcements

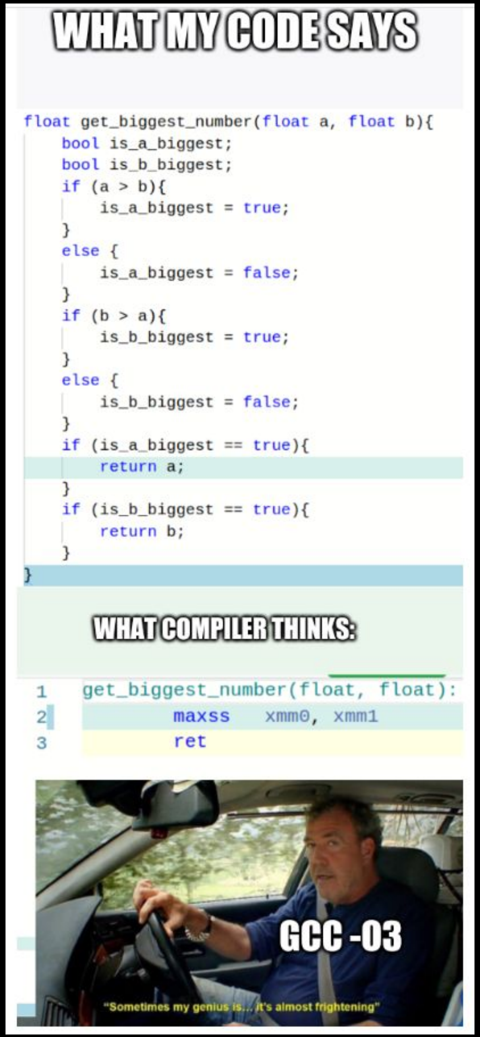
- Server and auth bot will be up by next Thursday
    - Email us if you need UIUC role

- O2F, 3rd Place! 100$!

- Fall recruitment event, need challenges!

- Purdue Oct 16-17
    - looking for PWN 2 presenter

sigpwny{plz_no_nsa_backdoor}



```
float get_biggest_number(float a, float b){
    bool is_a_biggest;
    bool is_b_biggest;
    if (a > b){
        is_a_biggest = true;
    }
    else {
        is_a_biggest = false;
    }
    if (b > a){
        is_b_biggest = true;
    }
    else {
        is_b_biggest = false;
    }
    if (is_a_biggest == true){
        return a;
    }
    if (is_b_biggest == true){
        return b;
    }
}
```

```
get_biggest_number(float, float):
        maxss   xmm0, xmm1
        ret
```

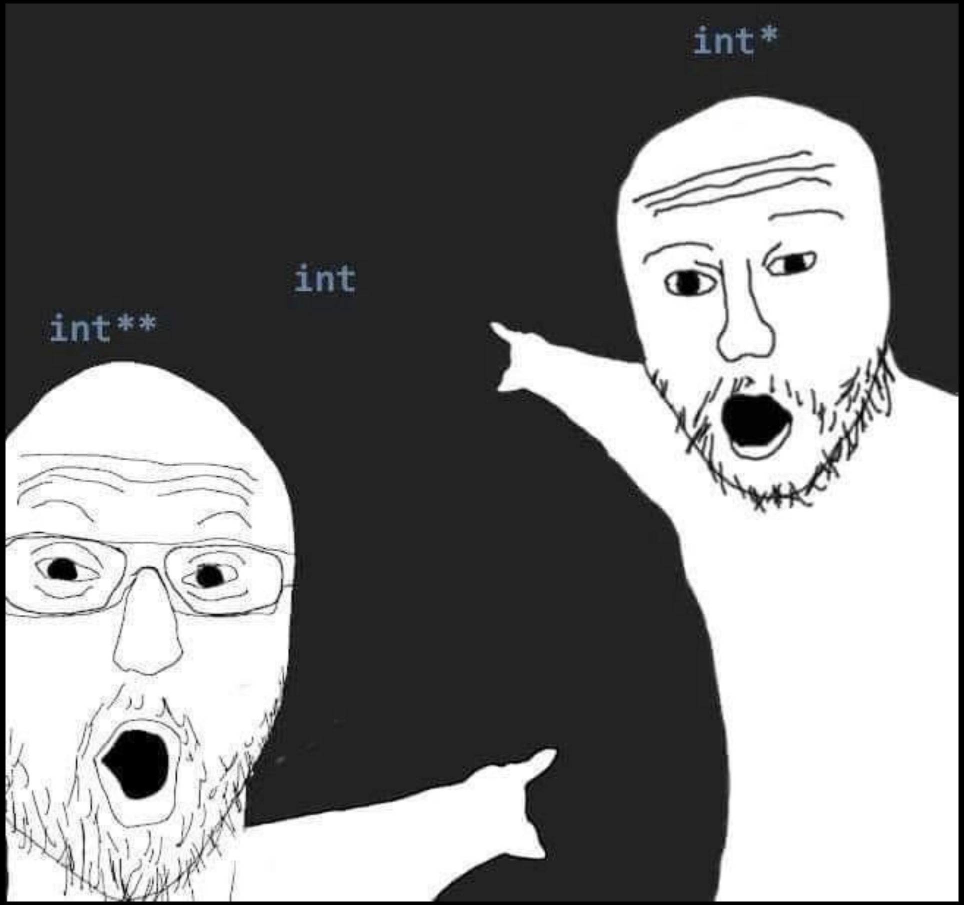


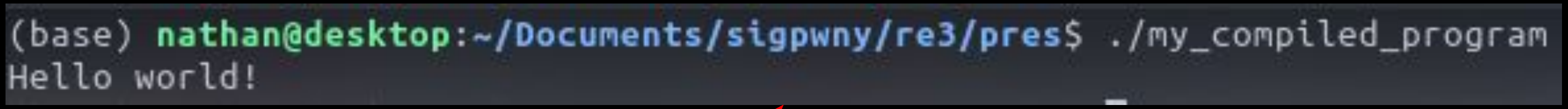
Table of Contents

- RE
  - What is reverse engineering?
  - Compilation
  - Executables
- Ghidra
- GDB

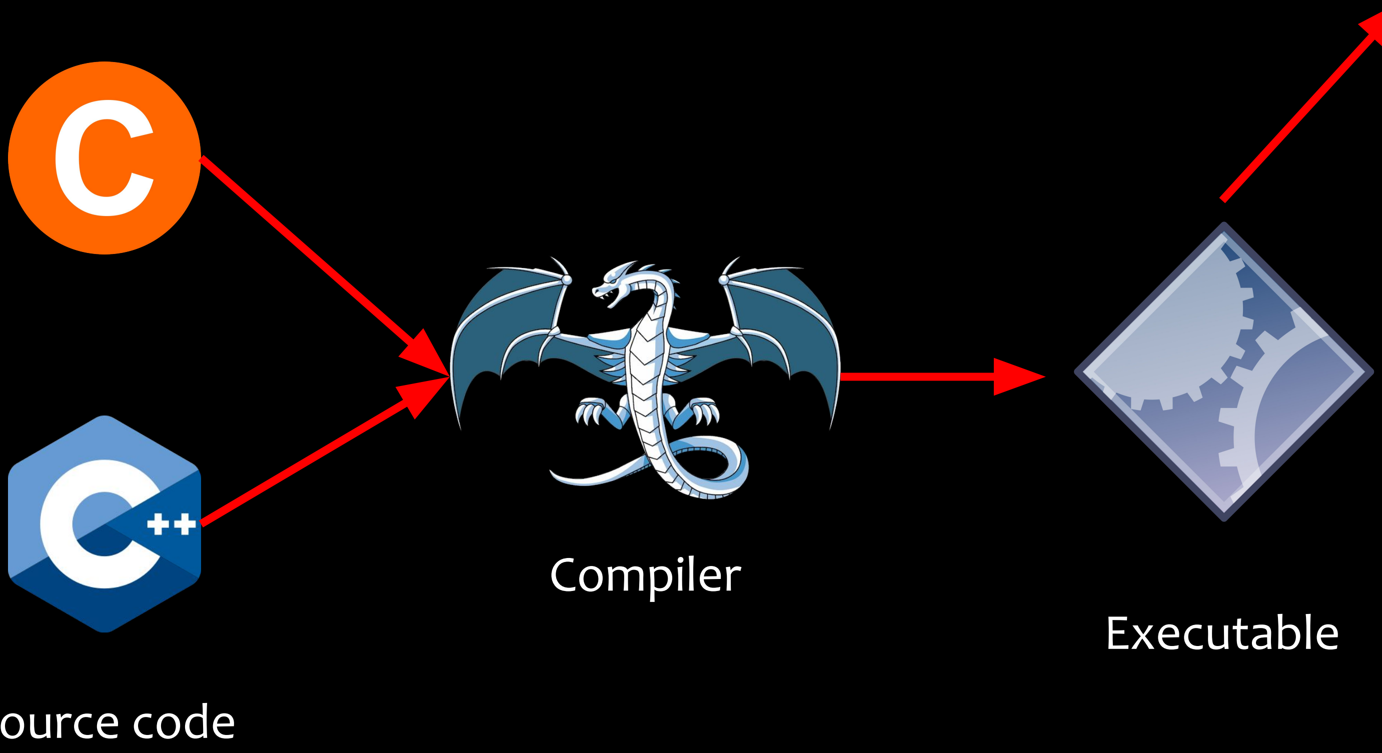
What is reverse engineering?

- Given a program, figure out what it does and how it works
  - Can we crack programs and write keygens?
  - Can we obtain secrets from the program?
    - Rocket league decryption key for game assets
  - Can we look for a flaw in the logic to find bugs?
- Programs can be written in C/C++, Java, Python … which all require different strategies to RE
  - We will focus on C/C++ programs compiled for Linux

Compilation



```
(base) nathan@desktop:~/Documents/sigpwny/re3/pres$ ./my_compiled_program
Hello world!
```



Compiler

Executable

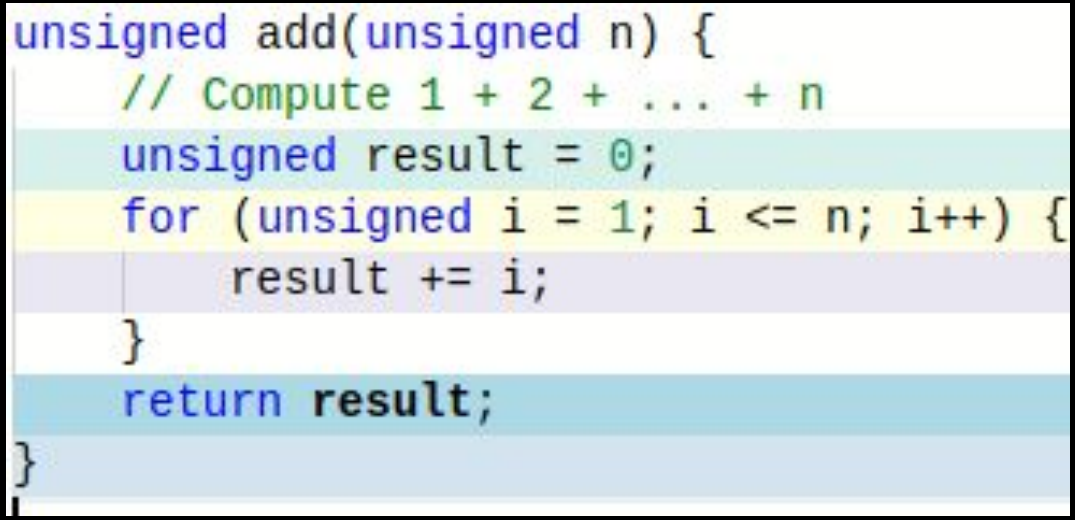
Source code

Executable

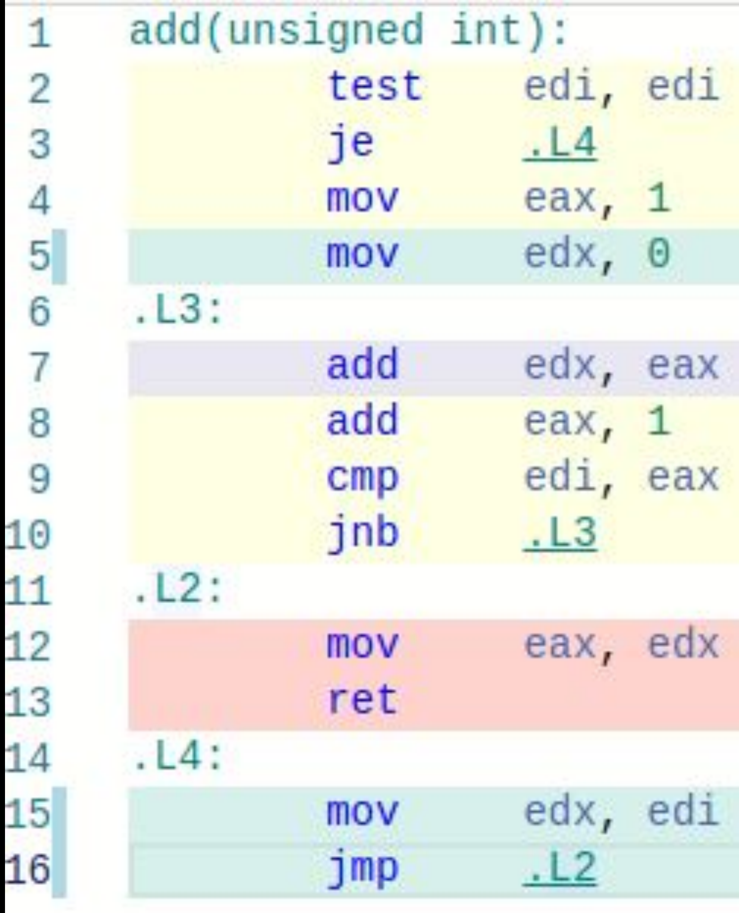
- Contains machine code (x86, ARM, …) that your processor understands
  - Hard for humans to understand, though!

- Uses registers and a stack, among other things
  - Register = 64 bit number (can be a number or a pointer)
    - Think of this as a general purpose variable
  - Stack = memory you can push and pop (used for function calls)
  - Heap = malloc'd memory
  - Data segment = memory where global variables are at

Reverse it!



```c
unsigned add(unsigned n) {
    // Compute 1 + 2 + ... + n
    unsigned result = 0;
    for (unsigned i = 1; i <= n; i++) {
        result += i;
    }
    return result;
}
```



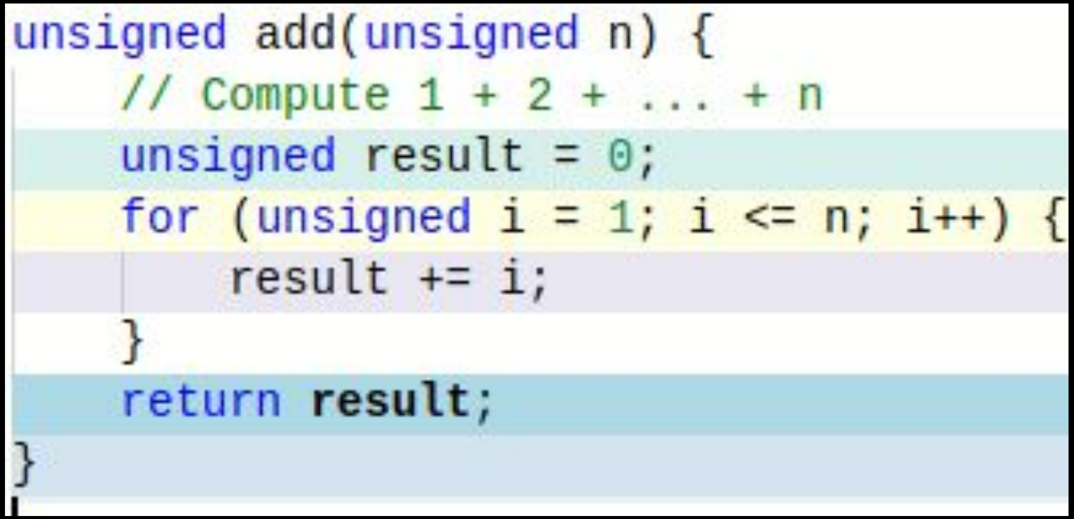
```asm
add(unsigned int):
        test    edi, edi
        je      .L4
        mov     eax, 1
        mov     edx, 0
.L3:
        add     edx, eax
        add     eax, 1
        cmp     edi, eax
        jnb     .L3
.L2:
        mov     eax, edx
        ret
.L4:
        mov     edx, edi
        jmp     .L2
```

https://godbolt.org/

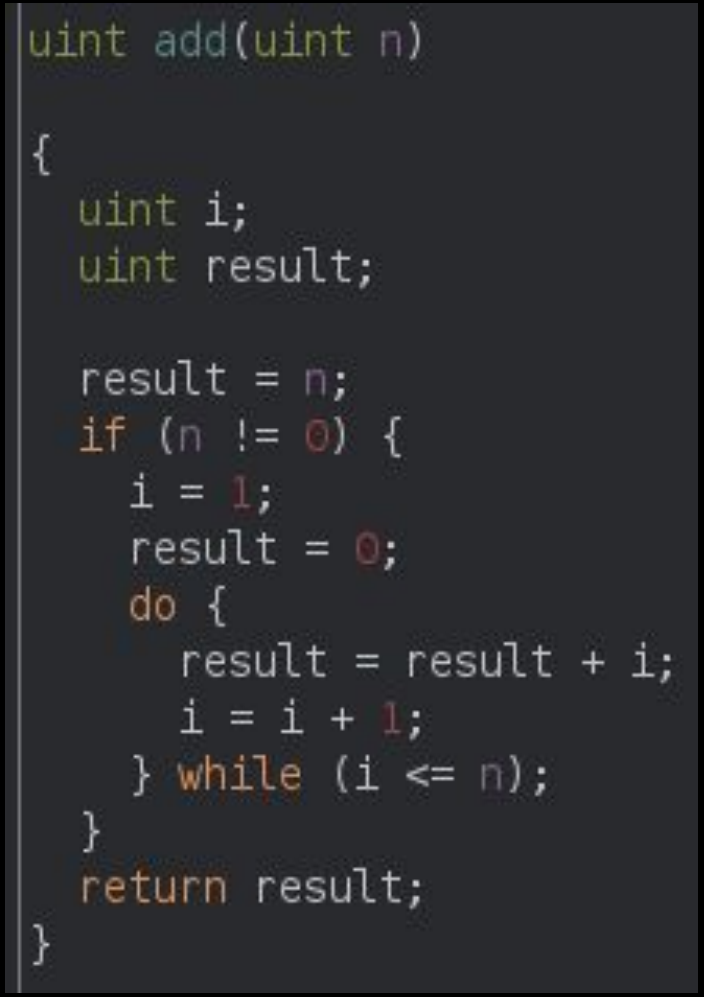
Ghidra to the rescue!

- Open source disassembler/decompiler
  - Transforms executable to disassembly
  - Can decompile disassembly to pseudo-C
- Written by the NSA 😳

Ghidra to the rescue!



```
unsigned add(unsigned n) {
    // Compute 1 + 2 + ... + n
    unsigned result = 0;
    for (unsigned i = 1; i <= n; i++) {
        result += i;
    }
    return result;
}
```



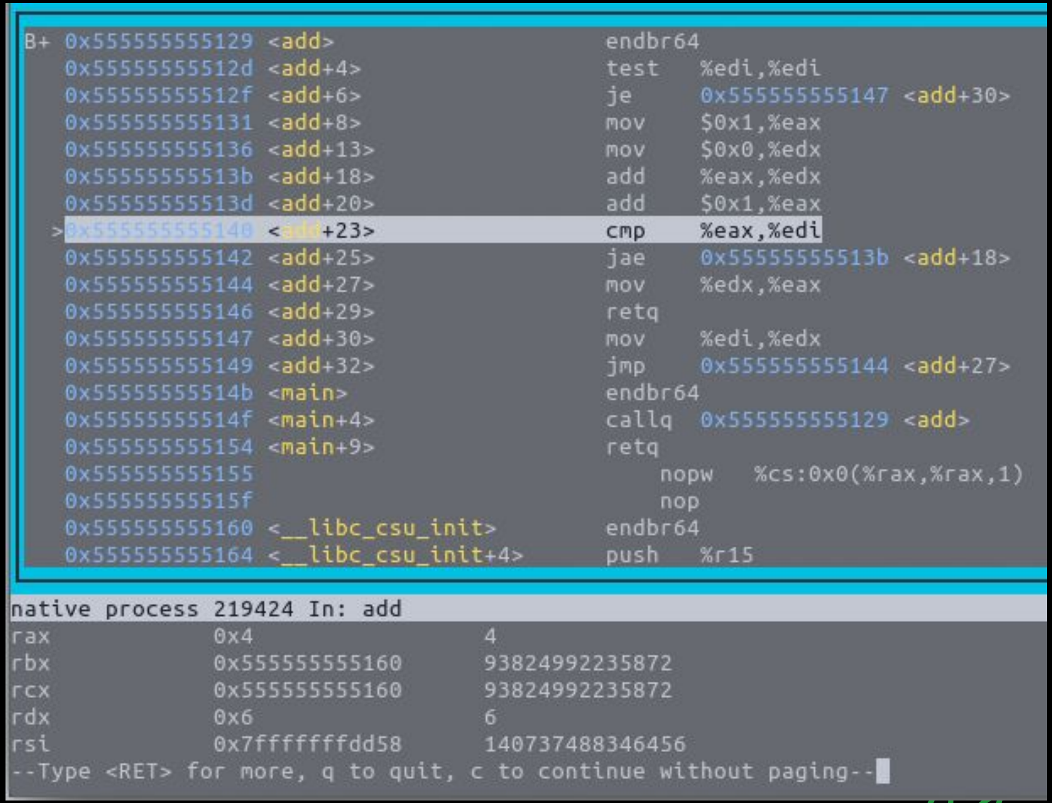
```
uint add(uint n)

{
  uint i;
  uint result;

  result = n;
  if (n != 0) {
    i = 1;
    result = 0;
    do {
      result = result + i;
      i = i + 1;
    } while (i <= n);
  }
  return result;
}
```

Ghidra follow along

Open Ghidra!

Dynamic Analysis with GDB

- GDB can debug assembly
- You can show the state of registers, the stack, and other memory
- Takes some getting used to!



```
B+ 0x555555555129 <add>                    endbr64
   0x55555555512d <add+4>                  test   %edi,%edi
   0x55555555512f <add+6>                  je     0x555555555147 <add+30>
   0x555555555131 <add+8>                  mov    $0x1,%eax
   0x555555555136 <add+13>                 mov    $0x0,%edx
   0x55555555513b <add+18>                 add    %eax,%edx
   0x55555555513d <add+20>                 add    $0x1,%eax
  >0x555555555140 <add+23>                 cmp    %eax,%edi
   0x555555555142 <add+25>                 jae    0x55555555513b <add+18>
   0x555555555144 <add+27>                 mov    %edx,%eax
   0x555555555146 <add+29>                 retq
   0x555555555147 <add+30>                 mov    %edi,%edx
   0x555555555149 <add+32>                 jmp    0x555555555144 <add+27>
   0x55555555514b <main>                   endbr64
   0x55555555514f <main+4>                 callq  0x555555555129 <add>
   0x555555555154 <main+9>                 retq
   0x555555555155                              nopw   %cs:0x0(%rax,%rax,1)
   0x55555555515f                              nop
   0x555555555160 <__libc_csu_init>        endbr64
   0x555555555164 <__libc_csu_init+4>      push   %r15

native process 219424 In: add
rax            0x4                 4
rbx            0x555555555160      93824992235872
rcx            0x555555555160      93824992235872
rdx            0x6                 6
rsi            0x7fffffffdd58      140737488346456
--Type <RET> for more, q to quit, c to continue without paging--
```

GDB follow along

GHIDRA CHEAT SHEET

Get started:

- View all functions in list on left side of screen. Double click main to decompile main

Decompiler:

- Middle click a variable to highlight all instances in decompilation
- Type “L” to rename variable
- “Ctrl+L” to retype a variable
- Type “;” to add an inline comment on the decompilation and assembly
- Alt+Left Arrow to navigate back to previous function

General:

- Double click an XREF to navigate there
- Search -> For Strings -> Search to find all strings (and XREFs)
- Choose Window -> Function Graph for a graph view of disassembly

GDB CHEAT SHEET

- “b main” - Set a breakpoint on the main function
  - “b *main+10” - Set a breakpoint a couple instructions into main
- “r” - run
  - “r arg1 arg2” - Run program with arg1 and arg2 as command line arguments. Same as ./prog arg1 arg2
  - “r < myfile” - Run program and supply contents of myfile.txt to stdin
- “c” - continue
- “si” - step instruction (steps into function calls)
- “ni” - next instruction (steps over function calls)
- “x /32xb 0x5555555551b8” - Display 32 hex bytes at address 0x5555555551b8
  - “x /4xg addr” - Display 4 hex “giants” (8 byte numbers) at addr
  - “x /16i $pc” - Display next 16 instructions at $rip
  - “x /s addr” - Display a string at address
- “info registers” - Display registers
- “info file” or “info proc map” - Display memory mappings
- “layout asm” - Get a split screen window to step through assembly

Go try for yourself!

https://ctf.sigpwny.com

- Start with re_intro
- All can be solved with Ghidra. (debugger will be very easy with GDB!)
- Practice practice practice! Ask for help!

Next Meetings

Weekend Seminar: Reverse Engineering II

- Explore more advanced RE tools + methods
- Explore more complicated obfuscation

Next Thursday: Pwn I

- Go over pwn fundamentals
- How to exploit programs with vulnerabilities