

FA2022 Week 04

Reverse Engineering I

Richard and Pete



Announcements

- Fall CTF 2022
 - This Saturday, CIF 3039 12 - 6PM!
 - Bring your friends
- No meeting this Sunday



ctf.sigpwny.com


sigpwny{plz_no_nsa_backdoor}

WHAT MY CODE SAYS

```
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;  
    }  
}
```

WHAT COMPILER THINKS:

```
1 get_biggest_number(float, float):  
2   maxss  xmm0, xmm1  
3   ret
```



GCC-03

"Sometimes my genius is... it's almost frightening"

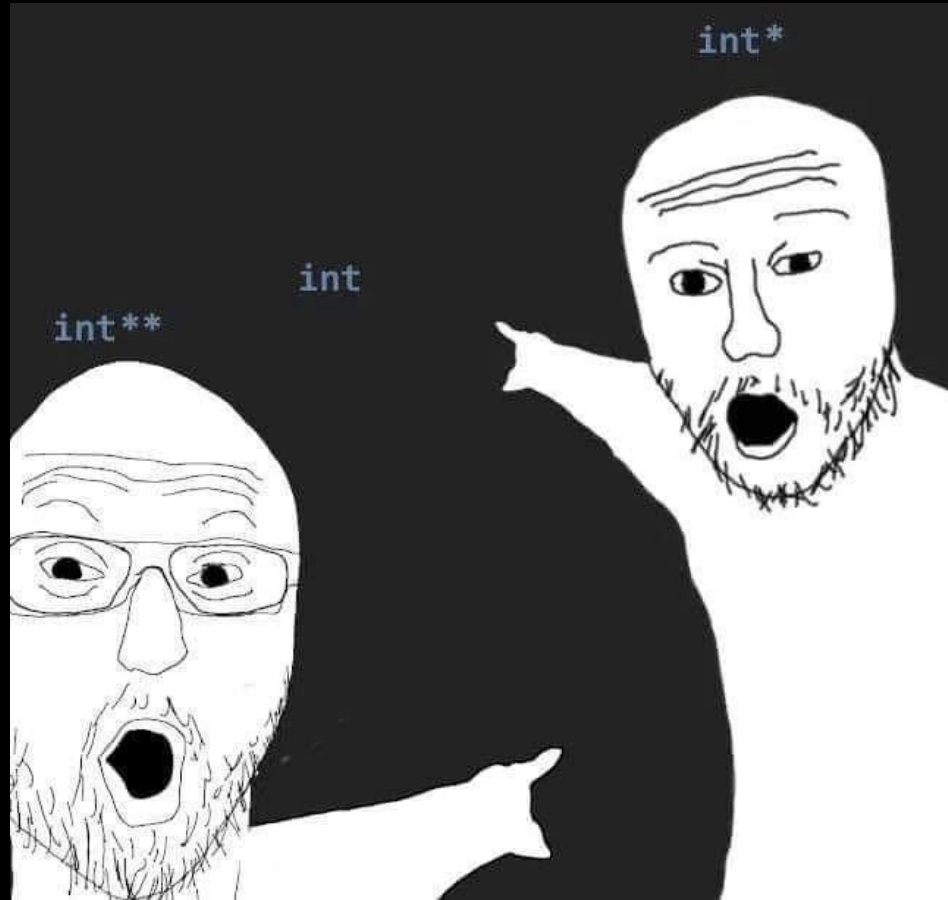


Table of Contents

- RE
 - What is reverse engineering?
 - Compilation
 - Executables
 - Static vs dynamic analysis
- Ghidra
 - Demo
- GDB
 - Demo



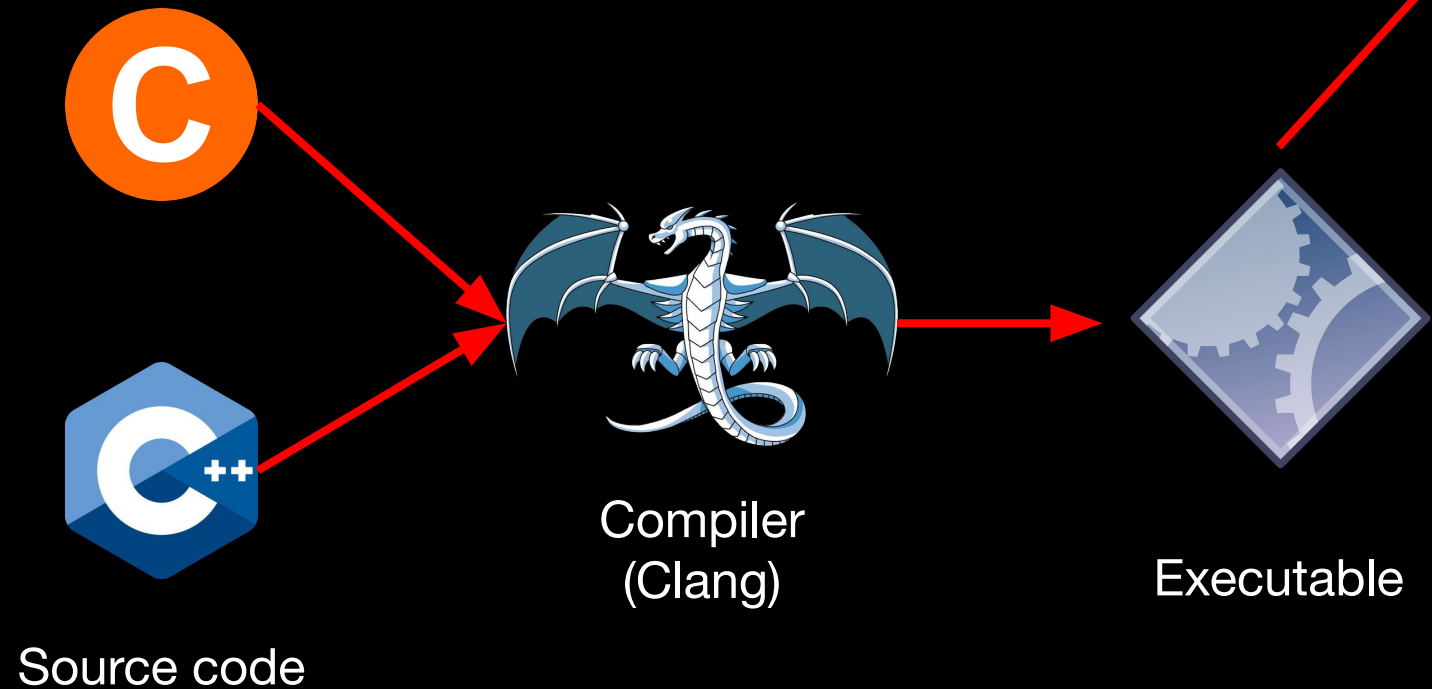
What is Reverse Engineering?

- Figure out how a program works
 - Crack programs and write keygens?
 - Find secrets in the program?
 - Find bugs in the code?
- Many different languages, different strategies for RE
 - Today: C/C++ on Linux ("ELF binaries")



Compilation

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



Executable

- Processor understands machine code
- Registers & stack
 - Register: store 64 bit number
 - Stack: function local variables
 - Heap: malloc'd memory
 - Data segment: global variables



Static vs Dynamic Analysis

- Static
 - Tools: Ghidra
 - Decompile
 - Debugging analogy: read source code
- Dynamic
 - Tools: GNU Debugger (GDB)
 - Run the program
 - Set breakpoints, step through, try various inputs
 - Debugging analogy: print statements after running



Reverse it!

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

```
1  add(unsigned int):  
2      test    edi, edi  
3      je     .L4  
4      mov    eax, 1  
5      mov    edx, 0  
6  .L3:  
7      add    edx, eax  
8      add    eax, 1  
9      cmp    edi, eax  
10     jnb   .L3  
11  .L2:  
12     mov    eax, edx  
13     ret  
14  .L4:  
15     mov    edx, edi  
16     jmp   .L2
```



Ghidra to the rescue!

- Open source disassembler/decompiler
 - Disassembler: binary to assembly
 - Decompiler: assembly 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!

sigpwny.com/rev_setup

Download "debugger" from <https://ctf.sigpwny.com/challenges>



Dynamic Analysis with GDB

- Run program, with the ability to pause and resume execution
- View registers, stack, heap
- Steep learning curve
- Important: `chmod +x ./chal` to run file

```
B+ 0x55555555129 <add>          endbr64
0x5555555512d <add+4>        test   %edi,%edi
0x5555555512f <add+6>        je     0x55555555147 <add+30>
0x55555555131 <add+8>        mov   $0x1,%eax
0x55555555136 <add+13>       mov   $0x0,%edx
0x5555555513b <add+18>       add   %eax,%edx
0x5555555513d <add+20>       add   $0x1,%eax
> 0x55555555140 <add+23>       cmp   %eax,%edi
0x55555555142 <add+25>       jae   0x5555555513b <add+18>
0x55555555144 <add+27>       mov   %edx,%eax
0x55555555146 <add+29>       retq
0x55555555147 <add+30>       mov   %edi,%edx
0x55555555149 <add+32>       jmp   0x55555555144 <add+27>
0x5555555514b <main>         endbr64
0x5555555514f <main+4>       callq 0x55555555129 <add>
0x55555555154 <main+9>       retq
0x55555555155                nopw  %cs:0x0(%rax,%rax,1)
0x5555555515f                nop
0x55555555160 <__libc_csu_init> endbr64
0x55555555164 <__libc_csu_init+4> push  %r15
```

native process 219424 In: add

```
rax      0x4          4
rbx      0x55555555160 93824992235872
rcx      0x55555555160 93824992235872
rdx      0x6          6
rsi      0x7fffffffdd58 140737488346456
```

--Type <RET> for more, q to quit, c to continue without paging--

pwndbg

git clone

<https://github.com/pwndbg/pwsndbg>

cd pwndbg

./setup.sh

Breakpoint 1, 0x0000000000401150 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA

[REGISTERS]

```
RAX 0x401150 (main) ← push rbp
RBX 0x0
RCX 0x401290 (__libc_csu_init) ← endbr64
RDX 0x7fffffff1a8 → 0x7fffffff49a ← 'DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus'
RDI 0x1
RSI 0x7fffffff198 → 0x7fffffff47d ← '/home/richyliu/temp/debugger'
R8 0x7ffff7f90f10 (initial+16) ← 0x4
R9 0x7ffff7fc9040 (_dl_fini) ← endbr64
R10 0x7ffff7fc3908 ← 0xd00120000000e
R11 0x7ffff7fde680 (_dl_audit_preinit) ← endbr64
R12 0x7fffffff198 → 0x7fffffff47d ← '/home/richyliu/temp/debugger'
R13 0x401150 (main) ← push rbp
R14 0x0
R15 0x7ffff7ffd040 (_rtld_global) → 0x7ffff7ffe2e0 ← 0x0
RBP 0x1
RSP 0x7fffffff088 → 0x7ffff7d9fd90 (__libc_start_call_main+128) ← mov edi, eax
RIP 0x401150 (main) ← push rbp
```

[DISASM]

```
► 0x401150 <main>      push rbp
0x401151 <main+1>     mov rbp, rsp
0x401154 <main+4>     sub rsp, 0x40
0x401158 <main+8>     mov dword ptr [rbp - 4], 0
0x40115f <main+15>    mov dword ptr [rbp - 8], edi
0x401162 <main+18>    mov qword ptr [rbp - 0x10], rsi
0x401166 <main+22>    cmp dword ptr [rbp - 8], 2
0x40116a <main+26>    jge main+59                <main+59>

0x401170 <main+32>    movabs rdi, 0x402004
0x40117a <main+42>    call puts@plt                <puts@plt>

0x40117f <main+47>    mov dword ptr [rbp - 4], 1
```

[STACK]

```
00:0000 | rsp 0x7fffffff088 → 0x7ffff7d9fd90 (__libc_start_call_main+128) ← mov edi, eax
01:0008 |     0x7fffffff090 ← 0x0
02:0010 |     0x7fffffff098 → 0x401150 (main) ← push rbp
03:0018 |     0x7fffffff0a0 ← 0x100000000
04:0020 |     0x7fffffff0a8 → 0x7fffffff198 → 0x7fffffff47d ← '/home/richyliu/temp/debugger'
05:0028 |     0x7fffffff0b0 ← 0x0
06:0030 |     0x7fffffff0b8 ← 0x8e4494d77c28027e
07:0038 |     0x7fffffff0c0 → 0x7fffffff198 → 0x7fffffff47d ← '/home/richyliu/temp/debugger'
```

pwndbg> █

GDB Follow Along

Same file as Ghidra follow along (debugger)



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
 - `x/4gx {void*}$rcx` - Dereference pointer at `$rcx`, display 4 QWORDS
 - `p/d {int*}{int*}$rcx` - Dereference pointer to pointer at `$rcx` as decimal
- `info registers` - Display registers (shorthand: `i r`)
- [x86 Linux calling convention](#) ("System V ABI"): `RDI, RSI, RDX, RCX, R8, R9`



Go try for yourself!

- <https://ctf.sigpwny.com>
- Start with first_re
- Practice practice practice! Ask for help!



Next Meetings

2022-09-24 - This Saturday

- Fall CTF!!!
- Play in our annual beginners CTF!

2022-09-25 - This Sunday

- **No meeting!**

2022-09-29 - Next Thursday

- OSINT
- Open Source Intelligence - stalk your targets!!





SIGPwny