



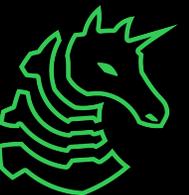
FA2023 Week 06 • 2023-10-05

Reverse Engineering II

Richard, Pete, Henry

Announcements

- DRM circumvention with Ojas this sunday
- No CTF this weekend
- Fall CTF solutions will be released soon™
- Fuzzing team meeting after this at 8PM
 - Weekly time moved to Thursday



ctf.sigpwny.com

sigpwny{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"

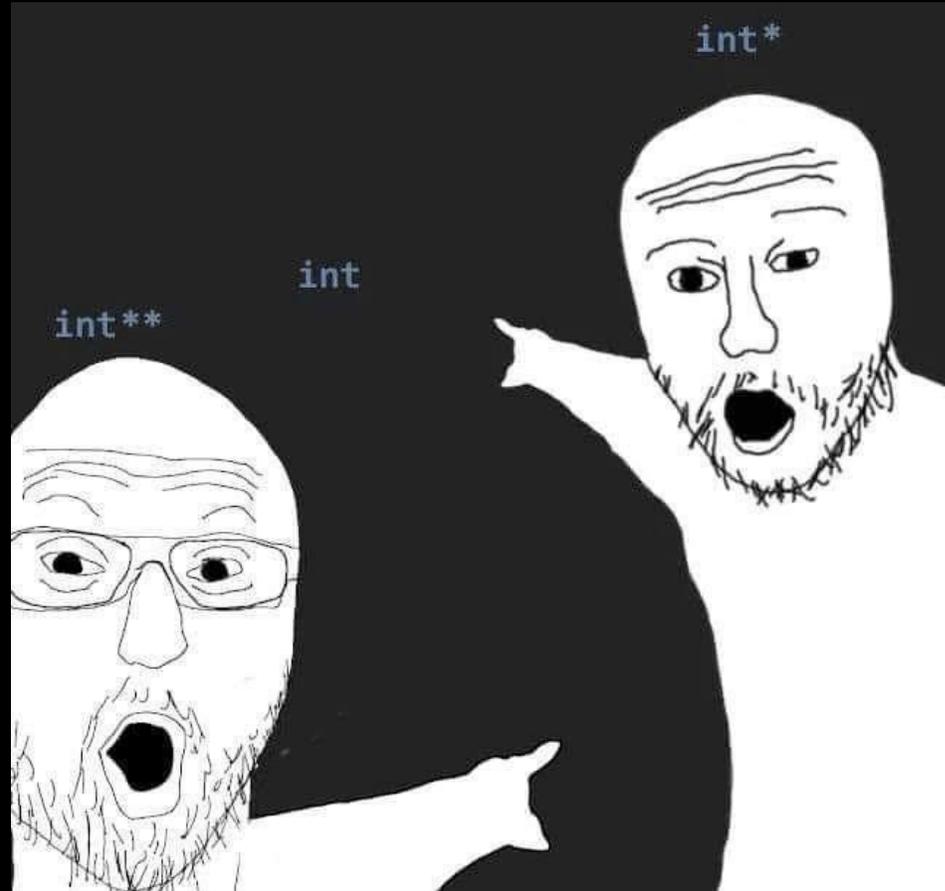


Table of Contents

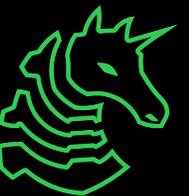
- Leaderboard / helper callout
- RE
 - Recap: What is reverse engineering?
 - Recap: Assembly
 - Compilation / Decompilation
 - Static analysis & optimization
- Ghidra
 - Demo
- GDB
 - Demo

If you haven't installed Ghidra yet, start downloading it through the slides here!
sigpwny.com/rev_setup23

Top 10 - differences from Week 4

Scoreboard

1	ronanboyarski	+3k points	26410
2	NullPoExc	+3k points	24515
3	caasher	+3k points	17065
4	mgcsstywth	+8k points (up 2 places)	14650
5	CBCicada	+2k points	9320
6	EhWhoAml	+.5k points	8645
7	aaronthewinner	+3k points (up 1 place)	7205
8	ilegosmaster	+4k points (NEW!)	6660
9	drizzle	+1.5k points	6175
10	SHAD0WV1RUS		5970



Want to be a helper?

Congratulate yourself - you made it to week 6 of meetings 🤘🤘🤘🤘

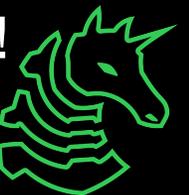
SIGPwny has a flipped leadership model - you get *invited* to become a helper

Some things we look for

- You frequently attend meetings and are actively engaged with the meeting content
- You interact with other club members
- You have a learning/teaching-focused mindset

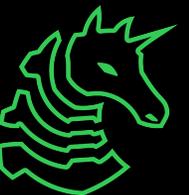
You demonstrate an interest in improving the club. This can be shown in various ways, such as contributing to **ongoing projects**, sharing your cybersecurity knowledge by **running a meeting / creating challenges / participating in CTFs**, or expressing interest in **{design, branding, outreach, or marketing}**

- talk to an admin / send a message on discord to let us know you want to help!
- See sigpwny.com/faq for more details



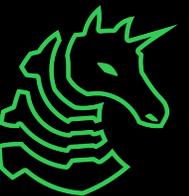
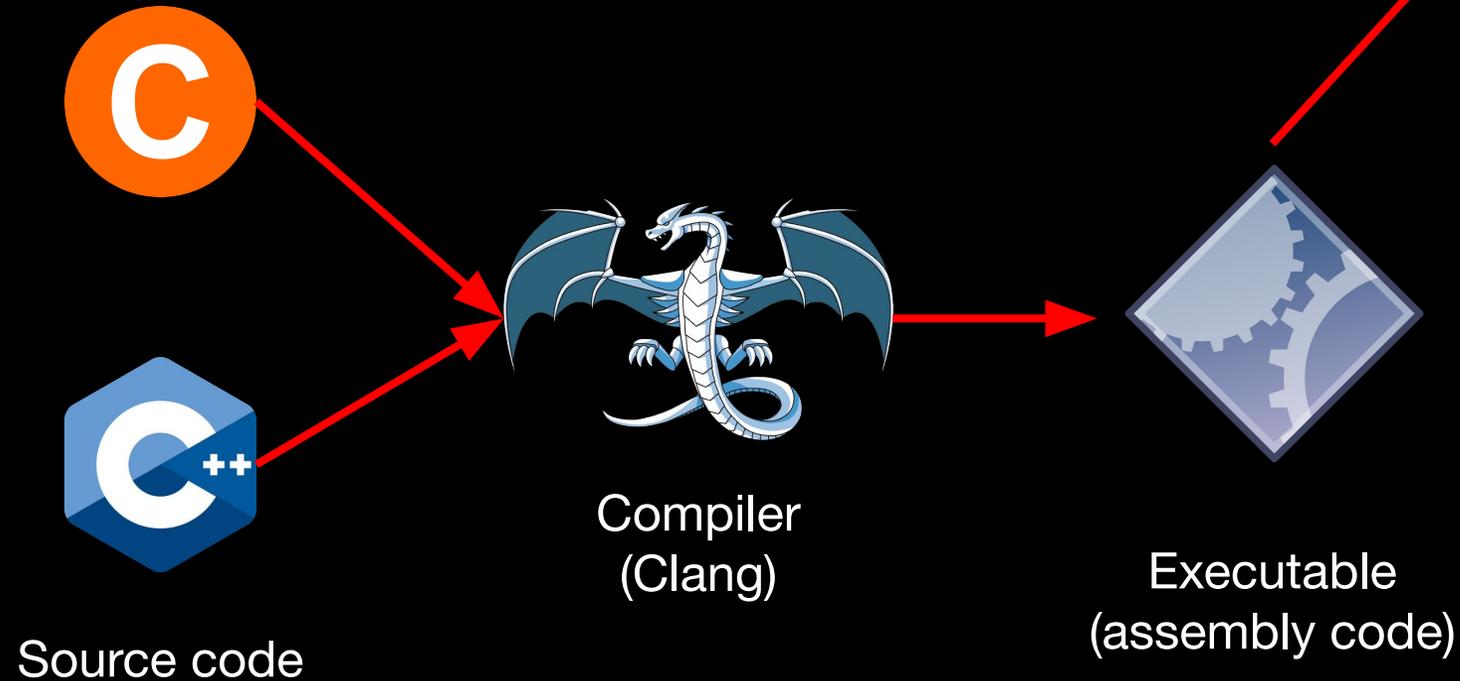
Recap: Reverse Engineering

- Reverse Engineering: Figure out how a program works
- Two major (non-exclusive) techniques
 - Static analysis (today: **Ghidra**)
 - Dynamic analysis (today: **GDB**)
- Different strategies for RE
 - Today: C / C++ on Linux ("ELF binaries")
 - Later: Java Rev, Rev III (Side channels, VMs, Symbolic execution)



Compilation

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



Recap: Assembly

[Sam's slides from Sunday](#)



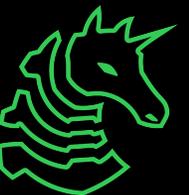
What is Assembly?

- A human-readable abstraction over CPU machine codes

0100100000000101110111011000000011011100010011

48 05 DE C0 37 13

add rax, 0x1337c0de

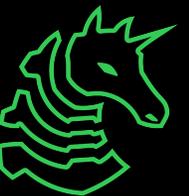


What is Assembly?

```
int method(int a){  
    int b = 6;  
    char c = 'c';  
    return a+b;  
}
```

method:

```
push    rbp  
mov     rbp, rsp  
mov     DWORD PTR [rbp-20], edi  
mov     DWORD PTR [rbp-4], 6  
mov     BYTE PTR [rbp-5], 99  
mov     edx, DWORD PTR [rbp-20]  
mov     eax, DWORD PTR [rbp-4]  
add     eax, edx  
pop     rbp  
ret
```



Basic CPU Structures

Instruction Memory

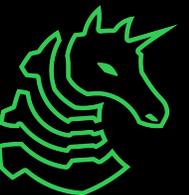
```
[0x00401000]
    ;-- section..text:
    ;-- segment.LOAD1:
entry0 ();
push    rsp
pop     rsi
xor     dl, 0x60
syscall
ret
```

Registers

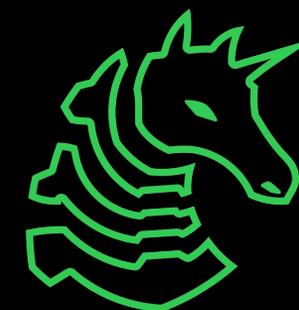
```
*RAX 0x3e8
*RBX 0x401300 (__libc_csu_init) ←
*RCX 0x7ffff7ea311b (getegid+11) ←
RDX 0x0
*RDI 0x7ffff7fad7e0 (_IO_stdfile_1) ←
RSI 0x0
R8 0x0
*R9 0x7ffff7fe0d60 (_dl_fini) ←
*R10 0x400502 ← 0x646967657465567
*R11 0x202
*R12 0x401110 (_start) ← endbr64
*R13 0x7ffff7ffddc0 ← 0x1
R14 0x0
R15 0x0
*RBP 0x7ffff7ffddcd0 ← 0x0
*RSP 0x7ffff7ffdcbb0 ← 0x0
*RIP 0x401220 (main+42) ← mov
```

Stack

```
0x7ffff7ffdcbb0 ← 0x0
0x7ffff7ffdcbb8 → 0x401110 (_start)
0x7ffff7ffddcc0 → 0x7ffff7ffddc0
0x7ffff7ffddcc8 ← 0x0
0x7ffff7ffddcd0 ← 0x0
0x7ffff7ffddcd8 → 0x7ffff7de3083
```



Compilation / Decompilation



We can go from C code to assembly...

```
1 int some_mathz() {  
2     int res = 0;  
3     for (int i = 9; i > 1; i++) {  
4         res -= i;  
5     }  
6 }
```

```
some_mathz():  
    push    rbp  
    mov     rbp, rsp  
    mov     DWORD PTR [rbp-4], 0  
    mov     DWORD PTR [rbp-8], 9  
    jmp     .L2  
  
.L3:  
    mov     eax, DWORD PTR [rbp-8]  
    sub     DWORD PTR [rbp-4], eax  
    add     DWORD PTR [rbp-8], 1  
  
.L2:  
    cmp     DWORD PTR [rbp-8], 1  
    jg     .L3  
    ud2
```

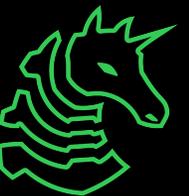
Now go from assembly to C code 🤖

```
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
```

Challenge: What does this do?

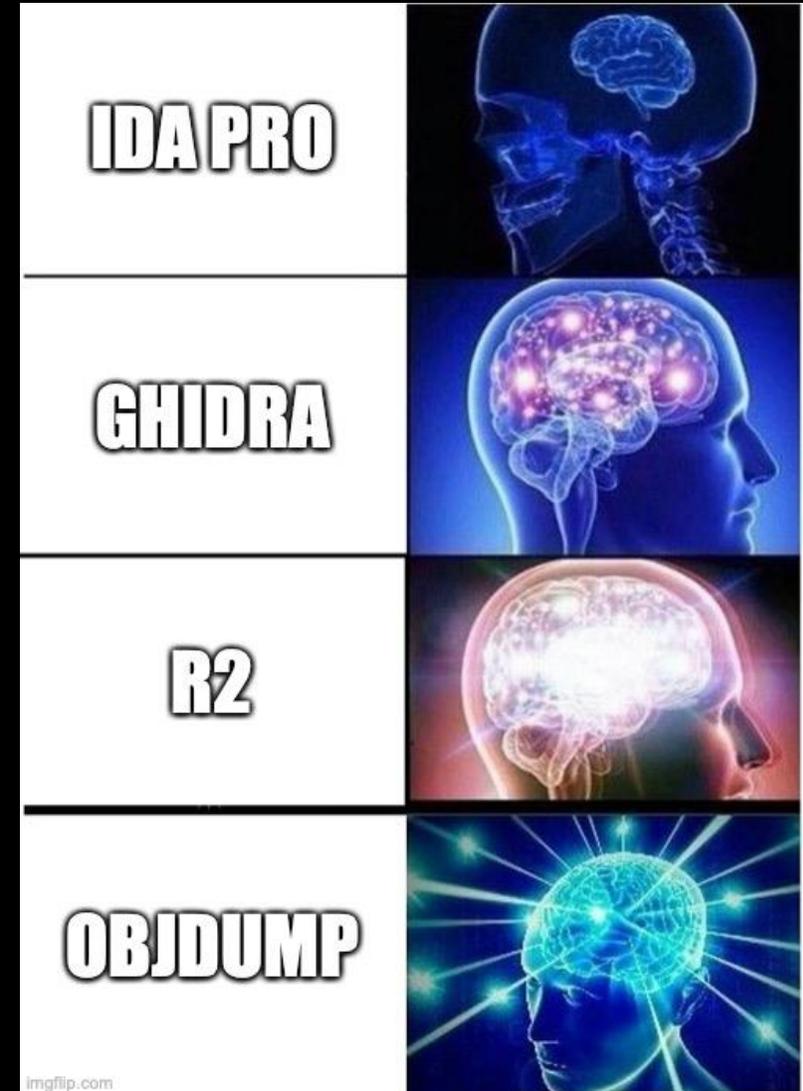
```
unsigned add(unsigned n) {
    // Compute 1 + 2 + ... + n
    unsigned result = 0;
    for (unsigned i = 1; i <= n; i++) {
        result += i;
    }
    return result;
}
```

Not perfect!



Ghidra to the rescue!

- Open source disassembler/decompiler
 - **Disassembler**: binary machine code to assembly
 - **Decompiler**: assembly to pseudo-C
- Written by the NSA 🤪

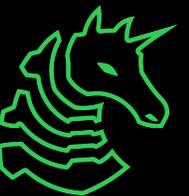


Ghidra to the rescue!

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

Decompilation not always the same! Many ways to write equivalent code

```
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;  
}
```



Common Optimizations

Loading an array with bytes

- Loading first 8 bytes simultaneously into stack (in one instruction)

```
#include <stdio.h>

int main() { 48 65 6c 6c 6f 20 77 6f 72 6c 64

    char string[] = "Hello world";
    printf("%s", string);

    return 0;
}
```

Challenge: why is the text of the decoded number backwards?

```
int __cdecl main(int _Argc, char **_Argv, char **_Env)
{
    undefined8 local_14;
    undefined4 local_c;

    __main();
    local_14 = 0x6f77206f6c6c6548;
    local_c = 0x646c72;
    printf("%s",&local_14);
    return 0;
}
```

"ow olleH" in hex

"dlr" in hex

Common Optimizations (Cont.)

Modulo replaced with mask

- % 4 replaced with & 0b11 (Taking the last two bits of unsigned int)

```
#include <stdio.h>

int main() {

    unsigned int A,B;
    scanf ("%u", &A);
    B = A % 4;
    printf ("%u", B);

    return 0;
}
```

```
int __cdecl main(int _Argc, char **_Argv, char **_Env)
{
    uint A;
    uint B;

    __main();
    scanf ("%u", &A);
    B = A & 0b00000011;
    printf ("%u", (ulonglong)B);
    return 0;
}
```

Ghidra Follow Along

Open Ghidra!

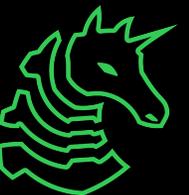
sigpwny.com/rev_setup23

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



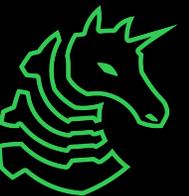
Ghidra Cheat Sheet

- Get started:
 - View all functions in list on left side of screen inside “Symbol Tree”. Double click **main** to decompile main
- Decompiler:
 - Middle click a variable to highlight all instances in decompilation
 - Type “L” to rename variable (after clicking on it)
 - “Ctrl+L” to retype a variable (type your type in the box)
 - 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 (Dynamic Analysis)

- Able to inspect a program's variables & state as it runs
- Set breakpoints, step through, try various inputs
- Debugging analogy: print statements after running



Dynamic Analysis with GDB

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

```
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>       jnp   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/pwndbg>

cd pwndbg

./setup.sh

Breakpoint 1, 0x000000000401150 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> █

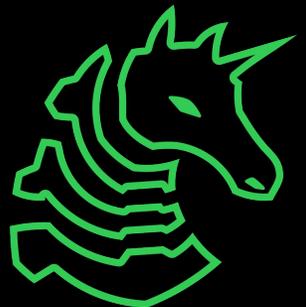
Windows users - WSL

Intel mac users - [pwn-docker](#)

M1/2 mac users - [still broken](#)

GDB Follow Along

Same file as Ghidra follow along (debugger)



GDB Cheat Sheet

[gdb](#)

[pwndbg](#)

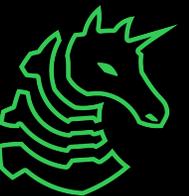
- `b main` - Set a breakpoint on the main function
 - `b *main+10` - Set a breakpoint a couple instructions into main
- `r` - run
 - `r arg1 arg2` - Run program with arg1 and arg2 as command line arguments. Same as `./prog arg1 arg2`
 - `r < myfile` - Run program and supply contents of myfile.txt to stdin
- `c` - continue
- `si` - step instruction (steps into function calls)
- `ni` - next instruction (steps over function calls) (`finish` to return to caller function)
- `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

*syscall calling convention is RDI, RSI, RDX, **R10**, R8, R9



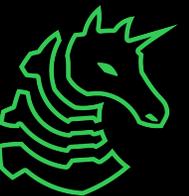
Pwndbg cheat sheet

- `emulate #` - Emulate the next # instructions
- `stack #` - Print # values on the stack
- `vmmap` - Print memory segments (use `-x` flag to show only executable segments)
- `nearpc` - Disassemble near the PC
- `tel <ptr>` - Recursively dereferences <ptr>
- `regs` - Use instead of `info reg` (gdb's register viewing)



Go try for yourself!

- <https://ctf.sigpwny.com>
- Start with Crackme 0
- Practice practice practice! Ask for help!



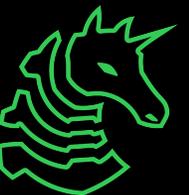
Next Meetings

2023-10-08 - This Sunday

- DRM circumvention with Ojas this sunday

2023-10-12 - Next Thursday

- Crypto I
- Learn the fundamentals of cryptography and encryption!



ctf.sigpwny.com

sigpwny{nsa_backdoor?}

Thanks for listening!

