



CS177 Computer Security Discussion

Spring 2020 - Week 8

Dongyu (Hector) Meng
May 18th



Today

- Administration changes for project 4
- Popular questions about `minecraft_hello`
- Hints and background knowledge for `lazy panel`



Some announcements

- Deadline for project 4 is midnight today (no change)
- Office hour 7-10pm today
- Office hour tomorrow will be canceled
- Quick walkthrough for the challenges of project 4 at the beginning of my office hour next week (May 26th, 1pm)



Questions about minecraft(_hello)

- In case you're still working on the first two challenges
- We got many private questions on piazza
- Being able to ask the right question is the first step towards a solution
- There's still time, not much, but enough for the first two challenges



Environment setup

- Pwntools installation
 - Try use a linux machine if you can (it's just easier...)
 - Working with vm is preferred (root, GUI)
 - If you're on CSIL, use python virtual environment tools ([@187](#))
- Mac user pwnlib.shellcraft not working
 - Use a linux machine
 - Generate the shellcode on a linux machine and load it in your script
 - Other tools / google for shellcode / DIY
 - It's the most common shellcode anyways



Environment setup

- GDB problems
 - Pwntools attach doesn't work
 - Make sure you have the latest version ([@187](#))
 - If you're on CSIL, need to set terminal context ([@187](#))
 - Last resort: `wait_for_debugger` and manually attach gdb ([@195](#))
 - “`./sysdeps/unix/sysv/linux/read.c: No such file or directory`”
 - GDB's fine, it's working
 - You're stepping through libc function, not very helpful

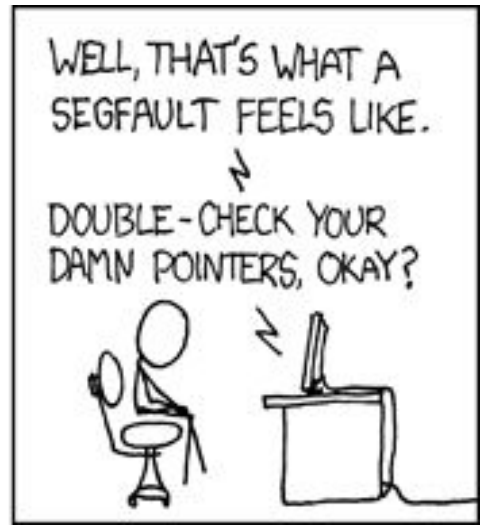


“How do I know I successfully get a shell?”

```
[DEBUG] Received 0x10 bytes:
  b'Very good, bye.\n'
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
  b'ls\n'
[DEBUG] Received 0x8e bytes:
  b'Dockerfile\t config  exploit.py  minecraft_hello.bin  minecraft_hello.c\n'
  b'build_docker.sh  core\t makefile  minecraft_hello.bin.idb  run.sh\n'
Dockerfile  config  exploit.py  minecraft_hello.bin  minecraft_hello.c
build_docker.sh  core  makefile  minecraft_hello.bin.idb  run.sh
$ cat /flag
```

“I got a segfault. Any idea?”

- Yes and no. You can get a segfault for many reasons.
- But in general, this means you're handling some pointer incorrectly





“I got a segfault. Any idea?”

- For this assignment, most likely your *esp* or *eip* is invalid
 - GDB is always a way out
-
- `>>> dmesg | tail -1`
 - `>>> [7027.856094] minecraft_hello[22542]:
segfault at deadbeef ip 0000000008048726 sp
00000000deadbeef error 4 in
minecraft_hello.bin[8048000+1000]`

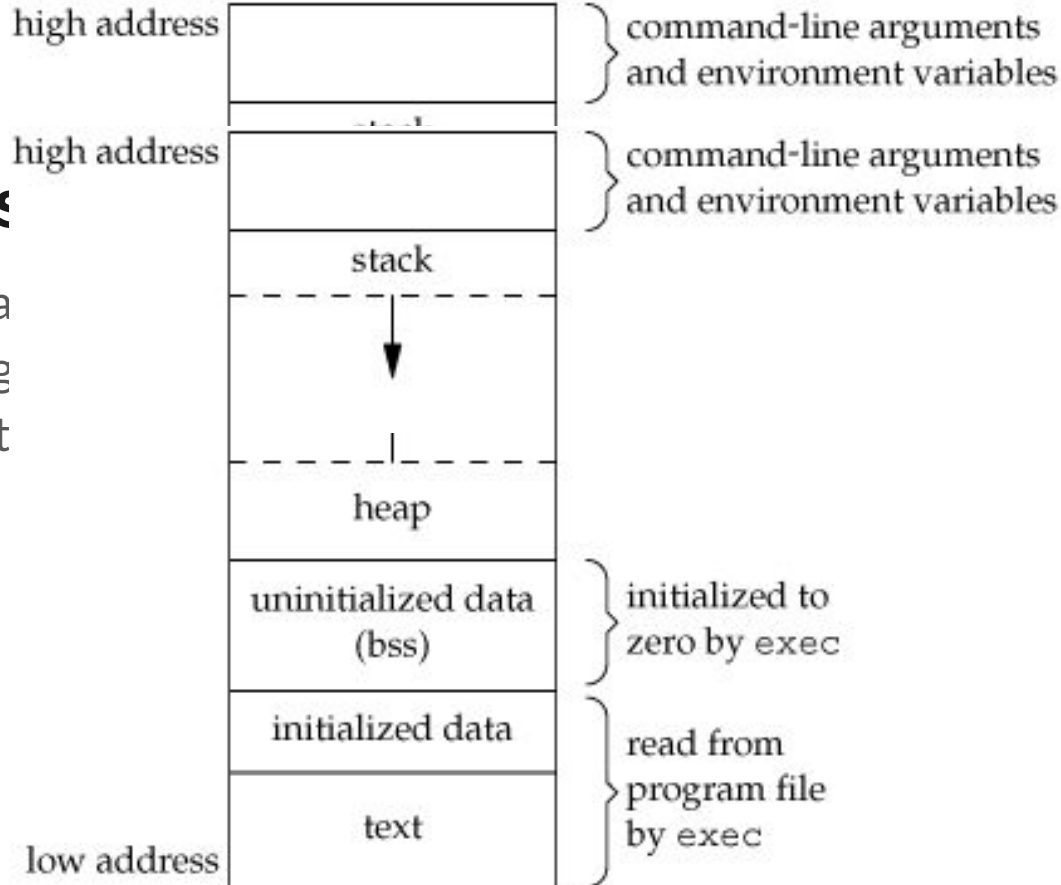


Diagnostic message

```
Minecraft_hello[22542]:                               filename[pid]
segfault at deadbeef
ip 0000000008048726
sp 00000000deadbeef
error 4
in minecraft_hello.bin[8048000+1000]   your code? lib?
```

“What is

- And “Ca
- Once ag
- That is t



address?”



“What is the address of win function?”

- Ghidra? IDA?
- Or if you prefer lightweight approaches:
- **objdump** -d minecraft_hello.bin | grep win



The dynamic way and the static way

- As an exercise, we hope you understand the details so that you can learn something
- Static way:
 - Reverse the code, understand what ecx does, where the return address is, one-shot perfect overflow
- Dynamic way:
 - “I know I want to keep ecx intact, but where is the return addr?”
 - “Let’s debug and see which offset match the final eip!”
 - “I’ll just copy paste the secret from memory!”



Extra credit: Lazy Panel



Lazy Panel

```
@mdy-personal > INSERT > ~/repo/cs...on/as...ts/4/lazy_panel >
./lazy_panel.bin
Hi there, welcome to the cs177 admin panel!
Please login...
Username:
aloha
Now tell me your password:
password
Ops, wrong password!
```



Lazy Panel

```
* bed @mdy-personal INSERT ~/repo/cs...d
./lazy_panel.bin
Hi there, welcome to the cs177 admin panel!
Please login...
Username:
aloha
Now tell me your password:
Super secure password
Welcome to the panel! What can I do for you?
give me the flag
Nah, I'm too lazy... Maybe next time...
```




Lazy Panel: reversing

- You don't have source code this time. Need to analyse the binary.
- What interesting (library) functions does the binary use?
- Use Ghidra decompiled code to have a quick grasp of program logic!
- The purpose of reversing is not to reconstruct the original program pitch perfect. It's more about finding the critical information for you to proceed.

Program Trees

- lazy_panel.bin
 - .bss
 - .data
 - .got.plt
 - .got
 - dynamic
 - .fini_array
 - .init_array
 - .eh_frame
 - .eh_frame_hdr
 - .rodata
 - .fini

Symbol Tree

- panel_out
- puts
- puts
- rand
- register_tm_clones
- setbuf
- setbuf
- strncmp
- strncmp

Data Type Manager

- Data Types
 - BuiltinTypes
 - lazy_panel.bin
 - generic_clib
 - generic_clib_64

Listing: lazy_panel.bin

0040077a	108	65 66 67 68	MOV	RDX, 0x706f6e6dc6b6a69
00400784	108	48 ba 69	MOV	qword ptr [RBP + local_98], RAX
0040078b	108	48 89 85	MOV	qword ptr [RBP + local_90], RDX
00400792	108	78 ff ff ff	MOV	RAX, 0x7877767574737271
0040079c	108	72 73 74	MOV	RDX, 0x4645444342417a79
004007a6	108	75 76 77 78	MOV	qword ptr [RBP + local_88], RAX
004007aa	108	48 ba 79	MOV	qword ptr [RBP + local_80], RDX
004007ae	108	7a 41 42	MOV	RAX, 0x2e2e2e4b4a494847
004007b8	108	43 44 45 46	MOV	qword ptr [RBP + local_78], RAX
004007bc	108	48 89 45 80	MOV	byte ptr [RBP + local_70], 0x0
004007c0	108	48 89 55 88	MOV	dword ptr [RBP + local_c], 0x0
004007c7	108	48 b8 47	JMP	LAB_0040081c
004007c9	108	48 49 4a	CALL	XREF[1]: 00400820(j)
004007ce	108	4b 2e 2e 2e	MOV	ECX, EAX
004007d0	108	48 89 45 90	MOV	EDX, 0x66666667
004007d5	108	c6 45 98 00	MOV	EAX, ECX
004007d7	108	c7 45 fc	IMUL	EDX
004007d9	108	00 00 00 00	SAR	EDX, 0x4
004007dc	108	eb 53	MOV	EAX, ECX
004007de	108	LAB_004007c9	SAR	EAX, 0x1f
004007e1	108	e8 12 fe	SAR	EDX, 0x1f
004007e3	108	ff ff	SUB	EDX, EAX
004007e5	108	89 c8	MOV	EAX, EDX
004007e5	108	89 d0	MOV	dword ptr [RBP + local_2c], EAX

Decompile: main - (lazy_panel.bin)

```

14 char acStack84 [20];
15 char acStack64 [20];
16 int local_2c;
17 char *local_28;
18 char *local_20;
19 undefined *local_18;
20 int local_c;
21
22 setbuf(stdout, (char *)0x0);
23 local_18 = local_68;
24 local_20 = acStack84;
25 local_28 = acStack64;
26 local_98 = 0x6867666564636261;
27 local_90 = 0x706f6e6dc6b6a69;
28 local_88 = 0x7877767574737271;
29 local_80 = 0x4645444342417a79;
30 local_78 = 0x2e2e2e4b4a494847;
31 local_70 = 0;
32 local_c = 0;
33 while (local_c < 0x10) {
34     iVar1 = rand();
35     local_2c = iVar1 % 0x28;
36     local_20[local_c] = *(char *)((long)&local_98 + (long)(iVar1 % 0x28));
37     local_c = local_c + 1;
38 }
39 local_20[0x10] = '\0';
40 puts("Hi there, welcome to the cs177 admin panel!");
41 puts("Please login...");
42 puts("Username:");
43 panel_in(local_18, 0x14);
44 puts("Now tell me your password:");
45 panel_in(local_28, 0x14);
46 iVar1 = strncmp(local_28, local_20, 0x10);
47 if (iVar1 == 0) {
48     panel_out("Welcome to the panel! What can I do for you?\n", 0x2d);
49     panel_in(local_108, 400);
50     puts("Nah, I'm too lazy... Maybe next time...");
51 }
52 else {
53     puts("Oops, wrong password!");
54 }
55 return 0;

```

Console - Scripting

<https://ghidra-sre.org/CheatSheet.html>

Function Call Trees: rand - (lazy_panel.bin)

Incoming Calls

Outgoing Calls

```
26 local_98 = 0x6867666564636261;
27 local_90 = 0x706f6e6d6c6b6a69;
28 local_88 = 0x7877767574737271;
29 local_80 = 0x4645444342417a79;
30 local_78 = 0x2e2e2e4b4a494847;
31 local_70 = 0;
32 local_c = 0;
33 while (local_c < 0x10) {
34     iVar1 = rand();
35     local_2c = iVar1 % 0x28;
36     local_20[local_c] = *(char *)((long)&local_98 + (long)(iVar1 % 0x28));
37     local_c = local_c + 1;
38 }
39 local_20[0x10] = '\0';
40 puts("Hi there, welcome to the cs177 admin panel!");
41 puts("Please login...");
42 puts("Username:");
43 panel_in(local_18,0x14);
44 puts("Now tell me your password:");
45 panel_in(local_28,0x14);
46 iVar1 = strncmp(local_28,local_20,0x10);
47 if (iVar1 == 0) {
48     panel_out("Welcome to the panel! What can I do for you?\n",0x2d);
49     panel_in(local_108,400);
50     puts("Nah, I\'m too lazy... Maybe next time...");
51 }
52 else {
53     puts("Oops, wrong password!");
54 }
```



rand

- The rand() function returns a **pseudo-random integer** in the range 0 to RAND_MAX inclusive (i.e., the mathematical range [0, RAND_MAX]).
- Read the [document](#)
- Learn about what [pseudo-random integer](#) means



strncmp

- `int strncmp(const char *s1, const char *s2, size_t n);`
- The `strncmp()` function compares the two strings `s1` and `s2`.
- It returns an integer less than, equal to, or greater than zero if `s1` is found, respectively, to be less than, to match, or be greater than `s2`.
- It compares the first (at most) `n` bytes of `s1` and `s2`.
- No, “`\x00`” is not equal to anything!



Static + Dynamic

- They actually help each other.
- After you get the essence of what the program is trying to do, debug wisely to get the information you need!

```
26 local_98 = 0x6867666564636261;
27 local_90 = 0x706f6e6d6c6b6a69;
28 local_88 = 0x7877767574737271;
29 local_80 = 0x4645444342417a79;
30 local_78 = 0x2e2e2e4b4a494847;
31 local_70 = 0;
32 local_c = 0;
33 while (local_c < 0x10) {
34     iVar1 = rand();
35     local_2c = iVar1 % 0x28;
36     local_20[local_c] = *(char *)((long)&local_98 + (long)(iVar1 % 0x28));
37     local_c = local_c + 1;
38 }
39 local_20[0x10] = '\0';
40 puts("Hi there, welcome to the cs177 admin panel!");
41 puts("Please login...");
42 puts("Username:");
43 panel_in(local_18,0x14);
44 puts("Now tell me your password:");
45 panel_in(local_28,0x14);
46 iVar1 = strncmp(local_28,local_20,0x10);
47 if (iVar1 == 0) {
48     panel_out("Welcome to the panel! What can I do for you?\n",0x2d);
49     panel_in(local_108,400);
50     puts("Nah, I\'m too lazy... Maybe next time...");
51 }
52 else {
53     puts("Oops, wrong password!");
54 }
```



libc

- libc has all the code you need to create a shell.
- C standard library and more
- `system("/bin/sh");`
- Need to know the address of `system`
- To know that, you first need to find the address of libc



GOT and PLT

- Global Offset Table and Procedure Linkage Table
- When the program is executed, addresses of library functions it uses will be loaded to GOT
- A table of function pointers
- Your program call the dummy functions in PLT to jump to those library functions
- `.got.plt` and `.plt` in Ghidra (Program Trees window)



Use leaked address

- To get to the function you care about, which is `system`
- ASLR only relocates an executable entirely
- The relative address between functions within library is not messed up!
- Find out which libc version it is and where each function is [here](#)
- Or load your libc into pwntools (ELF) and use it for analysis



ROP tools

- [angrop](#)
- [pwnlib.rop](#)



Thanks!

Good luck if you're not done yet!

Individual office hour starts at 7:00pm.