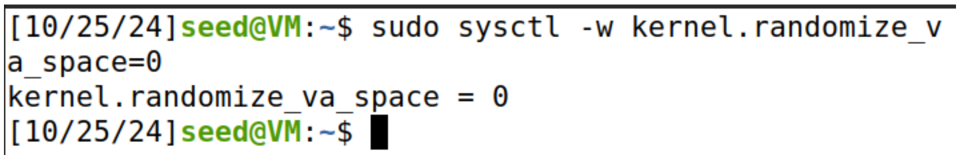
Buffer Overflow Attack Lab (Set-UID Version)

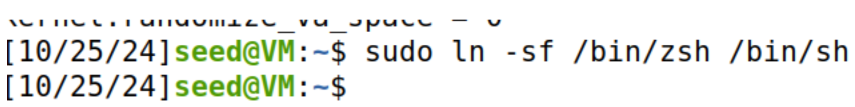
Name: Malithi Mithsara, Index Number 856558563

Environment Setup

Following codes related to setup the environment as given in lab.



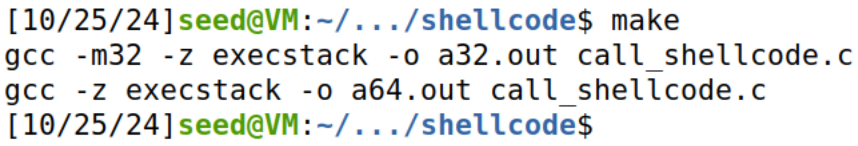
```
[10/25/24]seed@VM:~$ sudo sysctl -w kernel.randomize_v
a_space=0
kernel.randomize_va_space = 0
[10/25/24]seed@VM:~$ 
```



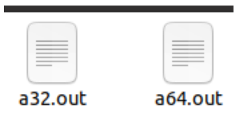
```
[10/25/24]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
[10/25/24]seed@VM:~$
```

Task 1: Getting Familiar with Shellcode

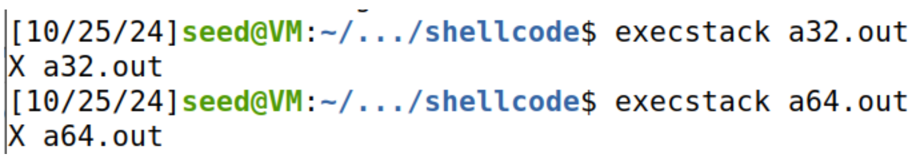
The goal of buffer-overflow attacks is to inject malicious code into the target program.



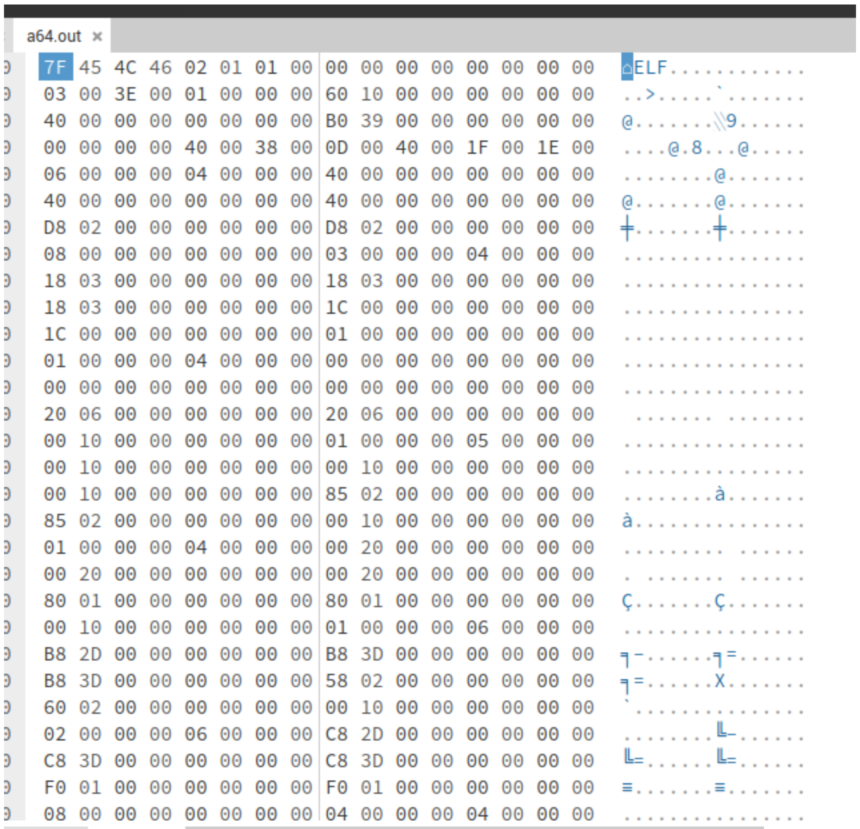
```
[10/25/24]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[10/25/24]seed@VM:~/.../shellcode$
```



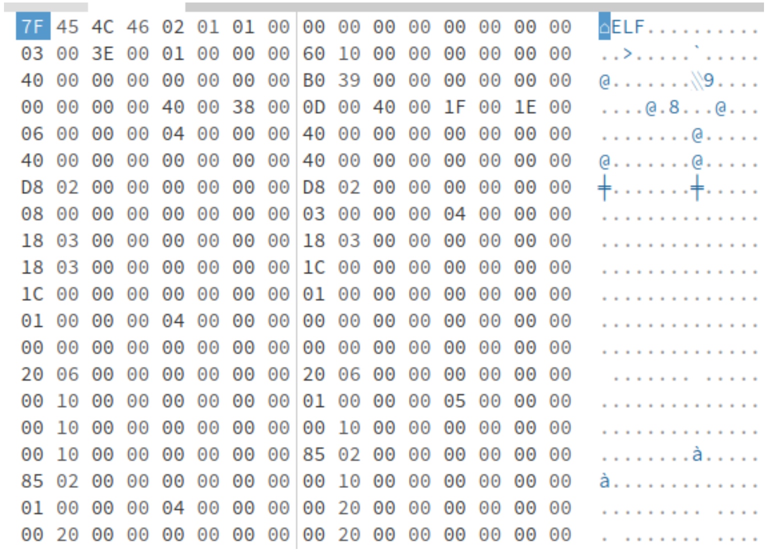
Running the a32.out and a64.out



```
[10/25/24]seed@VM:~/.../shellcode$ execstack a32.out
X a32.out
[10/25/24]seed@VM:~/.../shellcode$ execstack a64.out
X a64.out
```



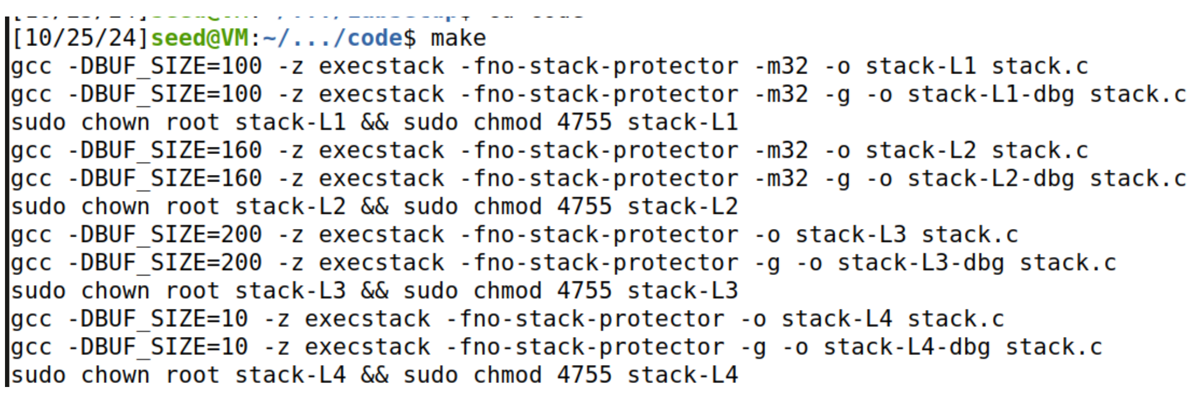
```
7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00  ⌂ELF............
03 00 3E 00 01 00 00 00 60 10 00 00 00 00 00 00  ..>.....`.......
40 00 00 00 00 00 00 00 B0 39 00 00 00 00 00 00  @.......░9......
00 00 00 00 40 00 38 00 0D 00 40 00 1F 00 1E 00  ....@.8...@.....
06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00  ........@.......
40 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  @.......@.......
D8 02 00 00 00 00 00 00 D8 02 00 00 00 00 00 00  ╪.......╪.......
08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00  ................
18 03 00 00 00 00 00 00 18 03 00 00 00 00 00 00  ................
18 03 00 00 00 00 00 00 1C 00 00 00 00 00 00 00  ................
1C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
20 06 00 00 00 00 00 00 20 06 00 00 00 00 00 00   ....... .......
00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00  ................
00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00  ................
00 10 00 00 00 00 00 00 85 02 00 00 00 00 00 00  ........à.......
85 02 00 00 00 00 00 00 00 10 00 00 00 00 00 00  à...............
01 00 00 00 04 00 00 00 00 20 00 00 00 00 00 00  ......... ......
00 20 00 00 00 00 00 00 00 20 00 00 00 00 00 00  . ....... ......
80 01 00 00 00 00 00 00 80 01 00 00 00 00 00 00  Ç.......Ç.......
00 10 00 00 00 00 00 00 01 00 00 00 06 00 00 00  ................
B8 2D 00 00 00 00 00 00 B8 3D 00 00 00 00 00 00  ╕-......╕=......
B8 3D 00 00 00 00 00 00 58 02 00 00 00 00 00 00  ╕=......X.......
60 02 00 00 00 00 00 00 00 10 00 00 00 00 00 00  `...............
02 00 00 00 06 00 00 00 C8 2D 00 00 00 00 00 00  ........╚-......
C8 3D 00 00 00 00 00 00 C8 3D 00 00 00 00 00 00  ╚=......╚=......
F0 01 00 00 00 00 00 00 F0 01 00 00 00 00 00 00  ≡.......≡.......
08 00 00 00 00 00 00 00 04 00 00 00 04 00 00 00  ................
```



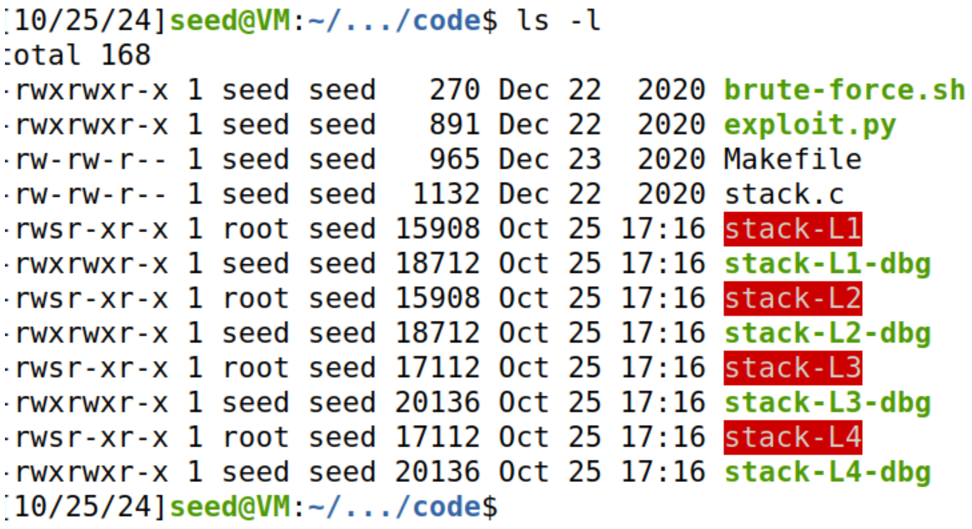
```
7F 45 4C 46 02 01 01 00 00 00 00 00 00 00 00 00  ⌂ELF..........
03 00 3E 00 01 00 00 00 60 10 00 00 00 00 00 00  ..>.....`.....
40 00 00 00 00 00 00 00 B0 39 00 00 00 00 00 00  @.......░9....
00 00 00 00 40 00 38 00 0D 00 40 00 1F 00 1E 00  ....@.8...@...
06 00 00 00 04 00 00 00 40 00 00 00 00 00 00 00  ........@.....
40 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  @.......@.....
D8 02 00 00 00 00 00 00 D8 02 00 00 00 00 00 00  ╪.......╪.....
08 00 00 00 00 00 00 00 03 00 00 00 04 00 00 00  ..............
18 03 00 00 00 00 00 00 18 03 00 00 00 00 00 00  ..............
18 03 00 00 00 00 00 00 1C 00 00 00 00 00 00 00  ..............
1C 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ..............
01 00 00 00 04 00 00 00 00 00 00 00 00 00 00 00  ..............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ..............
20 06 00 00 00 00 00 00 20 06 00 00 00 00 00 00   ....... .....
00 10 00 00 00 00 00 00 01 00 00 00 05 00 00 00  ..............
00 10 00 00 00 00 00 00 00 10 00 00 00 00 00 00  ..............
00 10 00 00 00 00 00 00 85 02 00 00 00 00 00 00  ........à.....
85 02 00 00 00 00 00 00 00 10 00 00 00 00 00 00  à.............
01 00 00 00 04 00 00 00 00 20 00 00 00 00 00 00  ......... ....
00 20 00 00 00 00 00 00 00 20 00 00 00 00 00 00  . ....... ....
```

Task 2: Understanding the Vulnerable Program

The compilation and setup commands are already included in Makefile, so we just need to type make to execute those commands. The variables L1, ..., L4 are set in Makefile; they will be used during the compilation.



```
[10/25/24]seed@VM:~/.../code$ make
gcc -DBUF_SIZE=100 -z execstack -fno-stack-protector -m32 -o stack-L1 stack.c
gcc -DBUF_SIZE=100 -z execstack -fno-stack-protector -m32 -g -o stack-L1-dbg stack.c
sudo chown root stack-L1 && sudo chmod 4755 stack-L1
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -o stack-L2 stack.c
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -g -o stack-L2-dbg stack.c
sudo chown root stack-L2 && sudo chmod 4755 stack-L2
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -o stack-L3 stack.c
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -g -o stack-L3-dbg stack.c
sudo chown root stack-L3 && sudo chmod 4755 stack-L3
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -o stack-L4 stack.c
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -g -o stack-L4-dbg stack.c
sudo chown root stack-L4 && sudo chmod 4755 stack-L4
```

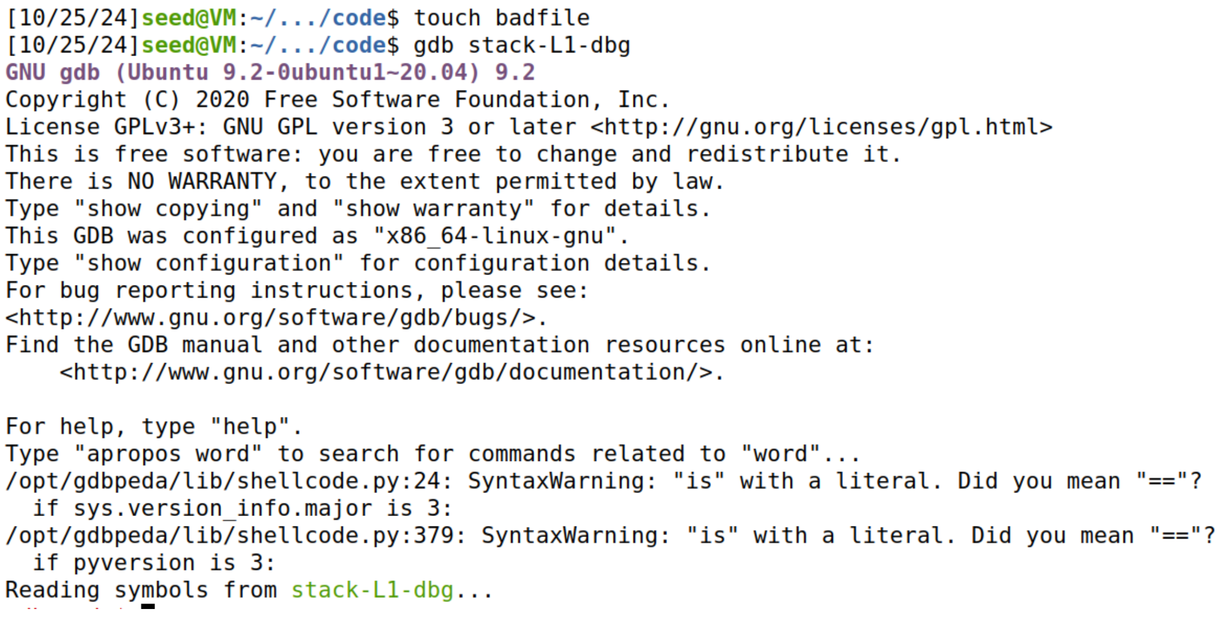


```
[10/25/24]seed@VM:~/.../code$ ls -l
total 168
-rwxrwxr-x 1 seed seed   270 Dec 22  2020 brute-force.sh
-rwxrwxr-x 1 seed seed   891 Dec 22  2020 exploit.py
-rw-rw-r-- 1 seed seed   965 Dec 23  2020 Makefile
-rw-rw-r-- 1 seed seed  1132 Dec 22  2020 stack.c
-rwsr-xr-x 1 root seed 15908 Oct 25 17:16 stack-L1
-rwxrwxr-x 1 seed seed 18712 Oct 25 17:16 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Oct 25 17:16 stack-L2
-rwxrwxr-x 1 seed seed 18712 Oct 25 17:16 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Oct 25 17:16 stack-L3
-rwxrwxr-x 1 seed seed 20136 Oct 25 17:16 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Oct 25 17:16 stack-L4
-rwxrwxr-x 1 seed seed 20136 Oct 25 17:16 stack-L4-dbg
[10/25/24]seed@VM:~/.../code$
```

Task 3: Launching Attack on 32-bit Program (Level 1)

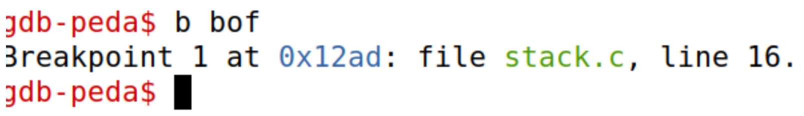
To exploit the buffer-overflow vulnerability in the target program, the most important thing to know is the distance between the buffer's starting position and the place where the return-address is stored.

First create an empty badfile. Then debug the stack-L1-dbg.



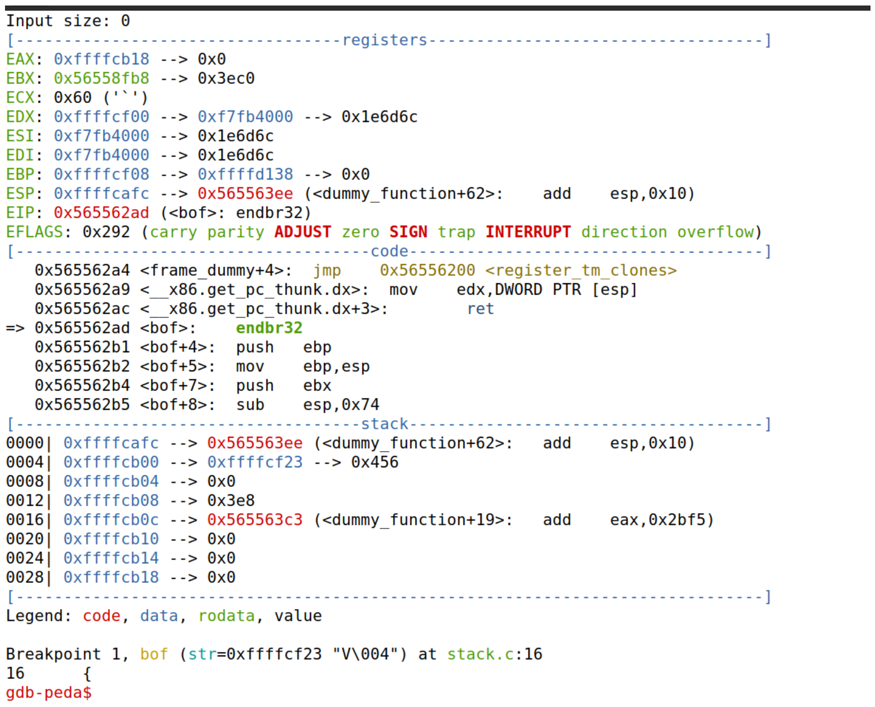
```
[10/25/24]seed@VM:~/.../code$ touch badfile
[10/25/24]seed@VM:~/.../code$ gdb stack-L1-dbg
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/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 stack-L1-dbg...
```

Set the break point into line 16.



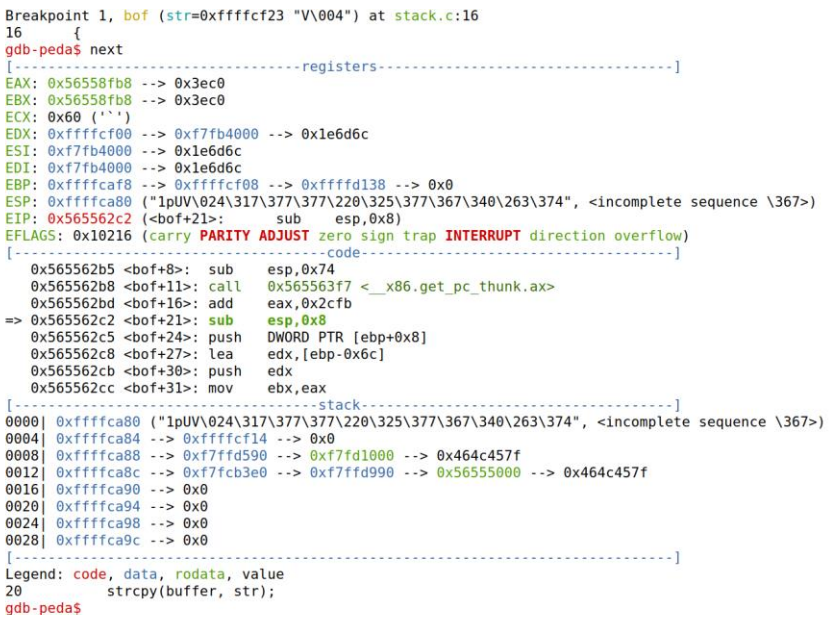
```
gdb-peda$ b bof
Breakpoint 1 at 0x12ad: file stack.c, line 16.
gdb-peda$
```

Start the execution of the program.



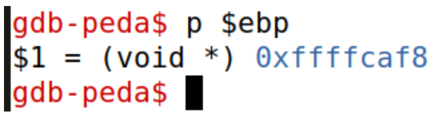
```
Input size: 0
[----------------------------------registers-----------------------------------]
EAX: 0xffffcb18 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf00 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcf08 --> 0xffffd138 --> 0x0
ESP: 0xffffcafc --> 0x565563ee (<dummy_function+62>:    add    esp,0x10)
EIP: 0x565562ad (<bof>: endbr32)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562a4 <frame_dummy+4>:  jmp    0x56556200 <register_tm_clones>
   0x565562a9 <__x86.get_pc_thunk.dx>:  mov    edx,DWORD PTR [esp]
   0x565562ac <__x86.get_pc_thunk.dx+3>:        ret
=> 0x565562ad <bof>:    endbr32
   0x565562b1 <bof+4>:  push   ebp
   0x565562b2 <bof+5>:  mov    ebp,esp
   0x565562b4 <bof+7>:  push   ebx
   0x565562b5 <bof+8>:  sub    esp,0x74
[------------------------------------stack-------------------------------------]
0000| 0xffffcafc --> 0x565563ee (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcb00 --> 0xffffcf23 --> 0x456
0008| 0xffffcb04 --> 0x0
0012| 0xffffcb08 --> 0x3e8
0016| 0xffffcb0c --> 0x565563c3 (<dummy_function+19>:   add    eax,0x2bf5)
0020| 0xffffcb10 --> 0x0
0024| 0xffffcb14 --> 0x0
0028| 0xffffcb18 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xffffcf23 "V\004") at stack.c:16
16      {
gdb-peda$
```

Next



```
Breakpoint 1, bof (str=0xffffcf23 "V\004") at stack.c:16
16      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf00 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcaf8 --> 0xffffcf08 --> 0xffffd138 --> 0x0
ESP: 0xffffca80 ("1pUV\024\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
EIP: 0x565562c2 (<bof+21>:      sub    esp,0x8)
EFLAGS: 0x10216 (carry PARITY ADJUST zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562b5 <bof+8>:  sub    esp,0x74
   0x565562b8 <bof+11>: call   0x565563f7 <__x86.get_pc_thunk.ax>
   0x565562bd <bof+16>: add    eax,0x2cfb
=> 0x565562c2 <bof+21>: sub    esp,0x8
   0x565562c5 <bof+24>: push   DWORD PTR [ebp+0x8]
   0x565562c8 <bof+27>: lea    edx,[ebp-0x6c]
   0x565562cb <bof+30>: push   edx
   0x565562cc <bof+31>: mov    ebx,eax
[------------------------------------stack-------------------------------------]
0000| 0xffffca80 ("1pUV\024\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
0004| 0xffffca84 --> 0xffffcf14 --> 0x0
0008| 0xffffca88 --> 0xf7ffd590 --> 0xf7fd1000 --> 0x464c457f
0012| 0xffffca8c --> 0xf7fcb3e0 --> 0xf7ffd990 --> 0x56555000 --> 0x464c457f
0016| 0xffffca90 --> 0x0
0020| 0xffffca94 --> 0x0
0024| 0xffffca98 --> 0x0
0028| 0xffffca9c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$
```

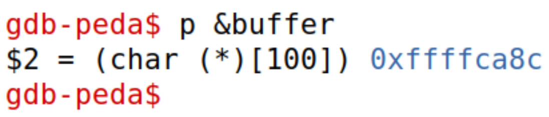
Get ebp value



```
gdb-peda$ p $ebp
$1 = (void *) 0xffffcaf8
gdb-peda$ 
```

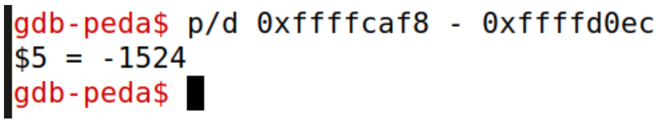
Ebp value is equal to 0xffffcaf8

Get the buffer's address



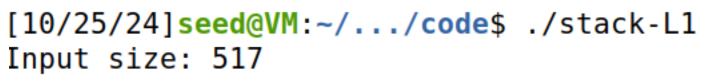
```
gdb-peda$ p &buffer
$2 = (char (*)[100]) 0xffffca8c
gdb-peda$
```

Buffer's address is equal to 0xfffca8c



```
gdb-peda$ p/d 0xffffcaf8 - 0xffffd0ec
$5 = -1524
gdb-peda$ 
```

Launching Attacks

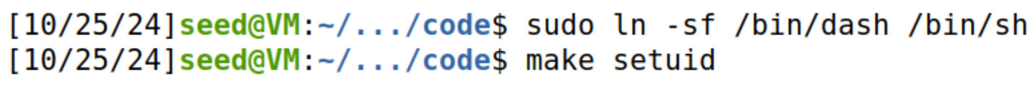


```
[10/25/24]seed@VM:~/.../code$ ./stack-L1
Input size: 517
```

To exploit the buffer-overflow vulnerability in the target program, we need to prepare a payload and save it inside badfile. We will use a Python program to do that.

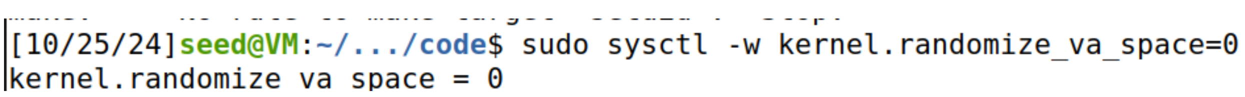
Tasks 7: Defeating dash's Countermeasure

/bin/sh points back to /bin/dash. And to defeat the countermeasure in buffer-overflow attacks, all we need to do is to change the real UID, so it equals the effective UID.

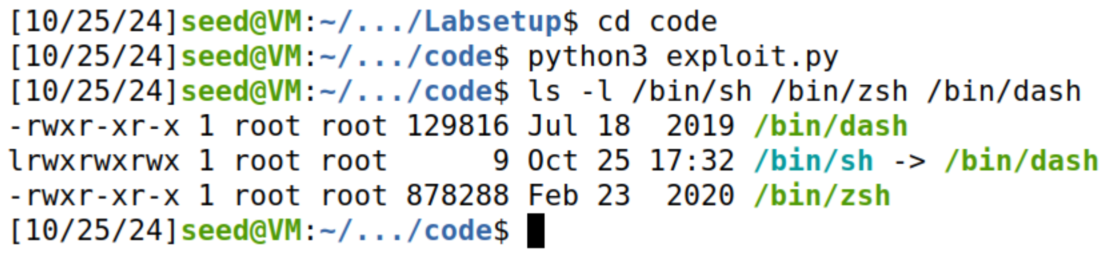


```
[10/25/24]seed@VM:~/.../code$ sudo ln -sf /bin/dash /bin/sh
[10/25/24]seed@VM:~/.../code$ make setuid
```

Launching the attack again. Now, using the updated shellcode, we can attempt the attack again on the vulnerable program, and this time, with the shell's countermeasure turned on.

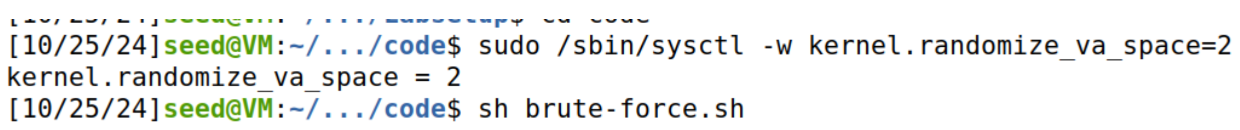


```
[10/25/24]seed@VM:~/.../code$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```



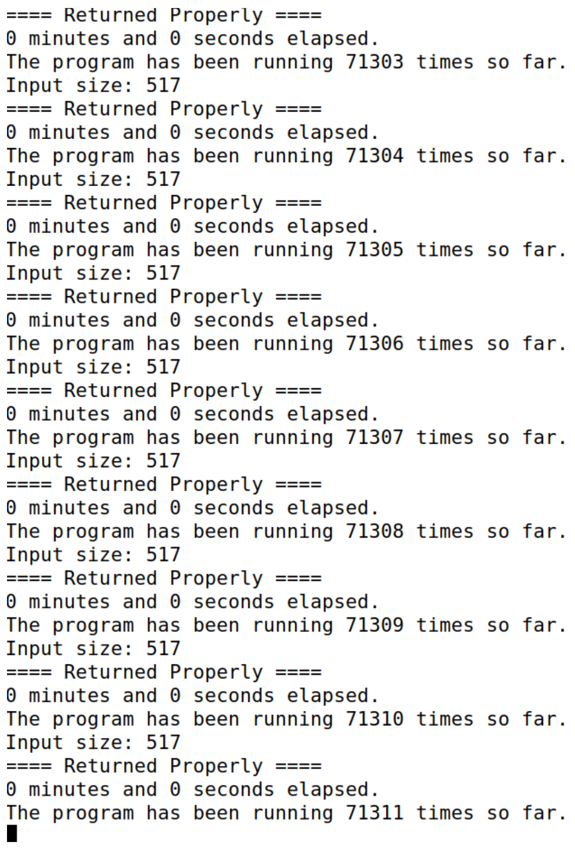
```
[10/25/24]seed@VM:~/.../Labsetup$ cd code
[10/25/24]seed@VM:~/.../code$ python3 exploit.py
[10/25/24]seed@VM:~/.../code$ ls -l /bin/sh /bin/zsh /bin/dash
-rwxr-xr-x 1 root root 129816 Jul 18  2019 /bin/dash
lrwxrwxrwx 1 root root      9 Oct 25 17:32 /bin/sh -> /bin/dash
-rwxr-xr-x 1 root root 878288 Feb 23  2020 /bin/zsh
[10/25/24]seed@VM:~/.../code$ 
```

Task 8: Defeating Address Randomization



```
[10/25/24]seed@VM:~/.../code$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[10/25/24]seed@VM:~/.../code$ sh brute-force.sh
```

After sh brute-force.sh command this programming running continuously.

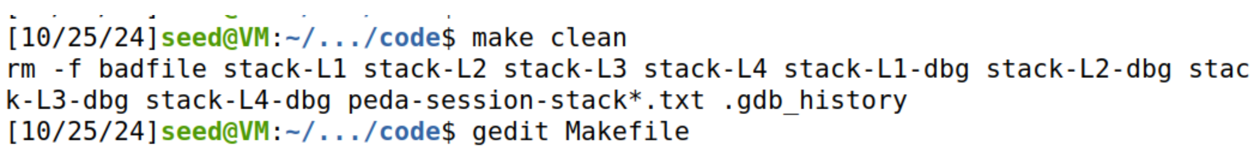


```
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71303 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71304 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71305 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71306 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71307 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71308 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71309 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71310 times so far.
Input size: 517
==== Returned Properly ====
0 minutes and 0 seconds elapsed.
The program has been running 71311 times so far.
```

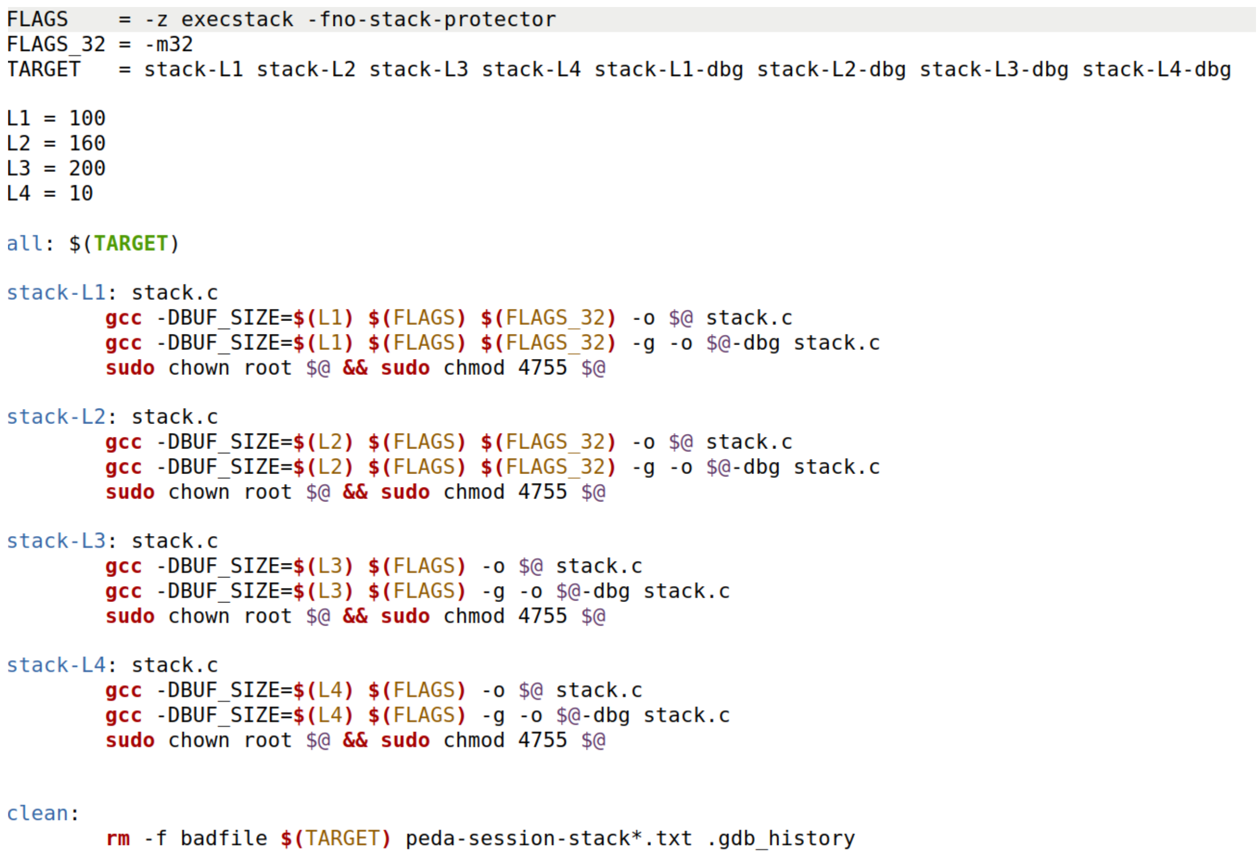
Tasks 9: Experimenting with Other Countermeasures

Task 9.a: Turn on the StackGuard Protection

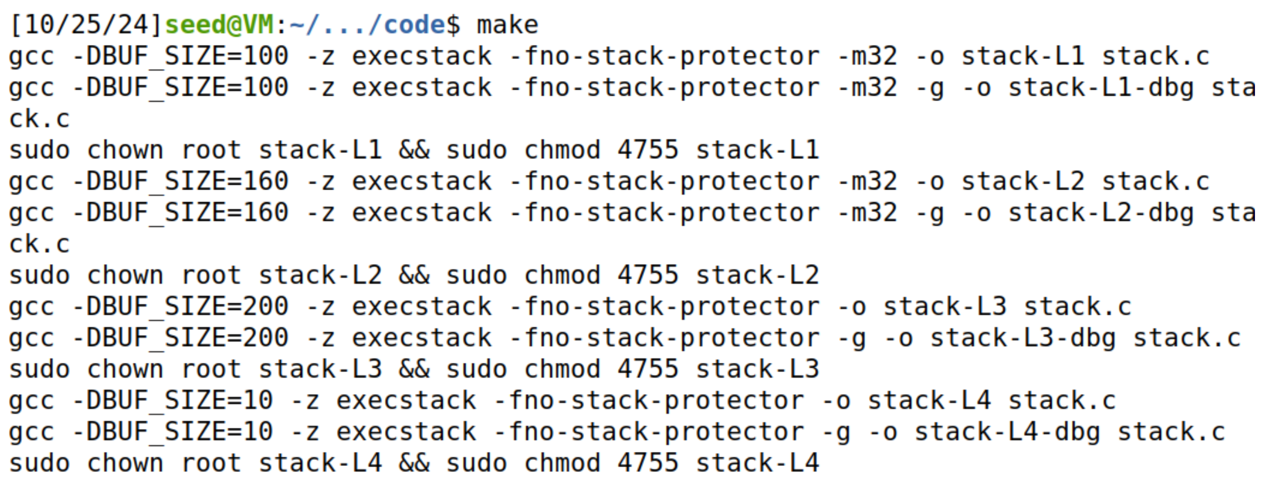
First, repeat the Level 1 attack with the StackGuard off, and make sure that the attack is still successful.



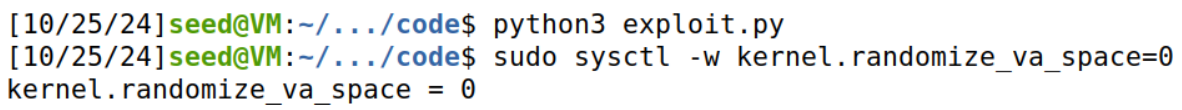
```
[10/25/24]seed@VM:~/.../code$ make clean
rm -f badfile stack-L1 stack-L2 stack-L3 stack-L4 stack-L1-dbg stack-L2-dbg stac
k-L3-dbg stack-L4-dbg peda-session-stack*.txt .gdb_history
[10/25/24]seed@VM:~/.../code$ gedit Makefile
```



```
FLAGS    = -z execstack -fno-stack-protector
FLAGS_32 = -m32
TARGET   = stack-L1 stack-L2 stack-L3 stack-L4 stack-L1-dbg stack-L2-dbg stack-L3-dbg stack-L4-dbg

L1 = 100
L2 = 160
L3 = 200
L4 = 10

all: $(TARGET)

stack-L1: stack.c
	gcc -DBUF_SIZE=$(L1) $(FLAGS) $(FLAGS_32) -o $@ stack.c
	gcc -DBUF_SIZE=$(L1) $(FLAGS) $(FLAGS_32) -g -o $@-dbg stack.c
	sudo chown root $@ && sudo chmod 4755 $@

stack-L2: stack.c
	gcc -DBUF_SIZE=$(L2) $(FLAGS) $(FLAGS_32) -o $@ stack.c
	gcc -DBUF_SIZE=$(L2) $(FLAGS) $(FLAGS_32) -g -o $@-dbg stack.c
	sudo chown root $@ && sudo chmod 4755 $@

stack-L3: stack.c
	gcc -DBUF_SIZE=$(L3) $(FLAGS) -o $@ stack.c
	gcc -DBUF_SIZE=$(L3) $(FLAGS) -g -o $@-dbg stack.c
	sudo chown root $@ && sudo chmod 4755 $@

stack-L4: stack.c
	gcc -DBUF_SIZE=$(L4) $(FLAGS) -o $@ stack.c
	gcc -DBUF_SIZE=$(L4) $(FLAGS) -g -o $@-dbg stack.c
	sudo chown root $@ && sudo chmod 4755 $@


clean:
	rm -f badfile $(TARGET) peda-session-stack*.txt .gdb_history
```



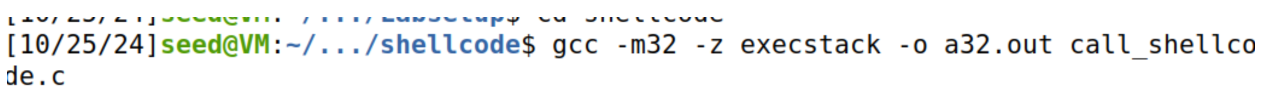
```
[10/25/24]seed@VM:~/.../code$ make
gcc -DBUF_SIZE=100 -z execstack -fno-stack-protector -m32 -o stack-L1 stack.c
gcc -DBUF_SIZE=100 -z execstack -fno-stack-protector -m32 -g -o stack-L1-dbg sta
ck.c
sudo chown root stack-L1 && sudo chmod 4755 stack-L1
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -o stack-L2 stack.c
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -g -o stack-L2-dbg sta
ck.c
sudo chown root stack-L2 && sudo chmod 4755 stack-L2
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -o stack-L3 stack.c
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -g -o stack-L3-dbg stack.c
sudo chown root stack-L3 && sudo chmod 4755 stack-L3
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -o stack-L4 stack.c
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -g -o stack-L4-dbg stack.c
sudo chown root stack-L4 && sudo chmod 4755 stack-L4
```



```
[10/25/24]seed@VM:~/.../code$ python3 exploit.py
[10/25/24]seed@VM:~/.../code$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Task 9.b: Turn on the Non-executable Stack Protection

make the stack non-executable. We will do this experiment in the shellcode folder. The call shellcode program puts a copy of shellcode on the stack, and then executes the code from the stack. Please recompile call shellcode.c into a32.out and a64.out, without the "-z execstack" option

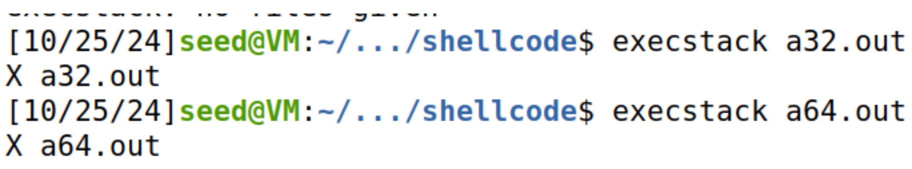


```
[10/25/24]seed@VM:~/.../Labsetup$ cd shellcode
[10/25/24]seed@VM:~/.../shellcode$ gcc -m32 -z execstack -o a32.out call_shellco
de.c
```

- **gcc**: This is the GNU Compiler Collection, used to compile the C code.
- **-m32**: This flag tells the compiler to generate a 32-bit binary, regardless of the architecture of the host system. It is commonly used when developing exploits, as many legacy vulnerabilities and shellcode are written for 32-bit systems.
- **-z execstack**: This flag marks the stack as executable. By default, modern systems have non-executable stack protection enabled (a security measure called **DEP**, Data Execution Prevention). The -z execstack option disables this protection, allowing code on the stack to be executed, which is necessary for executing shellcode that is stored on the stack.
- **-o a32.out**: This specifies the name of the output file. In this case, the compiled executable will be named a32.out.

- **call_shellcode.c**: This is the C source file that is being compiled. Typically, in exploit development scenarios, this file would contain shellcode or a function that attempts to execute shellcode.

These are the output when execute the binary files,



```
execstack: no files given
[10/25/24]seed@VM:~/.../shellcode$ execstack a32.out
X a32.out
[10/25/24]seed@VM:~/.../shellcode$ execstack a64.out
X a64.out
```

X indicates that the stack is non-executable.