jamk.fi

Software exploitation

Assignment 3

Mikael Romanov K1521

Maaliskuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma
Kyberturvallisuus

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

1 Table of Contents

1 Introduction

This assignments goal was to create customized executable shellcodes in assembly. The first program took shellcode as input and executed it as a function. Second program needed buffer overflow to execute buffer overflow

2 Testing platform

I used my Thinkpad w520 with VMware Workstation 14 (Figure 1)

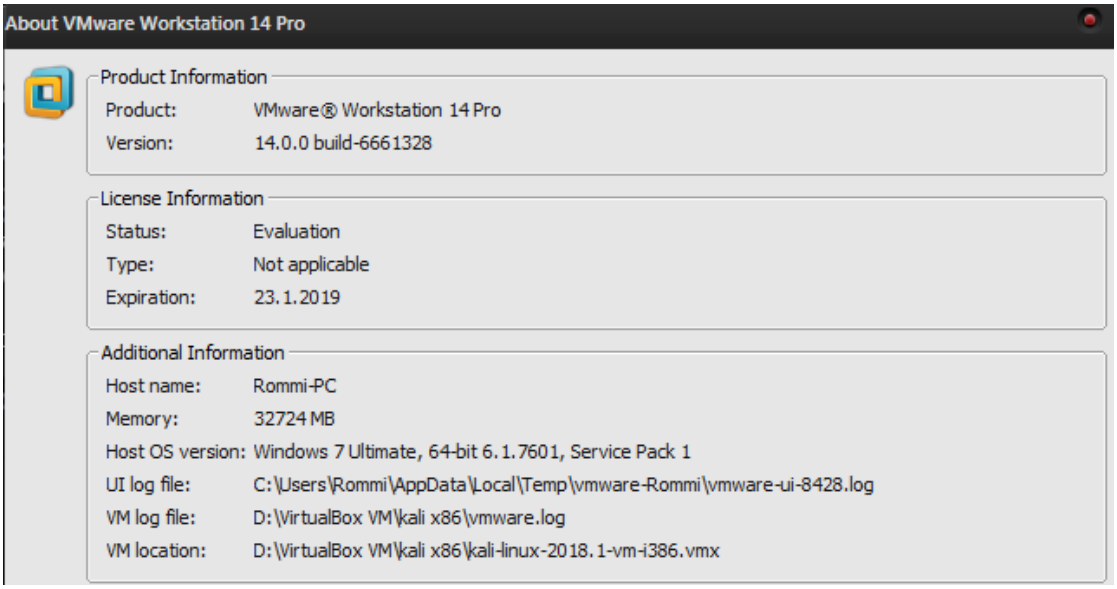


Figure 1 vmware

I reserved good amount of resources for the KALI vm and they were following(Figure 2)

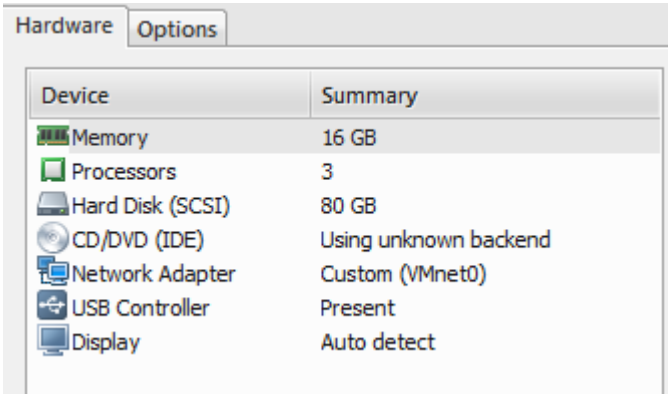


Figure 2 resources

I used Kali Linux 4.14.0-kali3-i686 (32 bit) since the amd64 had some problems with the **shellcode -f.** The gcc version was 7.2.0, the same as on the earlier assignments. (Figure 3)

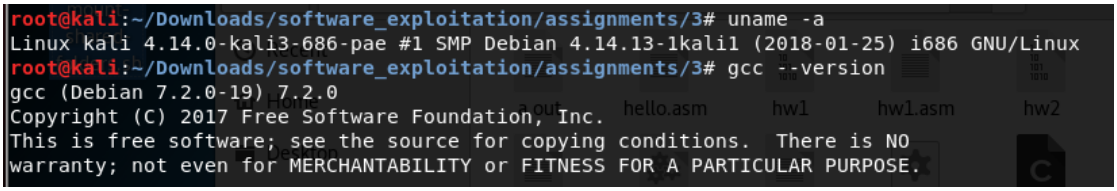


Figure 3 versions

I used the make file to compile the programs. The make file was simple to us to compile the file, it only required the command ***make.*** The MAKEFILE uses gcc to compile the program.

After compiling the programs I turned ASLR off, because it is hard to know where the memory is allocated next. ASLR can be set off by echoing ***echo 0 > /proc/sys/kernel/randomize_va_space*** (Figure4)

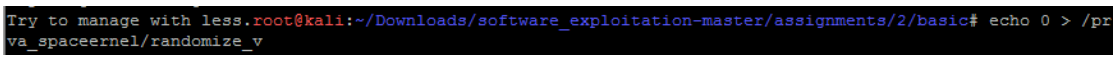


Figure 4 aslr off

Then I used readelf to get information of the executables headers, which indicated that the program is 32bit (Figure 5)

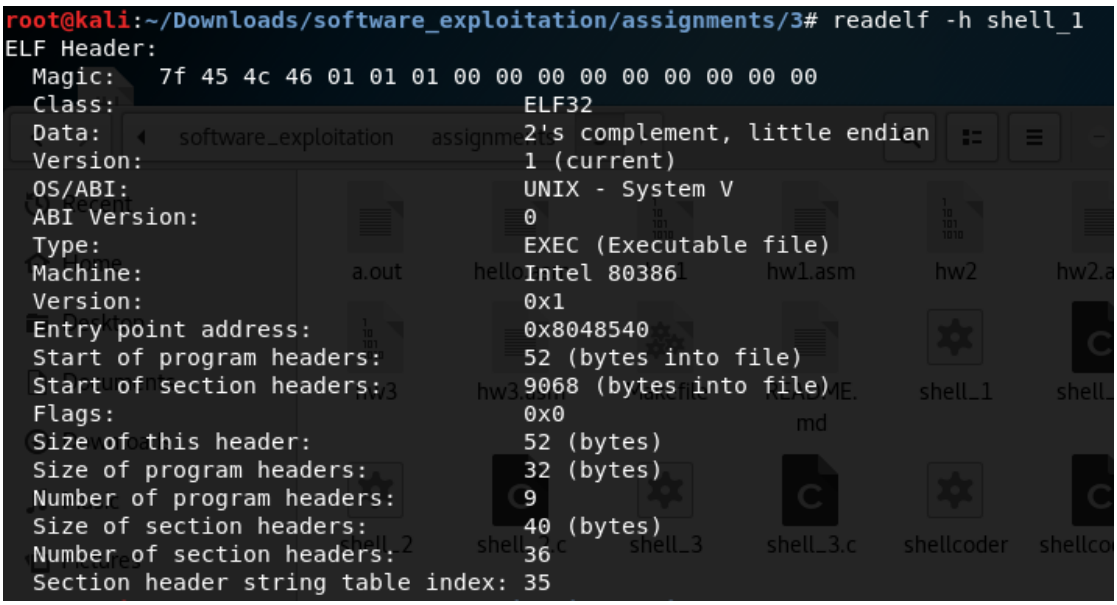


Figure 5 readelf

3 shell_1

I started this assignment by modifying the hello.asm file, so that it would work with the programs. (Figure 6)

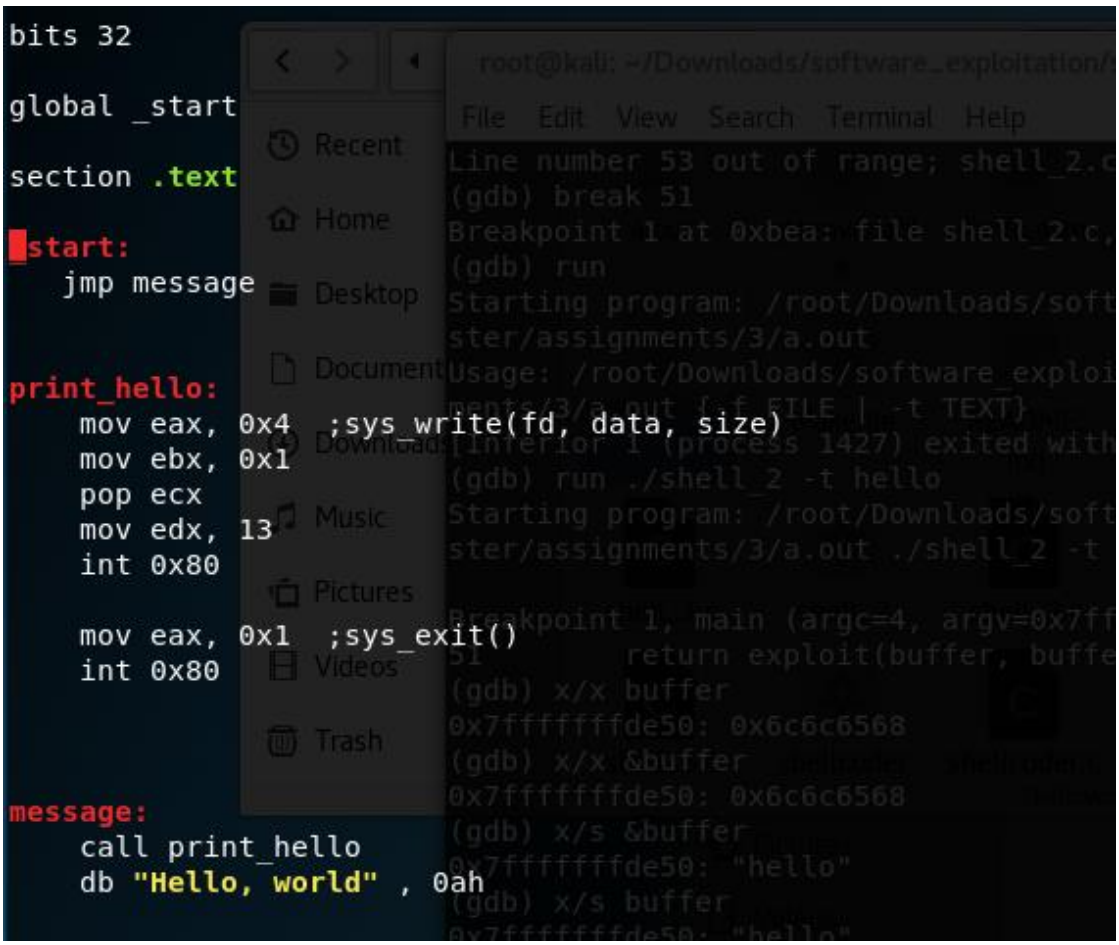


Figure 6 new hello.asm

I assembled it and then disassembled to see if there where null bytes in the register. And as seen below, the hello.asm had null bytes in few locations.(Figure 7) The null bytes need to be removed, because it means end of string in C.

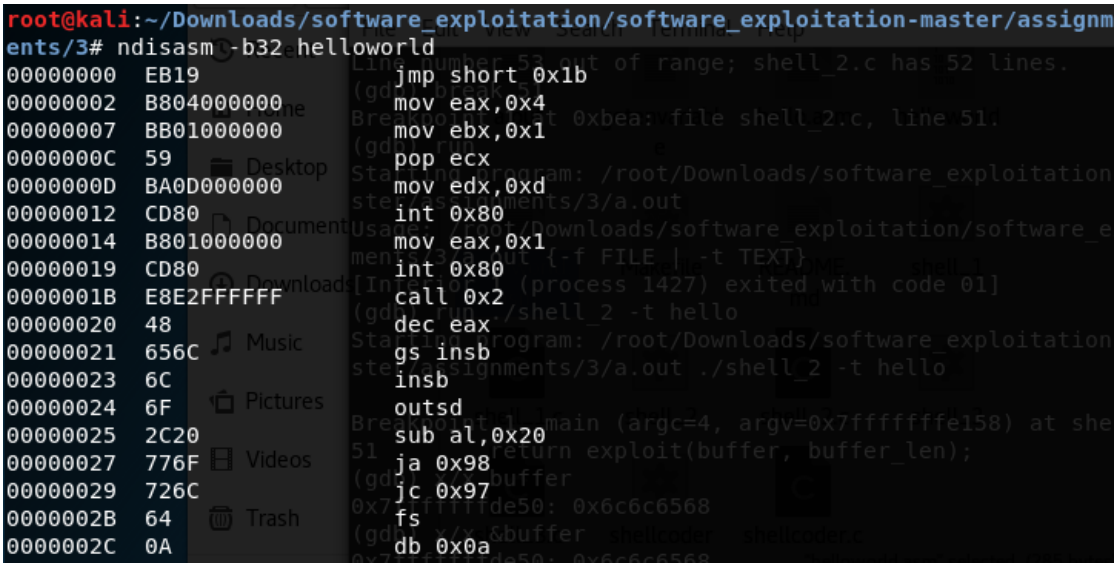


Figure 7 ndisasm

I then edited the hello.asm (renamed it to hw1) and removed all the null bytes by making the register 8bits long, because the values won't need extra space. First the register is cleared by Xor. Then changed the syswrite, stdout, message length and sys exit to 8 bits. And decreased ebx register to 0 for clean exit. (Figure 8)

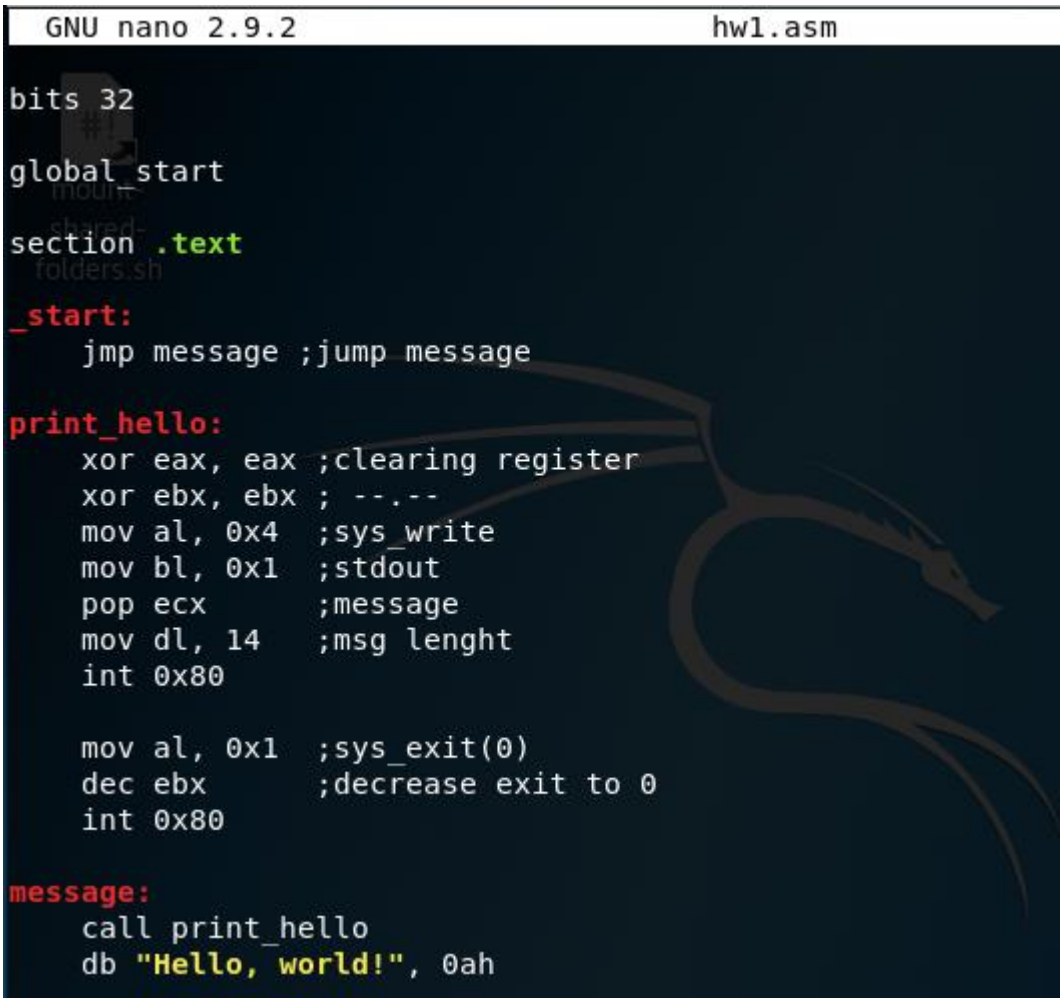


Figure 8 edited hello.asm

I saved the file and made 3 different versions of it, only thing that changed was the sentence that was in use. Then I assembled them (Figure 9)

Figure 9 assembling

To be sure about the removing of the null bytes, I disassembled the hello.asm again and this time the register didn't show any null bytes. This means that the shellcode should run. (Figure 10)

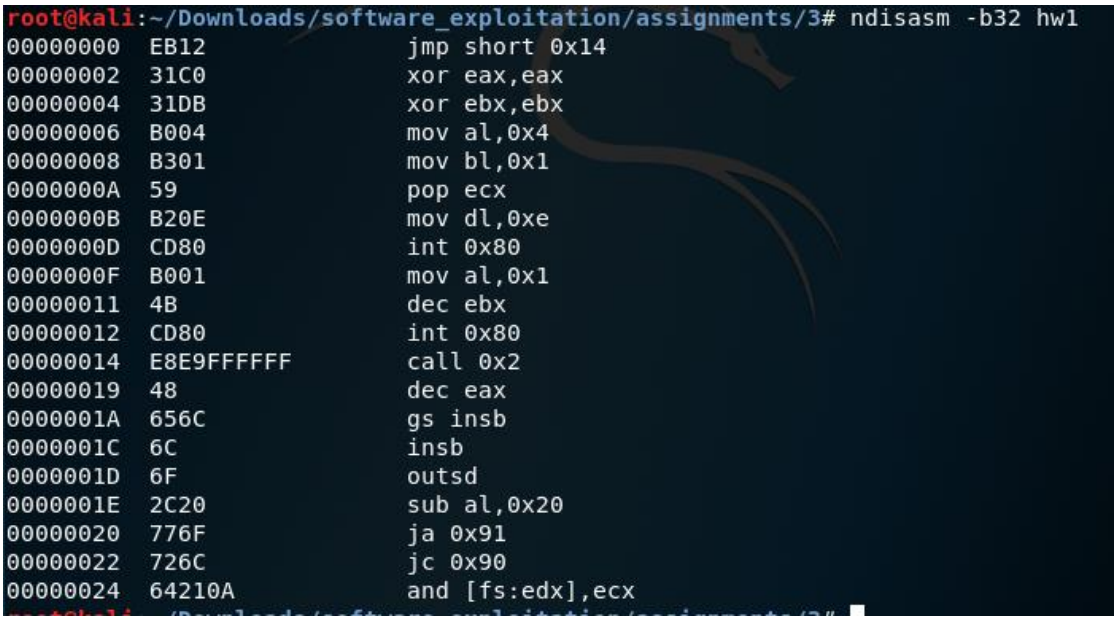


Figure 10

The I just started executing the shellcodes with cat. In the first one I used cat to pipe the command, because it wouldn't work any other way. (Figure 11)

Figure 11

The second one needed just the fie name, I inputted the filename and as expected it runned like it supposed to. (Figure 12)

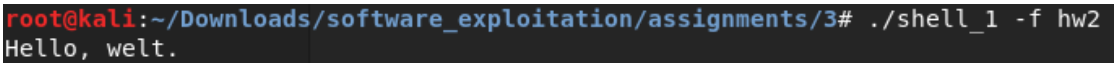


Figure 12 hello welt

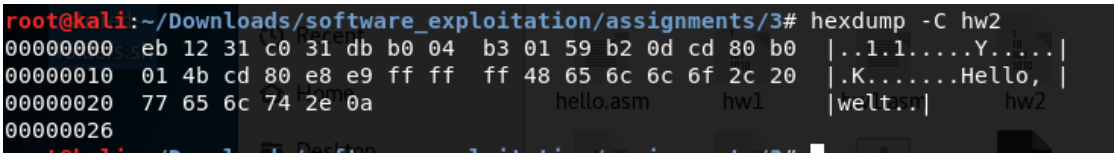


Figure 13 hexdump welt

The last one needed some padding to be executed correctly, since -t pushes to the stack also. (Figure 14)

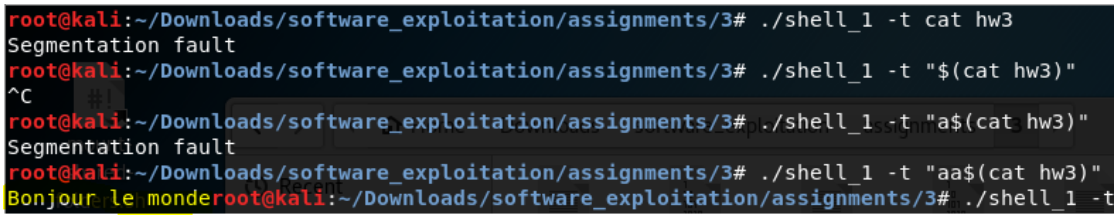


Figure 14 Bonjour le monde

4 shell_2

I started debugging the shell 2 and checked how many characters need to be printed until owerflow. (Figure 15)

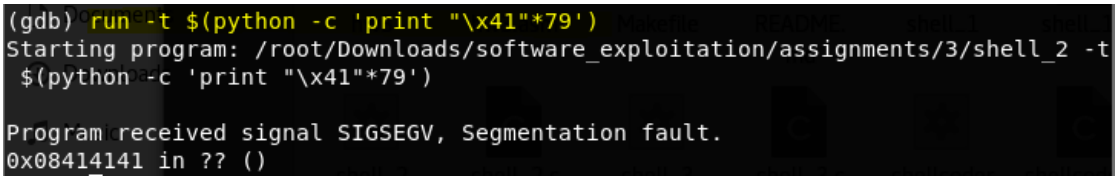


Figure 15 overflow try

The correct answer was 80 characters to fully overflow. (Figure 16)

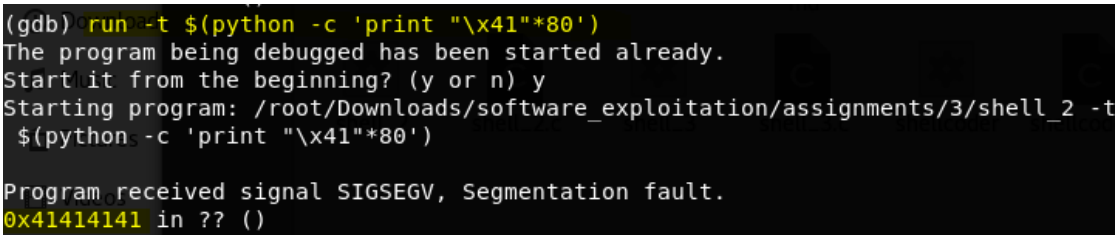
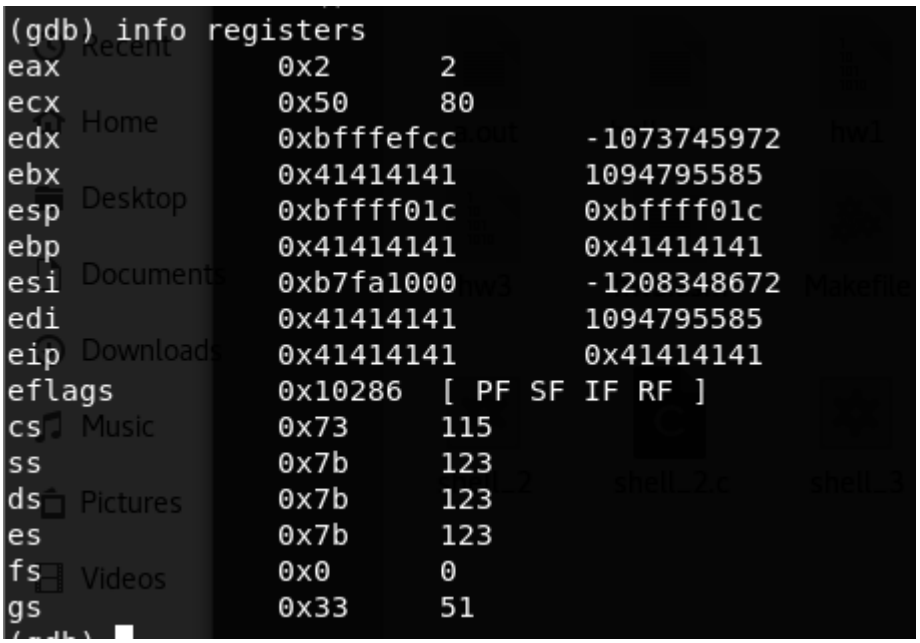


Figure 16 owerflow success



5 Conclusion

This assignment was also hard, way more than the earlier assignments. I tried the second shell but didn't manage to finish it sadly the shellcode didn't run for some reason, so my documentation ended to the overflow. First shell was pretty simple after reading the books chapter.