Maximiliano Brizzio
CS 576
Lab 7 - Shellcode

To begin with this assignment, I must first generate the null-byte shellcode that will be taking over the return address. Here is the assembly code:

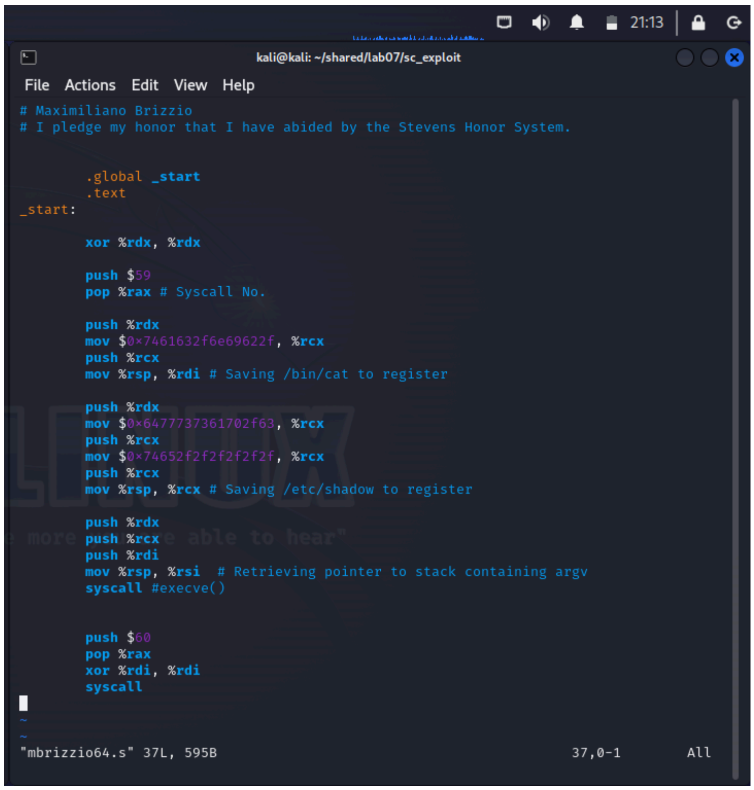

Fig. 1 - Assembly Code of mbrizzio64.s

As you can see, I am pushing all of the arguments of argv in order to the stack, followed by a null-terminator. Because of little endianness and the nature of pushing to the stack, I had to push each argument in reverse. I am also using hexadecimal numbers in order to avoid generating null-bytes in the shell code. Utilizing the stack as much as I do also helps in avoiding null-bytes that sometimes come as a result of using 'mov'.

Once I had compiled the code into mbrizzio64.sc, I was able to use hexdump in order to view the shellcode and verify no null-bytes were present:

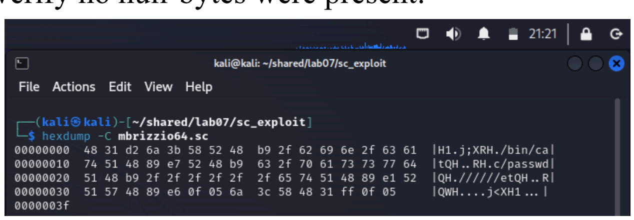

Fig. 2 - Hexdump of mbrizzio64.sc

Now that the shellcode is completely null-byte free, we are able to begin injecting the code into the vulnerable ‘exercise.c’ program. Before this can happen however, we need to find the address of buf1 in the copy function and the return address of the copy function. To do this, I use gdb on the compiled C code and set a breakpoint at the copy function. Once inside the copy function, I run `p &buf1` and `info frame`. The first command gives the address of buf1 to be “0x7fffffffdbf0” and the second command gives the saved return address which is “0x7fffffffdc88”. I used a hexadecimal calculator to subtract the buffer address from the return address which gives an offset of 152 bytes. I am now able to fill in the address in exploit.py in reverse order (due to little endianness):

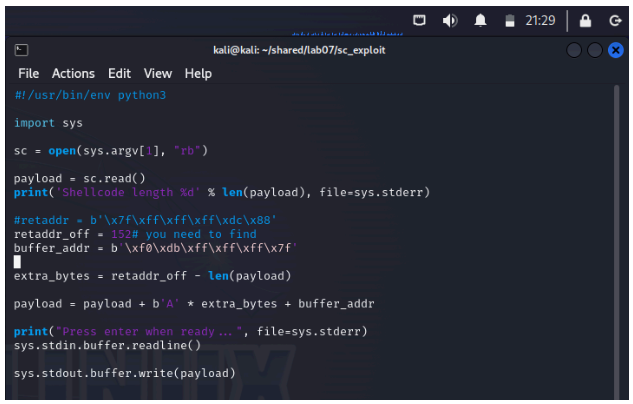

Fig. 3 - exploit.py

Now I am able to run the python script which generates the shellcode and stores it into the ‘payload’ file. I then start gdb on the compiled C program (exercise-64) and use the command r < sc_exploit/payload to input the shellcode and take over the control flow of the program once the copy function returns. The following is the result:

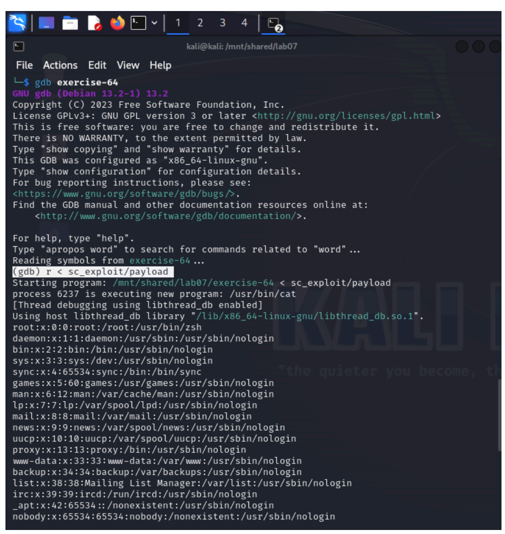

Fig. 4 - Execution of Injected Code