Buffer Overflow Assignment Report

The first step toward properly executing a buffer overflow attack is setting up the correct environment. In this assignment and demo, we are using running a Linux Redhat9 virtual machine located here <https://drive.google.com/file/d/1-4L2V2m3grNeQNF7pvcin8qGZhh4cbrg/view> in VirtualBox. After following the VirtualBox instructions to set up the VM, log in to the system using root as the username with no password and open up a web browser and download the getscore program files from here <http://analysis4ms.com/static/p1.zip> and put into the root directory.

Second, we need to compile the getscore program and get a segmentation fault to dump the core. To do this, open a command line and use the command *cd /p1/getscore* to change the present working directory to the getscore file that was just downloaded, then compile the program with the command *gcc getscore.c -o getscore* and set the proper permissions for both root and guest users with *chmod 755 getscore; chmod u+s getscore; chmod 600 score.txt* andcheck the permissions with *ls -l*. To test this is done correctly, run the program with *./getscore “Mary Doe” 123-45-6789*. We can now get the segmentation fault and dump the core. Before this, we need to run the command *ulimit -c unlimited* to enable core dumps, then run the get score program like this: *./getscore aaa AAAA……AAAA* where there are as many “A”s as necessary to get the segmentation fault to dump the core. We can check that the core is dumped by running another *ls* command, and we should see a file titled *core.xxxx* where the “x”s are any string of numbers following the “.” Operator.

Third we need to examine the core dump to determine where the addresses for the ESP, EIP, and string of “A”s so we can calculate the offset. We use the Linux GDB debugging tool to do this by running *gdb -c core.xxxx*, then check the value of the register and memory of the core dump for the $esp register with *info registers $esp* and *x/100x $esp-200* commands. To calculate the offset, we subtract the hexadecimal address of the start of the overflow, which is the row where the hexadecimal value 0x41414141 starts in the memory, from the hexadecimal address of the $esp pointer in the register.

The fourth and final step is creating the malicious input and writing the python script to hijack the jmp esp instruction to run our malicious shell code and gain root access. To do this, first find the hexadecimal address of the jmp esp instruction by disassembling the getscore program’s library. The getscore library is found by running *ldd getscore* command, then use the filepath returned in in place of the “x”s in the following command *objdump -D xxxx.lib.so > disasm.txt* to disassemble the library and write the results into the “disasm.txt” file. We then search the “disasm.txt” file to find the jmp esp instruction using *grep “\*%esp” disasm.txt* and the first row retuned contains the hexadecimal location of the jmp esp instruction in the first column. We can then create a guest user using *useradd guest* and change the user to guest using *su guest* andwrite the python script to hijack the jmp esp instruction and gain root access. Create the python script using *vim exploit.py* and write the following program:

A screenshot of a computer code

Description automatically generated

Where the second argument in the addr = strcut.pack() function is the hexadecimal value of the jmp esp instruction and 136 is the offset you calculated. We can then run the program using *python exploit.py* and we should have root access.