Description:

The goal of Lab 5 was to compromise a server program using a standard buffer overflow and brute force guess the secret password that is revealed. In this lab we learned how to examine one of the most common security vulnerabilities, the stack overflow. Our first objective of the lab involved obtaining the environment variables from the server. After that we attacked the parameters for the selfcomp.c file, we started with modifying the compromise1 array which is used to probe the stack to find the values needed to parameterize the attack to insert the shell code. To do this we adjusted the contents of the compromise1 array by adding more 'x' characters until running the program would cause a segmentation fault and the core would be dumped. We used the command "gdp selfcomp core" to analyze the core file look at the values of the registers in specific the value of the marker string to make sure it matched with the marker string in the compromise1 variable "MNOPWXYZ".

The next step of the lab was to write the shell code in the exploit.nasm file. The first part we had to modify was adding code to calculate the runtime address of each label and store it in the arrayAdr array. To do this we first used the mov instruction to move the values of the label into rsi, once they were moved into rsi the lea command was used to calculate the effective runtime address and store it in the specific byte allocated for that variable.

The third step of the lab was to get the final values from selfcomp, to do this we changed the variable in the doTest loop to use compromise instead of compromise1. The program is then run and selfcomp and then the environment variables are printed in the terminal.

The final step of the lab was to repeat the above steps for the client and server program. The client is used to send a string and prints the response from the server to the terminal. We used the same method above to replace 'x' characters until we crash the server which was 216 characters. We then used the debugger to determine the location of the environ variable as well as the length of the compromise string which matched the previous value of the environment variable. The next step was to copy our exploit code into the compromise array. Since more 'x' characters were used, more padding was added to the start of the array, also since it was being sent across a network a new line and null byte had to be added since it is two longer than the length of the attack array. Once that was done the client file was run to print the environment variables from the server and one of the variables MD5 contains the hash of a password that relates to a social media post given in the lab.

Testing:

To test the lab, we were expected to print the environment variables into the shell by running the program. For selfcomp.c, this required us to run the program after the ulimit was set to unlimited. When this was done, the environment variables should have been printed. For client.c, we first had to initialise the server using "./quoteserv 10000", and then we would have to run client.c on the same port with "./client 10000". This should then print the environment variables. Although we were unable to produce the correct environment variables as our output, the code was mostly correct, we were just unable to produce the result.