ELEC 377 Lab 5: Description of the Program

Name & Student ID: Shiyan Boxer (20106887) and Arsh Kochhar (20104779)
Date: Dec 7th, 2020

Overview of the Program

This program obtains a copy of the /ect/password file remotely by compromising a server program using a global buffer overflow.

Files

- **makefile** - The makefile to compile the programs.
- **selfcomp.c** - Can execute the exploit internally. Used to develop the exploit.
- **server** - The vulnerable server which is a binary executable file.
- **client.c** - Used to compromise the server program.

Our Solution

First, the program sets the function pointer to point to a default function “foo”, then, using a for loop, it copies the bytes from the variable “exploit” into the buffer. Finally, it calls a function using the function pointer.

The “exploit” variable is a string used to probe the program to find parameters about vulnerability and compromise the server. Each line contains 20 “X” characters which means a single string constant of 45 characters containing 40 “X” characters followed by “WXYZ” and a null character.

If the length of the string in the “exploit” is less than the buffer size, it will call the function foo. If it’s longer, it will overflow the buffer and modify the pointer so that it does not point to the function foo. Using the debugger message, we changed the selfcompe.c character by modifying the number of “X” characters in exploit until the core dump had WXYZ as hex values. From this, we derived the size of the buffer exploit is based on the number of “X” in selfcomp.

Using the info target and x/16 address commands we examined the memory command searching for “X” characters. From this, we found the address of the start of the buffer which was found in the .bss segment of the local executable file. Then, we wrote the compromise shell code in nasm based. The list file shows the listing of the shell code.

In the selfcomp.c, we changed the length of exploit2[105] to the length of the string which was the “X” characters + WXYZ) + null character. Then we copied the bytes from the list file to the array by deleting the first 2 columns and splitting the listing file hex numbers and added NOP instructions so the return address would be aligned with the existing return address.
We calculated the beginning of the code based on the length of the code and the value of the debugger then inserted that into the array. We then tested to see if it coped the contents of /ext/password. Then, we repeated the above section for the client and server program. Lastly, the exploit code was copied into client.c and changed the fprintf in the PutLine function to use our array.