ELEC 377 Lab 5 – Software Security

Problem Description

The objective of this lab is to compromise a server program using a standard buffer overflow and brute force guess the secret password that is revealed.

The server that was attacked has an encrypted password stored in the environment variables. The first part of the lab involved obtaining the environment variables from the server. They are a part of every Linux process, are usually passed to children processes, and are a mechanism that provides program customization without the need for a configuration file.

The program *selfcomp.c* attacks itself and starts by first adding a variable to the environment. It then calls the vulnerable function which contains a character array of 136 bytes long. There are also two global variables. The first, *compromise* contains the shell code and the second, *compromise1* probes the stack to find the values that are needed to parameterize the attack to insert the shell code. Then, the function *doTest* attacks itself by copying *compromise1* byte by byte into the array *buffer* and stops at a null byte. If the string is longer than the array, it extends itself into the rest of the stack frame.

The marker value "MNOPWXZ" is used to overwrite the saved base pointer for 64 bits and the final 'x' characters are used to overwrite the return value. The function adds and removes 'x' characters from *compromise1* until the program crashes and dumps the contents of the memory to the file *core*.

The next step is to write the shell code. The length of the string used (*compromise)* had 208 x's and the location of the return address on the stack was 0x7FFFFFFFDE60 for the server attack. Since the x64 is a little-endian machine, the non-null bytes occur first in memory and thus only the first 6 bytes of the return address must be overwritten.

The global variables *compromise* and *compromise1* are stored in the data segment, and the information stored at the top of the memory changes based on the information from the data segment. This results in it changing day-to-day and from login-to-login. Also, there may be a different length of netid and there may be a different return address due to this. The program is then compiled again to create a core dump and examines the stack pointer *rsp* using the command *info registers*. The stack pointer points to the first 'x' character after the 'MNOPWXYZ' marker characters that went into the base register. In the loop *doTest*, the *compromise1* is swapped for *compromise*, and then *selfcomp.c* prints out the environment variables.

The last step was repeat this exercise with the client and server program. The server program *./quoteserv* is started in one terminal window and takes a port number between 1024 and 65536 as an argument, a value of 10001 was used in testing. The server takes a name as input and returns a personalized quote based on the name. The server forks on each request and the child sends the response, which means it will continue running even when its being attacked until it is killed.