Kelsey Cameron

Justin Bates

Homework 5

**Objective:**

Take the four given files and reverse engineer the binary to find three hidden flags. Using the MSP430 and inputs through UART the flags must be triggered on a stock binary through certain conditions. This all must be done without modifying any memory or register in the microcontroller or modify the binary files in any way.

**Summary:**

The program Uniflash was used along with an online disassembler and various terminal programs. After trying multiple terminal programs it became clear that using a terminal that can enter non printable characters helps a lot when looking for these strings. After finding a suitable terminal application I began by configuring the terminal application to a baud of 9600 with 8 bits no parity and 1 stop bit (8N1.)

In order to capture flag A, we disassembled the binary .Elf file to read what was inside the file. We also viewed the .Hex files and the other two header files to understand how the program operates and make sure we have all the data. Due to several platform compatibility limitations, I used an online web disassembler to open the file. Once I opened the file, we noticed several strings commented out. The comments seemed to correlate to several addresses, so we read through the assembly code and figured out that one comment in particular maps to flag A. We tried using this input and then successfully captured flag A. The input to trigger this flag was 313373, in ascii characters.

Upon reading the assembly and analyzing the memory map, we realized that there are different functions that call printing the Flag A and Flag B captures. By utilizing these functions, we were able to capture Flag B. Within this program, there was a character buffer of 16 characters. By using the correct flag A code in conjunction with overflowing the character buffer, we discovered from the jumps within the assembly code that you can actually input an address and once the program buffer is full, it will simply jump to the address of the next UART values.

This absolutely taught us the importance of hardware security, since anyone could theoretically inject code into a program that has jump statements and then a hacker could gain total control over the program. Then, we used the disassembled binary to find the address of where flag B was located. This address was at 0xB2C4.

However, the only method of input is through the UART, and since the data being transferred can be expressed in eight bits, we had to break apart this address into two parts within the input. After the B2 and C4 were split up, we converted them to ASCII values. These values happen to correspond with the squared symbol and a capital Ä.

So as long as the jump to the flags is made by typing in the address of where flag A was found (313373), then we used 10 characters to fill the array and “crash” the program in a special way, and then whatever address is entered next is what the program jumps to, thanks to the printing function. In short, flag b was triggered using the following ASCII;

31337300000000002Ä

Where the underlined 2 stands for the squared symbol.

Unfortunately​, we were unable to find flag C because there was no printing function available to print flag C, and in order to find it we would have had to create a shell based program to print it. However, the general idea of how flag b should apply in conjunction with creating that printed program.

**Conclusion:**

In conclusion, there were several drawbacks to this assignment including microcontroller compatibility issues (which Orlando has now fixed), the fact that this program can only be run on windows using the G2553 chip, and the fact that the program was written in assembly with only the addresses of the code to view. In short, we learned a lot from this assignment, although it was certainly very difficult.