**ITT-210 SolarWinds Instructions**

In 2020, FireEye responsibly disclosed a hack of the SolarWinds vendor via their update system. Instead of attacking a SolarWinds product, attackers exploited the software process of the vendor itself, and instigated SolarWinds to unwittingly distribute malware to all their customers, including among them the United States Government. By exploiting SolarWinds itself, attacks were able to leverage the market reach of SolarWinds's own commercial success. At the time of disclosure, more than 18,000 customers were actively infected with the malware.

## **Part 1**

## As you work through the assignment, you will provide screenshots of the following:

1. Show each of the compiled and executed crc.exe and crc\_bad.exe.

1. The text difference between crc8.asm and crc-maleware.asm.
2. Identify in the screenshot the exploited string present in the infected assembly file.
3. Identify in the screenshot where the exploited string is used/referenced in the assembly. (Hint: In C, we define and use strings/data on the same line. In Assembly, the code and data are separate. Point a above is the string’s definition and point b, find the string's use.)
4. A different Bible verse used in main.c for both exploited and unexploited executables. (Hint: You should get different CRC values.)

### Steps

We have the original source code for our CRC module, and we have the assembly listing of our module. We are going to compose the C and ASM modules together and show how we can assemble both an exploited module and an unexploited using the same C source code!

There are two tools we will use:

1. ml the assembler
2. cl the C compiler

First, we need to convert the assembly module into object code so our C compiler can *link* the components together. As we learned in ITT-310, the C compiler is typically two pieces, a compiler that converts text to "object code," a linker that connects the object code to libraries, and any external dependencies. In order to bring together an assembly module and a C module, we will need to convert both to object code before the C compiler can work with it.

We want to convert the crc-malware.asm file into a crc-malware.obj.

C:\> ml crc8-malware.asm /c

Now we have the source code for the C file, so we will compile it to an intermediate object-file form.

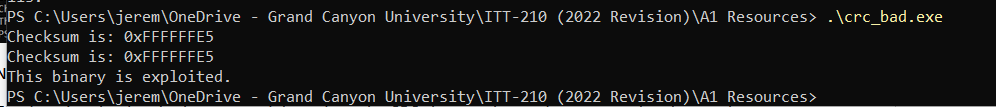
cl main.c /c

Now, we want to link the main.obj and crc8-malware.obj to make crc\_bad.exe.

cl crc8-malware.obj main.obj /Fecrc\_bad.exe

Run crc\_bad.exe.

Notice how the CRC module is getting the same comparison value when trying to check the two Bible verses. This is not how a CRC should behave. The CRC should be different, but our exploited module is hiding the exploited data.

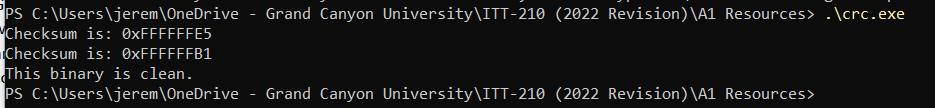
****

Now, let's assemble the good module.

ml crc8.asm /c

cl crc8.obj main.obj /Fecrc.exe

Run crc.exe

****

Notice how the CRC values are different.

Change the data in the second module, then see how the CRC changes. Leave the command and control server domain as a prefix of the data: exploitc2.io.

Notice how the exploited CRC module always gets the same value when exploitc2.io is at the beginning of the string.

Now, diff the 2 ASM files: crc8-malware.asm and crc8.asm.

## **Part 2**

Write a 100- to 150-word summary describing a security control you could integrate earlier into the SDLC process that may have reduced the scope of this hack. Identify how your control addresses the key security issues at each stage. Identify in your screenshot where exactly the exploit string is used in the instructions.