Jack Krejci

Metasploit Project

Buffer Overflow Documentation

In project 6, we performed a buffer overflow attack, which is an attack where more input is placed into some buffer than the capacity should allow which will then overwrite other important information. Our attack will make use of figuring out the size and location of stack data, we will then overwrite contents of the stack using the return address to run our executable shellcode written for both Metasploit, and without it for part one. So, the CPU will hit the return address and then jump to the shellcode that was injected into it, and have it run. The first step we have to do is to send an overflowing input string to our vulnerable server named “vulnc2” while running on a break in box. We will have to login with some user in the break in box so we can make sure to install our vulnerable server onto the machine, and then run it with the root account on the break in box.

So, we will have to make use of telnetting to that user so we can first put that vulnerable server file in the users account, so that it can let the root account of the break in box use it. We also have to make sure when logged in as the root, that we do the command “echo 0 > /proc/sys/kernel/randomize\_va\_space”. So that the ESP won’t be different every time the server is ran to make it easier on us, we then send that overflowing input consisting of a bunch of letters to help us determine the offset. For my project, I found that the offset was 223, we will then test this finding by running the program and attempting to insert four B’s into the return address. We first will attempt this in the root using GDB, and we shall check EIP to make sure it contains the four B’s which it did. Then we need to make sure that we are correct, so we use the ESP of what we got, and take note of the address in little endian, then run the buffer “buf = 'A'\*223 + "\xe8\xf4\xff\xbf" + "\x90"\*32 + "\xcc"\*(1024-223-4-32)”.

This will now attempt to use the address that we’re taking advantage of, and have it jump to run a simple placeholder command containing the address CC in hexadecimal. The hexadecimal address of 90 is used as a NOP sled to make it easier on us when targeting the return address. The return address can be very difficult to always accurately target, using this technique will help minimize this issue, and will run the NOP “do nothing” commands going through all of them until the shellcode gets reached. We had a success, and now we can give this an attempt using shellcode, making use of msfvenom to produce our shellcode for us in hexadecimal. We run the same command, but now we replace “\xcc” with our variable we used for the shellcode which will then allow us to be telnetted to the host we’re going after, and we’ve succeeded. When we run this outside of GDB our ESP address changed so to run it without GDB we’ll need to set buf as, “buf = 'A'\*223 + "\x08\xf5\xff\xbf" + "\x90"\*32 + shellcode”.

After this is done, we should then be able to run the command “nc -v 10.0.2.16 8675” in the shell, which should then connect us to the host’s shell from that vulnerable server. We can check if we’ve succeeded by checking the files using the “ls” command, and we can also use the “whoami” command to check where we are, and it should say “root” if it succeeded. We can also run this same exploit using Metasploit by taking a templet from Github which can be used so that we can make use of Metasploit on our computer. The Metasploit framework will then contain a function we can call “exploit” which will then contain the same logic that we used for our non-Metasploit program. To check if it works, we will need to make sure we save our vuln2 and the new Metasploit program to the .msf4/modules/exploits/linux folder so that it can be used. We will then connect to Metasploit using the command “msfconsole”, then we navigate to our exploit program so that it can be used by typing “use exploit/linux/mymsfexploit”. Lastly, we will have to set the payload which will be “set payload linux/x86/shell\_bind\_tcp”, then set the RHOSTS value to the break in box’s IP address as so, “set RHOSTS 10.0.2.16”. Finally, we just have to use the “run” command to run our Metasploit program, and if it worked, we should’ve telnetted over to the break in box and be in its shell, just like in the other program without Metasploit. The working attacks of the Metasploit and non-Metasploit programs can be seen in the images below.

**Non-Metasploit Program**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Metasploit Program**

**A screenshot of a computer

Description automatically generated**