**Jigyas Sharma**

**Dr. Alexandru Bardas**

**EECS 765**

**14 September 2023**

**Programming Assignment 1: Report**

**Introduction**

The aim of this programming assignment is to create an exploit that will take advantage of the buffer overflow in a program. The targeted operating system is RedHat8 and RedHat9(tentative) and the attack is going to be launched remotely. The exploit is going to target the buffer overflow segmentation faults in nweb. The exploit is going to be designed in a way that the EIP is going to be filled with an address to point to a location in memory which is going to contain a shellcode, and assuming that the nweb server is running with root privileges, the access to the shell provided by the shell code would also have root privileges.

**Running the Exploit**

**Developing the Exploit**

The initial step to creating the exploit is to check for vulnerabilities. The aim of the attack is to obtain a root shell by calling back from the nweb server. The first part of the process was to recreate the environment in which we would be attacking. This includes starting an nweb server on a Redhat8 machine with some defined parameters(ip address and port). The first step was to check for a buffer overflow vulnerability. To check for the buffer overflow vulnerability, use the metasploit frameworks “pattern\_create” and “pattern\_offset” scripts. The length for pattern create was 2000 and the offset received was 1032. However, upon testing the EIP overflows at 1030 instead of 1032. The next part is to develop a shell script that will return a shell to the attackers ip address who will be listening on some port. To create the shell script, use the metasploit framework’s, msfconsole which will create a shell script with LHOST and LPORT, the shell code can also be specified for multiple languages, we used python. Now with the above information we try to create the structure of the malicious input which we will send to the nweb server.

**Structure of the Input**

The structure of the input is as defined below.

Overflow\_string = b”\x90”\*825 + shellcode + b”\xa0\xfa\xff\xbf”

The shell code is 205 bytes, therefore, we added a padding of 825 NOP sleds before the shell code. Now, 825 + 205 = 1030 at which point the input starts overflowing to the EIP where we fill it with an address backwards(Little Endian) which will point to the ESP. The address will be filled with NOP sleds till it reaches our shell code.

**Determining the Parameters used in the Malicious Input**

Firstly, to figure out the buffer length we just keep feeding the program more and more data or you could create an absurdly large pattern using “pattern\_create” in the metasploit framework. I used a pattern of size 3000 and sent it to the nweb server through ncat. This created a core dump on our recreation of the environment which told us that there was a segmentation fault with signal 11. Using the pattern\_offset, we figured out

**Collaborations and References**

[**https://www.fortinet.com/resources/cyberglossary/buffer-overflow**](https://www.fortinet.com/resources/cyberglossary/buffer-overflow)

[**http://users.umiacs.umd.edu/~tdumitra/courses/ENEE657/Fall17/homeworks/enee657\_buffer\_overflow.pdf**](http://users.umiacs.umd.edu/~tdumitra/courses/ENEE657/Fall17/homeworks/enee657_buffer_overflow.pdf)

[**https://web.ecs.syr.edu/~wedu/seed/Book/book\_sample\_buffer.pdf**](https://web.ecs.syr.edu/~wedu/seed/Book/book_sample_buffer.pdf)

[**https://forums.kali.org/archive/index.php/t-28940.html**](https://forums.kali.org/archive/index.php/t-28940.html)

[**https://security.stackexchange.com/questions/157478/why-jmp-esp-instead-of-directly-jumping-into-the-stack**](https://security.stackexchange.com/questions/157478/why-jmp-esp-instead-of-directly-jumping-into-the-stack)