1. Write your shellcode to, instead of spawning a shell, do the “ls –a” command. Test you shellcode to show it works.

**Use the tools given to generate the shellcode for “ls –a”.**

**Steps:**

1. **Make odfhex:**

**g++ -o odfhex odfhex**

1. **Generate the shellcode:**

**./myBuildAsm Snyder\_HW\_02\_01\_ shellcode**

1. **Copy the data from hexdump\_Snyder\_HW\_02\_01\_shellcode.txt to Snyder\_HW\_02\_01\_test.c, then compile Snyder\_HW\_02\_01\_test:**

**./myCompileCC Snyder\_HW\_02\_01\_test**

1. **Verify the results:**

**ls -a**

1. **Copy the data from hexdump\_Snyder\_HW\_02\_01\_shellcode.txt to Snyder\_HW\_02\_01\_call\_shellcode.c, then compile Snyder\_HW\_02\_01\_call\_shellcode:**

**gcc –o Snyder\_HW\_02\_01\_call\_shellcode Snyder\_HW\_02\_01\_call\_shellcode.c**

1. **Verify the results:**

**ls -a**

**File names: Snyder\_HW\_02\_01\_shellcode.asm; hexdump\_Snyder\_HW\_02\_01\_shellcode.txt; Snyder\_HW\_02\_01\_test.c; Snyder\_HW\_02\_01\_a.png; Snyder\_HW\_02\_01\_call\_shellcode.c; Snyder\_HW\_02\_01\_b.png;**

1. Write a program that contains a stack-based (or stack smashing) buffer overflow. Explain in detail why the program has a stack based buffer overflow flaw. Demonstrate the flaw by causing the program to crash with a segmentation fault. Include the state of the stack before the crash, and determine exactly why the program crashed?

**A vulnerable strcpy command reads in an arbitrary length character array from a file and stores it in a fixed length character buffer of size 24. The program does not check the size of the data to be copied into the fixed sized buffer before it does the copy. Having more data than the buffer allows will cause the memory after the end of the fixed size buffer to be overwritten and a segmentation fault. File names: Snyder\_HW\_02\_02\_stack.c; Snyder\_HW\_02\_02\_exploit.c; Snyder\_HW\_02\_02.png; Snyder\_HW\_02\_02\_a.png; Snyder\_HW\_02\_02\_b.png;**

1. Employ your shellcode in question #1 and problem #2. Demonstrate that the exploited program will run the command “ls –a”. Display your results. (Hint: The lab will help)

**Snyder\_HW\_02\_02\_exploit.c was developed to allow us to run the “ls –a” command through the Snyder\_HW\_02\_01\_call\_shellcode.c program that had already been loaded in memory. By debugging using GDB and ‘x/128x buffer’, the address of str\_main in Snyder\_HW\_01\_02\_stack.c was obtained. This address was used to overflow the buffer for 64 bytes. In this case, 64 bytes worked because the distance between the “buffer” and the return address of the Snyder\_HW\_02\_01\_call\_shellcode in memory was less than 64 bytes. Several NOP operations were added to create a NOP sled for the new return address to point. At the end of the NOP sled, instructions to call the method used to open a shell were placed. The rest of the file used to overflow the buffer was filled with enough NOP bytes to overflow, in this case 256 bytes.**

**Steps:**

1. **As root, disable ASLR:**

**sysctl –w kernel.randomize\_va\_space=0**

1. **As root, compile the Snyder\_HW\_02\_01\_call\_shellcode.c program:**

**gcc –z execstack –o Snyder\_HW\_02\_01\_call\_shellcode Snyder\_HW\_02\_01\_call\_shellcode.c**

1. **As root, compile the Snyder\_HW\_02\_02\_stack.c program with the stack check disabled:**

**gcc –fno-stack-protector –z execstack –g –o Snyder\_HW\_02\_02\_stack Snyder\_HW\_02\_02\_stack.c**

1. **As root, change the permissions on the resulting executable file:**

**chmod 4755 Snyder\_HW\_02\_02\_stack**

1. **As a regular user, compile the Snyder\_HW\_02\_02\_exploit program:**

**gcc –o Snyder\_HW\_02\_02\_exploit Snyder\_HW\_02\_02\_exploit.c**

1. **As a regular user, run the exploit program**

**./Snyder\_HW\_02\_02\_exploit**

1. **As a regular user, run the stack program**

**./Snyder\_HW\_02\_02\_stack**

1. **The result is “ls –a” of the current directory.**

**“ls –a” to verify the results**

**File names: Snyder\_HW\_02\_01\_call\_shellcode.c; Snyder\_HW\_02\_02\_stack.c; Snyder\_HW\_02\_02\_exploit.c; Snyder\_HW\_02\_03.png**

1. Give an example of a program that contains an integer overflow error, and explain the problem. Correct the flaw by implementing appropriate range checking. Do not change the types (int, unsigned int, etc.) of any of the variables.

**In the program, 2 unsigned char integers are added together. The first time the two integers are added (254 and 1) resulting value (255) is not greater than the max for an unsigned char integer (0 to 255); therefore, the correct value displays. Then the second integer is updated (4) and the resulting addition will have a value (257) greater than the max for an unsigned char integer (0 to 255), overflowing the integer result. To avoid the overflow, both unsigned char integer values are range checked before the actual addition is executed. An error is printed if completing the addition will overflow the result integer. File names: Snyder\_HW\_02\_04\_a.c (pre-range checking); Snyder\_HW\_02\_04\_b.c (with range checking); Snyder\_HW\_02\_04.png**