1. Write a program with a race condition, either in threads, processes or signals. (The flaw need not be exploitable) Explain the software defect, and why it forms a race condition.

**The code will create four (4) threads an in each thread increment a counter. The majority of the time the code will create a thread and increase the counter before the main function creates another thread; however, there is a possibility that a later thread can read the counter before an earlier thread has time to finish assigning the incremented value. In this case, the final printed value will not be the total number of threads (4), but a lower number.**

**File name: Snyder\_HW\_03\_01.c**

1. Write a program with a race condition in the file system. Demonstrate how to exploit the flaw. You may modify the code somewhat to allow for control of the program’s timing.

**<Lab 3?>**

**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. Write a program that has a format string flaw. Make the program SUID root, and exploit it to obtain a root shell. Overwrite the .dtors section to start the execution of the shellcode.

**<Lab?>**

**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. **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. Write a program that demonstrates the use of format strings for output functions. It should be a suitable example for an introductory programming course.

**File name: Snyder\_HW\_02\_04.c**

1. Create a web application that uses a MySQL database backend. Demonstrate it.

**<Lab 4?>**

**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**

1. Create a web application that uses a MySQL database backend that suffers from a SQL injection vulnerability. Demonstrate it.

**<Lab 4?>**

**In the web application, “title” is not sanitized by mysql\_real\_escape\_string. This means a user can type a specifically crafted exploit string that can edit other user’s information.**

**Steps:**

1. **Open the web application and login as Mal.**
2. **Open my account for Mal and select edit**
3. **Update the “Title” field with the following exploit string:**
   1. **‘, title=‘Second Class Loser’ WHERE name= ‘Sal’;-- ‘**
4. **Press “Finish” to commit the changes.**
5. **Logout and login as Sal**
6. **Verify Sal’s information has been changed:**
   1. **Title: “Second Class Loser”**

**File names: Snyder\_HW\_03\_06.c; Snyder\_HW\_03\_06.c; Snyder\_HW\_03\_06.c**

1. Correct the flaw in the previous problem by correctly sanitizing the input.

**<Lab 4?>**

**mysql\_real\_escape\_string to sanitize the input for “company”**

**In the web application, “title” is now sanitized by mysql\_real\_escape\_string. This means even a specially crafted exploit string will not edit other user’s information.**

**Steps:**

1. **Open the web application and login as Mal.**
2. **Open my account for Mal and select edit**
3. **Update the “Title” field with the following exploit string:**
   1. **‘, title=‘Second Class Loser’ WHERE name= ‘Sal’;-- ‘**
4. **Press “Finish” to commit the changes.**
5. **Logout and login as Sal**
6. **Verify Sal’s information has NOT been changed:**
   1. **Title: “Junior Software Engineer”**
7. **Logout and login as Mal**
8. ***Verify Mal’s information has been changed:***
   1. ***Title: “Junior Software Engineer”***

**File names: Snyder\_HW\_03\_07.c; Snyder\_HW\_03\_07.c; Snyder\_HW\_03\_07.c**