1. The current permission of **myfile** is rw-rw-r--. Provide the commend to change its permission to r--r -- r--

Chmod 444 myfile

1. The current default permission of any files is rw-rw-rw-. What will be the new default permission if you give umask value of 067.

Rw- --- ---

1. Provide a command to change the owner of **myfile** to bob.

Chown bob myfile

1. Alice runs a Set-UID program that is owned by Bob. The program tries to read from /tmp/x, which is readable to Alice, but not to anybody else. Can this program successfully read from the file? (**Note**: Access control is based on EUID) No because the EUID is set to bob’s ID and he cannot read the file
2. A process tries to open a file for read. The process’s effective user ID is 1000, and real user ID is 2000. The file is readable to user ID 2000, but not to user ID 1000. Can this process successfully open the file? No because the the EUID is not able to open the file, and that is the ID that the system is checking to see if the user can open it.
3. We are trying to turn a program (myprogram) owned by the seed user into a Set-UID program that is owned by root. Can running the following commands achieve the goal?

$ sudo chmod 4755 myprogram

$ sudo chown root myprogram

Chown needs to be before chmod or it will not work.

1. Alice (User ID 1000) runs a Bob-owned normal program (Bob’s User ID is 1001). What will be the Real UserID and Effective User ID (EUID) of this program? Both will be 1000

For a normal program the RUID = EUID = Whoever runs it.

1. Alice (User ID 1000) runs a Bob-owned Set-UID program (Bob’s User ID is 1001). What will be the Real UserID and Effective User ID (EUID) of this program? RUID will be 1000 and the EUID will be 1001
2. **env** is a shell command for Linux and it can print a list of the current environment variables. Bob makes a copy of the **env** program and make it a Set-UID program named **myenv**. When bob runs **myenv**, which of the following environment variable will be printed out? Circle all that apply (**NOTE**: LD\_PRELOAD and LD\_LIBRARY\_PATH affect the dynamic linkers)
3. PATH
4. LD\_PRELOAD
5. LD\_LIBRARY\_PATH
6. PWD
7. ANY\_VALUE
8. What is the major problem of the following command line?

$ cat “aa; /bin/sh”

There is code (/bin/sh) where there should be a command.

A file name is expected.

1. What lesson could we learn from question 10?

Never mix data with code.

1. Bob says that he never uses any environment variable in his code, so he does not need to worry about any security problem caused by environment variables. Is he correct?

No, external programs can sill use environment variables and attack

1. What is the main issue with most insecure C functions (like strcpy() ) that deal with buffers?
   1. They pass pointers as arguments,
   2. They are not commented well enough,
   3. The size information for the buffer is not passed to the function.
   4. None of the Above.
2. Buffer Overflow Attack typically result from
   1. malformed inputs
   2. failure to allocate enough space for the buffer
   3. mistaken assumptions about the size of a piece of data
   4. let the user decide on the size of input
   5. all of the above
3. Which of the following code segments are vulnerable to Buffer Overflow Attack? (Circle all that apply)
4. int foo(char \*str) {  
     char buffer[100];  
     strcpy(buffer, str);  
   }
5. int foo(char \*str) {  
     char buffer[100];  
     strncpy(buffer, str, 200);  
    return 1;  
   }
6. int foo(char \*str) {  
     char buf[20];  
    char prefix[] = "http://";  
    strcpy(buf, prefix);;  
    strcat(buf, str);  
    return 1;  
   }
7. int foo(char \*str) {  
     char buf[20];  
    char prefix[] = "http://";  
    strcpy(buf, prefix);;  
    strcat(buf, str, sizeof(buf));  
    return 1;  
   }
8. int foo(char \*str) {  
     char buf[20];  
    scanf(“%9s”, buf);  
   }
9. Please draw the function stack frame for the following *add* function. How are the address decided for the variables a, b and x. (Note: the address of the ebp is *0xAABB0050*)

| |

| Arguments (a,b) |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

|\_\_\_\_\_return address\_\_|

|\_\_\_\_\_old ebp\_\_\_\_\_\_\_\_|\_\_\_ EBP

| x LOCAL |

int add(int a, int b) {  
  int x = a + b  
  return x;  
}

1. Stack guard works by doing which of the following?
   1. Separating stack and data into separate channels.
   2. Making the stack non-executable.
   3. Placing an area on the stack, known as a canary, that it fills with known data.
   4. Randomizing the stack starting address.
2. Why does non-executable stack make buffer-overflow attack more difficult? How can we bypass it.

Code can be injected into the stack, however it will not be executed as the stack has been made non-executable. You can put the code in a different place such as the library

1. Address Space Layout Randomization (ASLR) works better on
   1. 32 bit system
   2. 64 bit system
   3. Same on 32 bit and 64 bit system
   4. None of the above
2. Threads A and B have access to an integer variable C. C currently equals 0 and then both A & B execute the code and do “C = C + 1”. The pseudocode of A and B is as follows

int update(){

temp = get\_C\_value();

temp = temp + 1;

return temp ;

}

Does this program have a race condition problem?

Yes, Just like the bank problem

1. Does the following code has a race condition problem?

Function withdraw ($amount) {

$balance = getBalance();

if($amount <= $balance){

$temp = $balance - $amount;

echo “You have withdrawn: $amount”;

balance = saveBalance($temp);

}

}

YES

1. Which of the following is NOT one of the countermeasures for Race Condition Attack?
   1. Atomic Operations
   2. Stack-Guard
   3. Repeating Check and Use
   4. Sticky Symlink Protection
2. The permission of the file /home/seed/zzz is readable and writeable to the user seed. Does the following code (executed by seed) modify the contect of /home/seed/zzz? (optional)

int f = open(“/home/seed/zzz”, O\_RDWR)

fstat(f, &st);

map = mmap(NULL, st.st\_size, PROT\_READ| PROT\_WRITE, MAP\_PRIVATE, f, 0);

memcpy(map, “new content”, strlen(“new contecnt”));

No, you are triggering copy on write and writing to the duplicate file created and not the original copy since it is MAP\_PRIVATE

1. Which of the following statements about COPY-ON-WRITE is true?
2. If a resource, such as a file or a block of memory, is duplicated, but not modified, there's no need for the system to create a new version, as it can be shared between the copy and the original.
3. COPY-ON-WRITE avoids the overhead of creating a full copy.
4. COPY-ON-WRITE means the copy operation is deferred until the first write occurs.
5. All of the above
6. [True/False] DirtyCOW is a special type of race condition problem.

True

There are a few steps to COW so it can be exploited between those steps

1. When we read from a memory address multiple times, the second access is usually faster than the first access, what is the reason? The information is stored in the CPU cache which is much faster memory.
2. Regarding the Flush+ Reload technique used in the class, why do we use an array of size 256\*4096, why not use an array of size 256 \* 1? (optional) The cpu cache works in blocks and one bit is not big enough.
3. How do we use CPU cache as a side channel to send out a number 89?

By flushing the memory and finding the one that reads faster.

1. To improve the success rate of the Meltdown attack, it is better that the targeted kernel memory has already been cached by CPU. Is this correct? Why? Yes, You can judge if the information has been accessed recently by the speed it is called at. Since the CPU cache is faster.
2. If CPU does not do out-of-order execution, can we still launch the Meltdown attack? What is the downside? No, because it will wait for the permission check before letting the program run. Instead of running them at the same time.