**HOMEWORK -12**

**Question 1: List at least 5 defenses against buffer overflow attacks and provide a sentence or two describing what they are or how they work.**

1. Address space layout randomization - Initialize the stack to a different address in memory each time it is initialized, thus making it nearly impossible to find an address that allow for stack smashing.

2. Non-executable stack - Don't allow code from the stack to be executable, thus eliminating the possibility that the exploit can run code directly from the stack.

3. Bounds checking - Check that all memory addresses accessed by the code are within the proper bounds, preventing any return addresses from being overwritten.

Canaries - Known values that are placed between a buffer and control data on the stack to monitor buffer overflows. When the buffer overflows, the first data to be corrupted will be the canary, and a failed verification of the canary data is therefore an alert of an overflow, which can then be handled, for example, by invalidating the corrupted data.

4. Terminator canaries - Canaries built with NULL terminators (because most buffer overflow attacks are based on certain string operations which end at terminators).

5. Random canaries - A random value generated at program initialization and stored as a global variable. Thus the attacker cannot learn the canary value prior to the program start by searching the executable image. It is generally not possible to read the random canary (and most attempts result in segmentation faults that kill the program).

**Question 2: Describe the attack that defeated a random canary. Explain how the XOR canary defeats the attack.**

Stack canaries work by modifying every function's prologue and epilogue regions to place and check a value on the stack respectively. As such, if a stack buffer is overwritten during a memory copy operation, the error is noticed before execution returns from the copy function. When this happens, an exception is raised, which is passed back up the exception handler hierarchy until it finally hits the OS's default exception handler. If you can overwrite an existing exception handler structure in the stack, you can make it point to your own code. This is a Structured Exception Handling (SEH) exploit, and it allows you to completely skip the canary check.

XOR canaries - A random XOR Canary is one that randomly creates a Canary but it also uses an algorithm that looks at the control parts of the stack as well. One control part of the stack is the return address at the frame pointer. So if an attacker tries to change any part of these control elements of the stack as they normally intend to then the attacker has to change the canary too.

In order to complete an attack on one of these the attacker must know the original canary, and the algorithm and then compute the new canary with what the control value are going to become. This is the hardest canary to overcome however as with all of the canaries this is a protection for the user when they are using vulnerable software and does not help to protect valuable assets of the program.

Random canaries are generally not logically possible to read, unless the attacker knows where it is or can get the program to read from the stack. If the attacker does that, they know the canary value and can overwrite it with the known value.