Buffer Overflow Attack

A Buffer Overflow Attack is an attack that abuses a type of bug called a "buffer overflow", in which a program overwrites memory adjacent to a buffer that should not have been modified intentionally or unintentionally.

Buffer overflows are commonly associated with C-based languages, which do not perform any kind of array bounds checking. As a result, operations such as copying a string from one buffer to another can result in the memory adjacent to the new buffer to be overwritten with excess data.

When a buffer overflow occurs in a program, it will often crash or become unstable. An attacker attempting to abuse a buffer overflow for a more specific purpose other than crashing the target system, can purposely overwrite important values in the call stack of the target machine such as the instruction pointer or base pointer in order to execute his or her potentially malicious unsigned code.

Operating system and software vendors often employ countermeasures in their products to prevent Buffer Overflow Attacks.

A simple buffer overflow attack scenario can be with the code below.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char password[16];
    int passcheck = 0;

    printf("Enter password: \n");
    gets(password);

    if(strcmp(password, "password1"))
    {
        printf("Failure\n");
    }
    else{
        printf("Correct\n");
        passcheck = 1;
    }

    if(passcheck)
    {
        printf("Passcheck\n")
    }

    return 0;
}
```

If the code is run without stack protection and `hhhhhhhhhhhhhhhhhhhhhhhhhhhhh` is given as input, overflow will occur and the value of `passcheck` variable will be `1`.

There is a logic behind the output. What attacker did was, he/she supplied an input of length greater than what buffer can hold and at a particular length of input the buffer overflow so took place that it overwrote the memory of integer `passcheck`. So, despite of a wrong password, the value of `passcheck` became non zero and hence root privileges were granted to an attacker.

There are some ways to prevent buffer overflow attacks. They are as below.

- **Address space randomization**: Address space randomization randomly moves around the address space locations of data regions. Typically, buffer overflow attacks need to know the locality of executable code, and randomizing address spaces makes this virtually impossible.
- **Data execution prevention**: Data execution prevention process flags certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region.
- **Structured exception handler overwrite protection (SEHOP)**: SEHOP helps stop malicious code from attacking Structured Exception Handling (SEH), a built-in system for managing hardware and software exceptions. It thus prevents an attacker from being able to make use of the SEH overwrite exploitation technique. At a functional level, an SEH overwrite is achieved using a stack-based buffer overflow to overwrite an exception registration record, stored on a thread's stack.