**Buffer Overflow**

Data received can maliciously or unintentionally overrun a fixed capacity memory space and overwrite or corrupt data in adjacent spaces. This occurrence is termed as a buffer overflow.

Buffers are memory spaces that temporarily hold data while being transferred. They are vital in cases where the rate data is received is different from the rate data is processed. An example of a buffer overflow attack is when on a login interface page, the password input can be set to expect an 8-byte password, but instead when the user enters a 10-byte password, the excess data can overrun the adjacent memory spaces. A buffer overflow attack can cause system crashes, memory access errors or even make the program perform unauthorized tasks.

Among the various attacks of this nature, the common known ones are:

* Stack based buffer overflow, which occurs when a buffer located on the stack has more data written into it.
* Heap based buffer overflow, which occurs when the memory space is flooded having the application overwrite internal structures as a result.

Attackers use buffer overflows to alter the execution path of applications. When a carefully crafted malicious code is sent to the application and executed, the attacker can take over the application and cause harm.

C and C++ are programming languages that are highly susceptible to this kind of attack as they lack safeguarding techniques that guard against overwriting or data access in their memory. Programming languages that are relatively less susceptible to buffer overflow attacks are Java, Python and C# just to name a few.

**How to prevent/mitigate a buffer overflow attack**

All is not lost as you can protect your program against buffer overflow attacks and one way of doing so is address space randomization. This is randomly moving around the address space locations of data regions making it virtually impossible for the buffer overflow attack to happen for it depends on the locations of the executable code.

Data execution prevention, flagging certain memory areas as non-executable or executable is another data protection method. Furthermore, bounds checking can be applied at runtime, automatically checking that data written to a buffer is within acceptable boundaries.

A structured exception handler overwrite protection can be applied to assist in stopping harmful code from manipulating the Structured Exception Handling, a built in mechanism for managing software and hardware exceptions.

Veracode *What Is a Buffer Overflow? Learn About Buffer Overrun Vulnerabilities, Exploits & Attacks* <https://www.veracode.com/security/buffer-overflow> (no date)(Accessed on: 28 January 2021).

Imperva *Buffer Overflow Attack* <https://www.imperva.com/learn/application-security/buffer-overflow/> (no date)(Accessed on 28 January 2021).

Welekwe, A. (2020) *Buffer overflow vulnerabilities and attacks explained.* Available at:<https://www.comparitech.com/blog/information-security/buffer-overflow-attacks-vulnerabilities/> (Accessed on 28 January 2021).