**SQL Injection:-**

**Occurrence:-**

SQL injection errors occur when:

* Data enters a program from an untrusted source.
* The data used to dynamically construct a SQL query

The main consequences are:

Confidentiality: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.

Authentication: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

Authorization: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.

Integrity: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.

**Risk Factors:-**

The platform affected can be:

* Language: SQL
* Platform: Any (requires interaction with a SQL database)

SQL Injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind.

Essentially, the attack is accomplished by placing a meta character into data input to then place SQL commands in the control plane, which did not exist there before. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.

**Prevention**:-

One traditional approach to prevent SQL injection attacks is to handle them as an input validation problem and either accept only characters from a whitelist of safe values or identify and escape a blacklist of potentially malicious values. Whitelisting can be a very effective means of enforcing strict input validation rules but parameterized SQL statements require less maintenance and can offer more guarantees with respect to security. As is almost always the case, blacklisting is riddled with loopholes that make it ineffective at preventing SQL injection attacks.

Another solution commonly proposed for dealing with SQL injection attacks is to use stored procedures.

**Command Injection:-**

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application.

**Occurrence:-**

Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation. In Command Injection, the attacker extends the default functionality of the application, which execute system commands, without the necessity of injecting code.

**Cross-site Scripting:-**

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

**Occurrence:-**

Cross-Site Scripting (XSS) attacks occur when:

1. Data enters a Web application through an untrusted source, most frequently a web request.
2. The data is included in dynamic content that is sent to a web user without being validated for malicious content.

**Risk factors:-**

XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. The most severe XSS attacks involve disclosure of the user’s session cookie, allowing an attacker to hijack the user’s session and take over the account. Other damaging attacks include the disclosure of end user files, installation of Trojan horse programs, redirect the user to some other page or site, or modify presentation of content.

**Prevention:-**

 it's crucial that you turn off HTTP TRACE support on all web servers. An attacker can steal cookie data via Javascript even when document.cookie is disabled or not supported by the client. This attack is mounted when a user posts a malicious script to a forum so when another user clicks the link, an asynchronous HTTP Trace call is triggered which collects the user's cookie information from the server, and then sends it over to another malicious server that collects the cookie information so the attacker can mount a session hijack attack. This is easily mitigated by removing support for HTTP TRACE on all web servers.

**Xpath Injection:-**

**Occurrence:-**

XPath Injection attacks occur when a web site uses user-supplied information to construct an XPath query for XML data.

**Risk factors:-**

By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured, or access data that he may not normally have access to. He may even be able to elevate his privileges on the web site if the XML data is being used for authentication (such as an XML based user file).

**Prevention**:-

There is a need to use a parameterized XPath interface if one is available, or escape the user input to make it safe to include in a dynamically constructed query. Using quotes to terminate untrusted input in a dynamically constructed XPath query can lead to the attack so there is a need to escape that quote in the untrusted input to ensure the untrusted data can't try to break out of that quoted context.

**Code Injection:-**

Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application. This type of attack exploits poor handling of untrusted data.

**Occurrence:-**

These types of attacks are usually made possible due to a lack of proper input/output data validation, for example:

* allowed characters (standard regular expressions classes or custom)
* data format
* amount of expected data

**Risk factors:-**

* These types of vulnerabilities can range from very hard to find, to easy to find
* If found, are usually moderately hard to exploit, depending of scenario
* If successfully exploited, impact could cover loss of confidentiality, loss of integrity, loss of availability, and/or loss of accountability.

**Prevention**:-

**HTTP Response Splitting:-**

**Occurrence:-**

HTTP response splitting occurs when:

* Data enters a web application through an untrusted source, most frequently an HTTP request.
* The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

**Risk factors:-**

To mount a successful exploit, the application must allow input that contains CR (carriage return, also given by %0d or \r) and LF (line feed, also given by %0a or \n)characters into the header AND the underlying platform must be vulnerable to the injection of such characters. These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allow them to create additional responses entirely under their control.

**Prevention:-**

Testing on the platform of concern to see if the underlying platform allows for CR or LF characters to be injected into headers. In general, this vulnerability has been fixed in most modern application servers, regardless of what language the code has been written in.