**Software security refers to the protection of the programs that are either bought from an outside vendor or are created in-house by the user.**

**Software security engineering aims at developing secure systems that remain trusted in the event of an attack. Activities include: identifying security objectives that systems should satisfy; identifying security risks that threaten system operation; elicitation of security requirements that should be enforced on the system to achieve the expected security level; developing security architectures and designs that deliver the security requirements and integrates with the operational environment; and developing, deploying, and enforcing the developed or purchased security controls.**

**SQL injection**is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution.

SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

It can lead to disclosure of sensitive information, information compromises, and even spread of malicious software.

Attackers use this Injection vulnerability to bypass application security protocols by going around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database (i.e.; add, modify, and delete records in the database).

Content and behavior of many systems/application/websites is built on data in a database server and thus any successful attack on the database that drives a web application or a website, such as a SQL injection login bypass attack, can potentially give a malicious person access to deleting or modifying application content, sensitive information such as account credentials or internal business data.

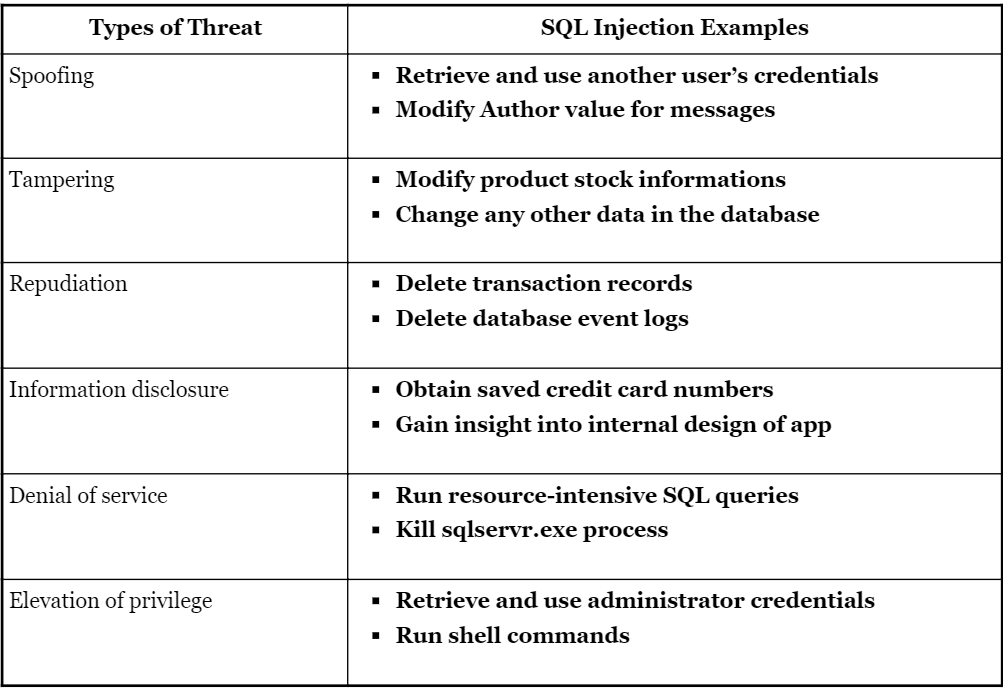


Fig 1: <http://cis1.towson.edu/~cssecinj/modules/other-modules/database/sql-injection-introduction/>

How to address SQL injection attack;

1. **Trust no one:** Assume all user-submitted data is evil so use input validation through a function checks to ensure that any dangerous characters specified are not passed to a SQL query in data. Filtering user data by context. i.e, phone numbers should be filtered to allow only the digits allowed in a phone number.
2. **Don't use dynamic SQL or allow user-defined queries with user input:**
3. **Adopt the latest technologies**: Older web development technologies may not have SQL injection protection protocols, so it is a good practice to use the latest version of the development environment and language and the latest technologies associated with that environment.
4. **Update and patch**: Vulnerabilities in applications and databases that hackers can exploit using SQL injection are regularly discovered, so it's vital to apply patches and updates as soon as practical.
5. **Employ Firewall:** Open source module ModSecurity, which is available for Apache, Microsoft IIS, and nginx web servers. ModSecurity provides a sophisticated and ever-evolving set of rules to filter potentially dangerous web requests. Its SQL injection defenses can catch most attempts to sneak SQL through web channels.
6. Employ verified mechanisms: Don’t try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures.
7. Do not embed the database login password in the application or connection string and storing the database configuration information in the web root.

**Web Server–Related Vulnerabilities**

Cross Site Scripting (XSS)

The concept of Cross-site scripting is to manipulate client server-side scripts of a website to execute in the manner desired by the attacker. XSS targets an application's users by injecting code, usually a client-side script such as JavaScript, into a web application's output. it allows attackers to execute scripts in the browser which can redirect the user to malicious sites, deface websites or tamper with user sessions.

Cross-Site Request Forgery.

Cross-Site Request Forgery (CSRF) is a malicious attack where a user is tricked into performing an action he or she didn't intend to do. An attacker website sends a request to a web application that a user is already authenticated against. The attacker then access functionality via the victim's already authenticated browser. Target Victims may include online banking applications, social media applications, email clients, and web interfaces for network devices.

Insecure Direct Object References

Insecure direct object reference is when a web application exposes a reference to an internal implementation object. Internal implementation objects include files, database records, directories and database keys. When an application exposes a reference to one of these objects in a URL, hackers can manipulate it to gain access to a user's personal data.

How to address Web Server–Related Vulnerabilities attacks;

Check all web-based input for validity and trustworthiness, encode all output originating from user input, add timestamps or timeouts to sessions that are subject to XSRF attacks, regularly test your Web application’s entry points with malformed and escaped script input to test for XSS and related vulnerabilities.

## **Use of Magic URLs and Hidden Form Fields Summary**

Magic URLs, are URLs that contain sensitive information or information that could lead an attacker to sensitive information. Sensitive Data held in the URL, or the HTTP body for that matter is sinful if the payload is not protected by the appropriate cryptographic defenses. If the URL data is used for authentication purposes, then there is probably a security issue.

How to address Web Server–Related Vulnerabilities attacks;

Test all web input, including forms, with malicious input, Understand the strengths and weaknesses of your approach if you’re not using cryptographic primitives to solve some of these issues, do not embed confidential data in any HTTP or HTML construct, such as the URL, cookie, or form, if the channel is not secured using an encryption technology such as SSL, TLS, or IPSec, or it uses application-level cryptographic defenses, do not trust any data, confidential or not, in a web form, because malicious users can easily change the data to any value they like, regardless of SSL use or not.

**Format String Problems**

A format string is an ASCII string that contains text and format parameters. For Example: printf(“my name is:%s\n”,”saif”); Format strings vulnerability exists in most of the printf family in many programming langguages (Printf, vsprintf, and vsnprintf).

If the programmer passes an attacker-controlled buffer as an argument to a printf (or any of the related functions, including sprintf, fprintf, etc), the attacker can perform writes to arbitrary memory addresses.

Format string attack is usually a casual programming error regarding format strings that allows the user to provide the format string portion.

How to address **Format String Vulnerabilities**

* Always specify a format string as part of program, not as an input. Most format string vulnerabilities are solved by specifying “%s” as format string and not using the data string as format string
* Make the format string a constant.
* use defenses such as Format\_Guard
* always use fixed format strings, or format strings from a trusted source
* always check and limit locale requests to valid values

**Catching Exceptions**

Exception handling includes; try-catch blocks implemented in several languages. It is the process of responding to exceptions when a computer program is executed. these exceptions occur when an unexpected event happens that requires special processing.

It is provided by specialized programming language code brakes, computer hardware mechanisms like interrupts or operating system IPC facilities like signals.

Examples include a program attempting to divide by zero, or a user providing abnormal input, or a file system error being encountered when trying to read or write a document file.

For example:

line = console.readLine();

**if** (line.length() == 0) {

**throw** **new** EmptyLineException("The line read from console was empty!");

}

console.printLine("Hello %s!" % line);

console.printLine("The program ran successfully.");

}

**catch** (EmptyLineException e) {

console.printLine("Hello!");

}

Sinful Structured Exception Handling

Current Microsoft Windows operating systems support structured exception handling (SEH). SEH includes the keywords \_\_try, \_\_except, and \_\_finally. Each keyword precedes a block of code, and effectively allows C programs to mimic what C++ accomplishes with try, catch, and destructors