**Security Implementation in Website**

### 1. SQL injection

SQL injection attacks are when an attacker uses a web form field or URL parameter to gain access to or manipulate your database.

When you use standard Transact SQL, it is easy to unknowingly insert rogue code into your query that could be used to change tables, get information and delete data.

Prevention: You can easily prevent this by always using **parameterized queries**

**2. Error messages**

Be careful with how much information you give away in your error messages.

Prevention: Provide only **minimal errors** to your users, to ensure they do not leak secrets present on your server (e.g. API keys or database passwords).

**Keep detailed errors in your server logs**, and show users only the information they need. Do not provide full exception detail either, as these can make complex attacks like SQL injection far easier.

**3. Passwords**

Everyone knows they should use complex passwords, but that does not mean they always do. It is crucial to use **strong passwords** to your server and website admin area, but equally also important to insist on good password practices for your users to protect the security of their accounts.

Prevention: Passwords should always be stored as encrypted values, preferably using a **one-way hashing algorithm such as SHA**.

Using this method means when you are authenticating users you are only ever **comparing encrypted values.** For extra website security, it is a good idea to salt the passwords, using a new salt per password.

**Hashing passwords** is a way of taking a variable-length password and creating a cryptic, fixed-length password from it. You do this by generating and using a salt value. A salt value is a random value that you use to generate the **hashed password**. This topic shows how to generate salt values and **hash passwords**.

### 4. HTTPS

Prevention: HTTPS is a protocol used to provide security over the Internet. HTTPS guarantees to users that they are talking to the server they expect, and that nobody else can intercept or change the content they are seeing in transit.

.

### 5. Server side validation/form validation

Validation should always be done both on **the browser and server side**. The browser can catch simple failures like mandatory fields that are empty and when you enter text into a numbers only field.

These can however be bypassed, and you should make sure you check for these validation server side as failing to do so could lead to malicious code or scripting code being inserted into the database or could cause undesirable results in your website.

Prevention: In the **Server Side Validation**, the input submitted by the user is being sent to the server and validated using one of server side scripting languages such as ASP.Net, PHP, NodeJSetc. After the validation process on the **Server Side**, the feedback is sent back to the client by a new dynamically generated web page.

### 6. XSS

Cross-site scripting (XSS) attacks inject malicious JavaScript into your pages, which then runs in the browsers of your users, and can change page content, or steal information to send back to the attacker.

You need to ensure that users cannot inject active JavaScript content into your pages.

The key here is to focus on how your user-generated content could escape the bounds you expect and be interpreted by the browser as something other that what you intended. This is similar to defending against SQL injection. When dynamically generating HTML, use functions which explicitly make the changes you're looking for (e.g. use element.setAttribute and element.textContent, which will be automatically escaped by the browser, rather than setting element.innerHTML by hand), or use functions in your templating tool that automatically do appropriate escaping, rather than concatenating strings or setting raw HTML content.

Prevention: The powerful tool in the XSS defender's toolbox is Content Security Policy (CSP). CSP is a header your server can return which tells the browser to limit how and what JavaScript is executed in the page, for example to disallow running of any scripts not hosted on your domain, disallow inline JavaScript, or disable eval()..

**8. Unwritable file system**

Prevention: Making the website code and webserver configs on the file system **unwritable by the web user** is a huge security advantage post-compromise.

Almost no websites take this preventative action but it makes many forms of exploitation nearly impossible