**SQL Injection:**

A SQL Injection attack is a form of attack that comes from user input that has not been checked to see that it is valid. The objective is to fool the database system into running malicious code that will reveal sensitive information or otherwise compromise the server.

There are two main types of attacks. First-order attacks are when the attacker receives the desired result immediately, either by direct response from the application they are interacting with or some other response mechanism, such as email. Second-order attacks are when the attacker injects some data that will reside in the database, but the payload will not be immediately activated.

Security is something that needs to be tackled on many levels because a chain is only as strong as its weakest link. When a user interacts with a piece of software, there are many links in the chain; if the user is malicious, he could attempt to attack these links to find the weak point and attempt to break the system at that point. With this in mind, it is important that the developer does not become complacent about the security of the system because one security measure is put in place, or a set of security measures are in place on only one part of the system.

For example, if a web application were to request that the user choose a date, then it would be normal that the values for the date are checked in some JavaScript function on the web page before any data was posted back to the server. This improves the user experience by reducing the wait between lots of server requests. However, the value needs to be validated again on the server as it is possible to spoof the request with a deliberately crafted invalid date.

In order to encrypt data for items such as passwords, the user's password can be stored as a "salted hash". What happens is that when a user creates a password, a randomly generated "salt" value is created by the application and appended to the password, and the password-and-salt are then passed through a one way encryption routine, such as found in the .NET Framework's helper class FormsAuthentication ([HashPasswordForStoringInConfigFile method](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemWebSecurityFormsAuthenticationClassHashPasswordForStoringInConfigFileTopic.asp" \t "_blank)). The result is a salted hash which is stored in the database along with the clear text salt string.

**Summary for preventing SQL Injection:**

* Encrypt sensitive data.
* Access the database using an account with the least privileges necessary.
* Install the database using an account with the least privileges necessary.
* Ensure that data is valid.
* Do a code review to check for the possibility of second-order attacks.
* Use parameterised queries.
* Use stored procedures.
* Re-validate data in stored procedures.
* Ensure that error messages give nothing away about the internal architecture of the application or the database.