SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

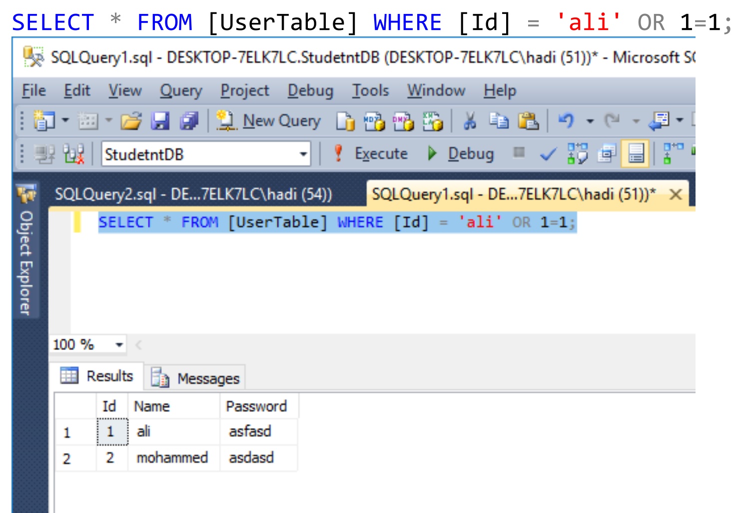
```
1. txtUserId = getRequestString("UserId")
2. txtSQL = "SELECT * FROM UserTable WHERE Id = " + txtUserId
```

1- SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id. If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: ali OR '1'='1

```
SELECT * FROM [UserTable] WHERE [Id] = 'ali' OR 1=1;
```



The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.

2- SQL Injection Based on ""="" is Always True

User name : " or ""="
Password : " or ""="

The code at the server will create a valid SQL statement like this:

```
1. SELECT * FROM Users WHERE Name = "" or "" = "" AND Pass = "" or "" = ""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE.

3- SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statement.

A batch of SQL statements is a group of two or more SQL statements, separated by semicolons.

The SQL statement below will return all rows from the "Users" table, then delete the "Suppliers" table.

```
1. SELECT * FROM Users; DROP TABLE Suppliers
```

Look at the following example:

```
1. txtUserId = getRequestString("UserId");
2. txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

And the following input:

User id: Ali; DROP TABLE Student

```
1. SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

4- Use SQL Parameters for Protection

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

First of all, update DataAccess class by adding optional parameter as list of dbparameter

```
    Public Shared Function GetSingleItems(ByVal sql As String, ByVal Optional dbParameter
As List(Of DbParameter) = Nothing) As Object
        Dim conn As New SqlConnection(ConnStr)
        Dim obj As Object
        Try
            conn.Open()
            Dim cmd As SqlCommand = New SqlCommand(sql, conn)
            If Not IsNothing(dbParameter) AndAlso dbParameter.Count > 0 Then
                For Each p As Object In dbParameter
                    cmd.Parameters.Add(p)
                Next
            End If
            obj = cmd.ExecuteScalar()
        Catch e As Exception
            Throw
        Finally
            conn.Close()
        End Try
        Return obj
    End Function
```

```
Public Shared Function InsertUpdateDelete(ByVal sql As String, ByVal Optional dbParameter As List(Of DbParameter) = Nothing) As Integer
        Dim rows As Integer = 0
        Dim conn As SqlConnection = New SqlConnection(ConnStr)
        Try
            conn.Open()
            Dim cmd As SqlCommand = New SqlCommand(sql, conn)
            If Not IsNothing(dbParameter) AndAlso dbParameter.Count > 0 Then
                For Each p As Object In dbParameter
                    cmd.Parameters.Add(p)
                Next
            End If
            rows = cmd.ExecuteNonQuery()
        Catch e As Exception
            Throw
        Finally
            conn.Close()
        End Try

        Return rows
    End Function
    Public Shared Function GetMultitpleItems(ByVal sql As String, ByVal Optional dbParameter As List(Of DbParameter) = Nothing) As DataTable
        Dim conn As New SqlConnection(ConnStr)
        Dim dt As New DataTable
        Try
            conn.Open()
            Dim da As SqlDataAdapter = New SqlDataAdapter()
            Dim cmd As SqlCommand = New SqlCommand(sql, conn)
            If Not IsNothing(dbParameter) AndAlso dbParameter.Count > 0 Then
                For Each p As Object In dbParameter
                    cmd.Parameters.Add(p)
                Next
            End If
            da.SelectCommand=cmd
            da.Fill(dt)
        Catch e As Exception
            Throw
        Finally
            conn.Close()
        End Try
        Return dt
    End Function
```

Then we need to modify repository and use SQL parameters

1- Convert sql string as follow:

```
Dim sql = String.Format("Select name from UserTable where name ='{0}' and password='{1}'", user.Name, user.Password)
```

```
Dim sql As String="select Name from UserTable where name='@Name' and password='@Password'"
```

The code for IsUserExist is rewritten as follow:

```
Public Function IsUserExist(user As AppUser) As Boolean
        Dim sql As String="select Name from UserTable where name='@Name' and 
password='@Password'"
        Dim plist As New List(of DbParameter)
        Dim p1=New SqlParameter("@Name",user.Name)
        plist.Add(p1)
        Dim p2=New SqlParameter("@Password",SqlDbType.NVarChar)
        p2.Value=user.Password
        plist.Add(p2)
        Dim name as string=DataAccess.GetSingleItems(sql,plist)
        Return name=user.Name
    End Function

The rest code of repository :

Public Function Add(user As AppUser) As Integer
        Dim sql ="INSERT INTO UserTable (Name,Password) VALUES ('@Name','@Password')"
        Dim plist As New List(of DbParameter)
        Dim p1=New SqlParameter("@Name",SqlDbType.NVarChar)
        p1.Value=user.Name
        plist.Add(p1)
        Dim p2=New SqlParameter("@Password",SqlDbType.NVarChar)
        p2.Value=user.Password
        plist.Add(p2)
        Dim row =DataAccess.InsertUpdateDelete(sql,plist)
        Return row
    End Function
    Public Function Update(user As AppUser) As Integer
        Dim sql ="UPDATE UserTable SET Name = '@Name',[Password] = '@Password' WHERE 
id='@id'"
        Dim plist As New List(of DbParameter)
        Dim p1=New SqlParameter("@Name",SqlDbType.NVarChar)
        p1.Value=user.Name
        plist.Add(p1)
        Dim p2=New SqlParameter("@Password",SqlDbType.NVarChar)
        p2.Value=user.Password
        plist.Add(p2)
        Dim p3=New SqlParameter("@id",SqlDbType.Int)
        p3.Value=user.id
        plist.Add(p3)
        Dim row =DataAccess.InsertUpdateDelete(sql,plist)
        Return row
    End Function
```

Interfaces

Interfaces define the properties, methods, and events that classes can implement. With an Interface we create a **contract**. Each Class that implements it must have certain Functions. Interfaces allow you to define features as small groups of closely related properties, methods, and events; this reduces compatibility problems because you can develop enhanced implementations for your interfaces without jeopardizing existing code. You can add new features at any time by developing additional interfaces and implementations.

There are several other reasons why you might want to use interfaces instead of class inheritance:

1. Interfaces are better suited to situations in which your applications require many possibly unrelated object types to provide certain functionality.
2. Interfaces are more flexible than base classes because you can define a single implementation that can implement multiple interfaces.
3. Interfaces are better in situations in which you do not have to inherit implementation from a base class.
4. Interfaces are useful when you cannot use class inheritance. For example, structures cannot inherit from classes, but they can implement interfaces.

Example:

```
Public interface IUserRepository
    Function IsUserExist(usr As AppUser) As Boolean
    Function GetUser() As List(Of AppUser)
    Function Add(user As AppUser) As Integer
    Function Update(user As AppUser) As Integer
end interface


Public Class UserRepository
    Implements IUserRepository

Public Function IsUserExist(user As AppUser) As Boolean Implements
IUserRepository.IsUserExist

...
```