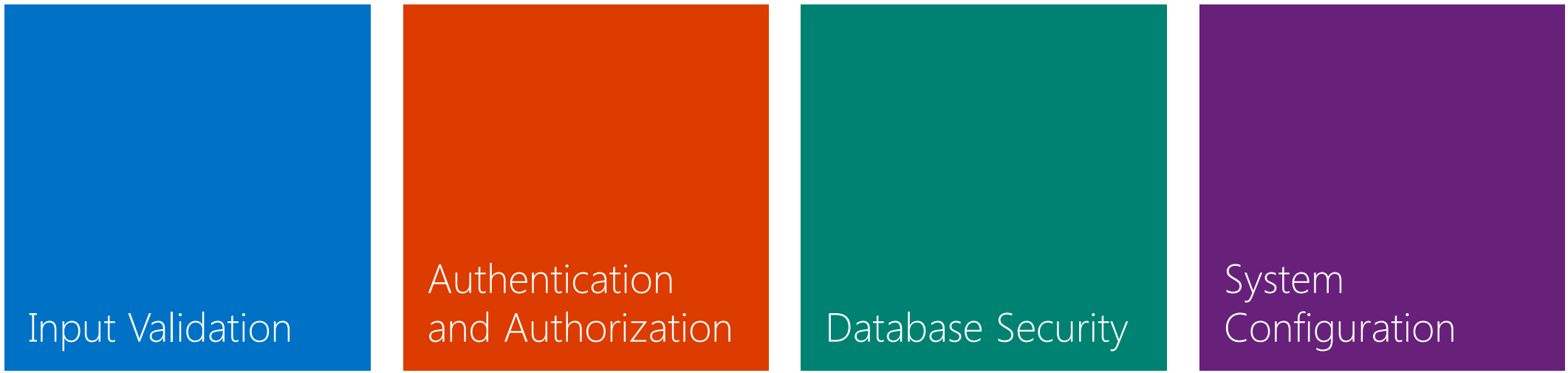
Modern Practices for Secure Code

Marios Patsias

Microsoft CERTIFIED Trainer

patsoft limited

Modern Practices for Secure Code



Modern Practices for Secure Code (cont.)

Input Validation

- ❑ Classify all data sources into trusted and untrusted
  - Treat all 3rd party databases, command line arguments, user input, uploaded files as *untrusted*
- ❑ Validate all user input on a trusted system (server-side)
  - Do not trust client side validations
- ❑ Develop a centralized input validation routine
- ❑ Encode data into a common character set before validation
- ❑ Validate data range
- ❑ Validate data length
- ❑ Validate for expected data types
- ❑ Validate all input a “white list” of allowed/expected characters (if possible)

Authentication and Authorization

- ❑ Require authentication for ALL pages except public
- ❑ Segregate authentication logic from the resource being requested and use redirection to and from the centralized authentication control
- ❑ Utilize authentication for connections to external systems that involve sensitive information or functions
- ❑ Enforce password complexity requirements
- ❑ Enforce account disabling after number of invalid login attempts
  - Disable account for a period of time

Database Security

- ❑ Always use strongly typed parameterized queries
- ❑ Ensure that all variables are strongly typed
- ❑ Use the lowest possible level of privileges when accessing the database
- ❑ Connection strings should always be encrypted in a separate configuration file
- ❑ Close the database connection as soon as possible
- ❑ Disable default accounts that are not required for the specific application
- ❑ Disable all unnecessary functionality
  - System stored procedures, services or features
- ❑ Application should use different credential for each user type
  - Use application roles instead of user roles (when possible)
  - Read-only, administrator , read-write …

System Configuration

- ❑ Ensure that server components are fully updated
- ❑ Disable directory listings
- ❑ Restrict web servers and service accounts to the minimum privileges required
- ❑ Remove all unnecessary functionality and files
- ❑ Encrypt all sensitive data contained in configuration files

Session Management

- ❑ Session identifier creation must always be done on a trusted system (e.g. the server)
- ❑ Logout functionality should fully terminate the associated session
- ❑ Establish a session inactivity timeout as short as possible
  - Based on balancing risk
  - Based on business requirements
- ❑ Disallow persistent logins and enforce periodic session terminations
  - Termination times should support business requirements
  - User should receive sufficient notifications
- ❑ Supplement standard session management for server-side operations

Error Handling

- ❑ Do no disclose sensitive information in error responses
- ❑ Properly free allocated memory when error conditions occur
- ❑ Restrict access to logs to only authorized individuals
- ❑ Use cryptographic hash function to validate log entry integrity
- ❑ Use error handlers that do not display debugging or stack trace information

Data Protection

- ❑ Protect all cached or temporary copies of sensitive data stored on the server
- ❑ Do not store passwords, connection strings or other sensitive information in clear text on the client side
- ❑ Remove sensitive data when the data is no longer required
- ❑ Implement appropriate access control for sensitive data stored on the server
- ❑ Remove unnecessary application and system documentation
- ❑ Disable client side caching on pages containing sensitive information

General Coding Practices

- ❑ Protect shared variables and resources from inappropriate concurrent access
- ❑ Elevated privileges should be raised as late as possible, and dropped as soon as possible
- ❑ Review secondary applications
  - 3rd party code and libraries can introduce new vulnerabilities
- ❑ Implement safe updating
  - Use cryptographic signatures
  - Use encrypted channels

Secure ASP.NET applications from SQL Injection

What is SQL Injection

SQL injection (SQLI) is a technique that allows a user to inject SQL commands into the database engine from a vulnerable application. By leveraging the syntax and capabilities of SQL, the attacker can influence the query passed to the back-end database in order to extract sensible information or to get control over the database.

- ❑ SQL injection attacks are among the most popular security issues today.
- ❑ Is not a DMBS related
- ❑ Caused by Dynamic Query Building

Dynamic Query Building

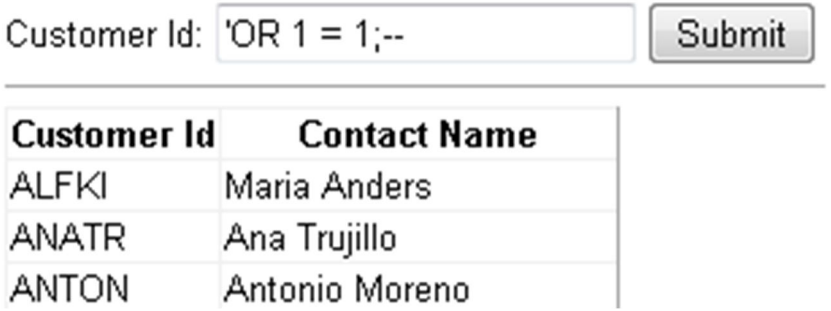
Typical Dynamic query code

```
protected void Submit(object sender, EventArgs e)
{
    string conString
= ConfigurationManager.ConnectionStrings["constr"].ConnectionString;
    using (SqlCommand cmd = new SqlCommand("SELECT * FROM Customers
WHERE CustomerId = '" + txtCustomerId.Text + "'"))
    {
        using (SqlConnection con = new SqlConnection(conString))
        {
            con.Open();
            cmd.Connection = con;
            GridView1.DataSource = cmd.ExecuteReader();
            GridView1.DataBind();
            con.Close();
        }
    }
}
```

Standard Usage

SELECT * FROM Customers WHERE CustomerId = 'ALFKI'

Hack #1: Get all records from table



SELECT * FROM Customers WHERE CustomerId = '' OR 1 = 1;--'

Hack #2: Get all tables names in database


```
SELECT * FROM Customers WHERE CustomerId = '' AND 1 = 2  UNION
SELECT table_schema, table_name, 1 FROM information_schema.tables;--'
```


```
SELECT * FROM Customers WHERE CustomerId = '' AND 1 =2 UNION
SELECT owner, table_name, 1 FROM all_tables;--'
```


```
SELECT * FROM Customers WHERE CustomerId = '' AND 1 =2 UNION
SELECT table_schema, table_name, 1 FROM information_schema.table
```

Hack #3: Deleting records

```
SELECT * FROM Customers WHERE CustomerId = '';DELETE FROM Persons;--'
```

Hack #4: Droping tables

```
SELECT * FROM Customers WHERE CustomerId = '';DROP TABLE Persons;--'
```

Solution #1:Use strongly typed parameters

Strongly typed Dynamic query

```
    using (SqlCommand cmd = new SqlCommand("SELECT * FROM Customers
WHERE CustomerId =@CustomerId"))


....
SqlParameter parameter = new SqlParameter("@CustomerId",
SqlDbType.VarChar);
....

      parameter.value =  txtCustomerId.Text.Trim;
      parameter.Size =10;
....
      cmd.Parameters.Add(parameter)
        }
    }
}
```

Solution #2: Allow only valid characters

Remove special characters used in SQL Queries:

- ❑ semi-colon (;)
- ❑ dash (-)
- ❑ percentage (%)