# OWASP

CODE FELLOWS

401 .NET CORE

# OWASP TOP 10 .NET MVC

1. SQL Injection
2. Weak Account Management
3. Cross Site Scripting
4. Insecure Direct object references
5. Security Misconfigurations
6. Sensitive data exposure
7. Missing function level access control
8. Cross site request forgery
9. Using components with unknown vulnerabilities
10. Unvalidated redirects and forwards

# 1. SQL INJECTION

- Modifying an input parameter that can force a SQL statement to execute in a different way than expected

- Example:
  - Imagine a query like this:
    - SqlCommand command = new SqlCommand($"SELECT * FROM NonSensitiveDataTable WHERE Id = {Request.Query["id"]}", connection);
  - Can easily be restructured to this:
    - SELECT * FROM NonSensitiveDataTable WHERE Id = 999 UNION SELECT * FROM SensitiveDataTable

# SQL INJECTION SOLUTION

```
1   int id = int.Parse(Request.Query["id"]);
2   SqlCommand command = new SqlCommand($"SELECT * FROM
    SuperSensitiveDataTable WHERE Id = {id}", connection);
```

- Sanitize your inputs
  - Know the expected data types

# SANITIZE YOUR INPUTS

- .NET CORE solution:
  - ROUTE Tag

```
1   [HttpGet]
2   [Route(“myroute/{id}")]
3   public string OnlyIntsAreAllowedHere(int id)
```

```
var sql = @"Update [User] SET FirstName = @FirstName WHERE Id = @Id";
context.Database.ExecuteSqlCommand(
    sql,
    new SqlParameter("@FirstName", firstname),
    new SqlParameter("@Id", id));
```

# PARAMETERIZED QUERIES

# STORED PROCEDURES

- Not as popular
- Allow you to run queries on the database side
- Send the name of the parameters to the SP similar to parametrized queries.

# USE AN ORM

- Entity Framework
- LINQ query gets packaged like a parametrized query
- Not impossible to have SQL injections in .NET Core

```
var user = "johndoe";

var blogs = context.Blogs
    .FromSql($"EXECUTE dbo.GetMostPopularBlogsForUser {user}")
    .ToList();
```

# 2. WEAK ACCOUNT MANAGEMENT

- Password Hashing
- Salting
- Session Identifiers
- Unencrypted connections

# PASSWORD HASHING

| 1. | 2. | 3. |
| --- | --- | --- |
| DO NOT save passwords in plain text | Use .NET CORE Identity<br>• PBKDF2 hashing<br>• Random salt per user | Don't "invent" a hashing algorithm |

# SALTING

- Adding a random string to your text before hashing
- Two exact same passwords with salts hash differently
- Keep it unique
- .NET CORE – saved in a separate column.
  - Still secure since they are all unique

# SESSION IDENTIFIERS

- Cookieless sessions
- Don’t share a url that could impersonate a user

# UNENCRYPTED CONNECTIONS

MAKE SURE YOU ARE CONNECTED OVER HTTP USING SSL/TLS

# CROSS-SITE SCRIPTING (XSS)

- Write scripts directly onto a page
- JavaScript most commonly
- Steal private data

# POTENTIAL WITH XSS

| | |
|---|---|
| JavaScript: | • Inject script tags onto a webpage<br>• Redirect the user to a different page<br>• Build a fake login page<br>• Steal login cookie |
| CSS | • Inject styles onto a page<br>• Change entire layout to trick the user |
| IFRAMES | • Can go undetected<br>• Inject “pay per view” ads<br>• Fake logins |

# SOLUTIONS

Don't allow "<scripts>" on your page

HTML Encoding user output

.NET Core always encodes output from users
Every JavaScript (including JQuery) will encode (may be manual)

URL Encoding

URLS do not encode same as HTML
.NET Core offers ability to encode user input for URLS

Browser Protection

Chrome doesn't allow XSS.
XSS Filters

.NET CORE TagHelpers

Not all frameworks support this yet.

# INSECURE DIRECT OBJECT REFERENCES

- Ids or reference variables that can be changed by an end user
- [Authorize] tag in .NET Core prevents anonymous users but not logged in usrs
  - Policies
  - Validate on the server side
    - Captured claims against current logged in user against resource
  - Anything on the browser side can be modified
    - JavaScript
    - Hidden Fields
    - Cookies

# CROSS-ORIGIN RESOURCE SHARING (CORS)

- Don't allow "just anyone"  to access your site
  - Unless its designed to be so
- .NET Core has a "UseCors()" configuration
  - Allow you to control who can make AJAX requests to the site

# UNAUTHORIZED DIRECTORY TRAVERSAL

- Don’t allow users to do a directory traversal
- Don’t allow users to have access to sensitive or bulk information
  - Backups
  - User profiles
  - “All” the images