Cross-Site Request Forgery (CSRF) and Mitigation Techniques using ASP.NET Core

Introduction to CSRF

Cross-Site Request Forgery (CSRF) is a security vulnerability where an attacker tricks a user into unknowingly executing actions on a website they are authenticated with. This is achieved by injecting malicious requests into the user's session, leveraging the trust relationship between the user and the target website. Imagine someone tricking you into clicking a link that makes you unintentionally change your password on a site you're logged into. That's CSRF.

Example of CSRF

Imagine you're logged into your favorite shopping website where you have saved payment information. You then visit a different website that has a hidden button saying "Get Free Gift!" When you click that button, without you knowing, it actually sends a request to the shopping website to buy expensive items using your stored payment information.

More Real-Life Use Cases of CSRF

Example 1: Unauthorized Email Change

Imagine you're logged into your email account where you have important messages and personal information. You then visit a malicious website that has a hidden form saying "Claim Your Prize!" When you visit this site, it secretly sends a request to your email provider to change your account's recovery email to the attacker's email address. Now, if you forget your password, recovery emails go to the attacker.

Example 2: Unauthorized Social Media Post

You're logged into your social media account where you regularly connect with friends and share updates. You visit a website that claims to offer "Exclusive Content!" and prompts you to click a hidden button. This button actually sends a request to your social media account to post a message promoting the attacker's website or content without your knowledge.

Example 3: Unauthorized Fund Transfer

You're logged into your online banking account where you manage your savings and payments. You then visit a gaming website that promises "Free In-Game Currency!" and encourages you to click a hidden link. This link sends a request to your banking website to transfer funds from your savings account to the attacker's account, without your consent.

Preventing CSRF

To prevent CSRF attacks, websites often use tokens that are included with each request. These tokens are unique to each user session and are checked by the server to ensure that the request came from a legitimate source (the actual website) and not from a malicious one (like the hidden button on another site).

Implementation in ASP.NET Core

Here’s a basic example of how you might implement CSRF protection in ASP.NET Core using Anti-CSRF tokens:

Generate Token: When a user visits a page where an action can be taken (like transferring money), generate a unique token.
Include Token in Form: Include this token as a hidden field in any forms that could potentially change data (like transferring money).
Validate Token: When the form is submitted, verify that the token sent matches the one expected for that user’s session. If not, reject the request.

Example Code

```
// HomeController.cs

using Microsoft.AspNetCore.Mvc;

public class HomeController : Controller
{
    // GET: TransferMoney page
    public IActionResult TransferMoney()
    {
        // Generate CSRF token
        string csrfToken = Guid.NewGuid().ToString();

        // Store CSRF token in session or view data
        HttpContext.Session.SetString("CSRF-Token", csrfToken);

        return View();
    }
    // POST: TransferMoney action
    [HttpPost]
   // [ValidateAntiForgeryToken] // This attribute validates the CSRF token
    public IActionResult TransferMoney(int amount, string csrfToken)
    {
        // Retrieve CSRF token from session or view data
        var expectedToken = HttpContext.Session.GetString("CSRF-Token");

        // Validate CSRF token
        if (csrfToken != expectedToken)
        {
            // Handle CSRF token validation failure (e.g., return to home page)
            return RedirectToAction("Index");
        }

        // Process the money transfer logic
        // ...

        return RedirectToAction("TransferSuccess");
    }
}
```

Example View (TransferMoney.cshtml)

```
@model MyWebApp.Models.TransferMoneyViewModel

<form asp-action="TransferMoney" method="post">
   // @Html.AntiForgeryToken() <!-- Generates and includes CSRF token -->

    <label for="amount">Amount:</label>
    <input type="text" id="amount" name="amount" />

    <button type="submit">Transfer Money</button>
</form>
```

Explanation

- **Generating Token**: In the `TransferMoney` action method of `HomeController`, a CSRF token (`csrfToken`) is generated using `Guid.NewGuid().ToString()` and stored in the user's session (`HttpContext.Session.SetString("CSRF-Token", csrfToken)`).

- **Including Token in Form**: The `TransferMoney.cshtml` view includes `@Html.AntiForgeryToken()`, which generates a hidden field containing the CSRF token (`<input name="__RequestVerificationToken" type="hidden" value="...">`).
- **Validating Token**: Upon form submission (`POST` request to `TransferMoney` action), the `ValidateAntiForgeryToken` attribute ensures that the submitted CSRF token (`csrfToken`) matches the expected token stored in the session (`expectedToken`). If not, the request is rejected to prevent CSRF attacks.

This approach helps protect sensitive actions (like transferring money) from CSRF attacks by ensuring that only requests originating from your application (with the correct CSRF token) are processed.

CSRF attacks exploit the trust between a user and a website they're logged into. By understanding how CSRF works and implementing simple measures like Anti-CSRF tokens in ASP.NET Core, you can protect your application from such attacks. Always validate and secure user actions to ensure the safety and integrity of your users' data and interactions.

Real-Life Example: Update Email Address

Imagine you have a web application where users can update their email addresses. To prevent CSRF attacks, you'll use Anti-CSRF tokens generated by `@Html.AntiForgeryToken()` and validated by `[ValidateAntiForgeryToken]`.

Controller Code (AccountController.cs)

```
using Microsoft.AspNetCore.Mvc;

public class AccountController : Controller
{
    // GET: UpdateEmail page
    public IActionResult UpdateEmail()
    {
        return View();
    }

    // POST: UpdateEmail action
    [HttpPost]
    [ValidateAntiForgeryToken] // Validates Anti-CSRF token
    public IActionResult UpdateEmail(string newEmail)
    {
        // Simulate updating email address (replace with real logic)
        ViewBag.Message = $"Email address updated to {newEmail}";

        return View();
    }
}
```

View (UpdateEmail.cshtml)

```
@model UpdateEmailViewModel

<h2>Update Email Address</h2>

<form asp-action="UpdateEmail" method="post">
    @Html.AntiForgeryToken() <!-- Generates Anti-CSRF token -->

    <label for="newEmail">New Email Address:</label>
    <input type="email" id="newEmail" name="newEmail" />

    <button type="submit">Update Email Address</button>
</form>

@if (ViewBag.Message != null)
{
    <p>@ViewBag.Message</p>
}
```

Explanation

Controller (AccountController.cs)

- **GET Action**: `UpdateEmail()` renders the initial view where users can enter their new email address.
- **POST Action**: `UpdateEmail(string newEmail)` handles the form submission. It is decorated with `[ValidateAntiForgeryToken]`, which automatically validates the Anti-CSRF token submitted with the form.

View (UpdateEmail.cshtml)

- **Anti-CSRF Token**: `@Html.AntiForgeryToken()` generates a hidden field (`__RequestVerificationToken`) containing a unique token.

- **Form Submission**: When users submit the form, the Anti-CSRF token is automatically validated by `[ValidateAntiForgeryToken]`.

CSRF Protection in Action

1. **Token Generation**: The `@Html.AntiForgeryToken()` generates a unique token and includes it in the form.
2. **Token Validation**: `[ValidateAntiForgeryToken]` compares the token submitted with the form to the token stored in the server-side session.
3. **Preventing Attacks**: If the tokens match, the request is processed (`UpdateEmail(string newEmail)`). If not, ASP.NET Core rejects the request, preventing CSRF attacks.

Real-Life Example: Change Password

Imagine you have a web application where users can change their passwords. To prevent CSRF attacks, you'll use Anti-CSRF tokens generated by `@Html.AntiForgeryToken()` and validated by `[ValidateAntiForgeryToken]`.

Controller Code (AccountController.cs)

```
using Microsoft.AspNetCore.Mvc;

public class AccountController : Controller
{
    // GET: ChangePassword page
    public IActionResult ChangePassword()
    {
        return View();
    }

    // POST: ChangePassword action
    [HttpPost]
    [ValidateAntiForgeryToken] // Validates Anti-CSRF token
    public IActionResult ChangePassword(string currentPassword, string newPassword)
    {
        // Simulate changing password (replace with real logic)
        ViewBag.Message = "Password changed successfully";

        return View();
    }
}
```

View (ChangePassword.cshtml)

```
@model ChangePasswordViewModel

<h2>Change Password</h2>

<form asp-action="ChangePassword" method="post">
    @Html.AntiForgeryToken() <!-- Generates Anti-CSRF token -->

    <label for="currentPassword">Current Password:</label>
    <input type="password" id="currentPassword" name="currentPassword" />

    <label for="newPassword">New Password:</label>
    <input type="password" id="newPassword" name="newPassword" />

    <button type="submit">Change Password</button>
</form>

@if (ViewBag.Message != null)
{
    <p>@ViewBag.Message</p>
}
```

Real-Life Example: Purchase Order Submission

Imagine you have a web application where authorized users can submit purchase orders for goods or services. To prevent CSRF attacks, you'll use Anti-CSRF tokens generated by `@Html.AntiForgeryToken()` and validated by `[ValidateAntiForgeryToken]`.

```
using Microsoft.AspNetCore.Mvc;

public class PurchaseController : Controller
{
    // GET: SubmitOrder page
    public IActionResult SubmitOrder()
    {
        return View();
    }

    // POST: SubmitOrder action
    [HttpPost]
    [ValidateAntiForgeryToken] // Validates Anti-CSRF token
    public IActionResult SubmitOrder(string productName, int quantity)
    {
        // Simulate submitting a purchase order (replace with real logic)
        ViewBag.Message = $"Order for {quantity} units of {productName} submitted
successfully";
```

```
        return View();
    }
}
```

View (SubmitOrder.cshtml)

```
@model SubmitOrderViewModel

<h2>Submit Purchase Order</h2>

<form asp-action="SubmitOrder" method="post">
    @Html.AntiForgeryToken() <!-- Generates Anti-CSRF token -->

    <label for="productName">Product Name:</label>
    <input type="text" id="productName" name="productName" />

    <label for="quantity">Quantity:</label>
    <input type="number" id="quantity" name="quantity" />

    <button type="submit">Submit Order</button>
</form>

@if (ViewBag.Message != null)
{
    <p>@ViewBag.Message</p>
}
```