**Preventing Cross-Site Request Forgery (CSRF) Attacks in ASP.NET MVC Application**

Cross-Site Request Forgery (CSRF) is an attack where a malicious site sends a request to a vulnerable site where the user is currently logged in

Here is an example of a CSRF attack:

1. A user logs into www.example.com using forms authentication.
2. The server authenticates the user. The response from the server includes an authentication cookie.
3. Without logging out, the user visits a malicious web site. This malicious site contains the following HTML form:

HTMLCopy

<h1>You Are a Winner!</h1>

<form action="http://example.com/api/account" method="post">

<input type="hidden" name="Transaction" value="withdraw" />

<input type="hidden" name="Amount" value="1000000" />

<input type="submit" value="Click Me"/>

</form>

Notice that the form action posts to the vulnerable site, not to the malicious site. This is the "cross-site" part of CSRF.

1. The user clicks the submit button. The browser includes the authentication cookie with the request.
2. The request runs on the server with the user's authentication context, and can do anything that an authenticated user is allowed to do.

Although this example requires the user to click the form button, the malicious page could just as easily run a script that submits the form automatically. Moreover, using SSL does not prevent a CSRF attack, because the malicious site can send an "https://" request.

Typically, CSRF attacks are possible against web sites that use cookies for authentication, because browsers send all relevant cookies to the destination web site. However, CSRF attacks are not limited to exploiting cookies. For example, Basic and Digest authentication are also vulnerable. After a user logs in with Basic or Digest authentication. the browser automatically sends the credentials until the session ends.

**Anti-Forgery Tokens**

To help prevent CSRF attacks, ASP.NET MVC uses anti-forgery tokens, also called *request verification tokens*.

1. The client requests an HTML page that contains a form.
2. The server includes two tokens in the response. One token is sent as a cookie. The other is placed in a hidden form field. The tokens are generated randomly so that an adversary cannot guess the values.
3. When the client submits the form, it must send both tokens back to the server. The client sends the cookie token as a cookie, and it sends the form token inside the form data. (A browser client automatically does this when the user submits the form.)
4. If a request does not include both tokens, the server disallows the request.

Here is an example of an HTML form with a hidden form token:

HTMLCopy

<form action="/Home/Test" method="post">

<input name="\_\_RequestVerificationToken" type="hidden"

value="6fGBtLZmVBZ59oUad1Fr33BuPxANKY9q3Srr5y[...]" />

<input type="submit" value="Submit" />

</form>

Anti-forgery tokens work because the malicious page cannot read the user's tokens, due to same-origin policies. ([Same-origin policies](http://www.w3.org/Security/wiki/Same_Origin_Policy) prevent documents hosted on two different sites from accessing each other's content. So in the earlier example, the malicious page can send requests to example.com, but it cannot read the response.)

To prevent CSRF attacks, use anti-forgery tokens with any authentication protocol where the browser silently sends credentials after the user logs in. This includes cookie-based authentication protocols, such as forms authentication, as well as protocols such as Basic and Digest authentication.

You should require anti-forgery tokens for any nonsafe methods (POST, PUT, DELETE). Also, make sure that safe methods (GET, HEAD) do not have any side effects. Moreover, if you enable cross-domain support, such as CORS or JSONP, then even safe methods like GET are potentially vulnerable to CSRF attacks, allowing the attacker to read potentially sensitive data.

**Anti-Forgery Tokens in ASP.NET MVC**

To add the anti-forgery tokens to a Razor page, use the **HtmlHelper.AntiForgeryToken** helper method:

CSHTMLCopy

@using (Html.BeginForm("Manage", "Account")) {

@Html.AntiForgeryToken()

}

This method adds the hidden form field and also sets the cookie token.

**Anti-CSRF and AJAX**

The form token can be a problem for AJAX requests, because an AJAX request might send JSON data, not HTML form data. One solution is to send the tokens in a custom HTTP header. The following code uses Razor syntax to generate the tokens, and then adds the tokens to an AJAX request. The tokens are generated at the server by calling **AntiForgery.GetTokens**.

HTMLCopy

<script>

@functions{

public string TokenHeaderValue()

{

string cookieToken, formToken;

AntiForgery.GetTokens(null, out cookieToken, out formToken);

return cookieToken + ":" + formToken;

}

}

$.ajax("api/values", {

type: "post",

contentType: "application/json",

data: { }, // JSON data goes here

dataType: "json",

headers: {

'RequestVerificationToken': '@TokenHeaderValue()'

}

});

</script>

When you process the request, extract the tokens from the request header. Then call the **AntiForgery.Validate** method to validate the tokens. The **Validate** method throws an exception if the tokens are not valid.

C#Copy

void ValidateRequestHeader(HttpRequestMessage request)

{

string cookieToken = "";

string formToken = "";

IEnumerable<string> tokenHeaders;

if (request.Headers.TryGetValues("RequestVerificationToken", out tokenHeaders))

{

string[] tokens = tokenHeaders.First().Split(':');

if (tokens.Length == 2)

{

cookieToken = tokens[0].Trim();

formToken = tokens[1].Trim();

}

}

AntiForgery.Validate(cookieToken, formToken);

}