A8 Cross Site Request Forgery Lab

We learned about the dangers of CSRF in the last lecture. Now let's work with it. In this lab we'll explore a vulnerable resource and then learn how to protect it.

Mounting the attack

1. Open your site and log in.
2. Now pretend that you were convinced to visit the evil attacker's site. In another tab, visit this url: http://localhost:8888/CSRF.html.
3. Once you've visited the site, go take a look at the aspnet_Membership table and find your entry. Look at the email address! The bad guy has just hijacked your account. Now he/she can say he forgot your password and get a reset password emailed to him/her.

Recognizing the vulnerability

How did this happen?!? Let's find out.

4. Make sure you're still logged in to the site.
5. Open Fiddler or any web developer tool in the browser.
6. Choose to browse to "My account" and then "change my email address"
7. Go ahead and put in another email address and click the button.
8. Look at your developer tools and notice that an Ajax request was sent from the browser to the server. Examine that request. Note the URL and any parameters that were sent.

This HTTP endpoint is pretty safe. If anyone were to send a manually crafted request to it while not logged in, it would fail because it is behind authentication. That's a good thing! The vulnerability can only be exploited if the victim is genuinely logged in. That's the key to CSRF.

9. Bonus!! If you feel like you have time, try mounting a manual request to that page when you're logged out and watch it fail. Do the same after logging in to make it succeed.

Reading the synchronizer token

We'll implement the synchronizer token pattern to protect our site. First the server side.

10. Open AccountController.cs.
11. Add a property to allow the Ajax endpoint to receive a token. Add this to the ChangeEmailAddressDTO class:
```
public string SynchronizerToken { get; set; }
```
12. Now find the "email" method that handles an HTTP POST request. Shouldn't be too hard to find. ;-)
13. In that method, pull the synchronizer token from session:
```
var sessionToken =
System.Web.HttpContext.Current.Session["SynchronizerToken"].ToString();
```
14. Now make sure they're synchronized. Add this to the email method:
```
if (ChangeEmailObject.SynchronizerToken != sessionToken)
  throw new Exception("Bad token!");
```
15. Run and test. Does it work? Of course not! We haven't put a token in session yet.

Writing the synchronizer tokens

16. Open ChangeEmail.aspx and add a hidden field anywhere on the page to hold the synchronizer token:
```
<asp:HiddenField ID="synchronizerToken" ClientIDMode="Static"
runat="server" />
```

17. ... and send the value in the Ajax call. Add this to the data field in the ajax call:
```
"SynchronizerToken": $('#synchronizerToken').val()
```

Last step. Let's generate the synchronizer token and load it into the hidden field and put it in session.

18. Open the ChangeEmail.aspx.cs code. In the Page_Load event, create the token by generating a random number. Here's some code that will help:
```
var token = (new Random().Next()).ToString();  //Get a random number
this.synchronizerToken.Value = token;          //Put it on the page
Session["SynchronizerToken"] = token;          //Load it into session
```
19. Run and test by browsing to our ChangeEmail.aspx page. Make sure you can still change the password. If it does, move on. If not, work on it until it does.
20. Now for the payoff ... Rerun the attack by visiting the attacker's CSRF page. Does it fail this time?

When you can legitimately change your email but you've hamstrung the CSRF attack, you can be finished. Take a well-earned break!