Unvalidated Redirects and Forwards Lab

Our awesome web site has got an unvalidated redirect vulnerability. Let's locate it and then protect ourselves against it.

Testing the attack

1. Run the site.
2. Go to PackageTracking.aspx.
3. Enter a tracking number and a carrier. Use a real one if you like. It really works.
4. Note that the page is written to accept a carrier and a tracking number in the url.
5. Try to track a package with a url like this:
   ```
   http://localhost:7777/PackageTracking.aspx?Carrier=UPS&TrackingNumber=1Z12345E1512345676
   ```
   Or one like this:
   ```
   http://localhost:7777/PackageTracking.aspx?Carrier=FedEx&TrackingNumber=111111111111
   ```

It should have sent you right to the tracking site.

6. Now note that the page was written to handle generic package tracking sites. Enter some other carriers and see if it works. It may or may not depending on how those sites were written.
7. Put in a carrier like “Yahoo.com”. What happens when you try that carrier?
   ____
8. Now try it with another site. Are you seeing the pattern?
9. Run and test the site putting in various 'bad' urls. Do they let you go wherever you want?
   _____

This is a great example of an unvalidated redirect. Any hacker could tempt one of our customers to navigate to our site and then have them redirected to an evil site. Since they trust our domain, they'll likely fall for the hacker's trap.

Fixing the problem on the client side

Our customers deserve better than that. Let's help them out by going back in and prevent the attack by changing the textbox to a dropdown, and whitelisting the destination pages.

10. Open PackageTracking.aspx. Find the textbox where the user enters the carrier.
11. Change the page to use a dropdownList instead of that textbox. The dropdownlist should only have approved redirects. Here's some code that can go in PackageTracking.aspx:
    ```
    <asp:DropDownList ID="ddlCarriers" runat="server">
        <asp:ListItem Text="Select One ..." />
        <asp:ListItem Text="UPS" Value="ups" />
        <asp:ListItem Text="FedEx" Value="fedex" />
        <asp:ListItem Text="U.S. Postal Service" Value="usps" />
    </asp:DropDownList>
    ```
12. Run and test.

Changing to a dropdownList doesn't completely fix the problem, because these kinds of values can still be tampered with. It does help because attackers can no longer do an unvalidated redirect by enticing someone to type into the textbox, but they can still do their trickery via a querystring.

Fixing the problem on the server side

Let's fix it on the server-side and not just on the client-side.

13. Find the order class in the Core project. Locate the GetPackageTrackingUrl static method.
14. In this method, make it accept a limited number of inputs; if it isn't one of UPS, FedEx, and USPS, then have it throw an exception:
    ```
    throw new Exception("Bad carrier. Try one of UPS, FedEx, or USPS.");
    ```
15. You'll have to handle this exception in the web page itself. So go back to PackageTracking.aspx.cs and add exception handling wherever you're calling Order.GetPackageTrackingUrl(). Do something like this:
    ```
    try
    {
      Response.Redirect(
        Order.GetPackageTrackingUrl(_carrier, _trackingNumber));
    }
    catch (Exception ex)
    {
      lblErrorMessage.Text =
        string.Format("An error has occurred: {0}", ex.Message);
    }
    ```
16. Run and test. You should not be able to go to any unapproved pages.

Once your page still works but doesn't allow unauthorized redirects, you can be finished.