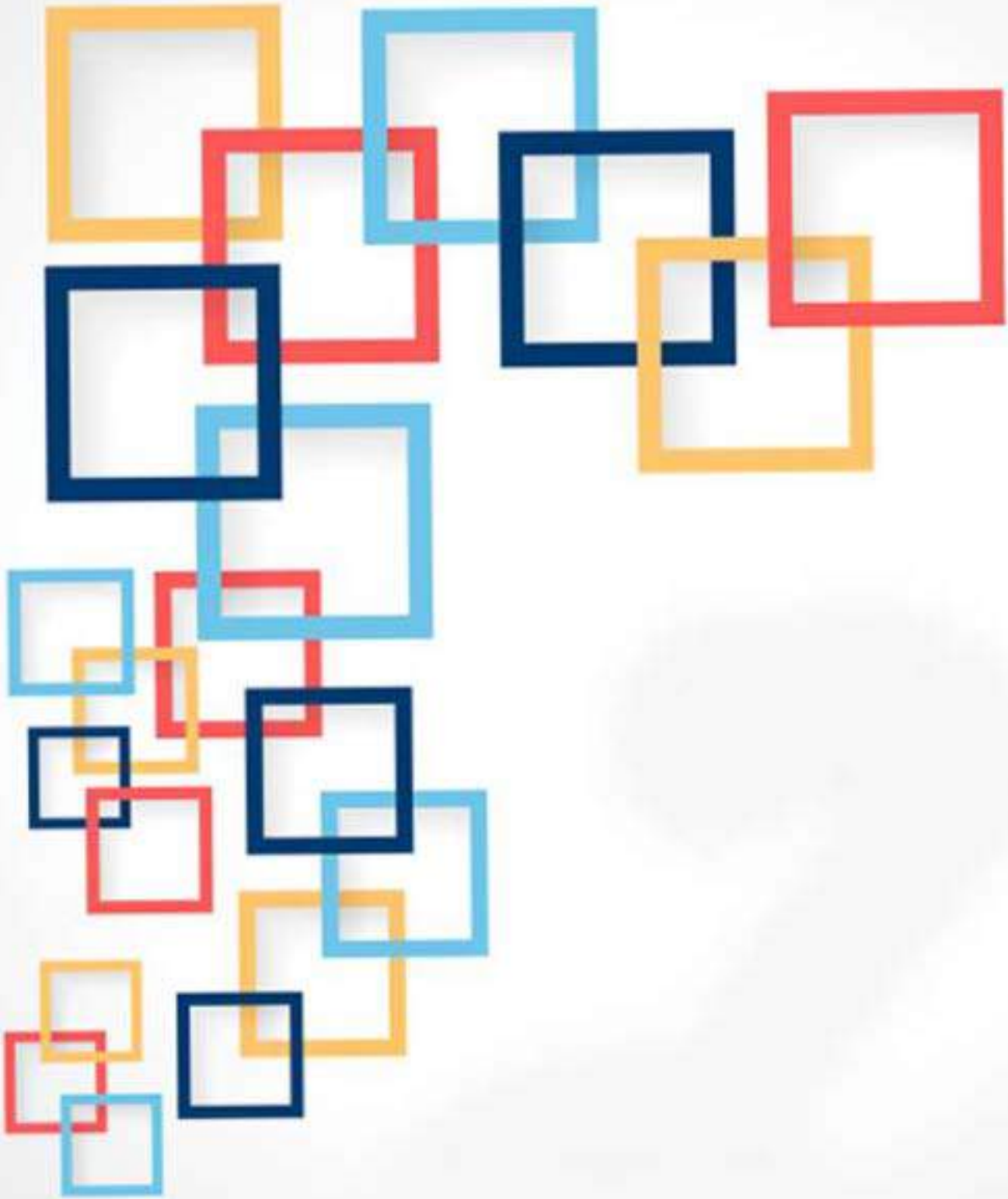
10 Points to Secure Your ASP.NET MVC Applications



Saineshwar Bageri

In case you are new to MVC I would suggest to start from this YouTube Tutorial

Click Here

INDEX

10 Points to Secure Your ASP.NET MVC Applications

INTRODUCTION

The Web is not a secure world here we need to take safe measure to make application Secure. There is good thought on this “Security should be baked from starting of new application else it will be costly”. I have seen that most people develop their application first then try to make it secure this cause a problem at some point we need to rewrite entire code this lead to bugs in application if you are adding security from start then you won’t need run again for having look on entire project is some loopholes open that attacker can attack.

Points that can Prevent Application from Attack.

1. Security Misconfiguration (Error Handling Must Setup Custom Error Page)
2. Cross-Site Request Forgery (CSRF)
3. Cross-Site Scripting (XSS) attacks
4. Malicious File Upload
5. Version Discloser
6. SQL injection attacks
7. Sensitive Data Exposure (Storing sensitive data in Encrypted form)
8. Audit trail
9. Broken authentication and session management
10. Unvalidated Redirects and Forwards.

- **Security Misconfiguration (Error Handling Must Setup Custom Error Page)**

In this kind of attack the hacker intercept Form which is submitted by User and change values and then send it to the server if you are now validating proper input then it can lead to error and leak some information to the hacker.

Example:-

For showing demo I have created an Employee form which takes basic Employee details.

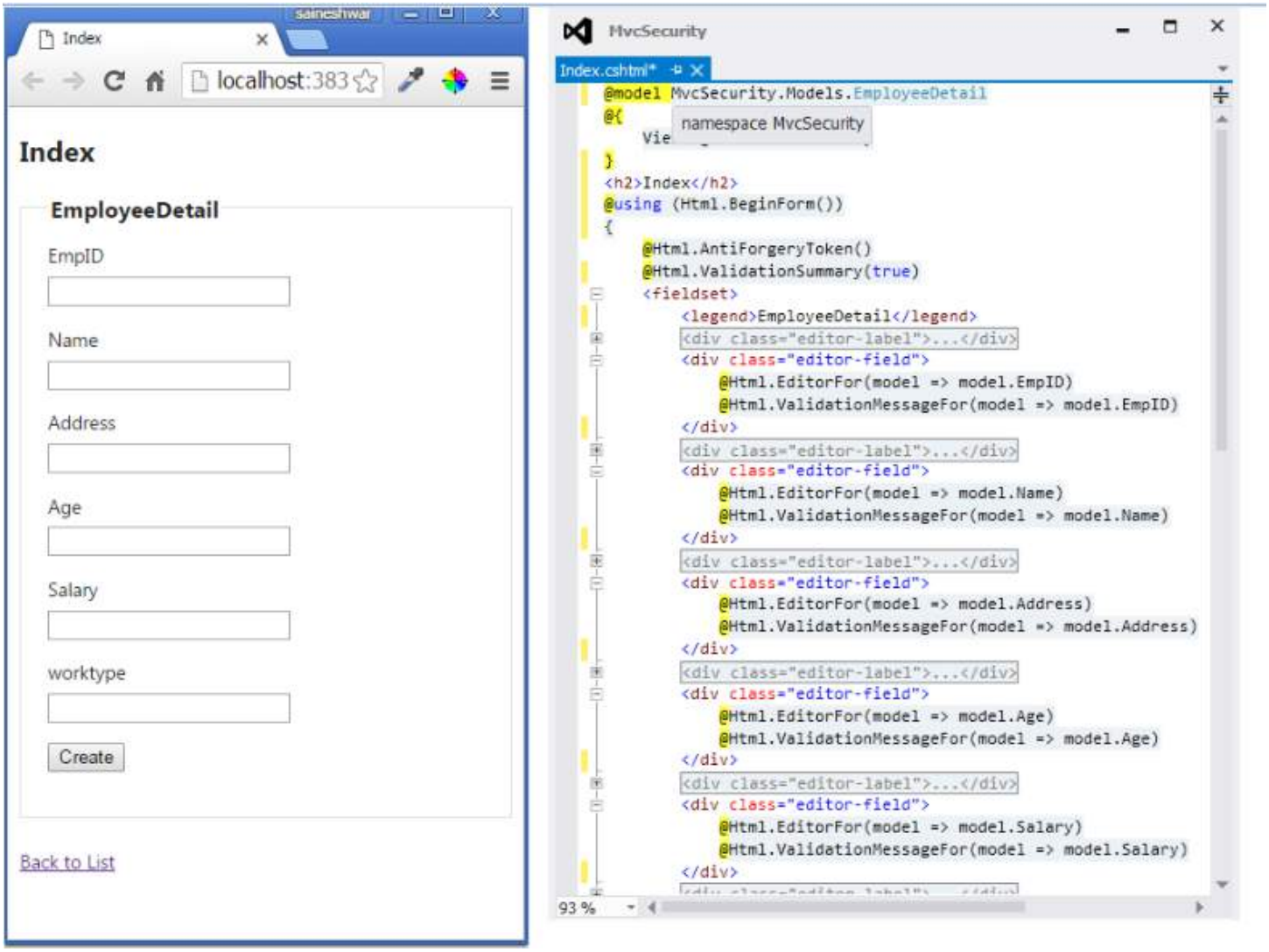


Fig 1. Add Employee form View in the browser along with Markup.

EMPLOYEE DETAIL MODEL VIEW.

```
namespace MvcSecurity.Models
{
    using System;
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;
    public partial class EmployeeDetail
    {
        public int EmpID { get; set; }
        [Required(ErrorMessage="Enter Name")]
        public string Name { get; set; }
        [StringLength(50)]
        [Required(ErrorMessage = "Enter Address")]
        public string Address { get; set; }
        [Required(ErrorMessage = "Enter Age")]
        public Nullable<int> Age { get; set; }
        [Required(ErrorMessage = "Enter Salary")]
        public Nullable<decimal> Salary { get; set; }
        [Required(ErrorMessage = "Enter worktype")]
        public string worktype { get; set; }
    }
}
```

Fig 2. EmployeeDetail Model.

Wow, we have a secured page now we have used Validation, AntiForgeryToken is that enough for securing page.

No that enough for securing page I will show you a small demo of intercept.

Below snapshot shows that address field is validating it ask for only 50 characters.

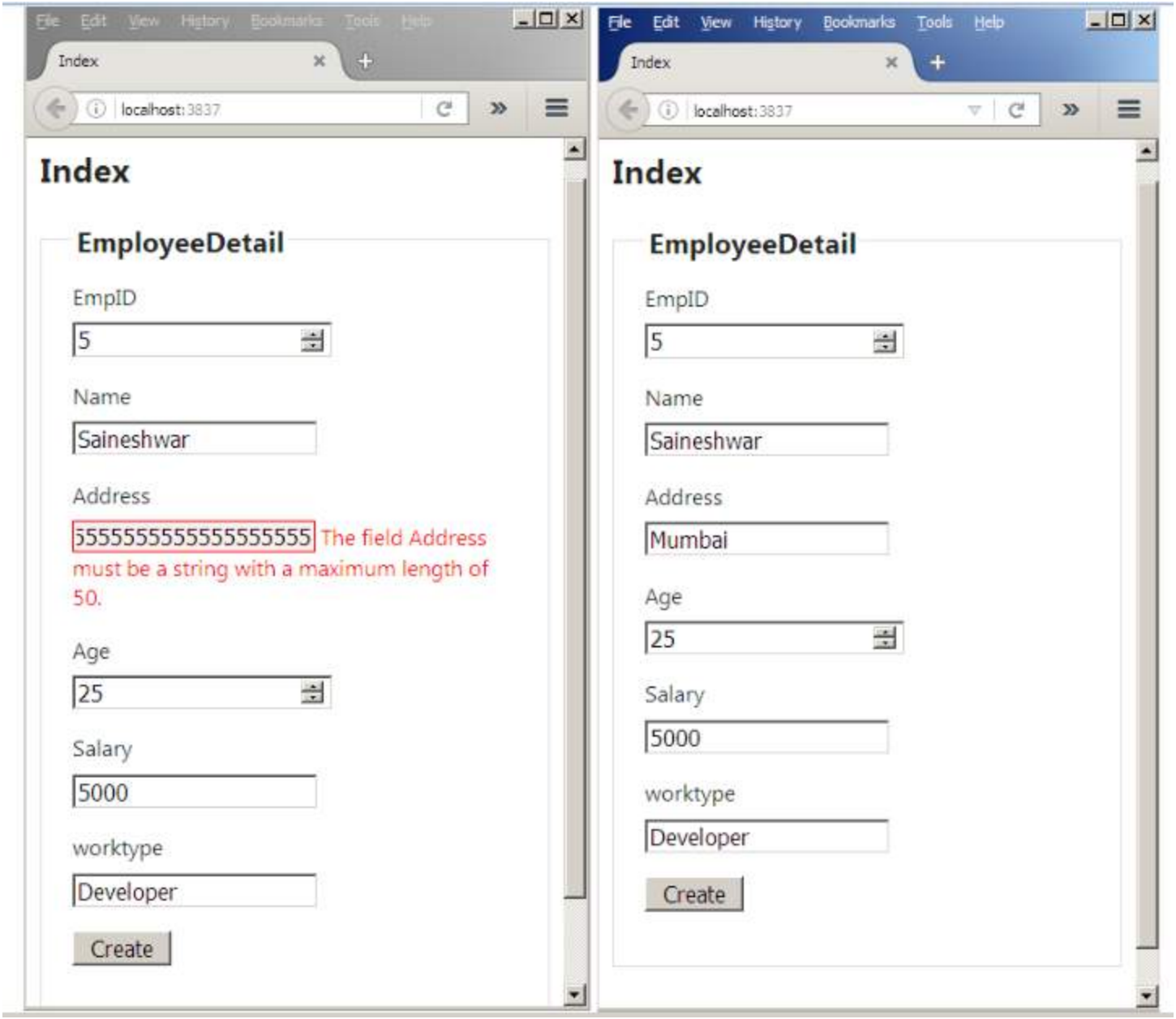


Fig 3.After adding validation to Model now we are validating form you can see Address input field which only accepting 50 characters it is not accepting more than 50 characters it showing Error and message .

INTERCEPTING ADD EMPLOYEE VIEW

Now let's intercept this form and then submit to the server for intercept I am using a tool called as burp suit which catches you request which is going to the server and coming from the server.

Below snapshot I have caught a request which is going to the server.

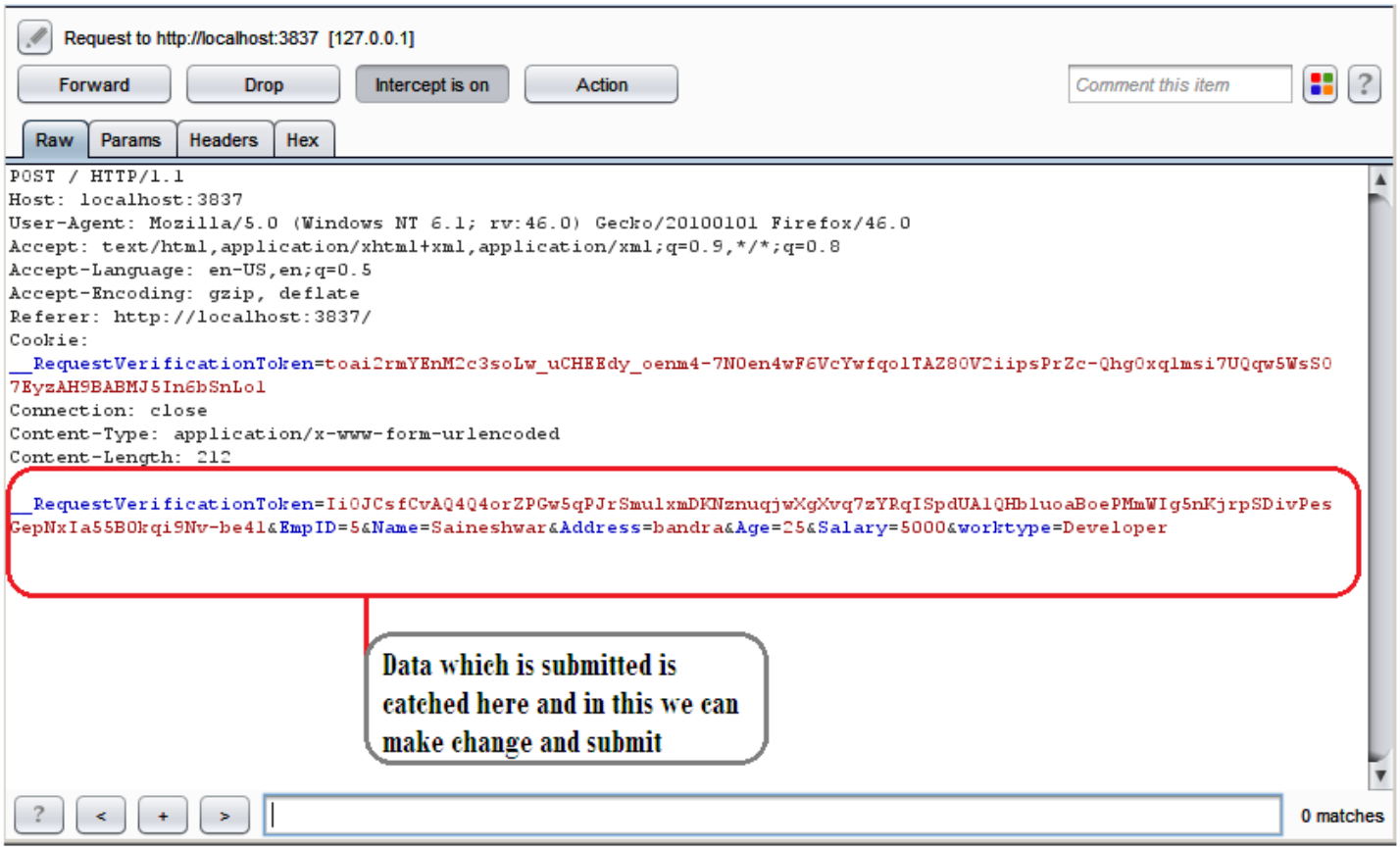


Fig 4.Intercepting Add Employee form using burp suite you can view that if the user submits the form then it catches data in this tool.

Below snapshot the request which I have caught is going to the server I have changed Address which is only taking 50 I have added more than 50 characters and then submitted to the server.

INTERCEPTED ADDRESS FIELD OF ADD EMPLOYEE FORM

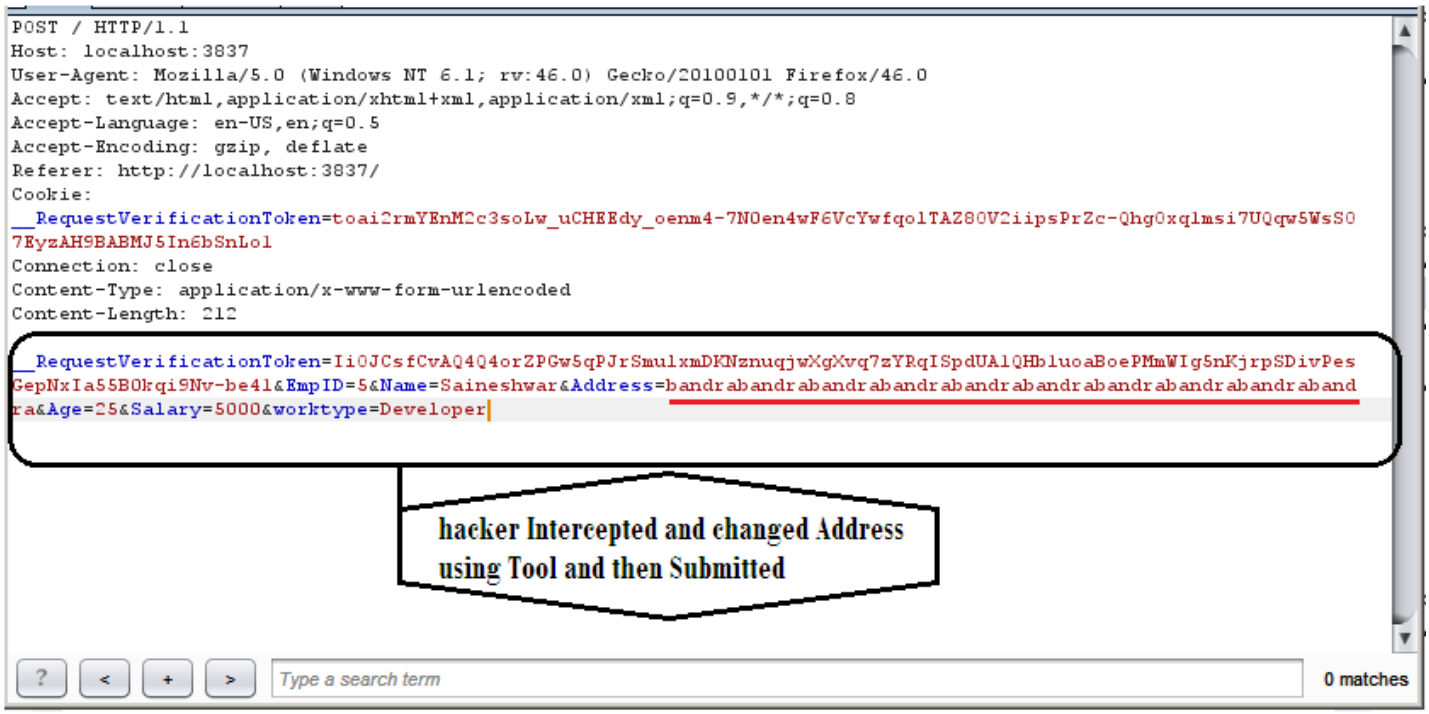


Fig 5.Intercepted Address field in burp suite and submitted to the server.

Below snapshot which shows the request which is submitted to the server which has more than 50 characters.

DEBUGGING MODE OF EMPLOYEE FORM

After submitting more than 50 characters of Address fields to the server we get an exception.

Because in the database we have taken data type of Address field as varchar(50) and value which we have submitted is greater than 50 characters it causes an exception which is thrown.

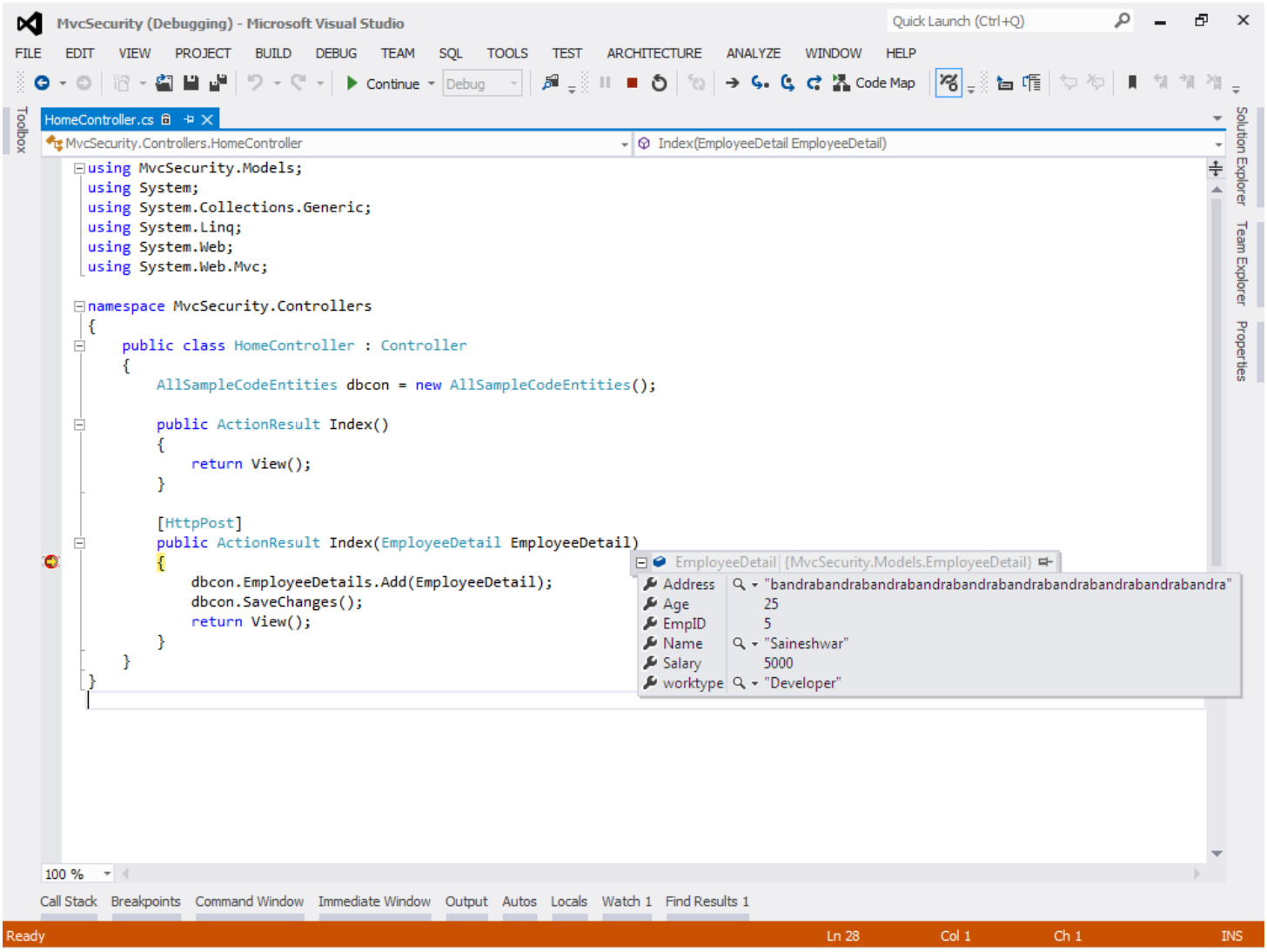


Fig 6.Intercepted Address field in burp suite and submitted to the server.

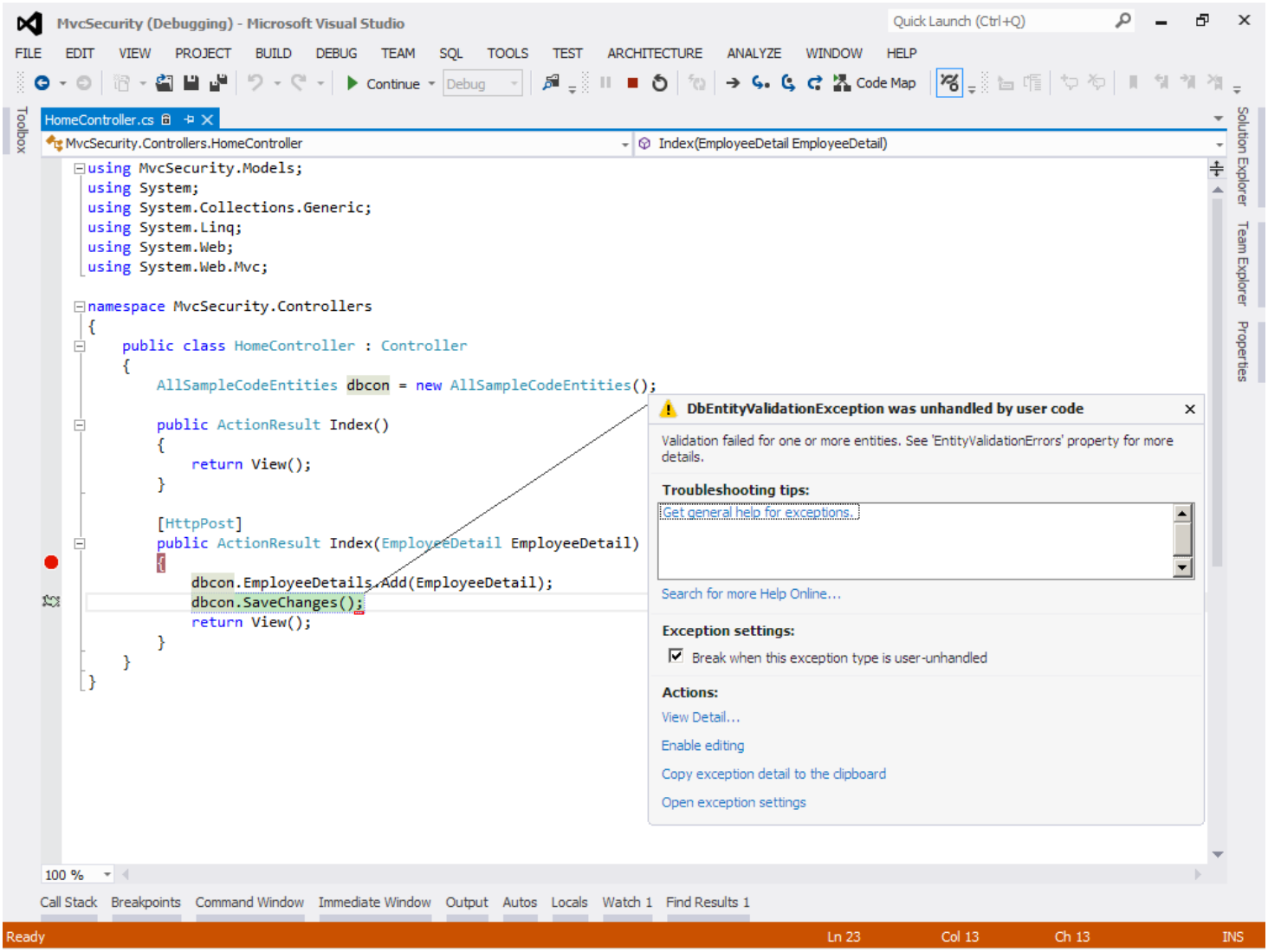


Fig 7.Interception of Address field caused an error.

PROBLEM WITH DISPLAYING ERROR OCCURRED TO USERS

Now exception which occurs is directly Displaying to User or Attacker which lets to leak of valued information to Attacker (Hacker). Using this error information he can try various permutation and combination to exploit the system.

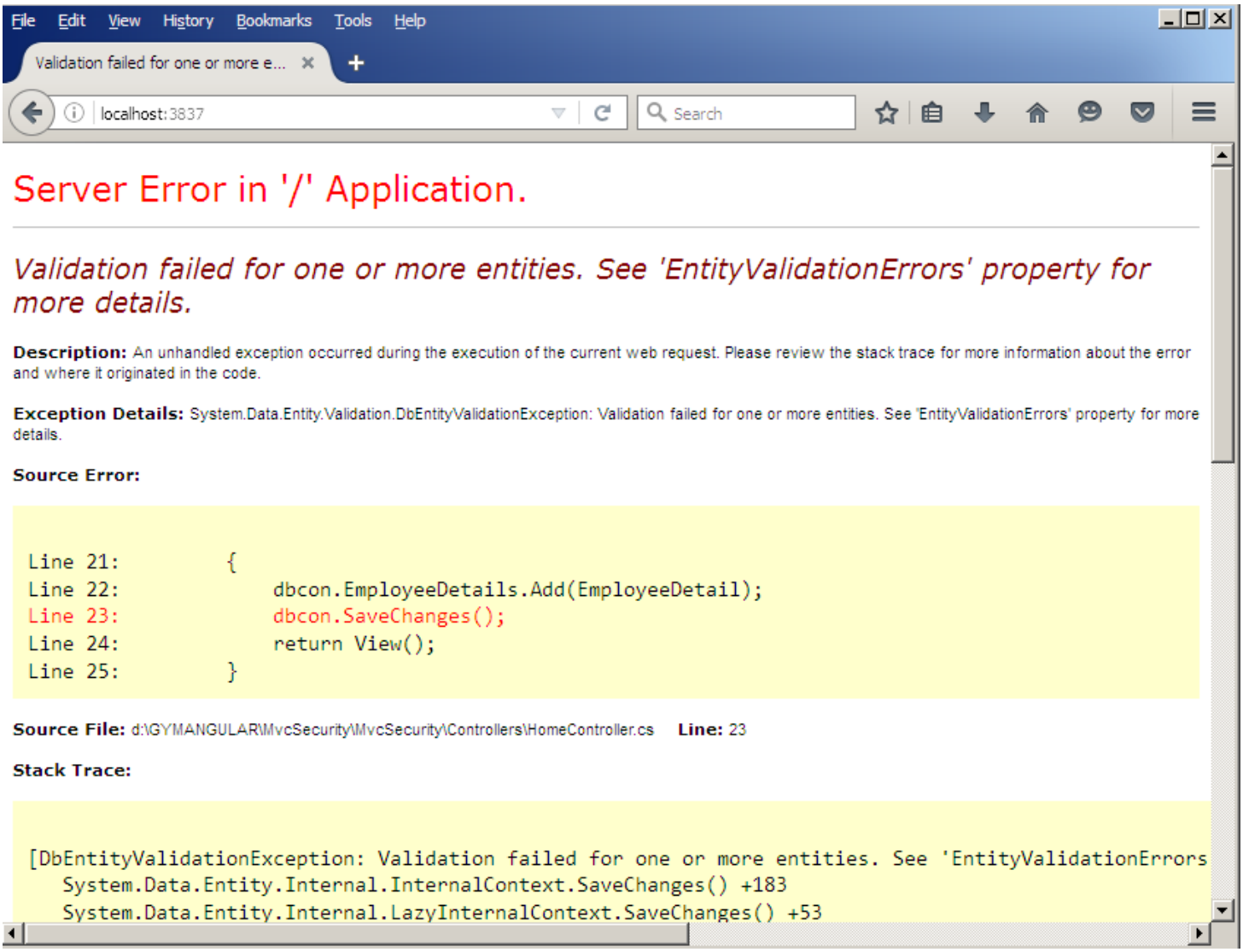


Fig 8.Displaying Error directly to Users.

Solution: -

Whenever any error occur we are not going to show it to the user we just going to show Error page this will hide to leak of valued information of our application its solution for it.

There are 2 solutions for hiding these details to Users.

1. Create a custom Error handling Attribute.
2. Setting Custom Error page from Web.config file

Solution 1:-

Create a custom Error handling Attribute using **HandleError Attribute** or using **IExceptionFilter** Filter

Showing example using **HandleErrorAttribute**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web.Mvc;

namespace MvcSecurity.Filters
{
    public class CustomErrorHandler : HandleErrorAttribute
    {
        public override void OnException(ExceptionContext filterContext)
        {
            Exception e = filterContext.Exception;
            filterContext.ExceptionHandled = true;
            var result = new ViewResult()
            {
                ViewName = "Error"
```

```
            }; ;
            result.ViewBag.Error = "Error Occur While Processing Your Request Please Check After Some
Time";
            filterContext.Result = result;
        }
    }
```

After creating custom Error attribute we need to apply this globally for entire application. For doing this we need to call this attribute in **FilterConfig** class which is in App_Start Folder as show below.

```
using MvcSecurity.Filters;
using System.Web;
using System.Web.Mvc;

namespace MvcSecurity
{
    public class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new CustomErrorHandler());
        }
    }
}
```

Whenever error occurs the CustomErrorHandler attribute will get called and it will redirect to Error.cshtml page. And any message you want to pass then you can pass through `@ViewBag.Error` from CustomErrorHandler attribute.

Html Error Page code snippet

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Error</title>
</head>
<body>
    <hgroup>
        <h1 class="text-danger">Error.</h1>
        <h2>@ViewBag.Error</h2>
        <h2></h2>
    </hgroup>
</body>
</html>
```

Error Page View

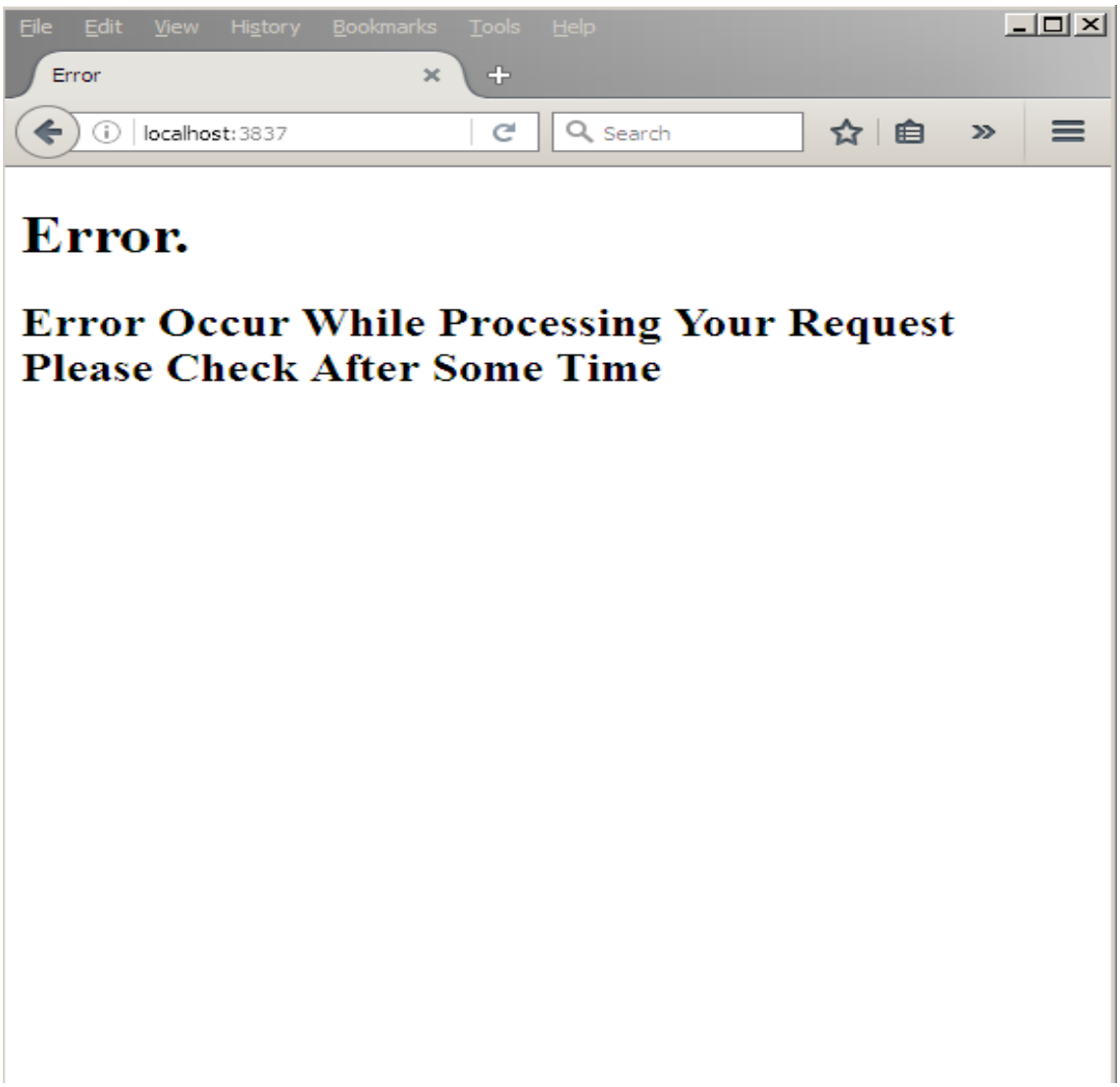


Fig 9.Displaying Custom Error page if any error occurs in Application.

Solution 2:-

1) Setting Custom Error page from Web.config file

If you do not want to write attribute then you can set Custom Error page in Web.config file.

Before doing that just create a simple Error page in HTML for displaying if any error occurs.

In Web.config file there is system.web tag inside that add Custom error tag as show below.

```
<customErrors mode="On" defaultRedirect="~/Error/Error.html" />
```

```
<system.web>
  <httpRuntime targetFramework="4.5" />
  <compilation debug="true" targetFramework="4.5"><assemblies>
    <add assembly="System.Data.Entity, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </assemblies></compilation>
  <authentication mode="Forms">
    <forms loginUrl="~/Account/Login" timeout="2880" />
  </authentication>
  <pages>...</pages>
  <profile defaultProvider="DefaultProfileP">...</profile>
  <membership defaultProvider="DefaultMembersh">...</membership>
  <roleManager defaultProvider="DefaultRoleProv">...</roleManager>
  ...
  <sessionState mode="InProc" customProvider="DefaultSessionProvider">
    <providers>
      <add name="DefaultSessionProvider"
       type="System.Web.Providers.DefaultSessionStateProvider, System.Web.Providers, Version=1.0.0.0, Culture=neutral
       connectionStringName="DefaultConnection" />
    </providers>
  </sessionState>
  <customErrors mode="On" defaultRedirect="~/Error/Error.html" />
</system.web>
```

Fig 10. Web.config File View while setting Custom Error page.

Html Error Page code snippet

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Error</title>
</head>
<body>
    <hgroup>
        <h1 class="text-danger">Error.</h1>
        <h2>An error occurred while processing your request.....</h2>
        <h2></h2>
    </hgroup>
</body>
</html>
```

Html Error Page View

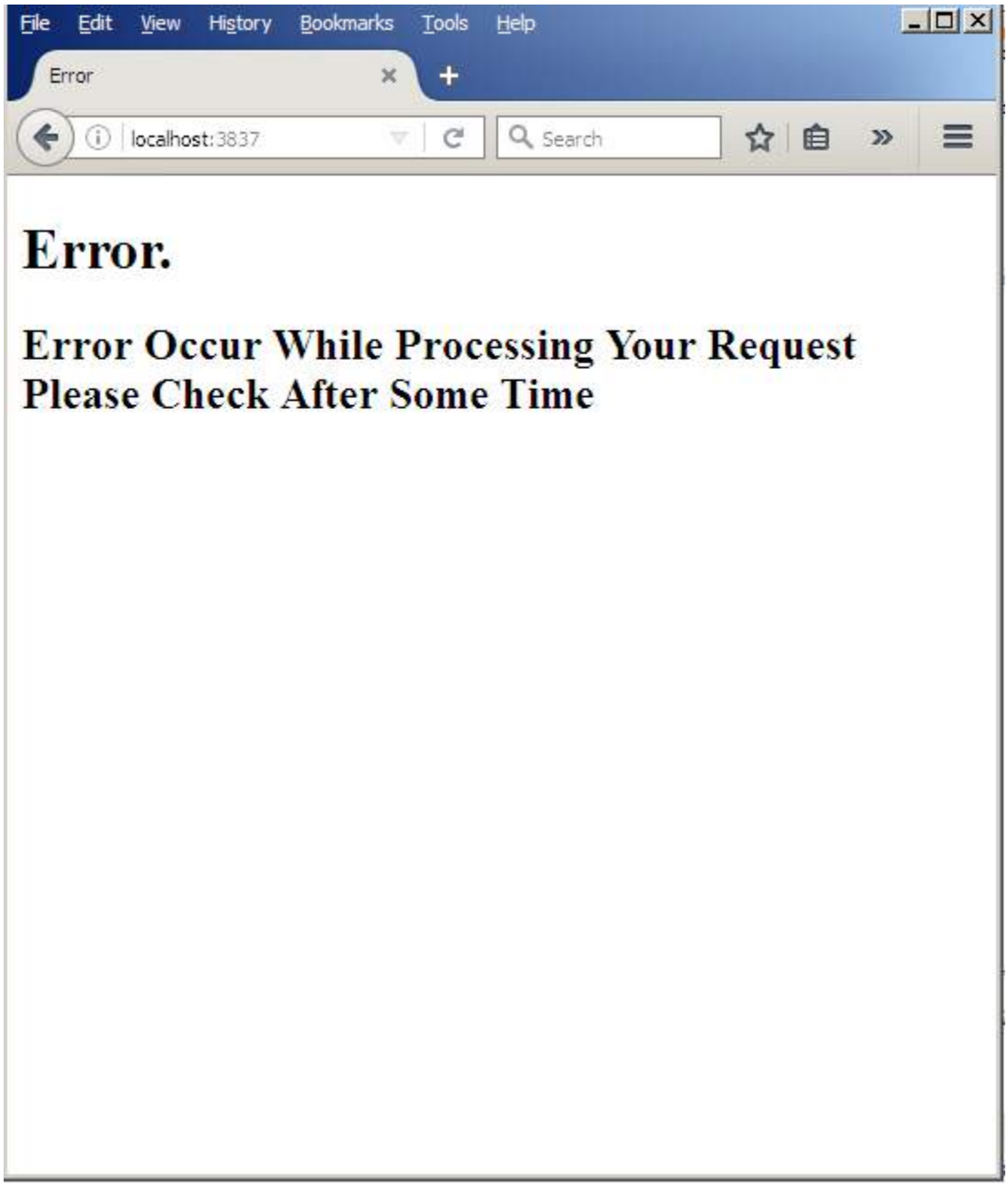


Fig 11.Displaying Custom HTML Error page if any error occurs in Application.

Wow, we are started Securing application step by step.

- **Cross-Site Request Forgery (CSRF)**

Cross Site Request Forgery is an attack in which User logs into his bank account then after login a session is sent to User browser and in meanwhile hacker will send an Email which contains Malicious link this Mail will look similar to Bank Email and User Click on this Malicious link this Link will connect to Bank session and User Account will be hacked and meanwhile money from User account will be Transferred to hacker account.

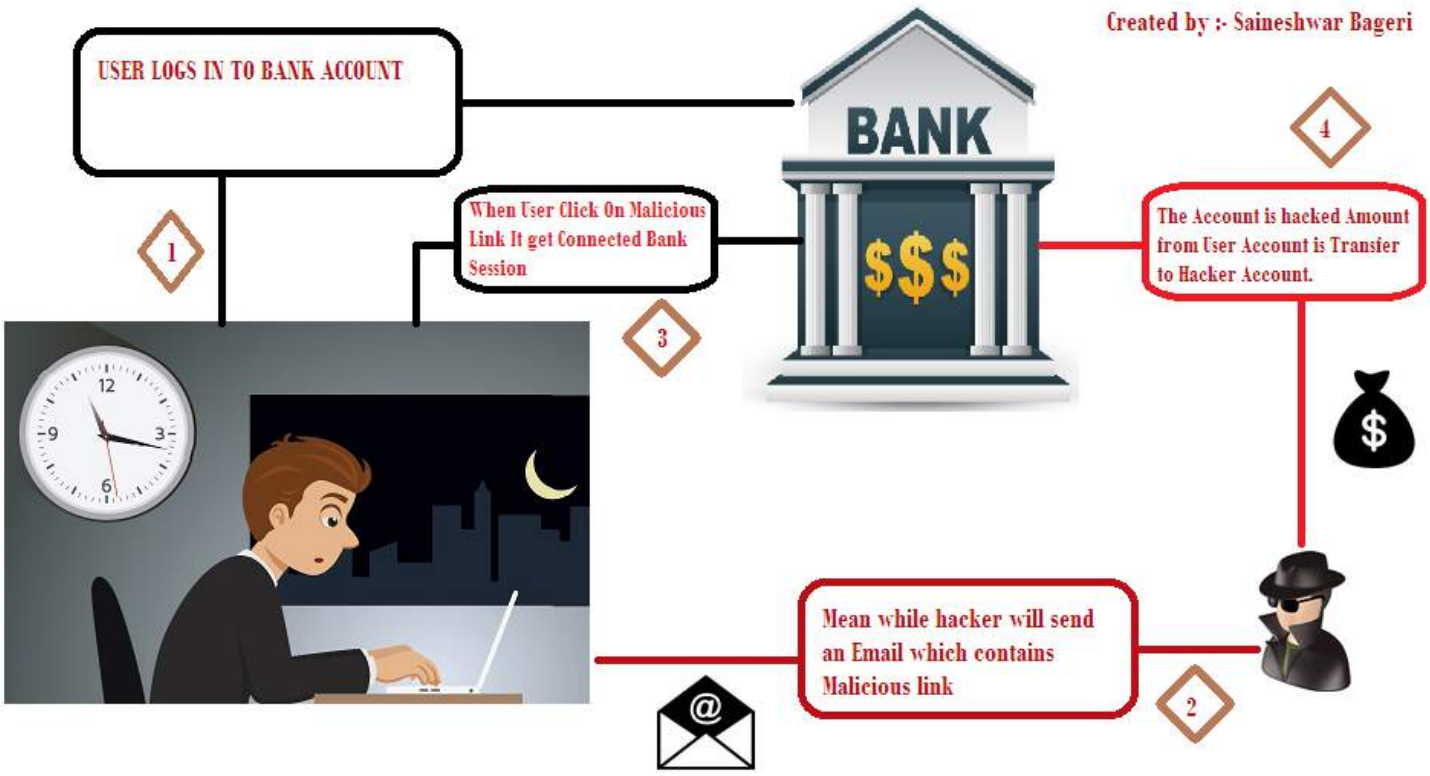


Fig 1.CSRF Attack.

Microsoft has recognized this threat and for preventing it has provided **AntiForgeryToken**.

Solution:-

We need to add @**Html**.**AntiForgeryToken**() helper to your form inside form tag and on Action Method which handles your post ([**HttpPost**])Request we need to [**ValidateAntiForgeryToken**] which will check the form which is submitted is having Token or not if not then it will show an error.

ADDING [ANTIFORGERYTOKEN] HELPER TO VIEW

Index.cshtml*

```
@model MvcSecurity.Models.EmployeeDetail
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>

@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>EmployeeDetail</legend>
        <div class="editor-label">
            @Html.LabelFor(model => model.EmpID)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.EmpID)
            @Html.ValidationMessageFor(model => model.EmpID)
        </div>
        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>
        <div class="editor-label">
            @Html.LabelFor(model => model.Address)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Address)
            @Html.ValidationMessageFor(model => model.Address)
        </div>
        <div class="editor-label">
```

Fig 2.Adding AntiForgeryToken on View.

Adding [**ValidateAntiForgeryToken**] Attribute to HttpPost **[HttpPost**] Method.

HomeController.cs* Index.cshtml*

MvcSecurity.Controllers.HomeController — Index(EmployeeDetail EmployeeDetail)

```
using MvcSecurity.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcSecurity.Controllers
{
    public class HomeController : Controller
    {
        AllSampleCodeEntities dbcon = new AllSampleCodeEntities();

        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Index(EmployeeDetail EmployeeDetail)
        {
            dbcon.EmployeeDetails.Add(EmployeeDetail);
            dbcon.SaveChanges();
            return View();
        }
    }
}
```

Fig 3.Adding ValidateAntiForgeryToken on [HttpPost] Method (Index).

WORKING OF ANTIFORGERYTOKEN

When we add this AntiForgeryToken helper on View it creates a hidden field and assign a unique token value to it and meanwhile a session Cookie is added to the browser. When we post form ASP.NET MVC framework check for **__RequestVerificationToken** Hidden field and **__RequestVerificationToken** Cookie are present or not. If either the cookie or the form **__RequestVerificationToken** Hidden field values are missing, or the values don't match, ASP.NET MVC does not process the action. This is how we can prevent cross-site request

forgery attack in asp.net MVC.

REQUESTVERIFICATIONTOKEN ON VIEW SNAPSHOT.

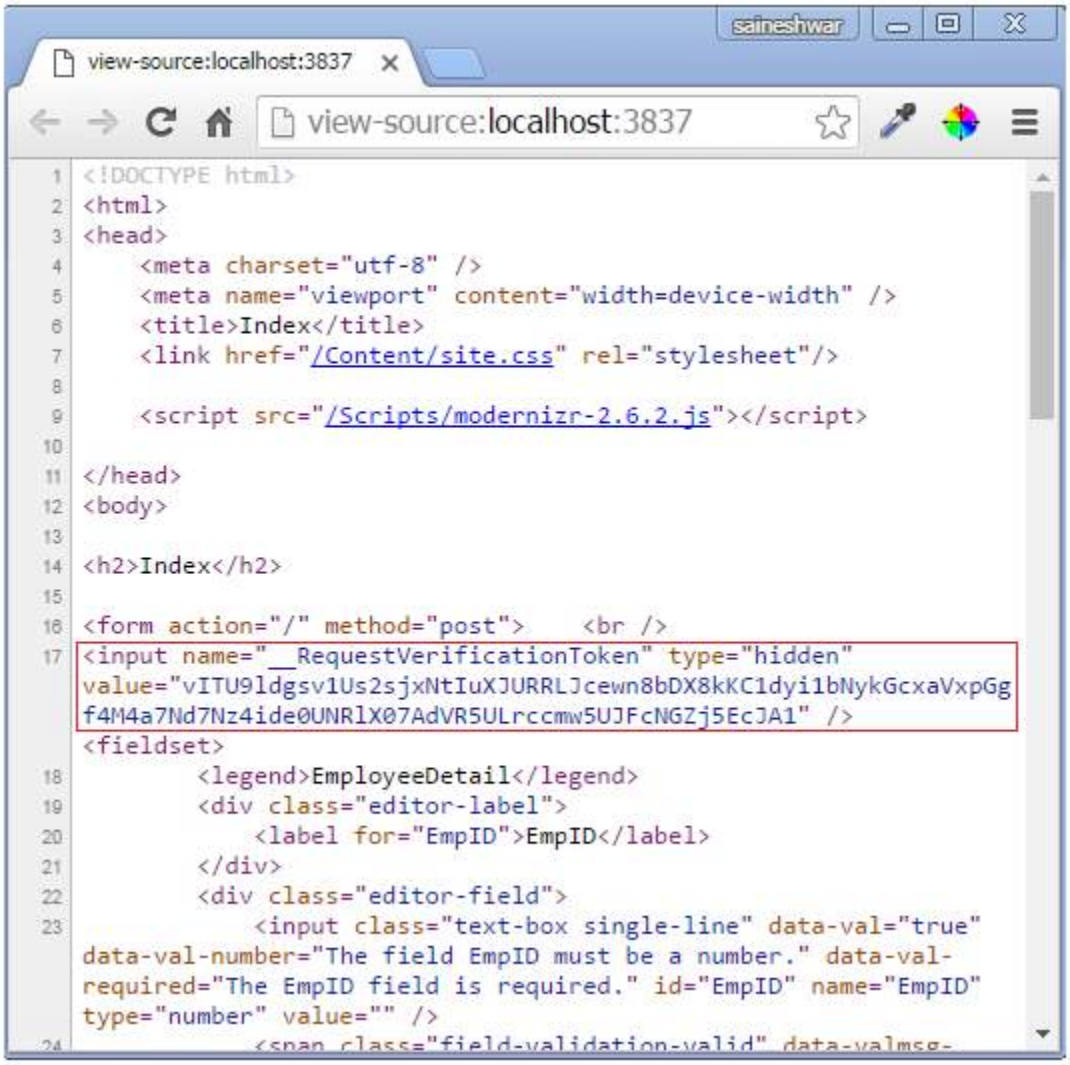


Fig 4. RequestVerificationToken generated the hidden field.

REQUESTVERIFICATIONTOKEN COOKIE SNAPSHOT.

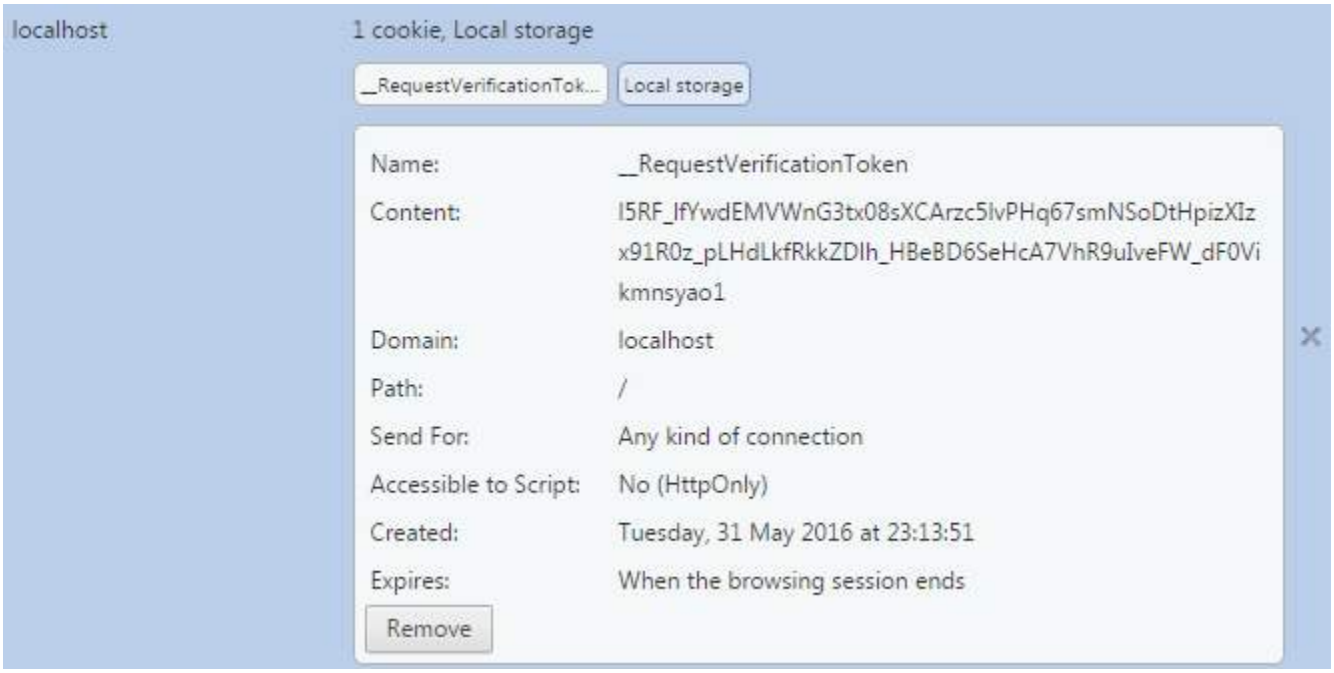


Fig 5. RequestVerificationToken generated a cookie.

- **Cross-Site Scripting (XSS) attacks**

Cross-site Scripting (XSS) is an attack in which malicious scripts is injected via input fields this attack is most common and allow an attacker to steal credentials and valuable data that can lead to a big security breach.

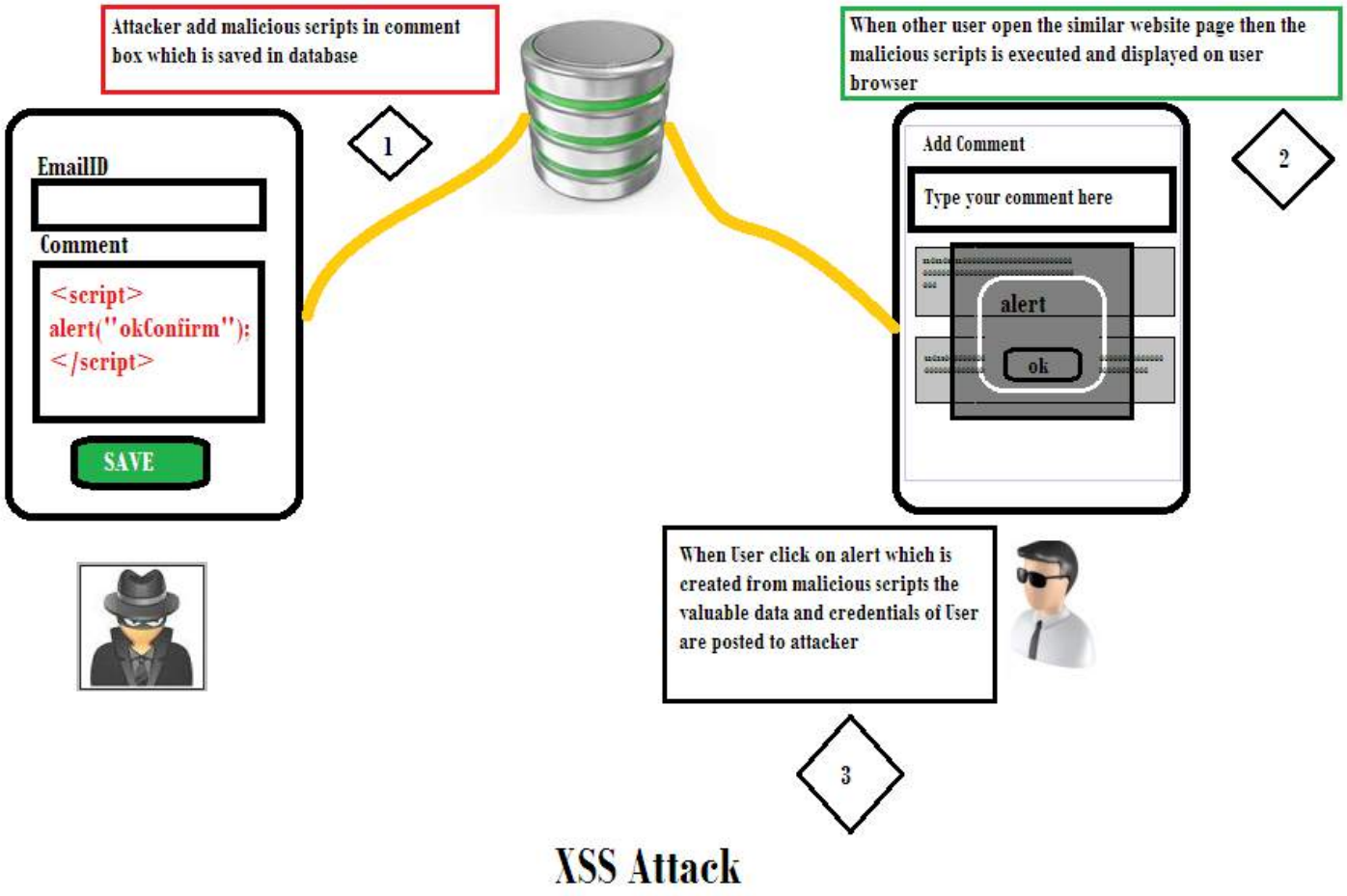


Fig 1. Cross Site Scripting (XSS).

In this attack attacker visits a website add malicious scripts in form comment in comment box and while website is not secured which take this input and save in database, meanwhile another user who ever visit that similar site this time the comment which attacker has added gets executed and displayed to User think it is and website alert and clicks on it the code behind this alert (malicious scripts) get its work done (steal credentials or

valuable data) and that can lead to big security issue.

Below is simple Employee form which we are trying to save data which contains Script tag which is not allowing us to save data but in show an error.

It prevents submit HTML for avoiding Cross Site Scripting attack.

Error Displayed

A potentially dangerous Request.Form value was detected from the client (worktype="<script>alert('hi');").

This error occurs because our form is validating data which is entered by User and User has entered (<script>alert('hi');</script>) in input field which causes this error.

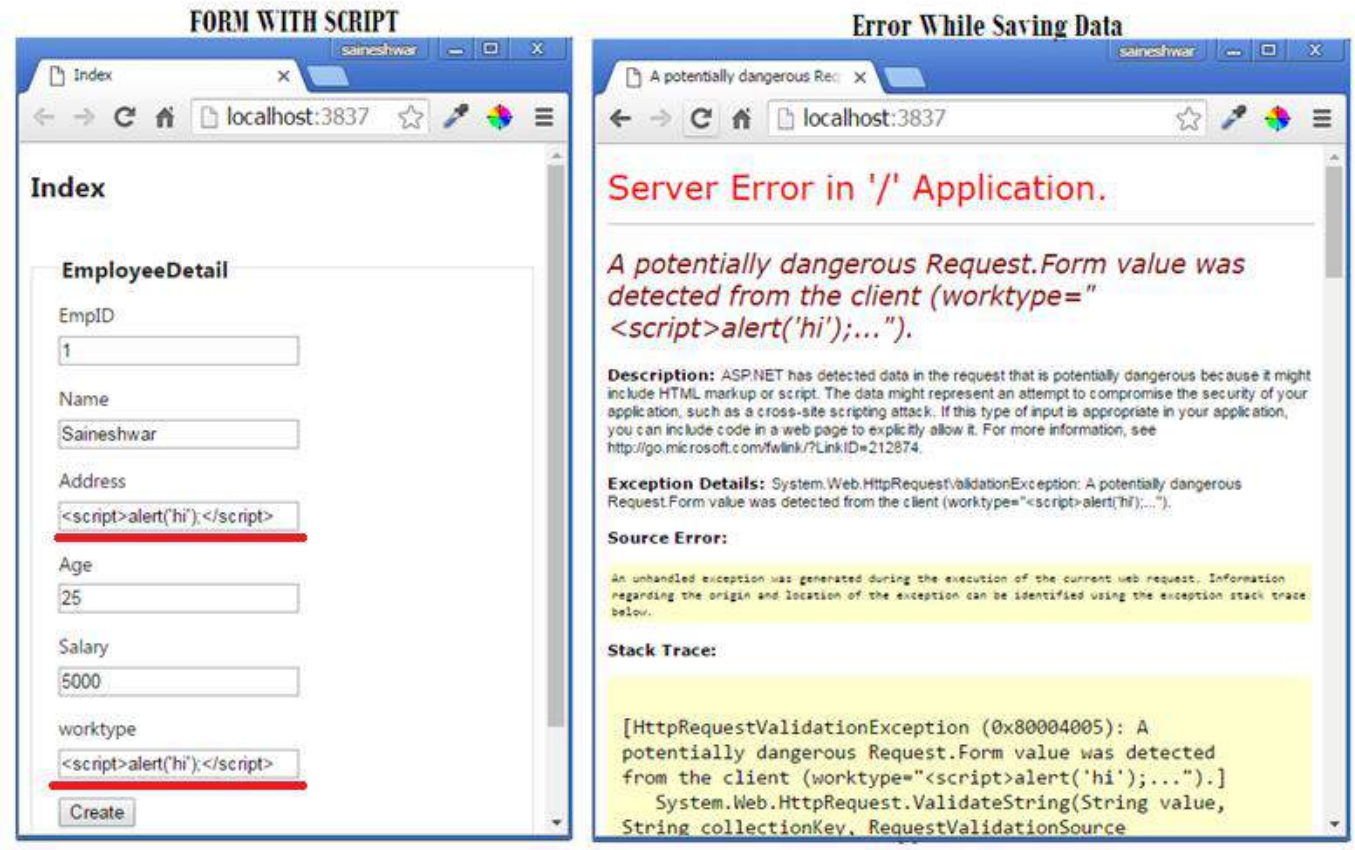


Fig 2. Submitting Malicious scripts in Input fields which lead to Error.

Solution to this problem is validating all input Fields.

Solution: -

1. **[ValidateInput(false)]**
2. **[AllowHtml]**
3. **[RegularExpressionAttribute]**
4. **AntiXSS Library**

Solution 1:-

ValidateInput

[ValidateInput] is an attribute which can be used on Controller or Action Method which will validate input. If we want Mark up to be allowed then we need to set **enableValidation** Properties to False **([ValidateInput(false)])** which will not validate input if we set to true then **[ValidateInput(true)])** It will validate input. In the same way, if you apply it on Controller then it applies to entire action methods inside the controller and if you apply it on Action Method then it will only be specific to that action method.

But ValidateInput attribute will apply to all properties of Model (EmployeeDetails**).**

Snapshot of Applying **ValidateInput Attribute** on HttpPost Method.

```
[HttpPost]
[ValidateInput(false)]
[ValidateAntiForgeryToken]
public ActionResult Index(EmployeeDetail EmployeeDetail)
{
    dbcon.EmployeeDetails.Add(EmployeeDetail);
    dbcon.SaveChanges();
    return View();
}
```

Fig 3. Applying ValidateInput Attribute on HttpPost Method.

Snapshot after Applying **ValidateInput Attribute**

```
[HttpPost]
[ValidateInput(false)]
[ValidateAntiForgeryToken]
public ActionResult Index(EmployeeDetail EmployeeDetail)
{
    dbcon.EmployeeDetails.Add(EmployeeDetail);
    dbcon.SaveChanges();
    return View();
}
```

EmployeeDetail {MvcSecurity.Models.EmployeeDetail}

Address	"<script>alert('hacked')</script>"
Age	25
EmpID	1
Name	"<script>alert('hacked')</script>"
Salary	50000
worktype	"<script>alert('hacked')</script>"

Fig 4.After adding ValidateInput Attribute on HttpPost Method it allows submitting script.

Solution 2:-

AllowHtml

[AllowHtml] attributes that is applied to Model properties such that it will not validate that particular Model property on which **AllowHtml** attribute is added. This allows submitting HTML for avoiding Cross Site Scripting attack.

In below snapshot, I have applied **AllowHtml** attribute on Address property of EmployeeDetails Model.

```
public partial class EmployeeDetail
{
    public int EmpID { get; set; }
    [Required(ErrorMessage="Enter Name")]
    public string Name { get; set; }

    [StringLength(50)]
    [Required(ErrorMessage = "Enter Address")]

    [AllowHtml]
    public string Address { get; set; }

    [Required(ErrorMessage = "Enter Age")]
    public Nullable<int> Age { get; set; }

    [Required(ErrorMessage = "Enter Salary")]
    public Nullable<decimal> Salary { get; set; }

    [Required(ErrorMessage = "Enter worktype")]
    public string worktype { get; set; }
}
```

Fig 1. Applying [AllowHtml] Attribute on Required Model Property.

After Applying AllowHtml attribute on that Address Property now Address property will not validate and allow HTML to be submitted in this field.

Fig 2. After adding [AllowHtml] Attribute on Address Model Property it allows submitting script.

Solution 3:-

Regular Expression

The third solution to XSS attack is validating all your Fields with Regular Expression such that only valid data can move in.

Use Regular Expression to validate input to protect from XSS attack below is Snapshot.

```
public partial class EmployeeDetail
{
    public int EmpID { get; set; }
    [Required(ErrorMessage = "Enter Name")]
    [RegularExpression("^[A-z]+$", ErrorMessage = "Enter Valid Name")]
    public string Name { get; set; }

    [StringLength(50)]
    [RegularExpression("[a-zA-Z ]+$", ErrorMessage = "Enter Valid Address")]
    [Required(ErrorMessage = "Enter Address")]
    [AllowHtml]
    public string Address { get; set; }

    [RegularExpression("^[0-9]+$", ErrorMessage = "Enter Valid Age")]
    [Required(ErrorMessage = "Enter Age")]
    public Nullable<int> Age { get; set; }

    [RegularExpression("(((\\d+)((\\.\\d{1,2})?))$", ErrorMessage = "Enter Valid Salary")]
    [Required(ErrorMessage = "Enter Salary")]
    public Nullable<decimal> Salary { get; set; }

    [Required(ErrorMessage = "Enter Worktype")]
    [RegularExpression("[a-zA-Z ]+$", ErrorMessage = "Enter Valid Worktype")]
    public string worktype { get; set; }
}
```

Fig 1. Applying Regular Expression to Model Property.

LIST OF REGULAR EXPRESSION TO USE.

Alphabets and Space

[a-zA-Z]+$

Alphabets

^[A-z]+$

Numbers

^[0-9]+$

Alphanumeric

^[a-zA-Z0-9]*$

Email

[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?

Mobile no.

^([7-9]{1})([0-9]{9})$

Date Format (mm/dd/yyyy | mm-dd-yyyy | mm.dd.yyyy **)**

/^(0[1-9]|1[012])[- /.](0[1-9]|[12][0-9]|3[01])[- /.](19|20)\\d\\d+$/

Website URL

^http(s)?://([\\w-]+.)+[\\w-]+(/[\\w- ./?%&=])?$

Credit Card Numbers

Visa

^4[0-9]{12}(?:[0-9]{3})?$

MasterCard

^5[1-5][0-9]{14}$

American Express

^3[47][0-9]{13}$

Decimal number

((\\d+)((\\.\\d{1,2})?))$

Solution 4:-

AntiXSS Library

The fourth solution to XSS attack is by using **Microsoft AntiXSS Library** which will help to protect your application.

Now let's start with install **Microsoft AntiXSS Library** from NuGet just right click on Project then select **Manage NuGet Packages.**

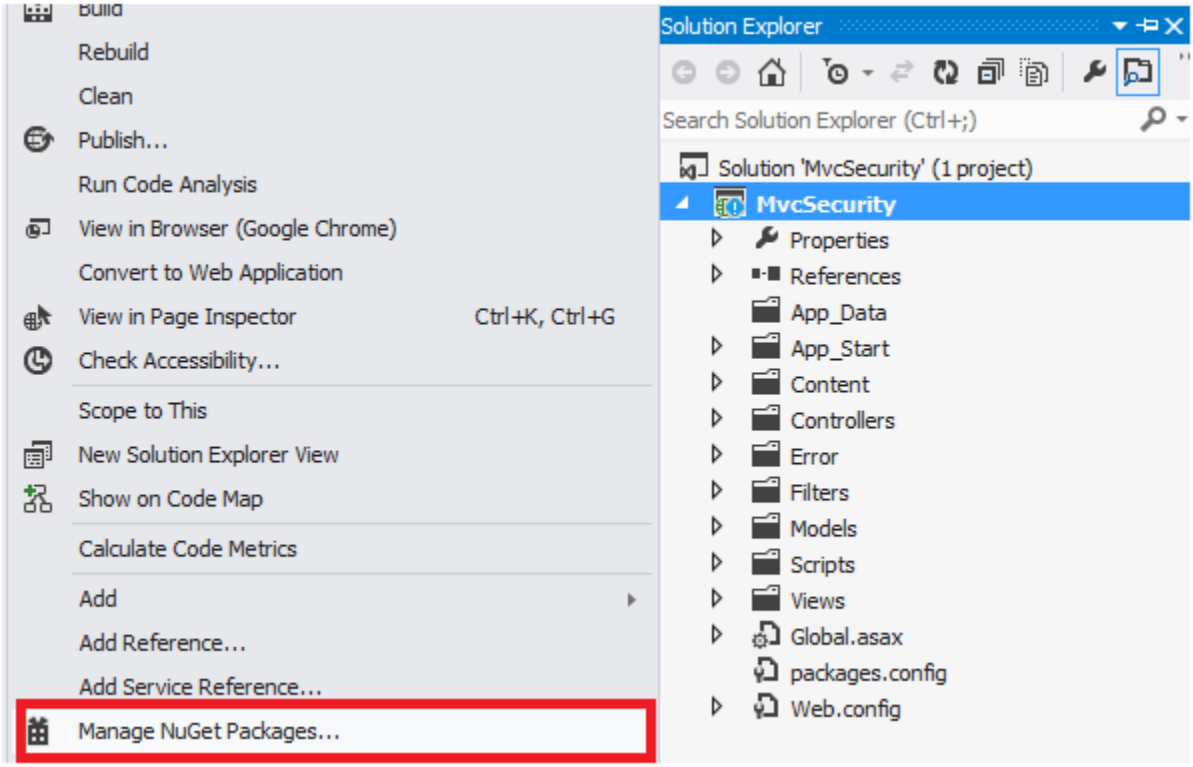


Fig 1.Choosing Manage NuGet Package for adding Package.

After selecting a new dialog will pop up with name **Manage NuGet Packages** in this dialog search **AntiXSS Library** in the search box then choose the first option which is **AntiXSS** click on install button.

Fig 2.Adding Microsoft AntiXSS Library to Project.

REFERENCE ADDED AFTER INSTALLING

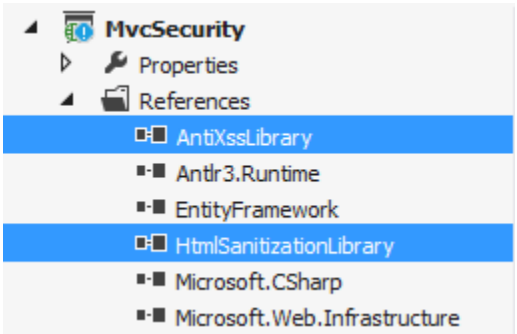


Fig 3.After adding Microsoft AntiXSS Library to Project.

After installing we are going to have a look on how to use AntiXSS Library.

SANITIZER CLASS

```
public static class Sanitizer
{
    public static string GetSafeHtml(string input);
    public static void GetSafeHtml(TextReader sourceReader, Stream destinationStream);
    public static void GetSafeHtml(TextReader sourceReader, TextWriter destinationWriter);
    public static string GetSafeHtmlFragment(string input);
    public static void GetSafeHtmlFragment(TextReader sourceReader, Stream destinationStream);
    public static void GetSafeHtmlFragment(TextReader sourceReader, TextWriter destinationWriter);
}
```

Fig 4. Sanitizer Class which we are going to use for sanitizes inputs.

BELOW SNAPSHOT SHOWS HOW TO USE SANITIZER CLASS METHOD

Sanitizer is a static class we can access anywhere we just need to provide input which field which we need to validate to Sanitizer class method (**`GetSafeHtmlFragment`**) it will check and return you Sanitize string.

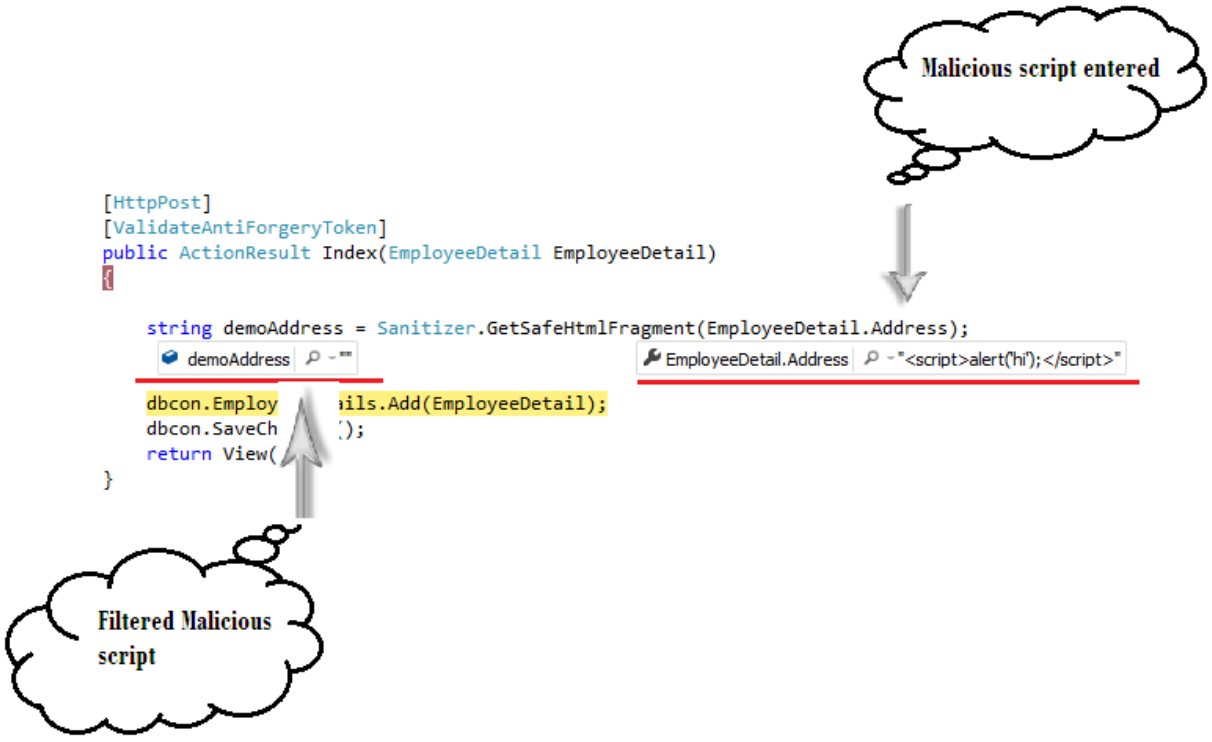


Fig 5.Here we are showing how to sanitize inputs using Sanitizer Class for that I have taken address field as input to sanitize.

We can use this Method to filter malicious script **while saving in the database** and **displaying in the browser**.

Tip: - Use [**ValidateInput**(**false**)] or [**AllowHtml**] before using this **AntiXSS library** else it will throw error “A potentially dangerous Request.Form”

- **Malicious File Upload.**

Till now we have learned how to protect all your input fields from attack but still, we are missing one main field it is File upload control we need to protect from taking invalid input most attackers try to upload a malicious file which may cause a security issue. The attacker can change file extension [tuto.exe to tuto.jpeg] and the malicious script can be uploaded as an image file. The Most of the developer just look on the file extension of the file and save in folder or database but file extension is valid not file it may have a malicious script.

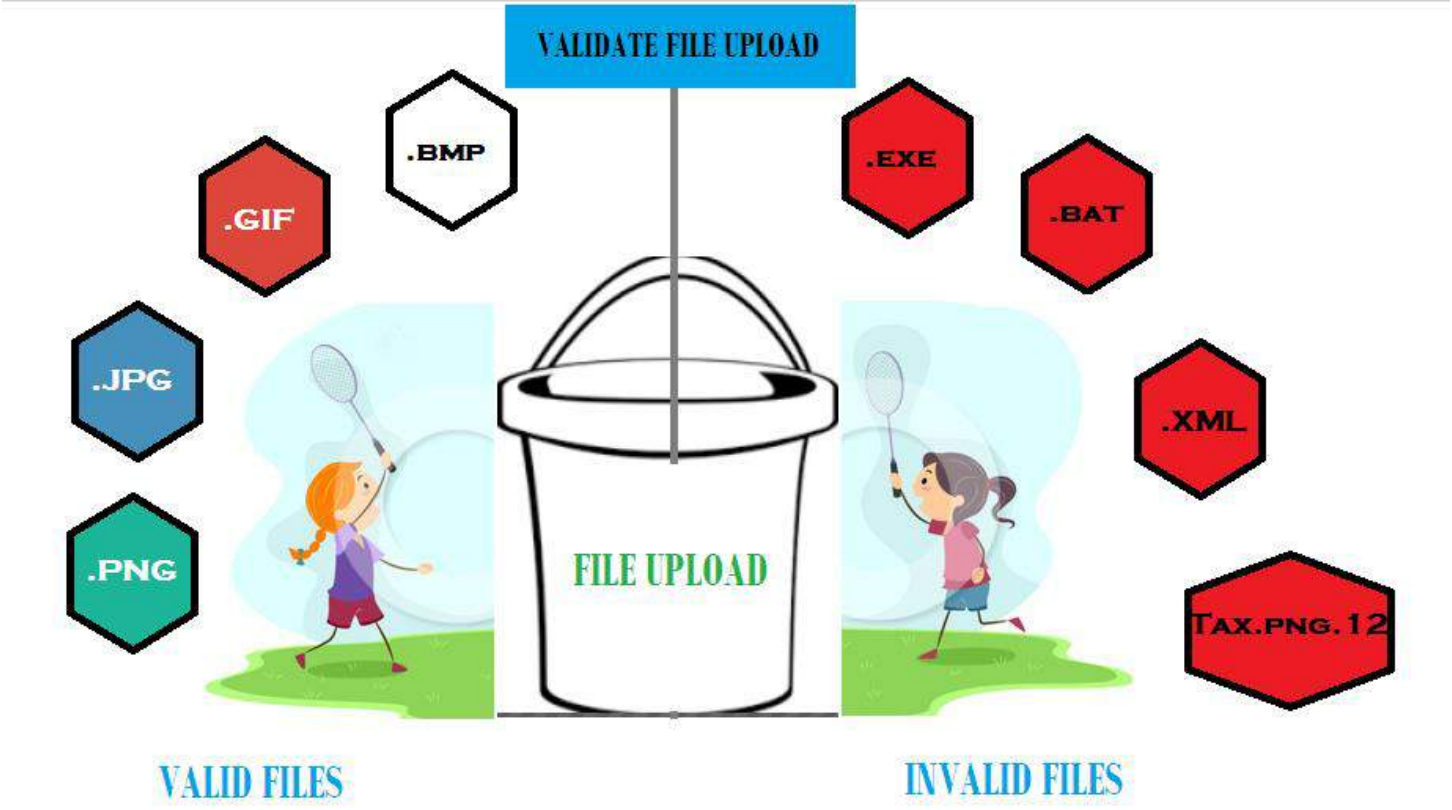


Fig 1.This image shows how people try to upload files some try valid files and some invalid files.

Solution:-

1) First thing we need to do is validate file uploads
2) Allow only access to files extension which are required
3) Check the file header.

First, the thing I am going to add a file upload control on View.

ADDING FILE UPLOAD CONTROL

```
@model MvcSecurity.Models.EmployeeDetail
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
@using (Html.BeginForm("Index", "Home",
    FormMethod.Post, new { enctype = "multipart/form-data" }))
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>EmployeeDetail</legend>
        <div class="editor-label">...</div>
        <div class="editor-field">...</div>
        <div class="editor-label">...</div>
        <div class="editor-field">...</div>
        <div class="editor-label">...</div>
        <div class="editor-field">...</div>
        <div class="editor-label">...</div>
        <div class="editor-field">...</div>
        <div class="editor-label">...</div>
        <div class="editor-field">...</div>
        <div class="editor-label">...</div>
        <div class="editor-field">
            @Html.EditorFor(model => model.worktype)
            @Html.ValidationMessageFor(model => model.worktype)
        </div>
        <br />
        <input type="file" id="Avatar" name="upload" />
        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
```

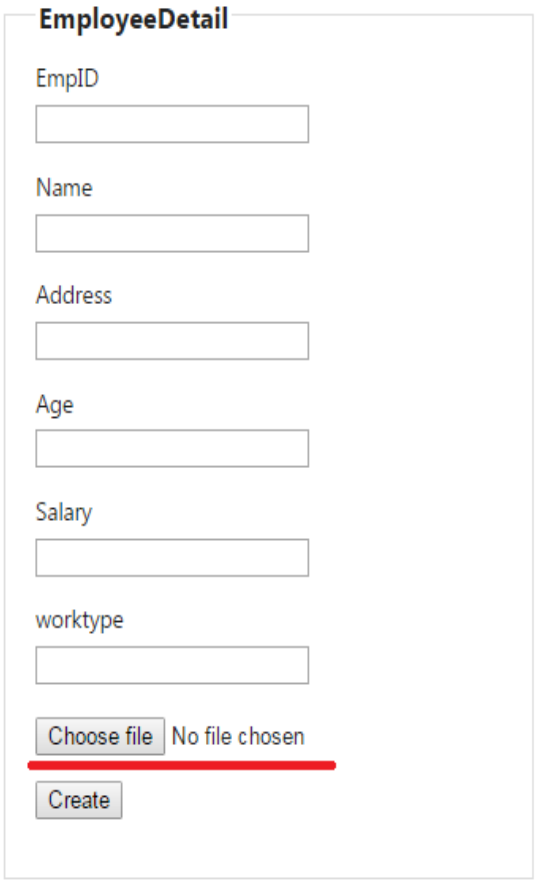


Fig 2.Adding File upload control on Add employee View.

We have added File upload control on View next we are going validate file on Submit.

VALIDATING FILE UPLOADS ON INDEX HTTPPOST METHOD

In this Method first, we are going to validate Content-Length of the file if it is zero [`upload.ContentLength == 0`] then user has not uploaded file.

If Content-Length is greater than zero then it has file [`upload.ContentLength > 0`] and we are going to reading Filename of File along with Content Type and File Bytes.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Index(EmployeeDetail EmployeeDetail)
{
    if (ModelState.IsValid)
    {
        HttpPostedFileBase upload = Request.Files["upload"];
        if (upload.ContentLength == 0)
        {
            ModelState.AddModelError("File", "Please Upload your file");
        }
        else if (upload.ContentLength > 0)
        {
         string fileName = upload.FileName; // getting File Name

         string fileContentType = upload.ContentType; // getting ContentType

         byte[]tempFileBytes = new byte[upload.ContentLength]; // getting filebytes

         var data = upload.InputStream.Read(tempFileBytes, 0,
Convert.ToInt32(upload.ContentLength));

         var types = MvcSecurity.Filters.FileUploadCheck.FileType.Image;  // Setting Image type
```

```
            var   result   =   FileUploadCheck.isValidFile(tempFileBytes,   types,   fileContentType);   //
Validate Header


                if (result == true)
                {
                    int FileLength = 1024 * 1024 * 2; //FileLength 2 MB
                    if (upload.ContentLength > FileLength)
                    {
                        ModelState.AddModelError("File", "Maximum allowed size is: " + FileLength + "
MB");
                    }
                    else
                    {
                        string demoAddress = Sanitizer.GetSafeHtmlFragment(EmployeeDetail.Address);
                        dbcon.EmployeeDetails.Add(EmployeeDetail);
                        dbcon.SaveChanges();
                        return View();
                    }
                }
            }
        }
        return View(EmployeeDetail);
    }
```

Till now it was basic validation we have done let's Validate file which is uploaded for doing that I have written a static class with name **FileUploadCheck** in this class there are Various Method for validating different file type for now I am going to show you how to validate Images files and only allow image files only.

FILEUPLOADCHECK CLASS

```
namespace MvcSecurity.Filters
{
    public static class FileUploadCheck
    {
        #region " Validations for File Types"
        private enum ImageFileExtension...
        private enum VideoFileExtension...
        private enum PDFFileExtension...
        public enum FileType...
        public static bool XLMimeType(string MimeType, string ext)...
        public static bool isValidFile(byte[] bytFile, FileType flType, String FileContentType)...
        public static bool isValidImageFile(byte[] bytFile, String FileContentType)...
        private static bool isValidVideoFile(byte[] bytFile, String FileContentType)...
        public static bool isValidPDFFile(byte[] bytFile, String FileContentType)...
        public static bool isValidDimension(byte[] bytFile, int maxRequiredWidth, int maxRequiredHeight)...
        #endregion
    }
}
```

Fig 3.View of FileUploadCheck Class which is custom created for validating File uploads.

In above snapshot there is **ImageFileExtension enum** which contains **Image formats and File types.**

```
private enum ImageFileExtension
{
    none = 0,
    jpg = 1,
    jpeg = 2,
    bmp = 3,
    gif = 4,
    png = 5
```

```
}

public enum FileType
{
    Image = 1,
    Video = 2,
    PDF = 3,
    Text = 4,
    DOC = 5,
    DOCX = 6,
    PPT = 7,
}
```

If it has passed basic validation then we are going to call **isValidFile** Method which take bytes, File type, FileContentType as input.

```
public static bool isValidFile(byte[] bytFile, FileType flType, String FileContentType)
{
    bool isvalid = false;
    if (flType == FileType.Image)
    {
        isvalid = isValidImageFile(bytFile, FileContentType); //we are going call this method
    }
    else if (flType == FileType.Video)
    {
        isvalid = isValidVideoFile(bytFile, FileContentType);
    }
    else if (flType == FileType.PDF)
    {
        isvalid = isValidPDFFile(bytFile, FileContentType);
    }
    return isvalid;
```

```
}
```

After calling isValidFile method it will call another static method based on File type.

If File type is image then it will call first Method [**isValidImageFile**] else File type is Video then second Method [**isValidVideoFile**] and in similar way if File type is PDF then last method [**isValidPDFFile**] will get called.

After completing with understanding **isValidFile** Method let's have a look on [**isValidImageFile**] Method which will get called.

Below is complete code snippet of [isValidImageFile] Method.

In this Method, we are allowing limited Image file extensions [jpg, jpeg, png, bmp, gif]

Working of this isValidImageFile method

When we pass bytes and FileContentType to this method then it will first check for FileContentType on the base of that it will set `ImageFileExtension` after that it is going to check header bytes which we have against uploaded file bytes if it matches that then File is valid [true] else file is invalid [false].

```
public static bool isValidImageFile(byte[] bytFile, String FileContentType)
{
    bool isvalid = false;

    byte[] chkBytejpg = { 255, 216, 255, 224 };
    byte[] chkBytebmp = { 66, 77 };
    byte[] chkBytegif = { 71, 73, 70, 56 };
    byte[] chkBytepng = { 137, 80, 78, 71 };

    ImageFileExtension imgfileExtn = ImageFileExtension.none;
```

```
if (FileContentType.Contains("jpg") | FileContentType.Contains("jpeg"))
{
    imgfileExtn = ImageFileExtension.jpg;
}
else if (FileContentType.Contains("png"))
{
    imgfileExtn = ImageFileExtension.png;
}
else if (FileContentType.Contains("bmp"))
{
    imgfileExtn = ImageFileExtension.bmp;
}
else if (FileContentType.Contains("gif"))
{
    imgfileExtn = ImageFileExtension.gif;
}

if (imgfileExtn == ImageFileExtension.jpg || imgfileExtn == ImageFileExtension.jpeg)
{
    if (bytFile.Length >= 4)
    {
        int j = 0;
        for (Int32 i = 0; i <= 3; i++)
        {
            if (bytFile[i] == chkBytejpg[i])
            {
                j = j + 1;
                if (j == 3)
                {
                    isvalid = true;
```

```
                }
            }
        }
    }
}



if (imgfileExtn == ImageFileExtension.png)
{
    if (bytFile.Length >= 4)
    {
        int j = 0;
        for (Int32 i = 0; i <= 3; i++)
        {
            if (bytFile[i] == chkBytepng[i])
            {
                j = j + 1;
                if (j == 3)
                {
                    isvalid = true;
                }
            }
        }
    }
}



if (imgfileExtn == ImageFileExtension.bmp)
{
    if (bytFile.Length >= 4)
```

```
    {
        int j = 0;
        for (Int32 i = 0; i <= 1; i++)
        {
            if (bytFile[i] == chkBytebmp[i])
            {
                j = j + 1;
                if (j == 2)
                {
                    isvalid = true;
                }
            }
        }
    }
}

if (imgfileExtn == ImageFileExtension.gif)
{
    if (bytFile.Length >= 4)
    {
        int j = 0;
        for (Int32 i = 0; i <= 1; i++)
        {
            if (bytFile[i] == chkBytegif[i])
            {
                j = j + 1;
                if (j == 3)
                {
                    isvalid = true;
                }
```

```
                }
            }
        }
    }
    return isvalid;
}
```

Calling isValidFile method from Action Method

We are going to call (`FileUploadCheck.isValidFile`) Method and then we are going to pass Parameter File Bytes, Types, FileContentType.

This Method will return Boolean value if it is valid file then true else it will return false.

```
string fileName = upload.FileName; // getting File Name

string fileContentType = upload.ContentType; // getting ContentType

byte[] tempFileBytes = new byte[upload.ContentLength]; // getting filebytes

var data = upload.InputStream.Read(tempFileBytes, 0, Convert.ToInt32(upload.ContentLength));

var types = MvcSecurity.Filters.FileUploadCheck.FileType.Image;   // Setting Image type

var result = FileUploadCheck.isValidFile(tempFileBytes, types, fileContentType); //Calling isValidFile method
```

After understanding code snippet now let's see a demo with how it works.

Below snapshot shows an Employee form with file upload control.

We are going to fill below form and choosing a valid file.

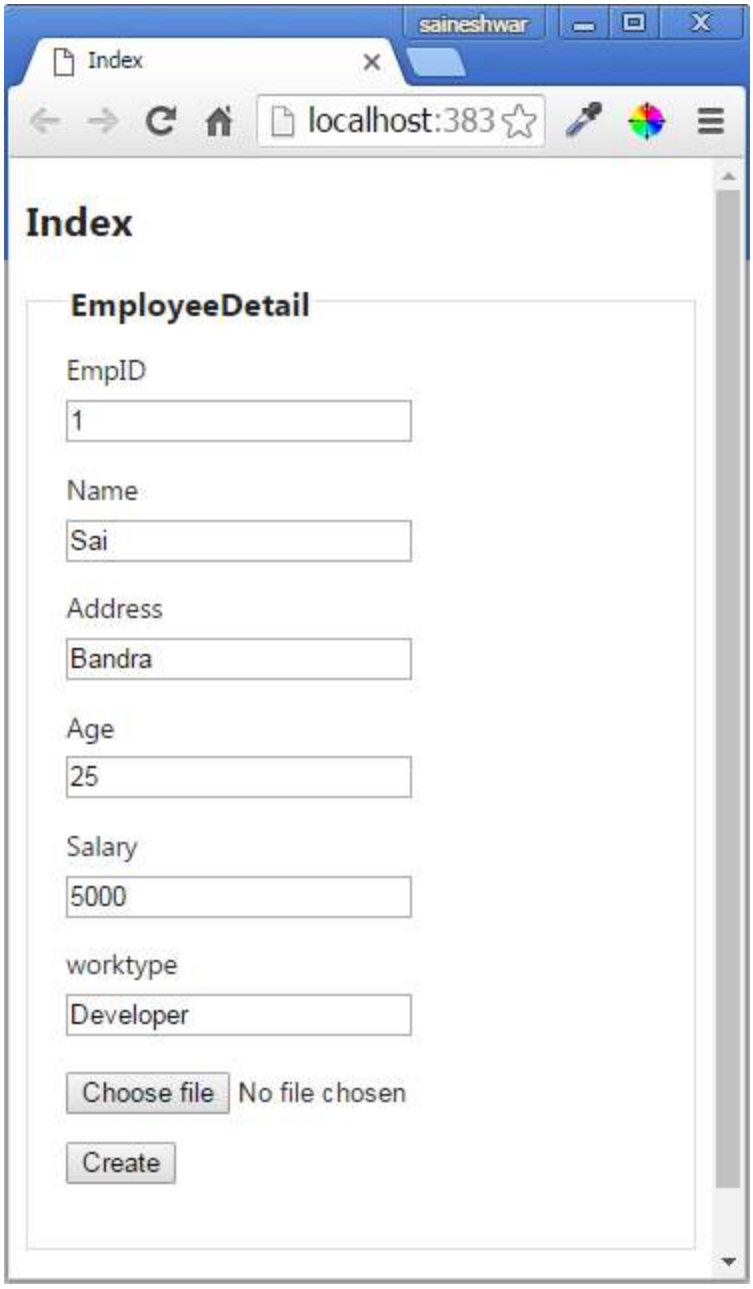


Fig 4.Add Employee form View after adding file upload.

CHOOSING A VALID .JPG FILE AND CHECK HOW IT WILL VALIDATE IN DETAILS

Choosing a .jpg image from disk.

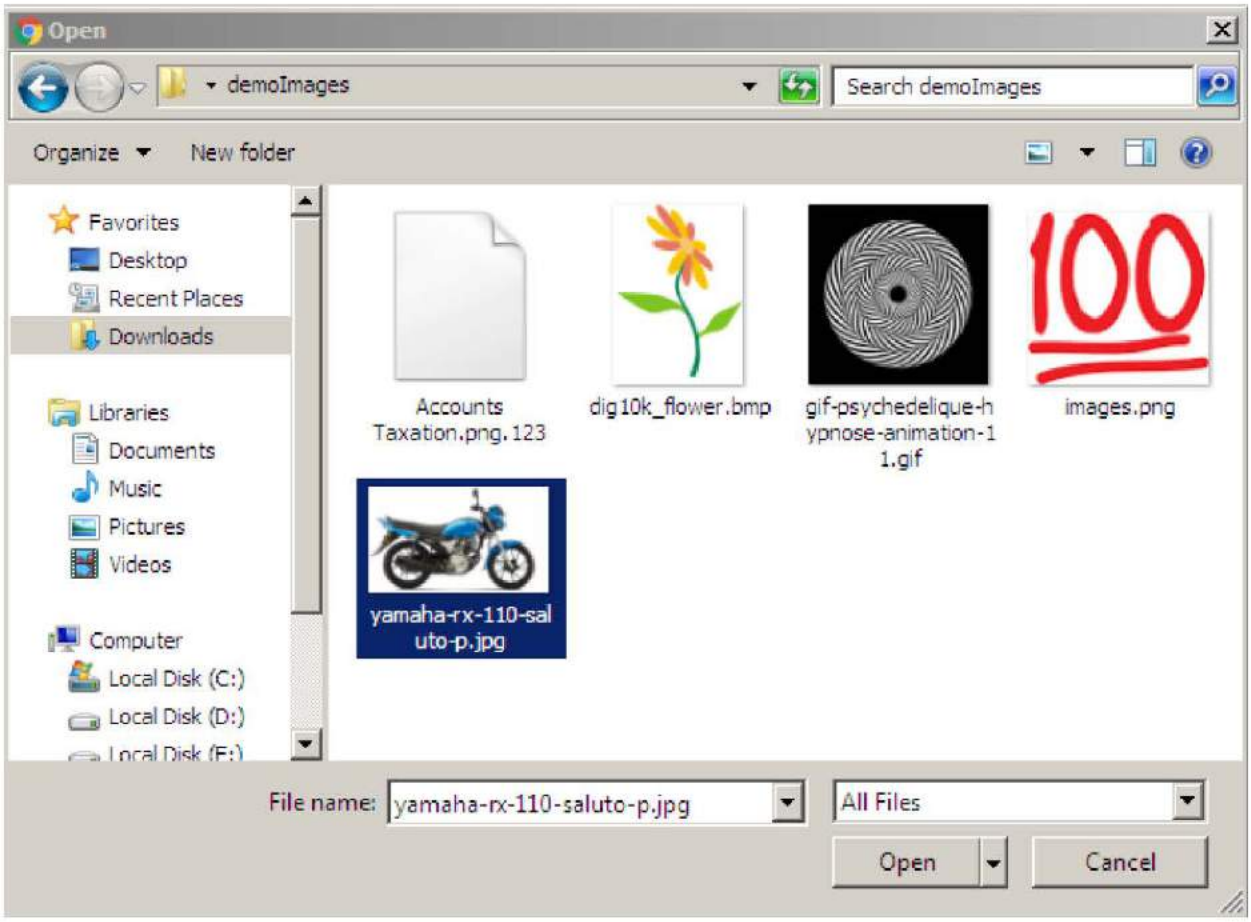


Fig 5.Choosing File for upload.

SNAPSHOT OF EMPLOYEE FORM AFTER CHOOSING A FILE.

In this part, we have chosen a file.

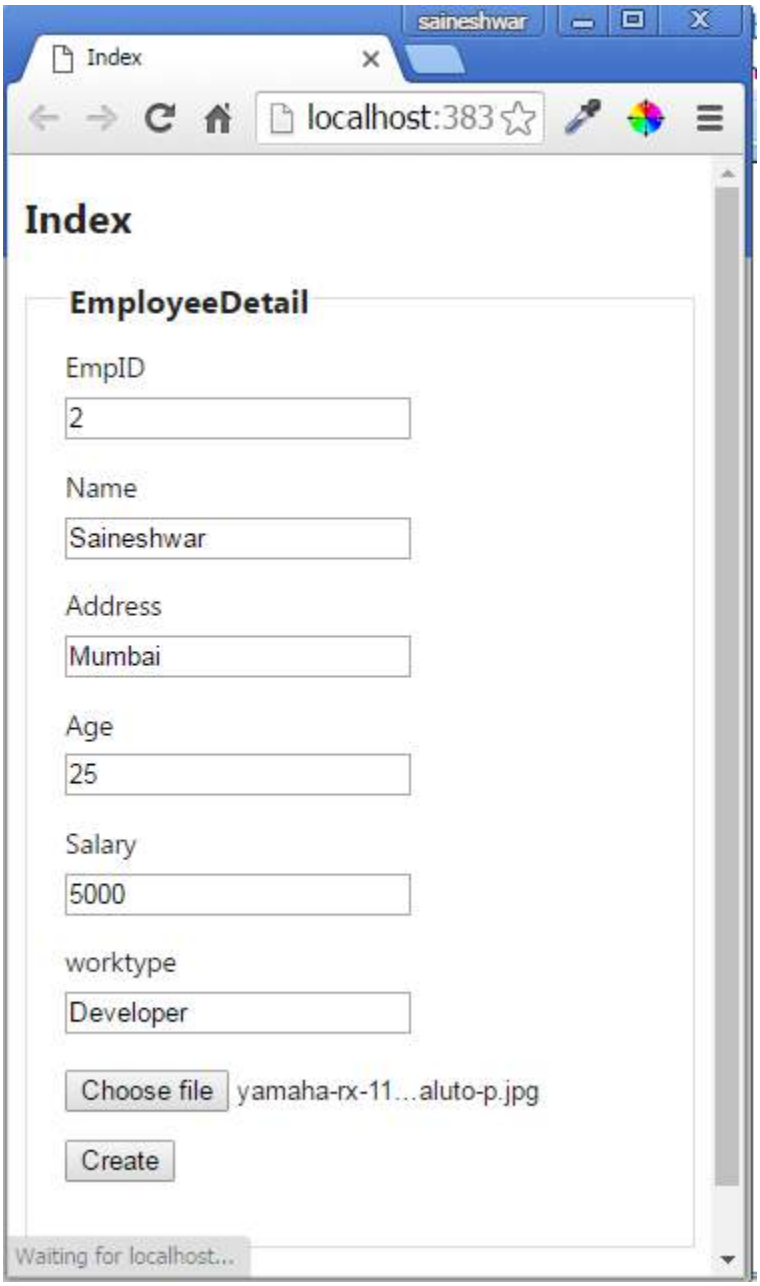


Fig 6.After Choosing File for upload.

SNAPSHOT OF DEBUGGING INDEX POST ACTION METHOD.

In this part we have posted an Employee form with a file here we can see have it is validating basic validations.

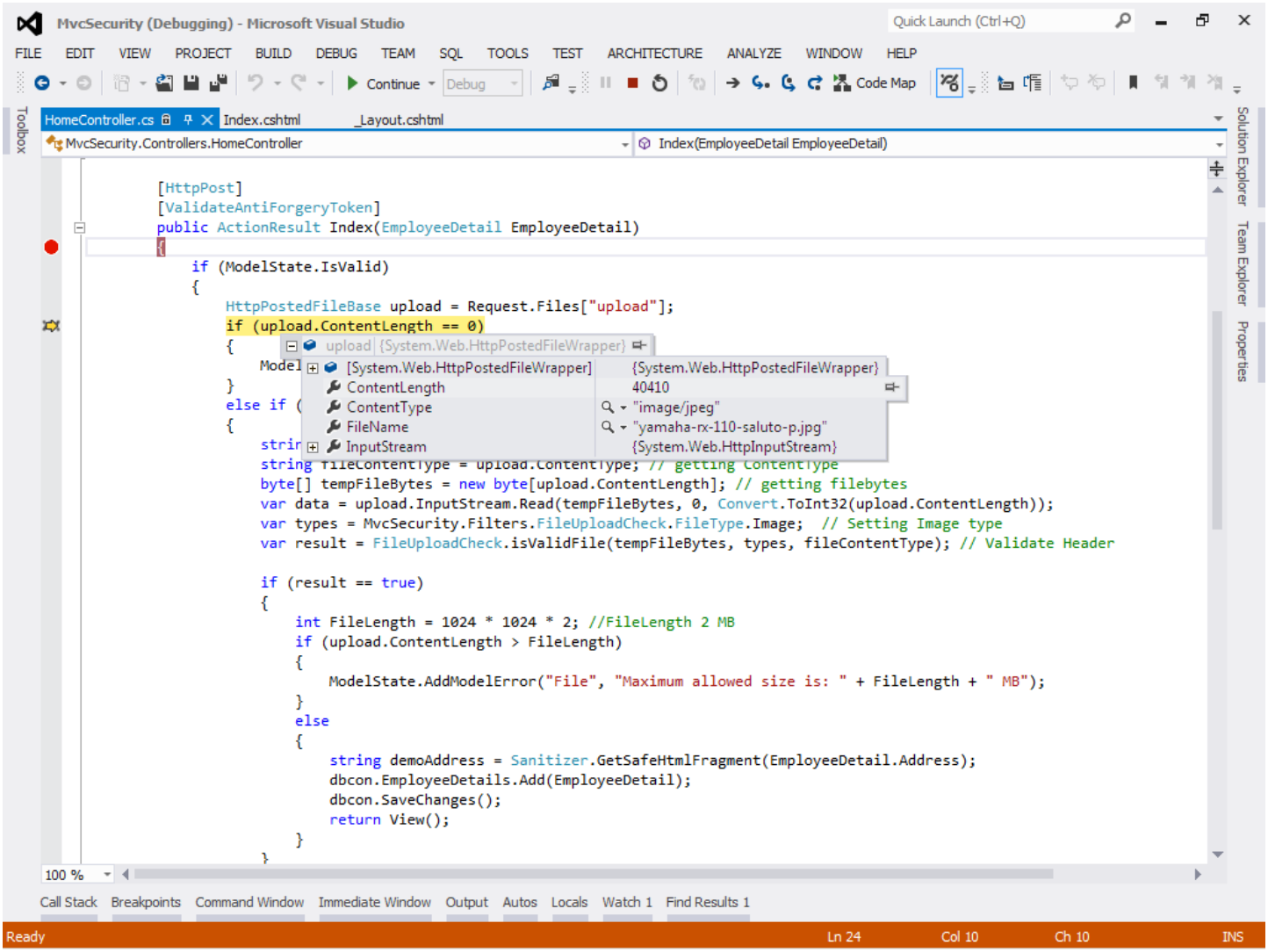


Fig 7.After Submitting Form for saving data, it shows real time value of file which we have uploaded.

SNAPSHOT OF DEBUGGING INDEX POST ACTION METHOD.

In this part, you can see the real-time value of file which we have uploaded it has passed basic validation.

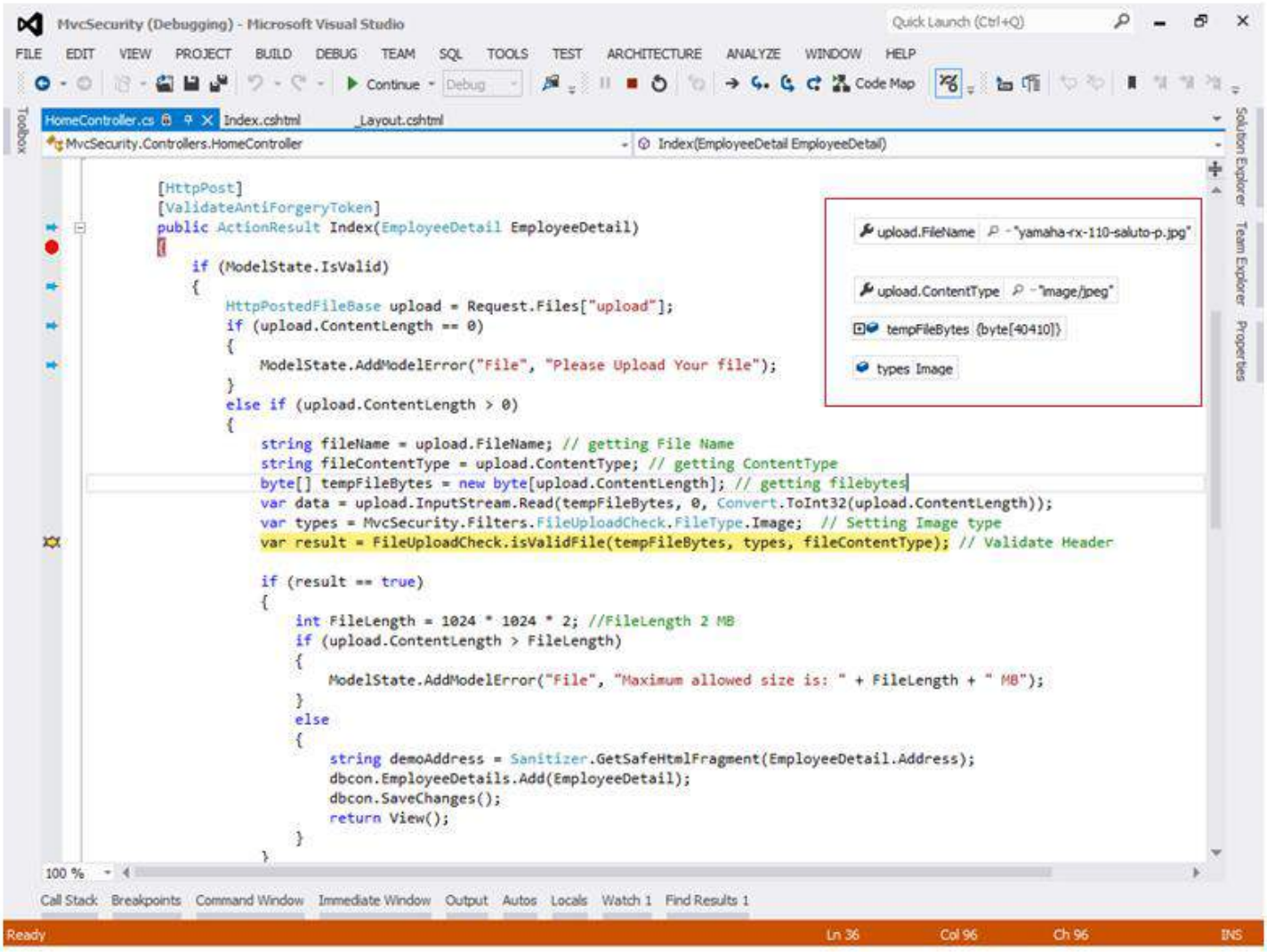


Fig 8.After Submitting Form for saving data, it shows real time value of file which we have uploaded.

SNAPSHOT OF FILEUPLOADCHECK CLASS WHILE ISVAILDFILE METHOD GETS CALLED.

In this part after calling isValidFile Method it going call another method according to its FileContentType.

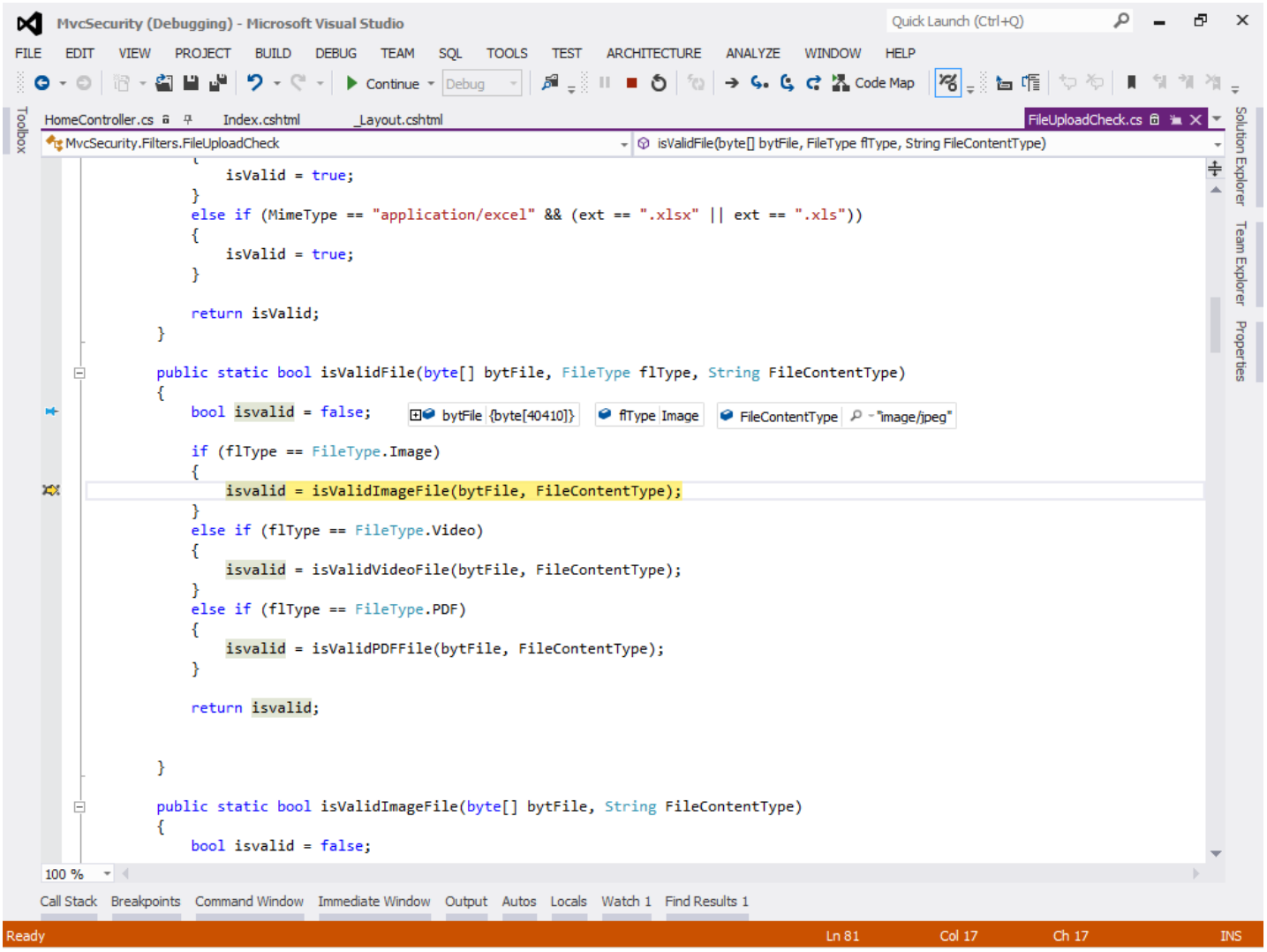


Fig 9.Calling method inside FileUploadCheck Class according to File Type.

SNAPSHOT OF ISVAILDIMAGEFILE METHOD WHILE CHECKING HEADER BYTES.

In this method, it will check header bytes of the image which is upload against the bytes which we have if it matches then it is a valid file else it is not a valid file.

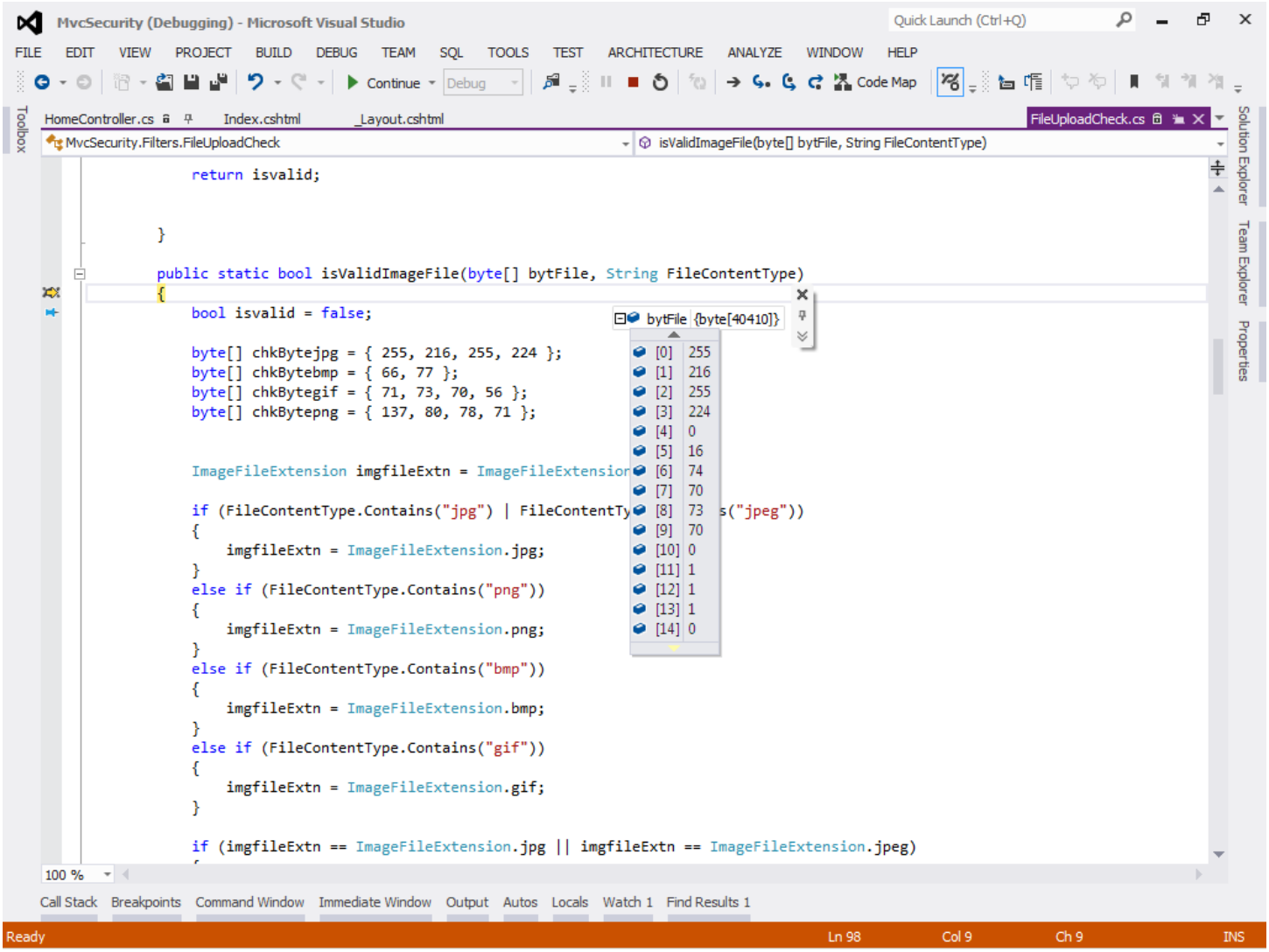


Fig 9. Check header bytes of the image which is upload against the bytes which we have.

- ## Version Discloser

Version information can be used by an attacker to target specific attack on that Version which is disclosed.

Whenever browser sends HTTP to request to the server in response we get response header which contains information of [**Server, X-AspNet-Version, X-AspNetMvc-Version, X-Powered-By**].

The server shows information of which web server is begin used.

X-AspNet-Version shows information of which specific **Asp.Net Version** Used.

X-AspNetMvc-Version shows information of which ASP.NET MVC version Used.

X-Powered-By shows information of which framework your website is running on.

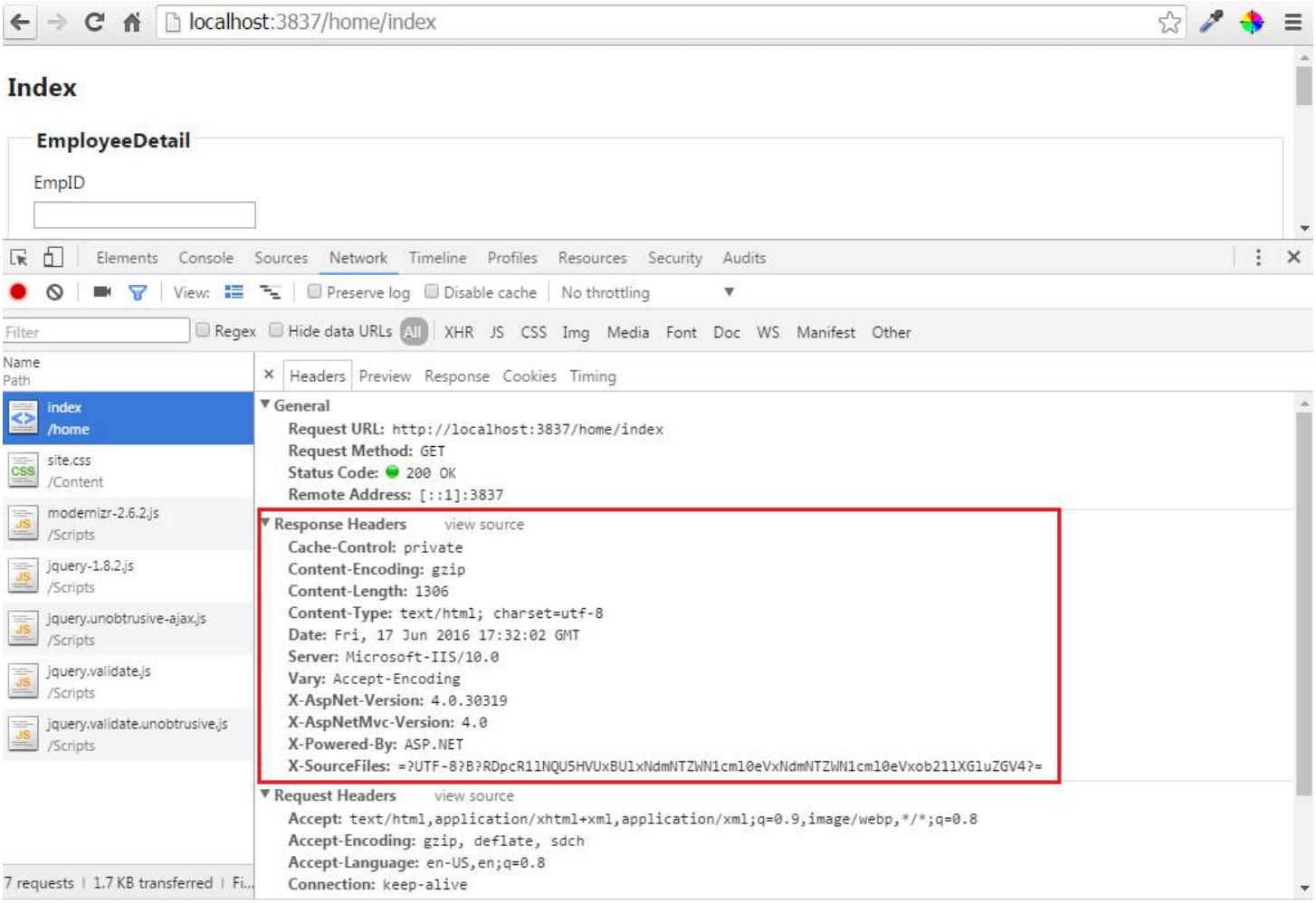


Fig 1. Response header disclosing Version Information.

Solution:-

1) For removing **X-AspNetMvc-Version header**

To remove response **X-AspNetMvc-Version** which shows information of which ASP.NET MVC version used we have built in property in MVC.

Just set [MvcHandler.DisableMvcResponseHeader = true;] in Global.asax Application start event [**Application_Start()**] this will remove header it won't be displayed any more.

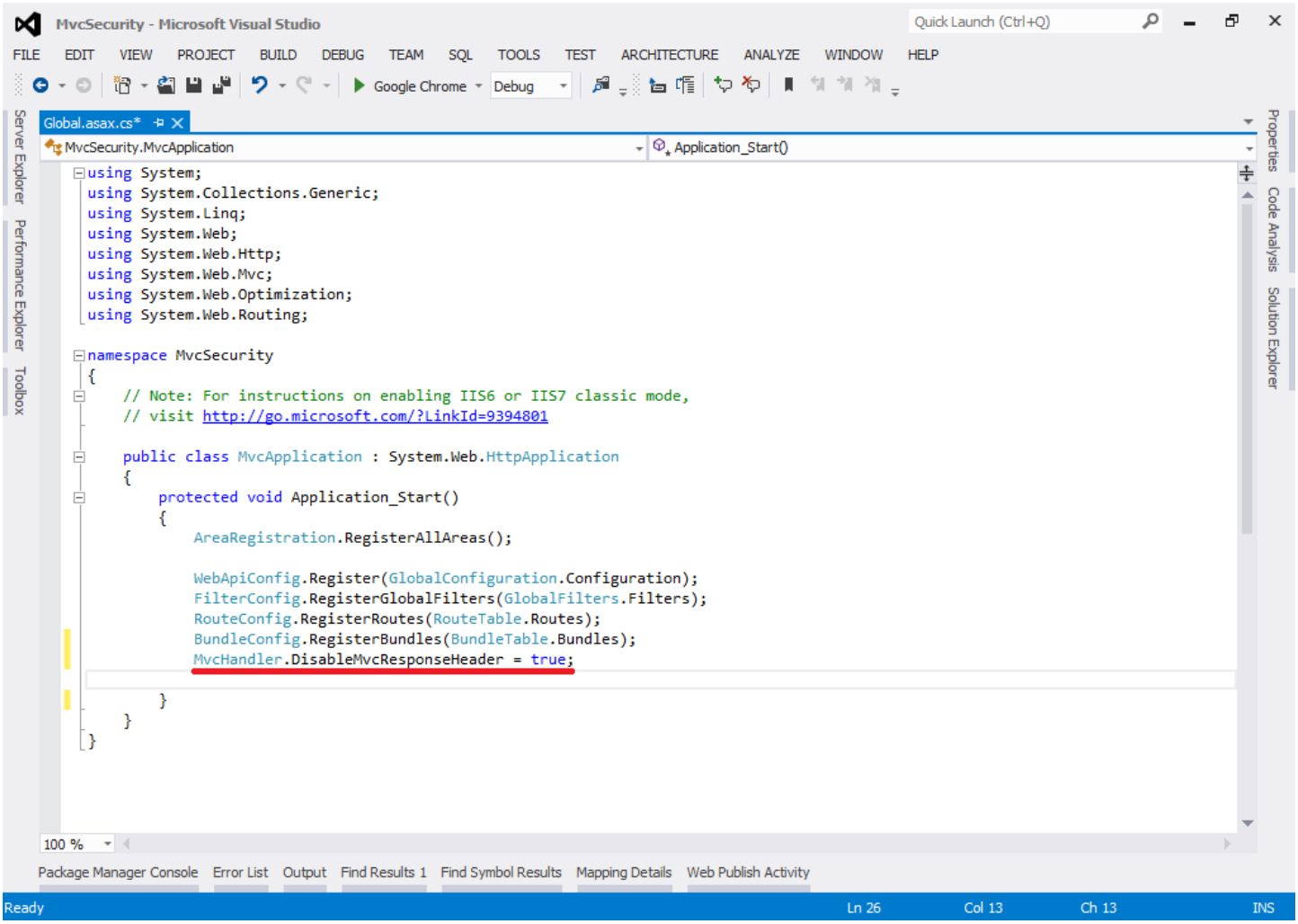


Fig 2.Setting property in Global.asax to remove X-AspNetMvc-Version from header.

Fig 3.Response after removing X-AspNetMvc-Version from header.

2) For removing **X-AspNet-Version and Server header**

To remove response of **Server header** which shows information of which web server is begin used and along with that **X-AspNet-Version header** shows information of which specific **Asp.Net Version** Used.

Just add an event in [**Application_PreSendRequestHeaders**()] in global.asax and then to remove header we need to set the property as below.

```
protected void Application_PreSendRequestHeaders()
{
    Response.Headers.Remove("Server");           //Remove Server Header
    Response.Headers.Remove("X-AspNet-Version"); //Remove X-AspNet-Version Header
}
```

Global.asax.cs

MvcSecurity.MvcApplication — Application_Start()

```
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace MvcSecurity
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801

    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            MvcHandler.DisableMvcResponseHeader = true;
        }

        protected void Application_PreSendRequestHeaders()
        {
            Response.Headers.Remove("Server");           //Remove Server Header
            Response.Headers.Remove("X-AspNet-Version"); //Remove X-AspNet-Version Header
        }
    }
}
```

Fig 4.Adding Application_PreSendRequestHeaders event in global.asax and then removing Response headers.

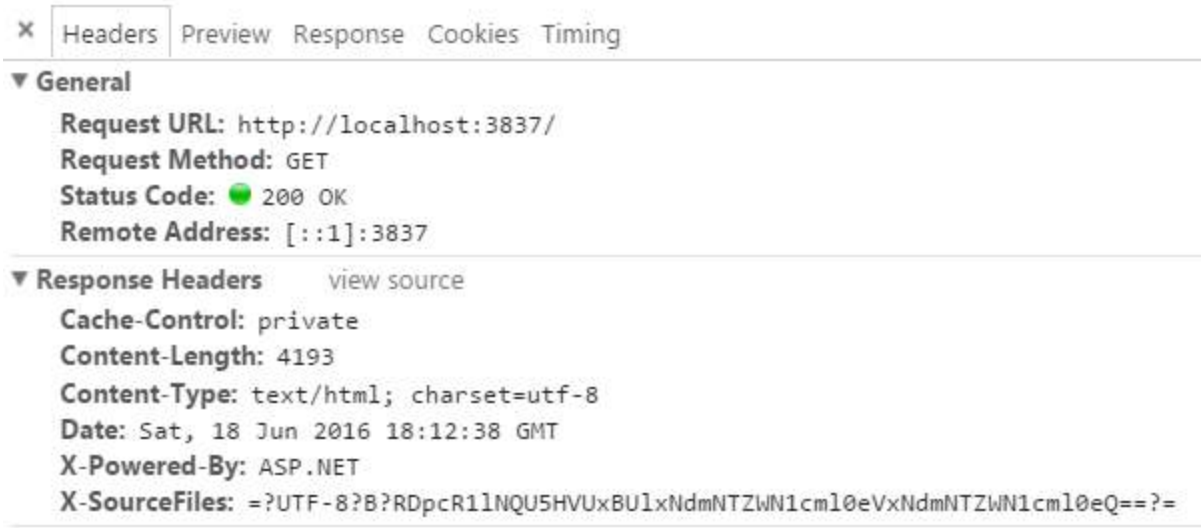


Fig 5.Response after removing X-AspNet-Version and Server from header.

3) For removing **X-Powered-By header**

To remove response of X-Powered-By **header** which shows information of which framework your website is running on.

Just add this tag under System.webServer in Web.config file will remove [**X-Powered-By**] header.

```
<httpProtocol>
  <customHeaders>
   <remove name="X-Powered-By" />
  </customHeaders>
</httpProtocol>
```

```
<system.webServer>
  <validation validateIntegratedModeConfiguration="false" />

  <handlers>
    <remove name="ExtensionlessUrlHandler-ISAPI-4.0_32bit" />
    <remove name="ExtensionlessUrlHandler-ISAPI-4.0_64bit" />
    <remove name="ExtensionlessUrlHandler-Integrated-4.0" />
    <add name="ExtensionlessUrlHandler-ISAPI-4.0_32bit"
         path="*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE,PATCH,OPTIONS"
         modules="IsapiModule" scriptProcessor="%windir%\Microsoft.NET\Framework\v4.0.30319\aspnet_isapi.dll"
         preCondition="classicMode,runtimeVersionv4.0,bitness32" responseBufferLimit="0" />
    <add name="ExtensionlessUrlHandler-ISAPI-4.0_64bit"
         path="*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE,PATCH,OPTIONS"
         modules="IsapiModule" scriptProcessor="%windir%\Microsoft.NET\Framework64\v4.0.30319\aspnet_isapi.dll"
         preCondition="classicMode,runtimeVersionv4.0,bitness64" responseBufferLimit="0" />
    <add name="ExtensionlessUrlHandler-Integrated-4.0"
         path="*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE,PATCH,OPTIONS" type="System.Web.Handlers.TransferRequestHandler"
         preCondition="integratedMode,runtimeVersionv4.0" />
  </handlers>

  <httpProtocol>
    <customHeaders>
      <remove name="X-Powered-By" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

Fig 6.Adding custom Header tag in Web.config for removing Response headers.

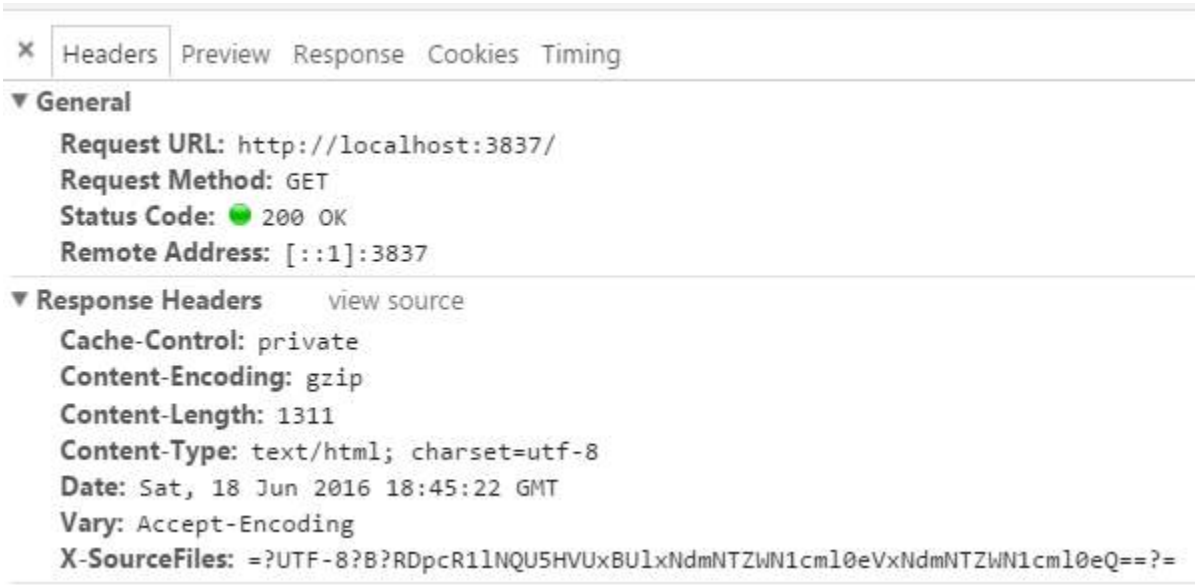


Fig 7.Response after removing X-Powered-By from header.

- ## SQL Injection Attacks

SQL Injection attack is one of the most dangerous attacks it is ranked 1 in top 10 Vulnerabilities by **OWASP** 2013 [Open Web Application Security Project] . SQL injection attack can give valuable data to the attacker that can lead to a big security breach and can also take full access to the database server.

In SQL Injection attacker always try to enter malicious SQL statement which will get executed in the database and return unwanted data to the attacker.

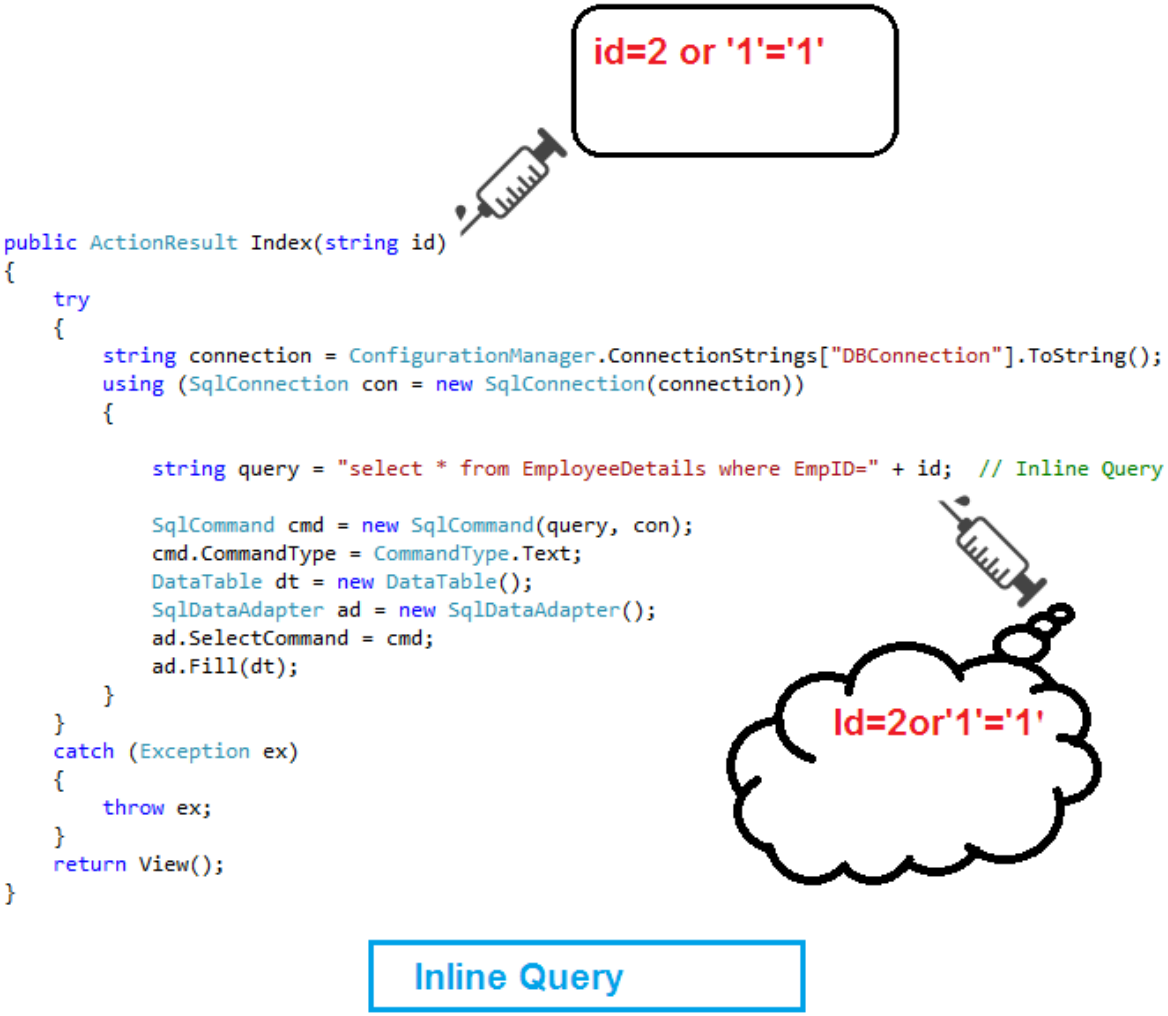


Fig 1.Sql injection attack example which shows how attack mostly occurs if you are using inline queries.

SIMPLE VIEW WHICH SHOWS USER DATA

View Shows Single Employee data based on EmployeeID as displayed in below Snapshot.

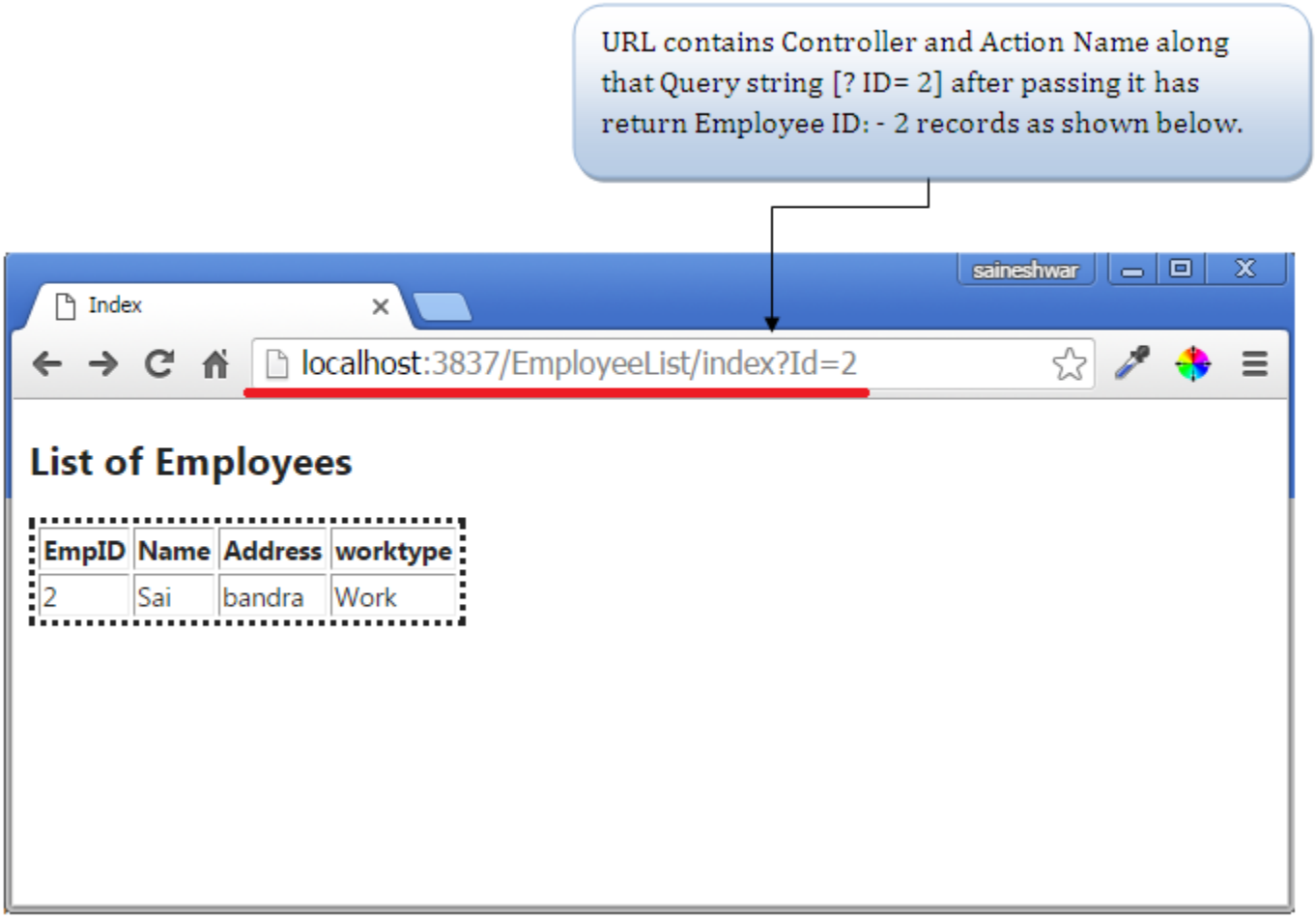


Fig 1.Employee View which displays user data.

SIMPLE VIEW WHICH SHOWS ALL USER DATA AFTER SQL INJECTION ATTACK

In this Browser View as attacker saw Application URL which contains some valuable data which is **ID** [**http://localhost:3837/EmployeeList/index?Id=2**] attacker tries SQL Injection attack as shown below.

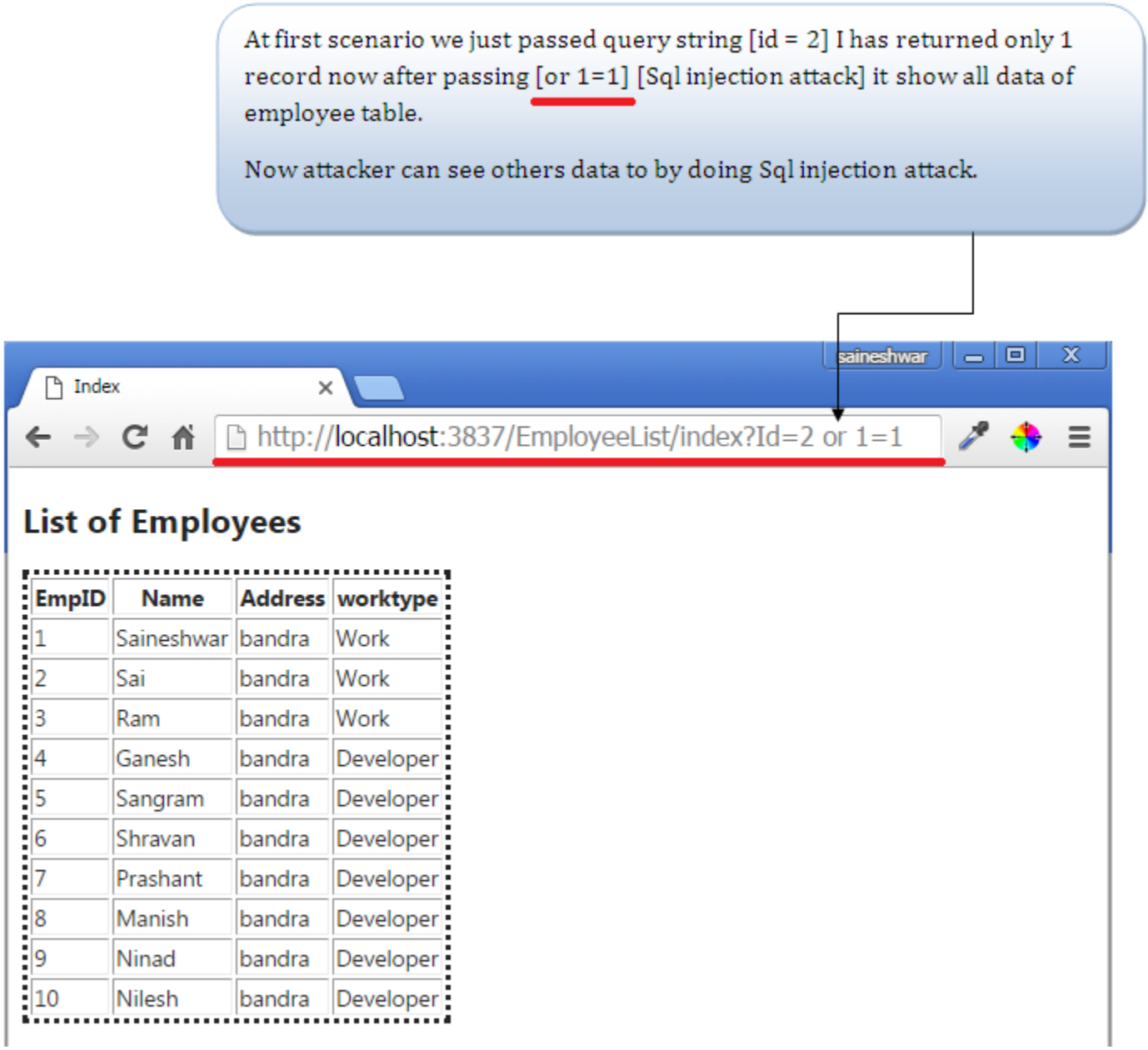


Fig 2.Employee View which displays all User data after SQL injection.

After trying permutation & combination of SQL injection attack , the attacker gets access to all User data.

DISPLAYING SQL INJECTION IN DEBUG MODE

Here we can see in details how attacker passed malicious SQL statement which gets executed in the database.

```
public ActionResult Index(string id)
{
    List<EmployeeDetail> employeeList = null;
    try
    {
        string connection = ConfigurationManager.ConnectionStrings["DBConnection"].ToString();
        using (SqlConnection con = new SqlConnection(connection))
        {

            string query = "select * from EmployeeDetails where EmpID=" + id;  // Inline Query

            SqlCommand cmd = new SqlCommand(query, con);
            cmd.CommandType = CommandType.Text;
            DataTable dt = new DataTable();
            SqlDataAdapter ad = new SqlDataAdapter();
            ad.SelectCommand = cmd;
            ad.Fill(dt);

           employeeList = dt.DataTableToList<EmployeeDetail>();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return View(employeeList);
}
```

id "2 or 1=1"

id "2 or 1=1"

cmd.CommandText "select * from EmployeeDetails where EmpID=2 or 1=1"

Fig 3.Debug mode View of index Action Method.

SQL PROFILER VIEW OF SQL STATEMENT

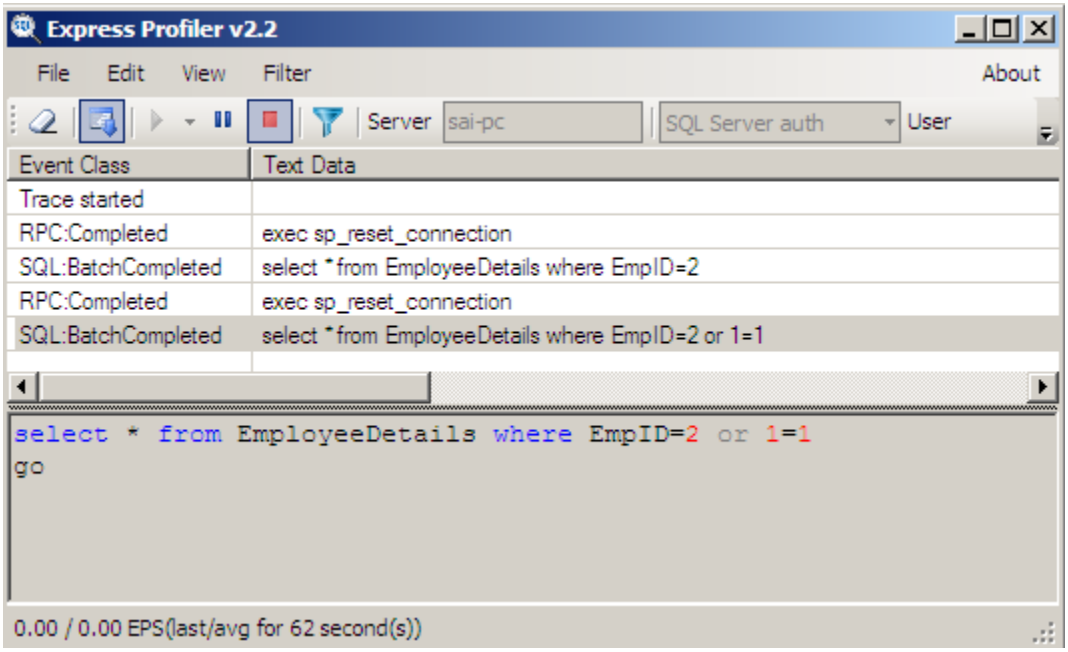


Fig 4. SQL Profiler View.

Solution:-

1) Validate inputs
2) Use of low-privileged database logins
3) Use Parameterized queries
4) Use ORM (e.g. Dapper , Entity framework)
5) Use Stored Procedures

1) Validate inputs

Always validate input in both side Clientside and Server side such that no special characters are entered in inputs which allow an attacker to get into the system.

In MVC we use Data Annotations for Validation.

Data Annotations attribute are simple rule that are applied on Model to validate Model data.

CLIENT SIDE VALIDATING INPUTS

```
public partial class EmployeeDetail
{
    public int EmpID { get; set; }
    [Required(ErrorMessage = "Enter Name")]
    [RegularExpression("^[A-z]+$",
    ErrorMessage = "Enter Valid Name")]
    public string Name { get; set; }

    [StringLength(50)]
    [RegularExpression("[a-zA-Z ]+$",
    ErrorMessage = "Enter Valid Address")]
    [Required(ErrorMessage = "Enter Address")]
    [AllowHtml]
    public string Address { get; set; }

    [RegularExpression("^[0-9]+$",
    ErrorMessage = "Enter Valid Age")]
    [Required(ErrorMessage = "Enter Age")]
    public Nullable<int> Age { get; set; }

    [RegularExpression("(((\\d+)((\\.\\d{1,2})?))$",
    ErrorMessage = "Enter Valid Salary")]
    [Required(ErrorMessage = "Enter Salary")]
    public Nullable<decimal> Salary { get; set; }

    [Required(ErrorMessage = "Enter Worktype")]
    [RegularExpression("[a-zA-Z ]+$",
    ErrorMessage = "Enter Valid Worktype")]
    public string worktype { get; set; }
}
```

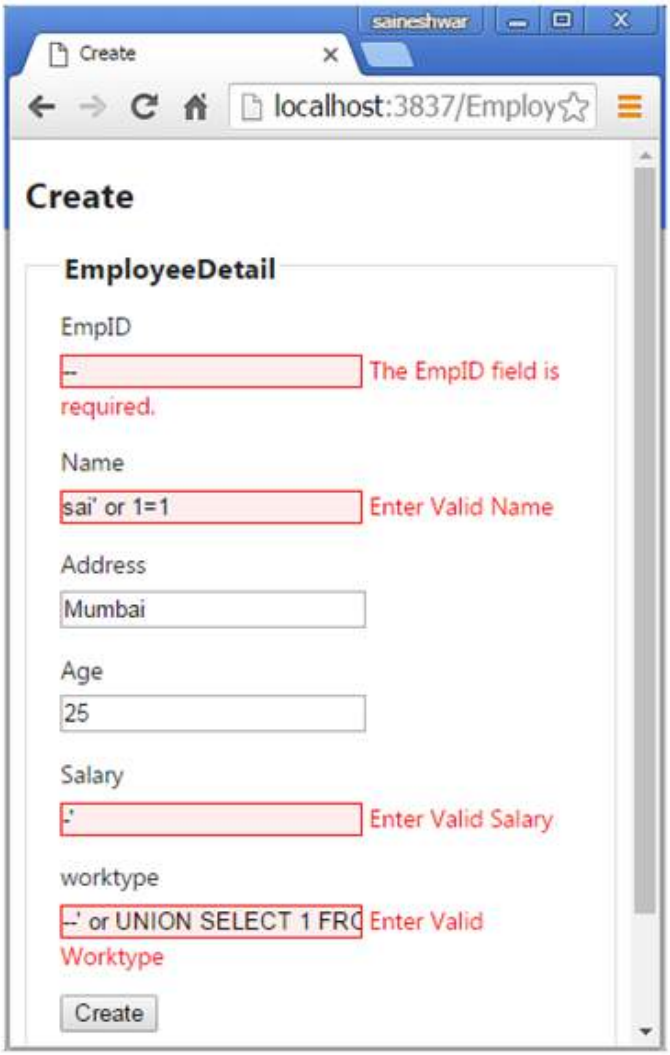


Fig 1. Client side validating inputs.

Model state is false this indicates that Model is not valid.

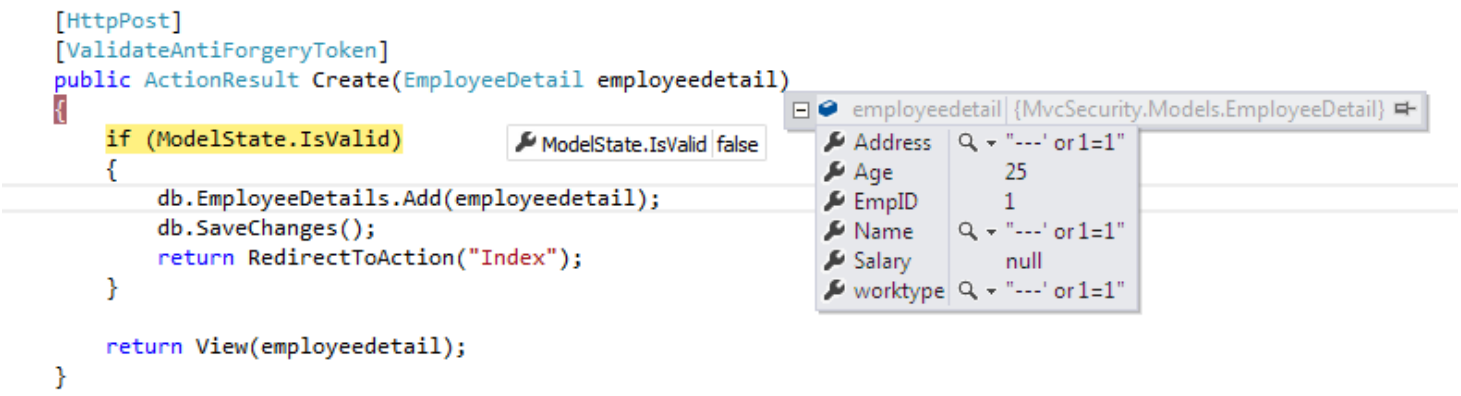


Fig 2. Server side validating inputs.

2) Give least-privileged database logins

Db_owner is default role for the database which can grant and revoke access, create tables, stored procedures, views, run backups, schedule jobs can even drop the database. If this kind of Role access is given to database then User who using can have full access and he can perform a various activity on it. We must create a new user account with least-privileged and give only that privilege that user must access.

e.g. if the user has to work related to Selecting , Inserting and Updating Employee details then only select, Insert and update permission should be provided.

STEP TO ADD NEW USER AND PROVIDE PRIVILEGE

In this part, I have shown a small sample of how to create User provide specific permission to table.

1] Creating New User

2] View after creating User

3] Selecting object that user can access (table)

4] Choosing specific table to provide permission

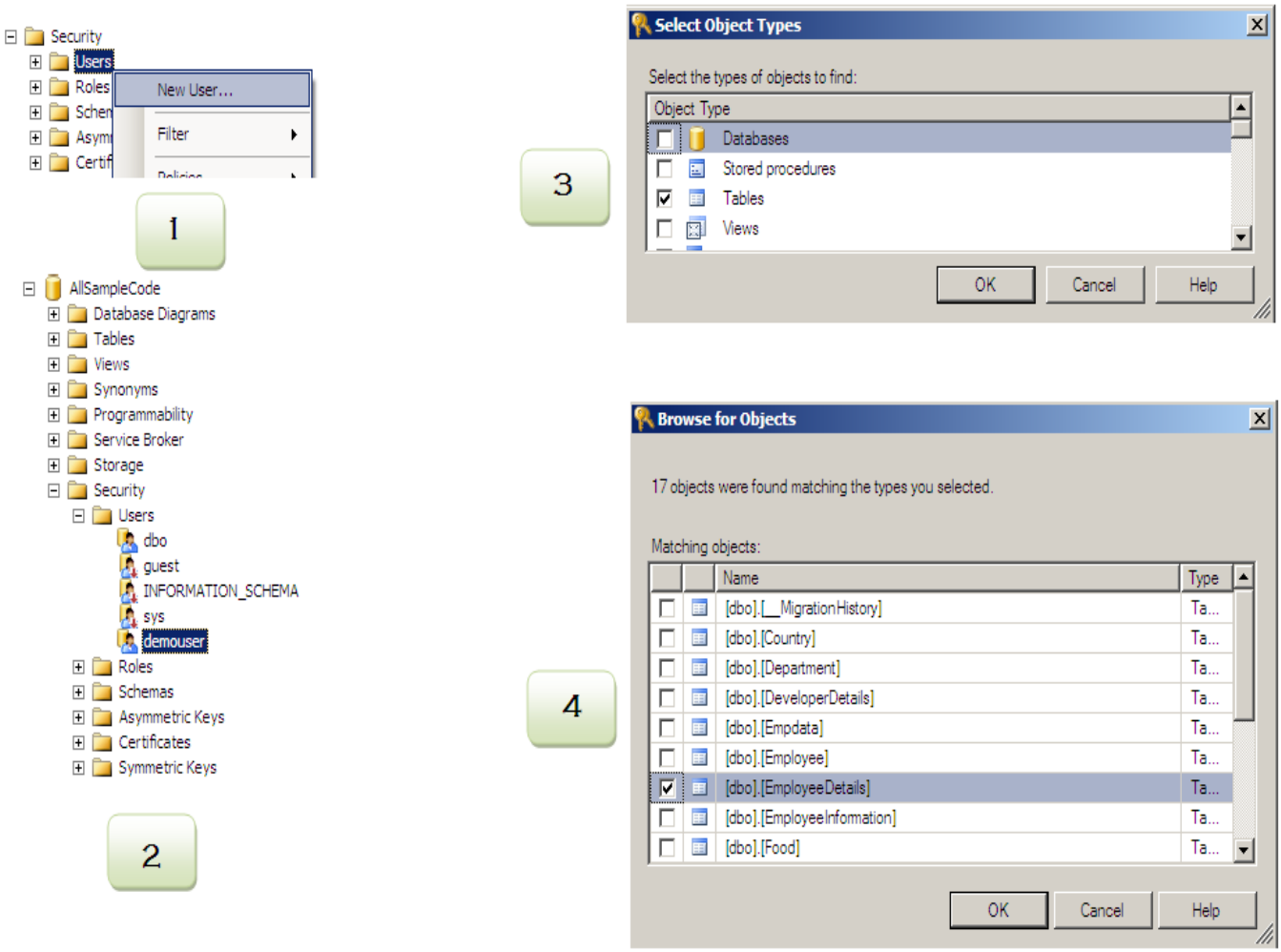


Fig 1. Creating New Users and providing rights to Users.

After choosing table we are going providing permission to table as shown below, you can see we have only given permission to ‘Insert’, ‘Select’, ‘Update’ for this User.

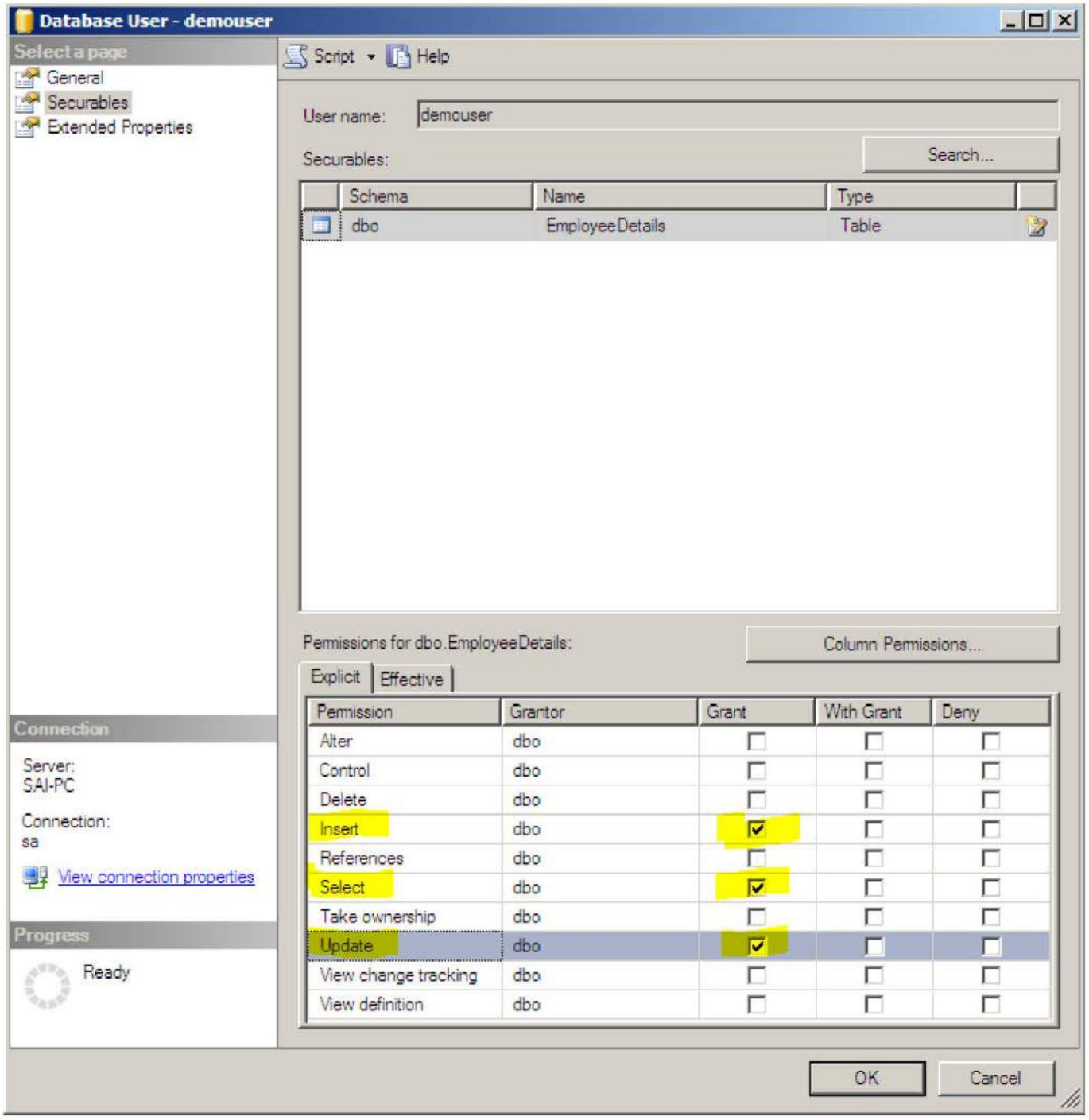


Fig 2. Giving permission for Insert, Select, Update.

With least-privileged, it will help us to safeguard database from attackers.

Use Stored Procedures

Stored procedures are a form of parameterized query .Using Stored Procedures is also one way to protect from SQL injection attack.

In below snapshot we have removed inline Query which we were using before now we have written code for getting data from database using stored procedure which helps us to protect against SQL injection attack ,then to stored procedure we need to pass parameter [@EmpID] and according to parameter it is going to get records from database after passing parameter we have used CommadType [CommadType.StoredProcedure] which indicate we are using stored procedure .

Note :- Always Use parameter with stored procedure if you do not use it still you are Vulnerable to SQL injection attack.

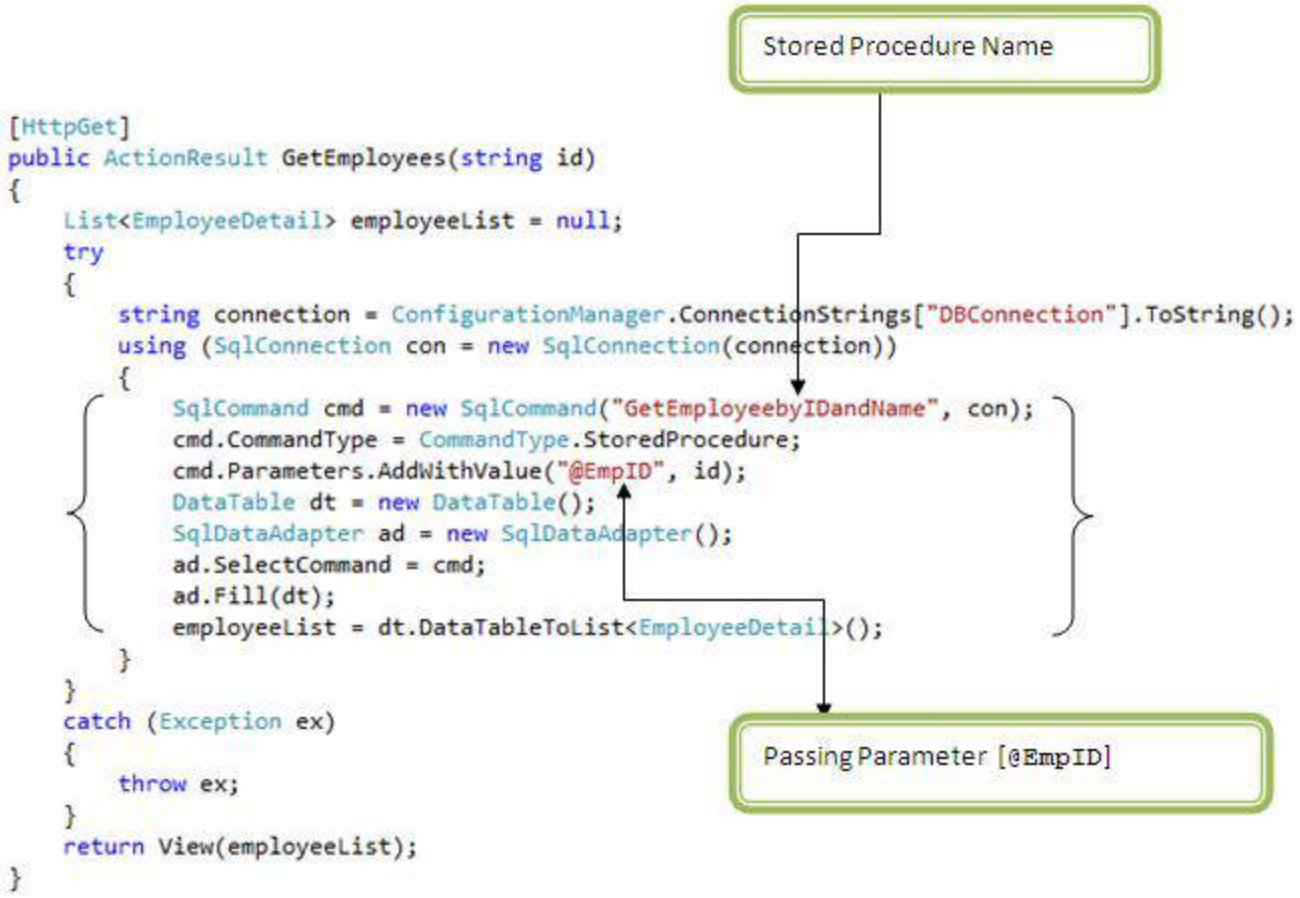


Fig 1. Using Stored Procedure for displaying Employees details.

After Using Stored Procedure lets Request GetEmployee View

Employee View which displays record.

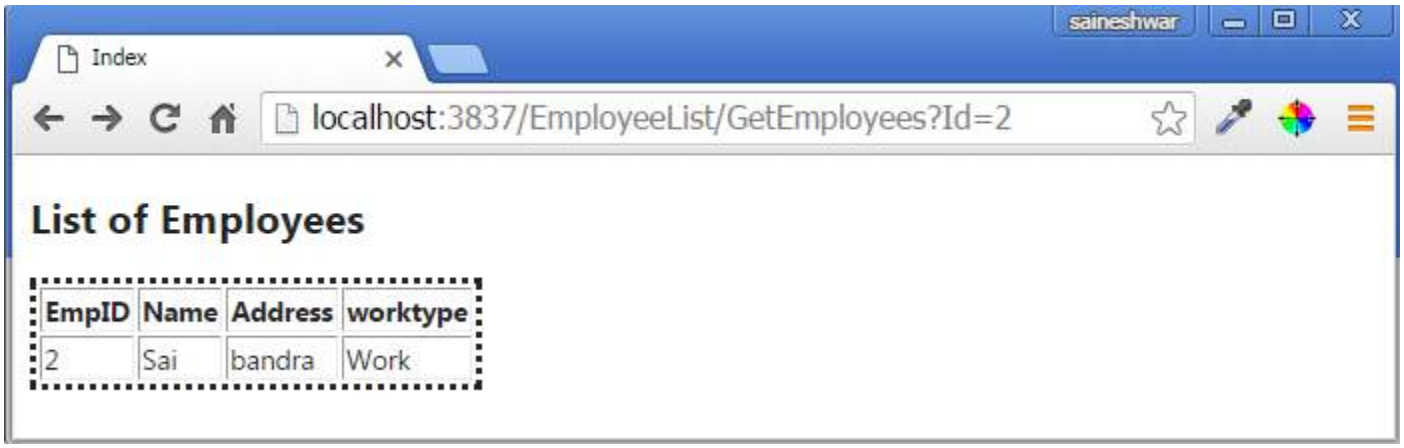


Fig 2. Employees details View after using stored procedure.

Profiler View after using Stored Procedure

Displaying Trace of store procedure which we have used.

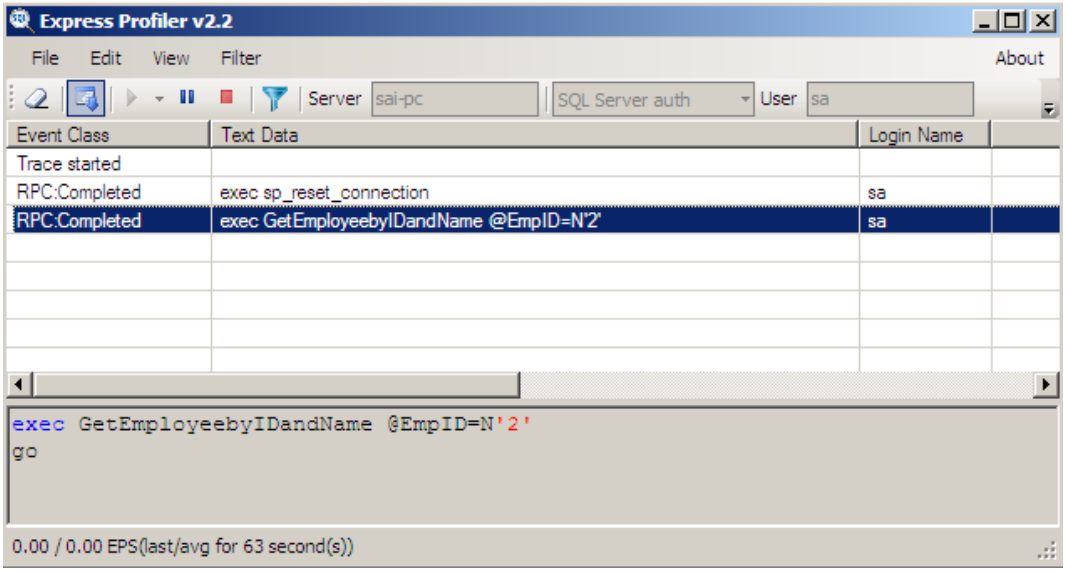


Fig 3. Trace of stored procedure Executed.

After Using Stored Procedure Let's try SQL injection Attack Again

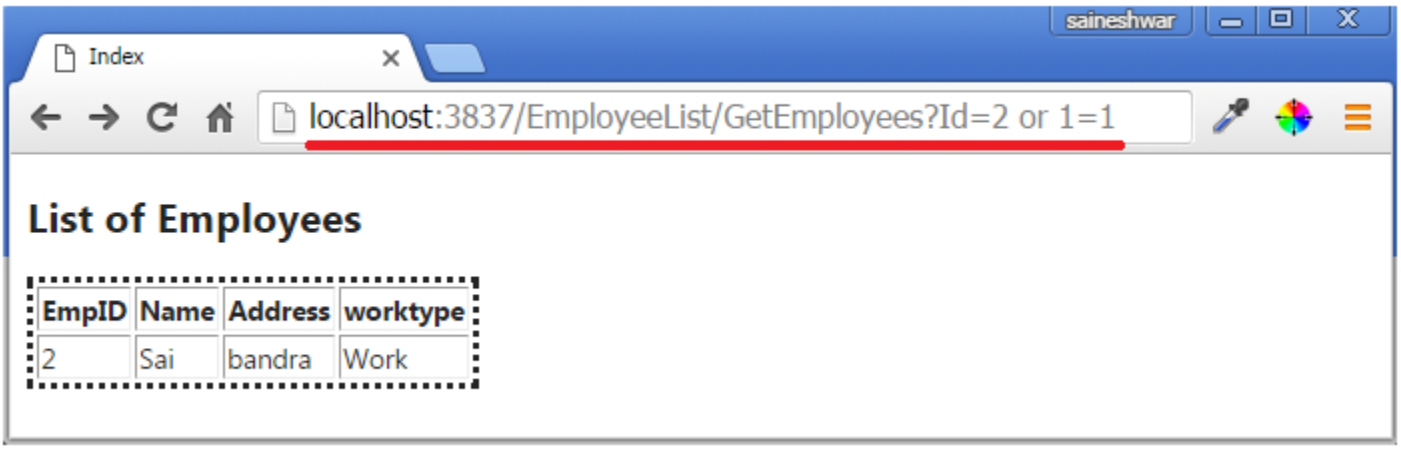


Fig 4. Trying SQL injection Attack after using stored procedure.

Displaying SQL injection attack execution in Debug Mode after using Stored Procedure

If you have look on parameter id [?Id=2 or 1=1] which contains Malicious SQL script after passing it to stored procedure it shows an error which indicates that it not get executed because parameter which we are passing is an integer which only take numeric values as input if we pass Malicious SQL script [?Id=2 or 1=1] it throws an error .

```
Create PROCEDURE GetEmployeebyIDandName
     @EmpID int

AS
BEGIN
    SELECT TOP 1000 [EmpID]
      ,[Name]
      ,[Address]
      ,[Age]
      ,[Salary]
      ,[worktype]
    FROM[EmployeeDetails]
    where [EmployeeDetails].EmpID =@EmpID
END
```

Fig 5. Stored procedure Used.

```
[HttpGet]
public ActionResult GetEmployees(string id)
{
    List<EmployeeDetail> employeeList = null;
    try
    {
        string connection = ConfigurationManager.ConnectionStrings["DBConnection"].ToString();
        using (SqlConnection con = new SqlConnection(connection))
        {
            SqlCommand cmd = new SqlCommand("GetEmployeebyIDandName", con);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@EmpID", id);
            DataTable dt = new DataTable();
            SqlDataAdapter ad = new SqlDataAdapter();
            ad.SelectCommand = cmd;
            ad.Fill(dt);
            employeeList = dt.DataTableToList<EmployeeDetail>();
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return View(employeeList);
}
```

id "2 or 1=1"

SqlException was caught

Conversion failed when converting the varchar value '2 or 1=1' to data type int.

Troubleshooting tips:

Get general help for this exception.

Fig 6. Trying SQL injection Attack after using stored procedure.

PROFILER VIEW AFTER USING STORED PROCEDURE

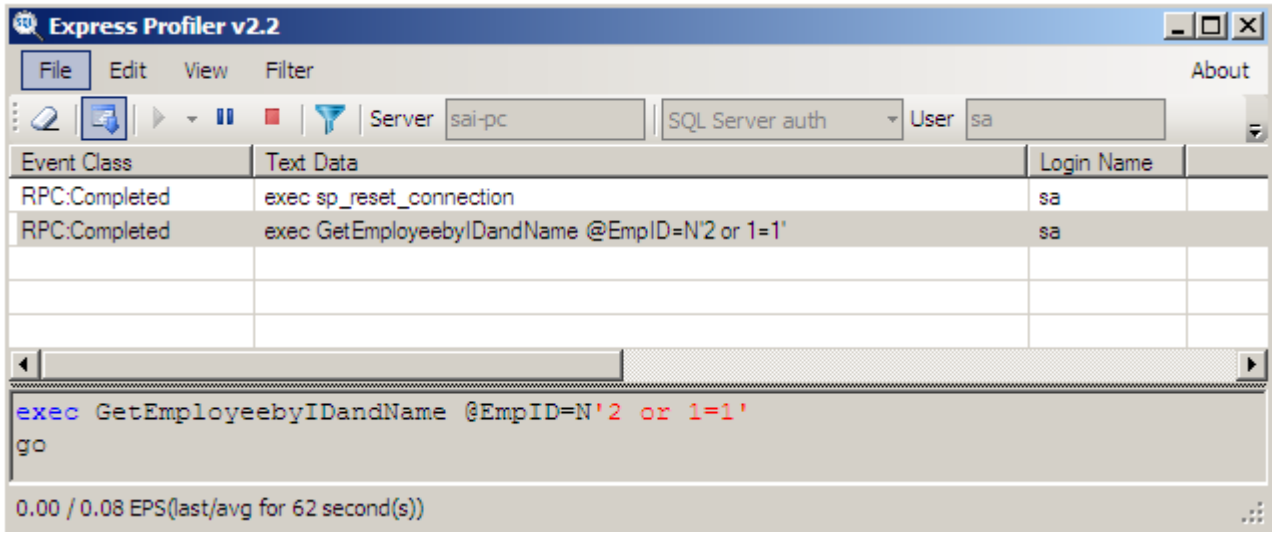


Fig 7. Trace of stored procedure Executed.

OUTPUT:-

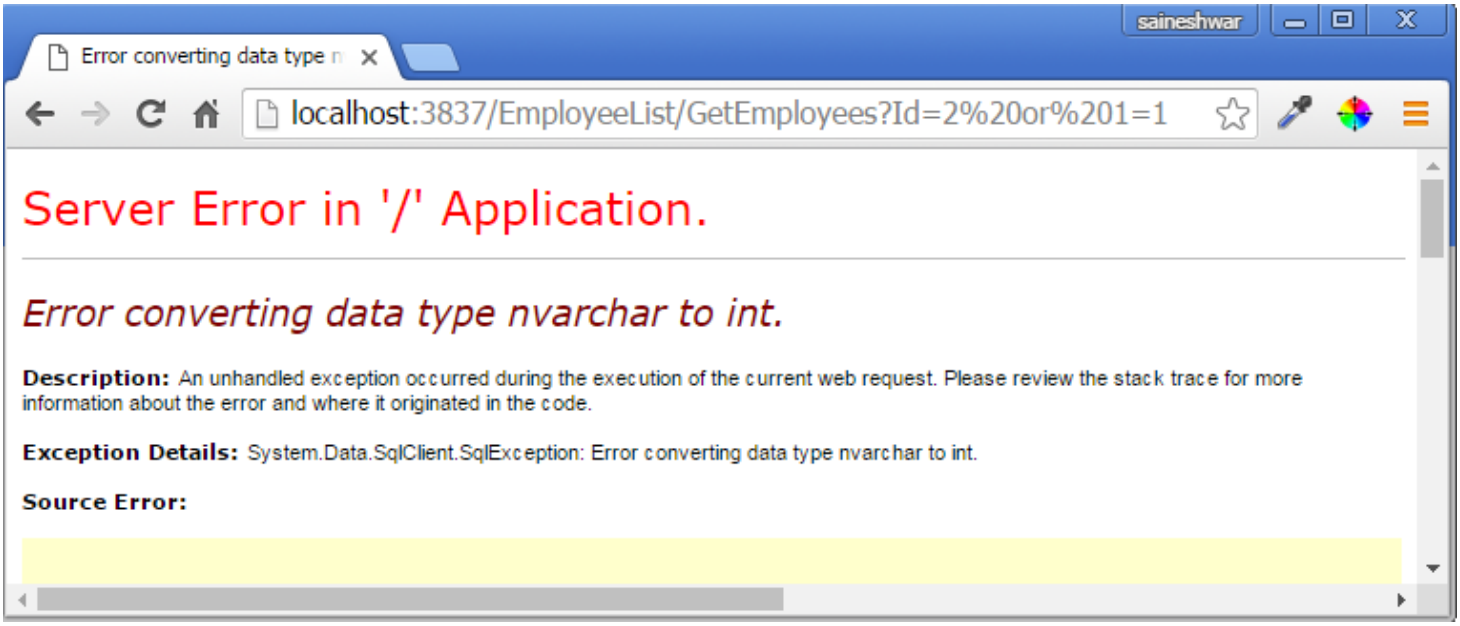


Fig 8. Displaying error after attacking by an attacker.

Meanwhile, you might think that if it was varchar datatype then we had successfully attack let's see a demo of it too.

Displaying SQL injection attack execution in Debug Mode after using Stored Procedure with different parameter @name

This is the second demo in which we have passed a different parameter to store procedure to see is it real protecting against SQL injection attack.

```
alter PROCEDURE GetEmployeebyIDandName
     @name varchar(50)
AS
BEGIN

    SELECT TOP 1000 [EmpID]
      ,[Name]
      ,[Address]
      ,[Age]
      ,[Salary]
      ,[worktype]
    FROM[EmployeeDetails]
    where [EmployeeDetails].name =@name

END
```

Fig 9. Stored procedure Used.

```
[HttpGet]
public ActionResult GetEmployees(string name)
{
    List<EmployeeDetail> employeeList = null;
    try
    {
        string connection = ConfigurationManager.ConnectionStrings["DBConnection"].ToString();
        using (SqlConnection con = new SqlConnection(connection))
        {
            SqlCommand cmd = new SqlCommand("GetEmployeebyIDandName", con);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.AddWithValue("@name", name);
            DataTable dt = new DataTable();
            SqlDataAdapter ad = new SqlDataAdapter();
            ad.SelectCommand = cmd;
            ad.Fill(dt);
            employeeList = dt.DataTableToList<EmployeeDetail>();
        }
    }
    catch (Exception
    {
        throw ex;
    }
    return View(emplo
}
```

name "Saineshwar or 1=1"

name "Saineshwar or 1=1"

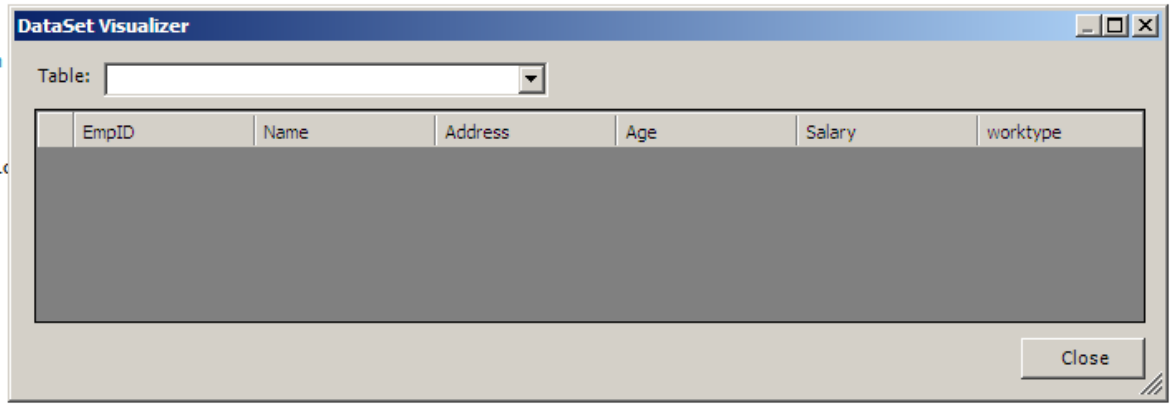


Fig 10. Trying SQL injection Attack after using stored procedure.

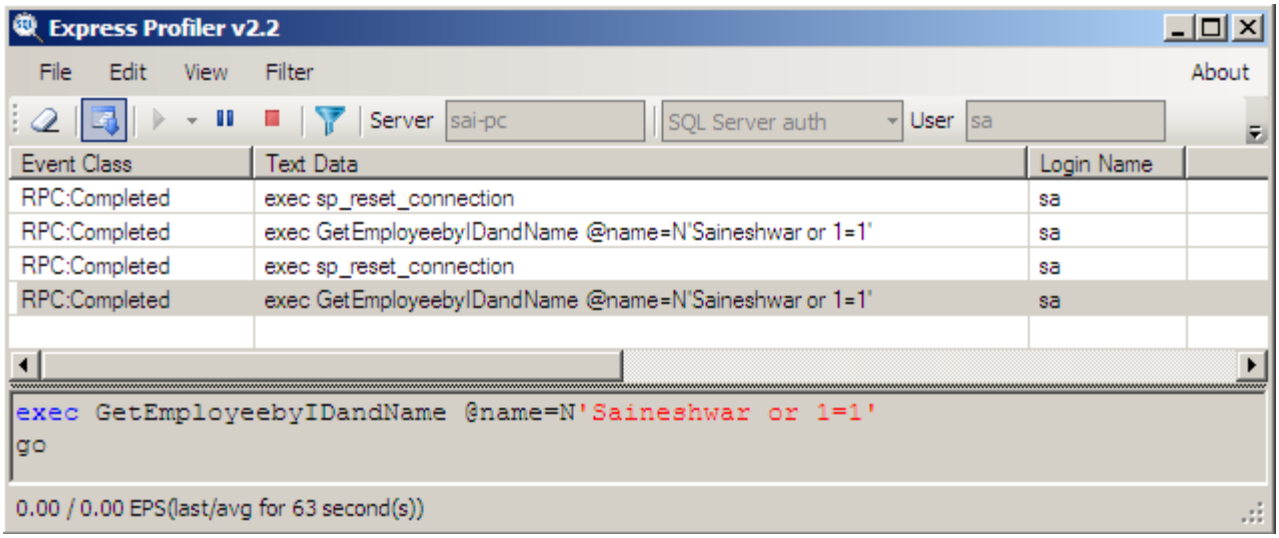


Fig 11. Trace of stored procedure Executed.

OUTPUT:-

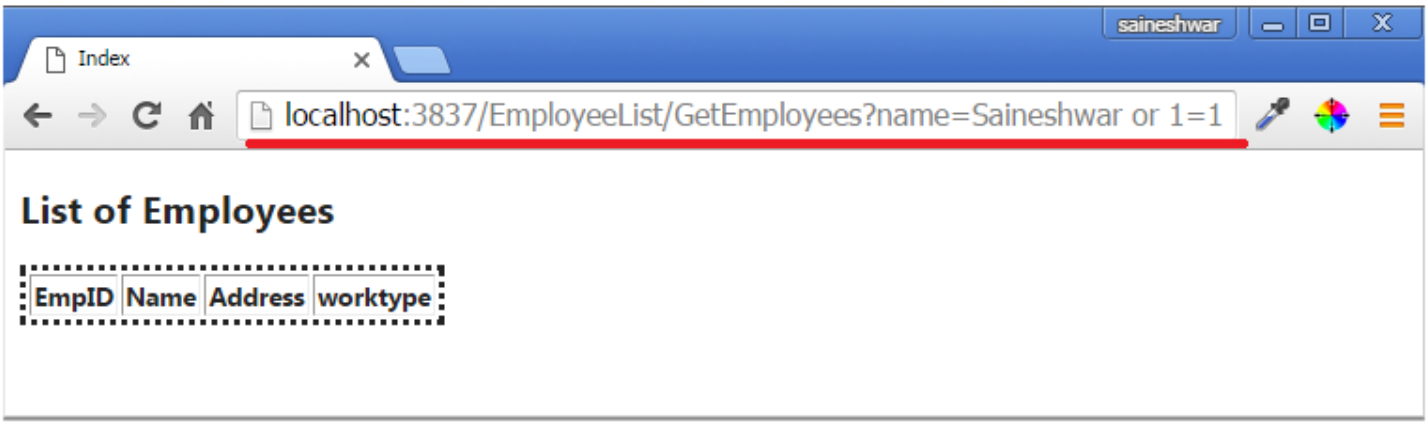
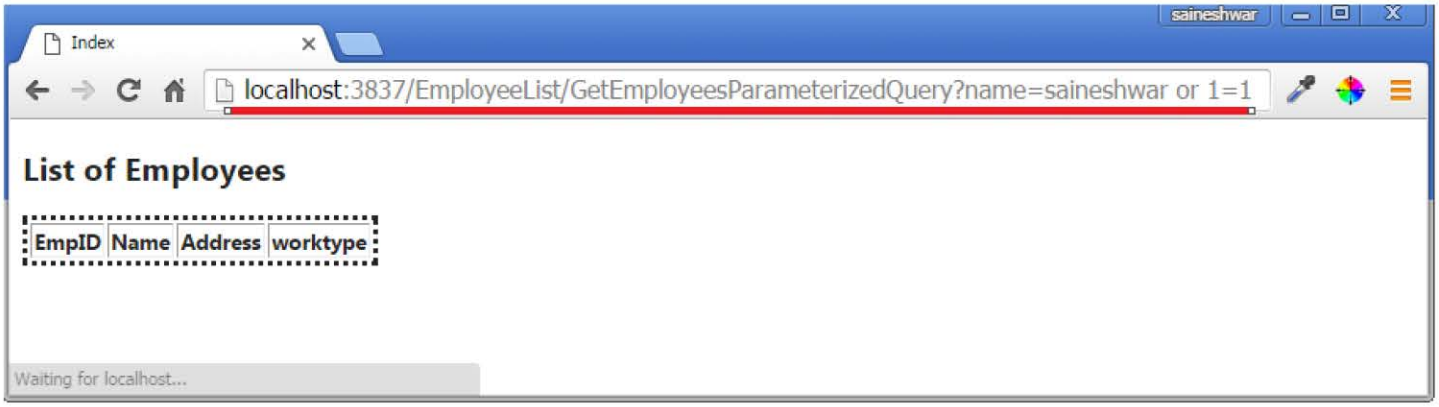


Fig 12. After using stored procedure if we try SQL injection attack then it does not get executed and no result is shown.

Use Parameterized queries

Using parameterized Queries is another solution to prevent against SQL injection attack it is similar to stored procedure. Instead of Concating string, you need to add parameters to the SQL Query and pass parameter value using the SqlCommand .



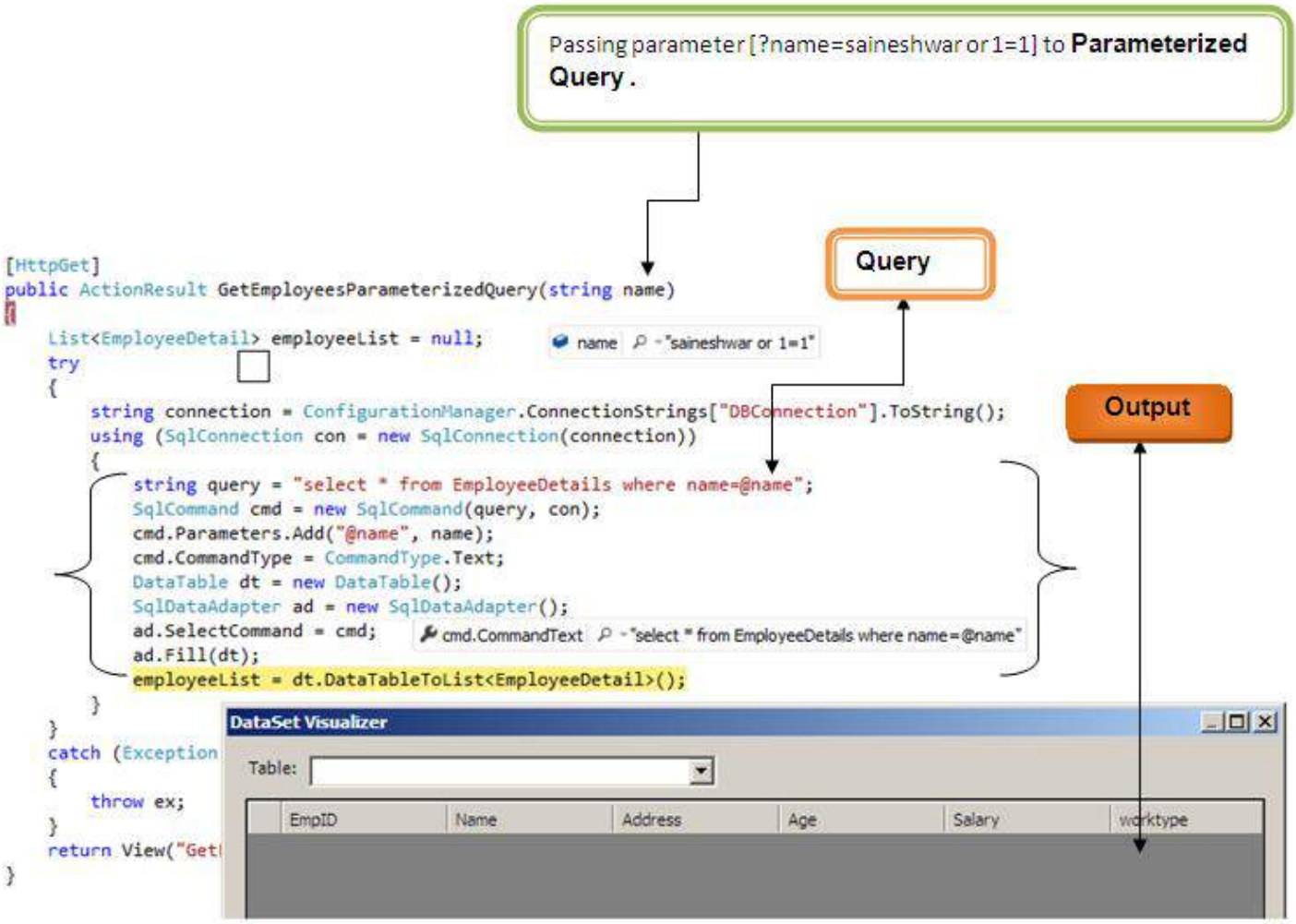


Fig 1. Parameterized Query also prevents against SQL injection attack.

PROFILER VIEW OF PARAMETERIZED QUERY

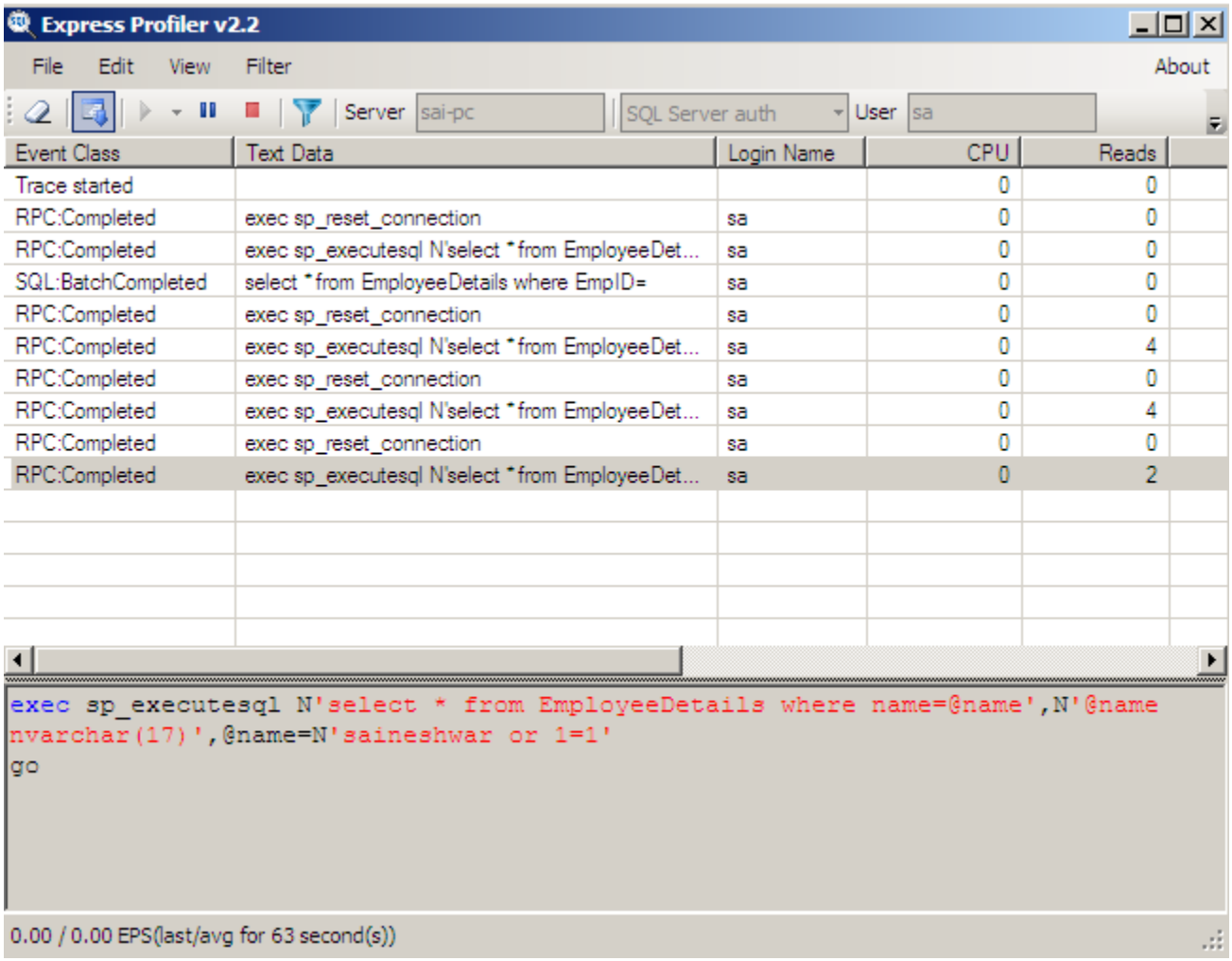


Fig 2. Trace of Parameterized query Executed.

OUTPUT:-

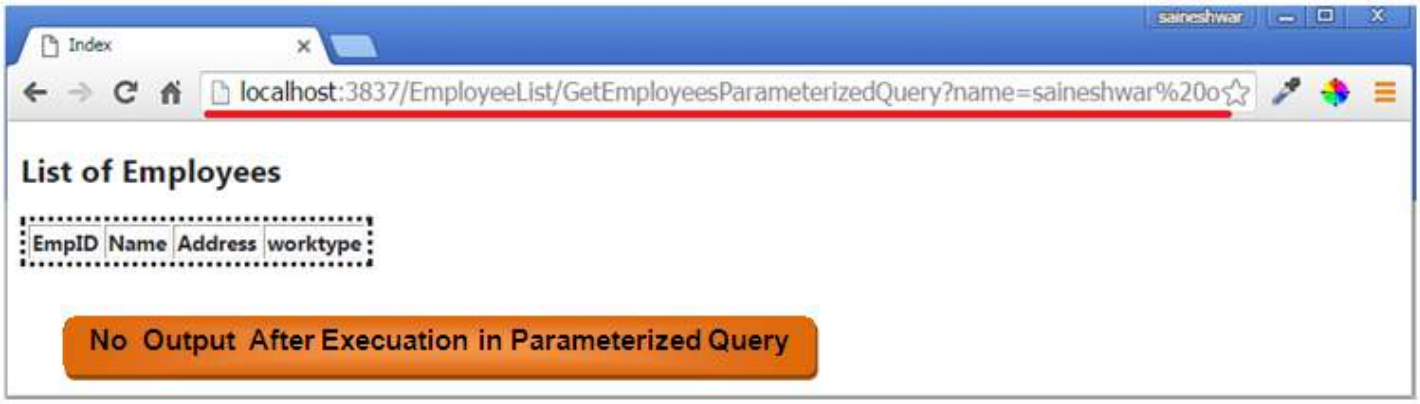


Fig 3. After using Parameterized query if we try SQL injection attack then it does not get executed and no result is shown.

Use ORM (Entity framework)

ORM stands for Object relational Mapper which maps SQL object to you domain object [**C#**].

If you are using entity framework properly you are not prone to SQL injection attack because entity framework internal uses parameterized query.

Normal execution of entity framework

Here we have passed the name as parameter [?name=Saineshwar].

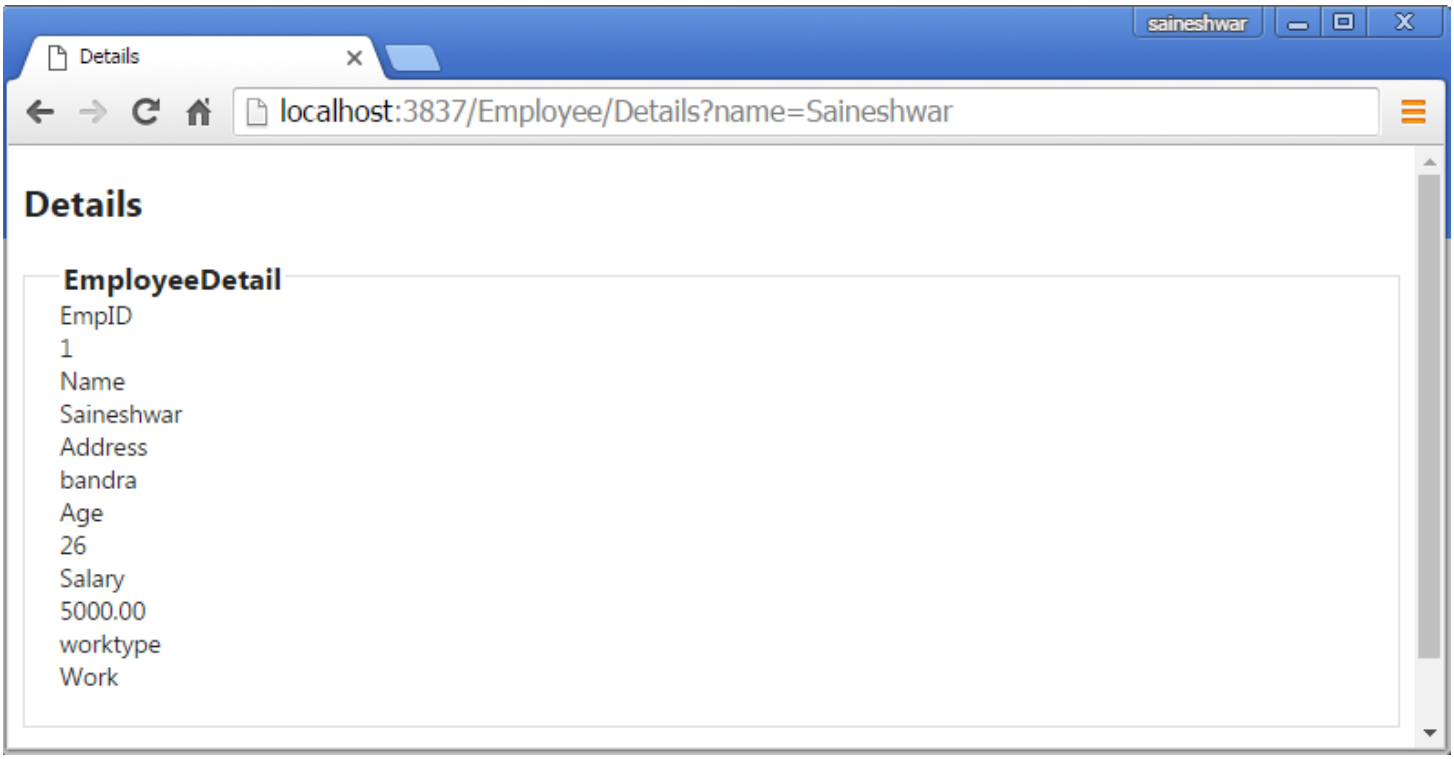


Fig 1.Passing parameter after using Entity Framework.

SNAPSHOT OF THE CONTROLLER WHILE EXECUTION .

In debug mode we can see that we have passed name parameter from Querystring which we then passed it to Linq query to getting records.

```
public class EmployeeController : Controller
{
    private AllSampleCodeEntities db = new AllSampleCodeEntities();  // DbContext class

    // ...

    public ActionResult Index()...

    // ...

    public ActionResult Details(string name)   name = "Saineshwar"
    {

        var data = (from a in db.EmployeeDetails
                    where a.Name == name
                    select a).SingleOrDefault();  //Displaying records According to Name


        EmployeeDetail employeedetail = data;   data {MvcSecurity.Models.EmployeeDetail}
        if (employeedetail == null)                 Address  "bandra"
        {                                           Age      26
            return HttpNotFound();                  EmpID    1
        }                                           Name     "Saineshwar"
        return View(employeedetail);                Salary   5000
    }                                               worktype "Work"


    protected override void Dispose(bool disposing)...
}
```

Fig 2. Controller Execution.

PROFILER VIEW OF LINQ QUERY

Entity Framework internally uses Parameterized query its true here is a snapshot.

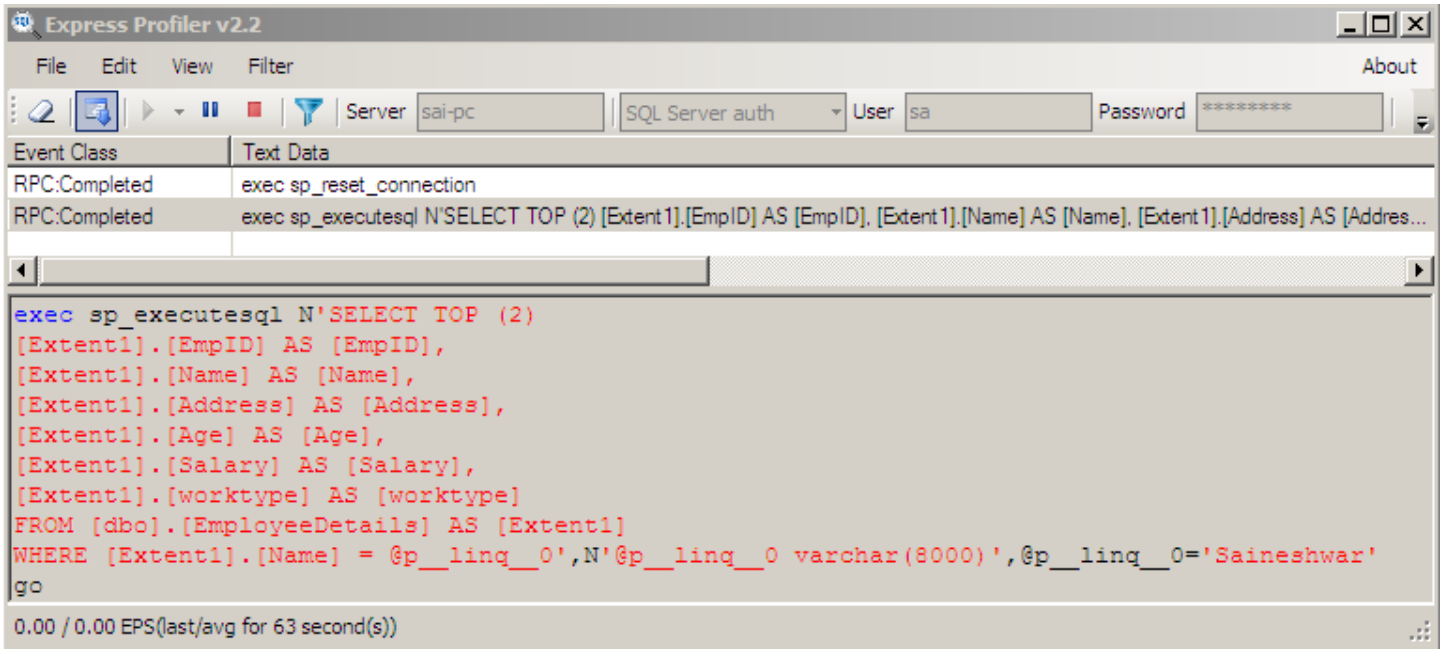


Fig 3. Trace of the query generated by Entity Framework.

LET'S TRY SQL INJECTION ATTACK ON ENTITY FRAMEWORK

In this part we are trying SQL injection attack on entity framework for that we have passed the parameter as [**?name=Saineshwar or 1=1**].

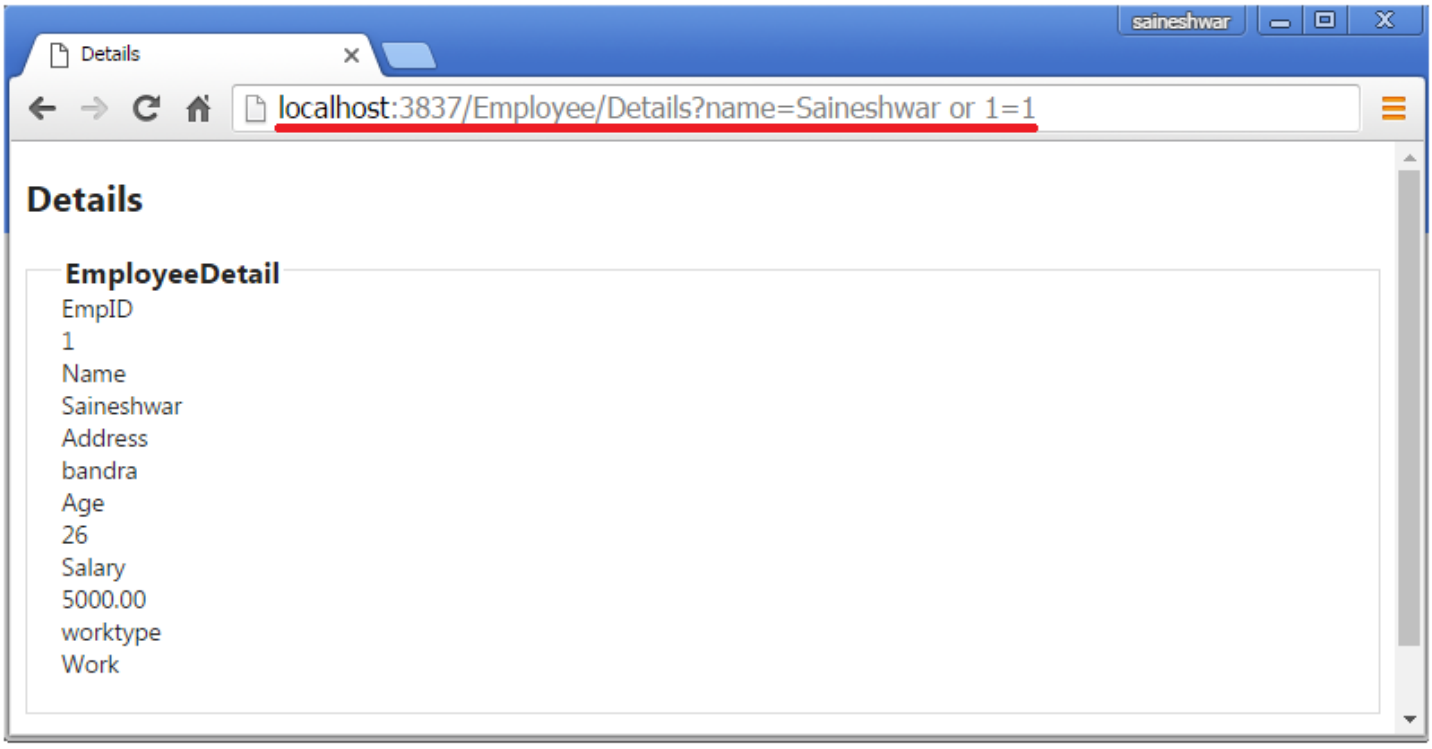


Fig 4. Trying SQL injection Attack after using Entity Framework.

SNAPSHOT OF THE CONTROLLER WHILE EXECUTION .

In debug mode we can see that we have passed name parameter from Querystring which we then passed it to Linq query to getting records but this time, it has malicious script along with normal parameter.

```
public class EmployeeController : Controller
{
    private AllSampleCodeEntities db = new AllSampleCodeEntities();  // DbContext class

    // ...

    public ActionResult Index()...

    // ...

    public ActionResult Details(string name)  name -"Saineshwar or 1=1"
    {

        var data = (from a in db.EmployeeDetails
                    where a.Name == name
                    select a).SingleOrDefault();  //Displaying records According to Name


        EmployeeDetail employeedetail = data;     data null
        if (employeedetail == null)
        {
            return HttpNotFound();
        }
        return View(employeedetail);
    }


    protected override void Dispose(bool disposing)...
}
```

Fig 5. No output after try SQL injection Attack on Entity Framework.

PROFILER VIEW OF LINQ QUERY

If you see trace closely it is considering the name and malicious script as a single parameter which prevents it from SQL injection attack.

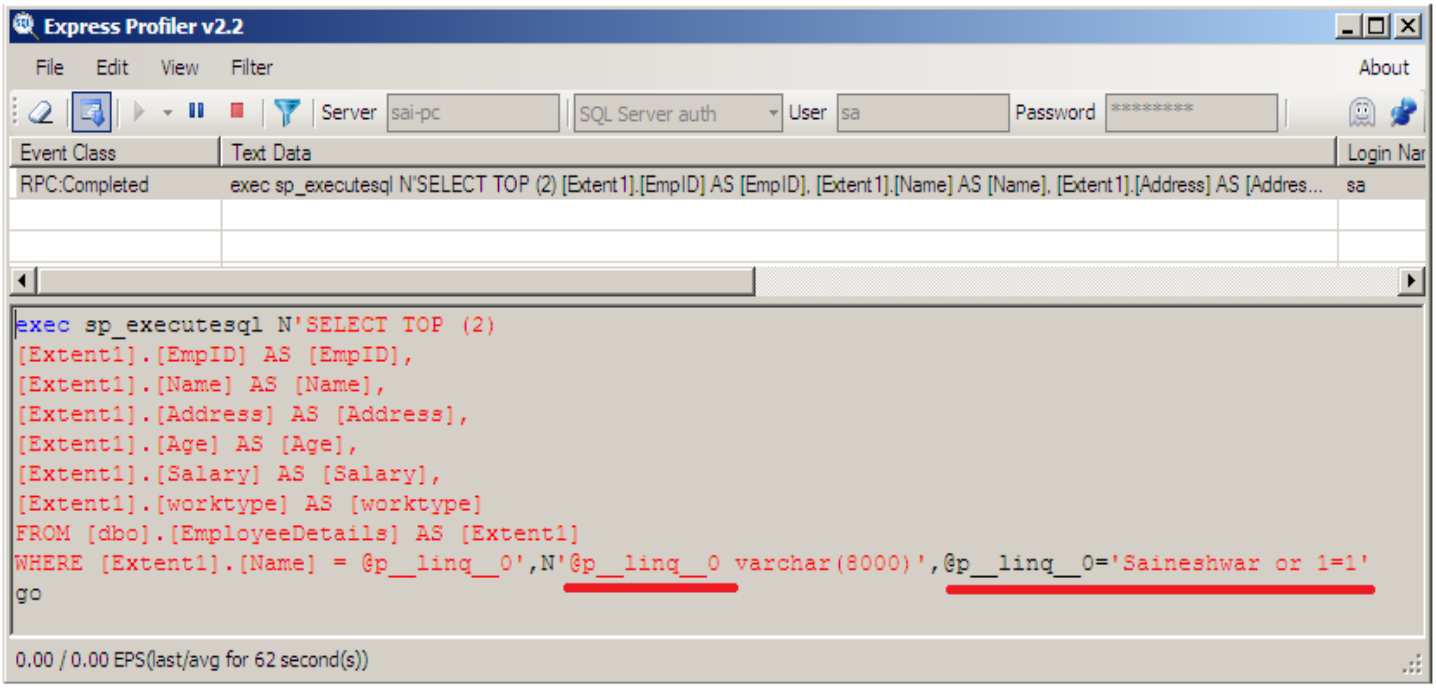


Fig 6. Trace of the query generated by Entity Framework after trying SQL injection attack.

- **Sensitive Data Exposure**

All Website and Application always have database in which entire data is been stored .mean while we store Users personal information (which may contains Password , PAN number , Passport details , Credit Card Numbers) in this we mostly encrypt only password right other data are stored in clear text which can lead to **Sensitive Data Exposure** whenever an attacker attack he gets access to database if he finds the table where all this personal and financial details stored steal that information .

Fig 1. Sensitive Data Exposure.

A simple demo of how Sensitive data is been sniffing.

SNAPSHOT OF LOGIN PAGE.

Simple code snippet of Login page which comes default if you are choosing Internet Template while creating a project.

```
<section id="loginForm">
<h2>Use a local account to log in.</h2>
@using (Html.BeginForm(new { ReturnUrl = ViewBag.ReturnUrl })) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Log in Form</legend>
        <ol>
            <li>
                @Html.LabelFor(m => m.UserName)
                @Html.TextBoxFor(m => m.UserName)
                @Html.ValidationMessageFor(m => m.UserName)
            </li>
            <li>
                @Html.LabelFor(m => m.Password)
                @Html.PasswordFor(m => m.Password)
                @Html.ValidationMessageFor(m => m.Password)
            </li>
            <li>
                @Html.CheckBoxFor(m => m.RememberMe)
                @Html.LabelFor(m => m.RememberMe, new { @class = "checkbox" })
            </li>
        </ol>
        <input type="submit" value="Log in" />
    </fieldset>
    <p>
        @Html.ActionLink("Register", "Register") if you don't have an account.
    </p>
```

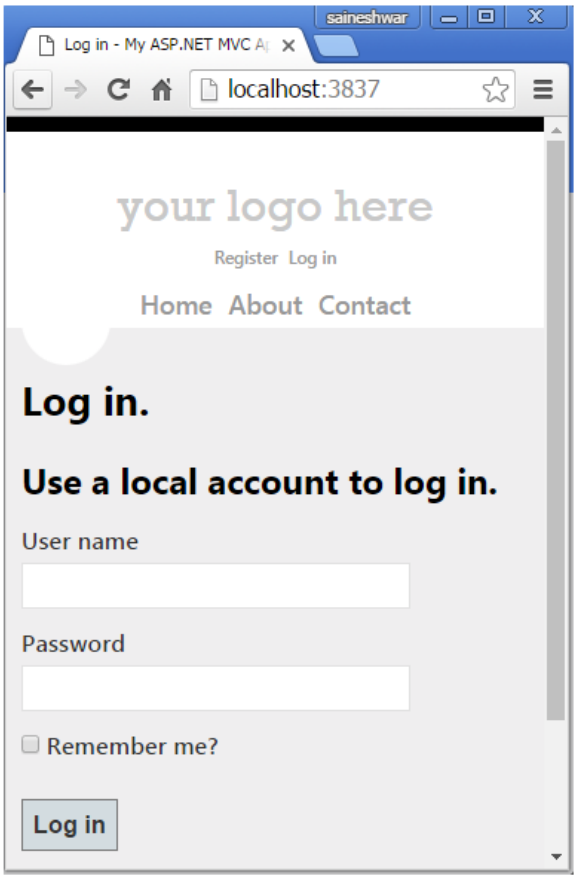


Fig 2. Login page Markup and Snapshot.

After having look at how Login page Mark and View looks now next we are going enter credentials and log in.

ENTERING CREDENTIALS FOR LOGIN

In Login page we are going to enter Username and Password to login into the application.

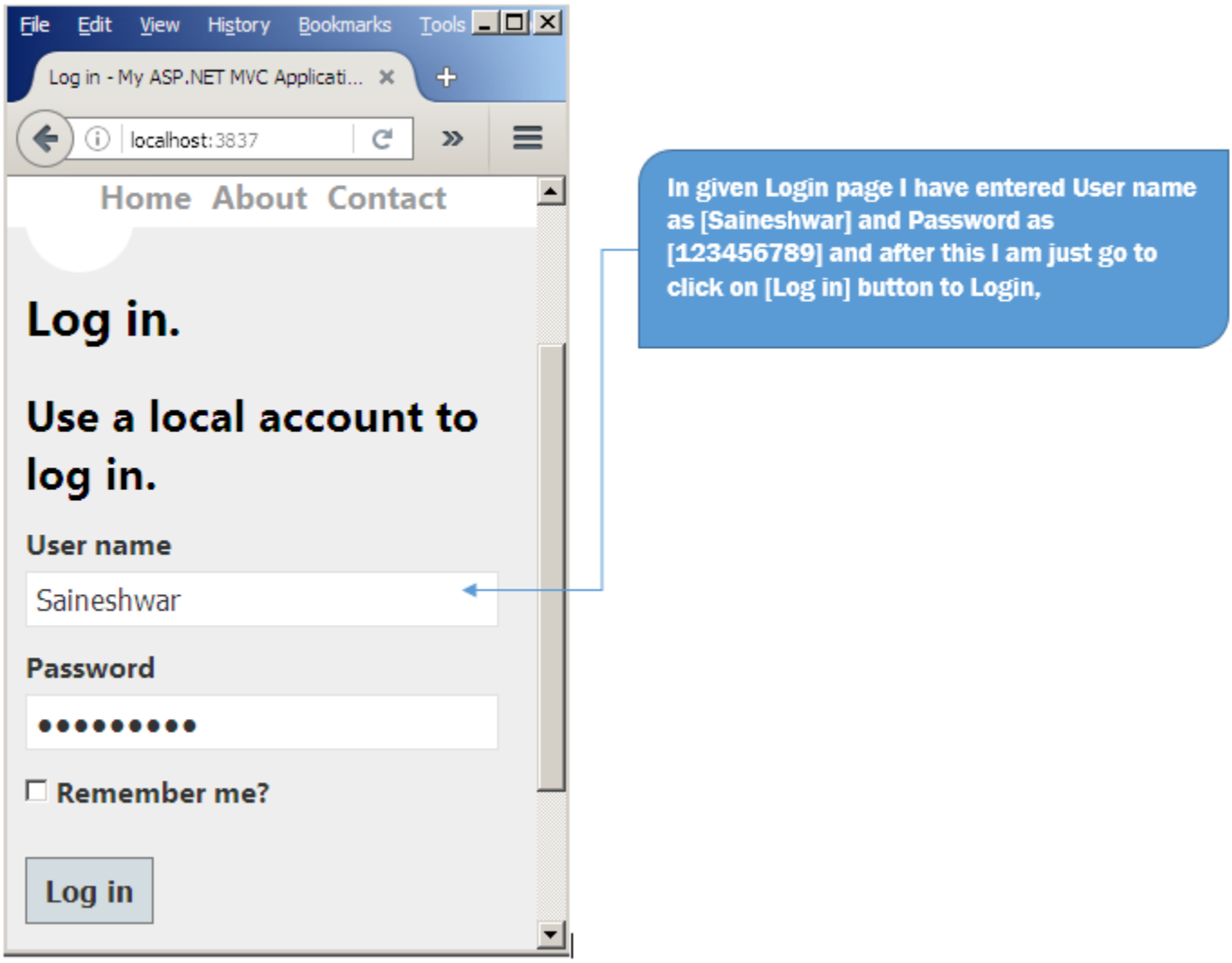


Fig 3. Entering Credentials.

When User enter Credentials in Login page and submit to server the data [**username and password**] is transferred to the server in the clear text this data [**username and password**] can be intercepted by the attacker and steal your Credentials.

Below snapshot shows how your values can be intercepted by an attacker.

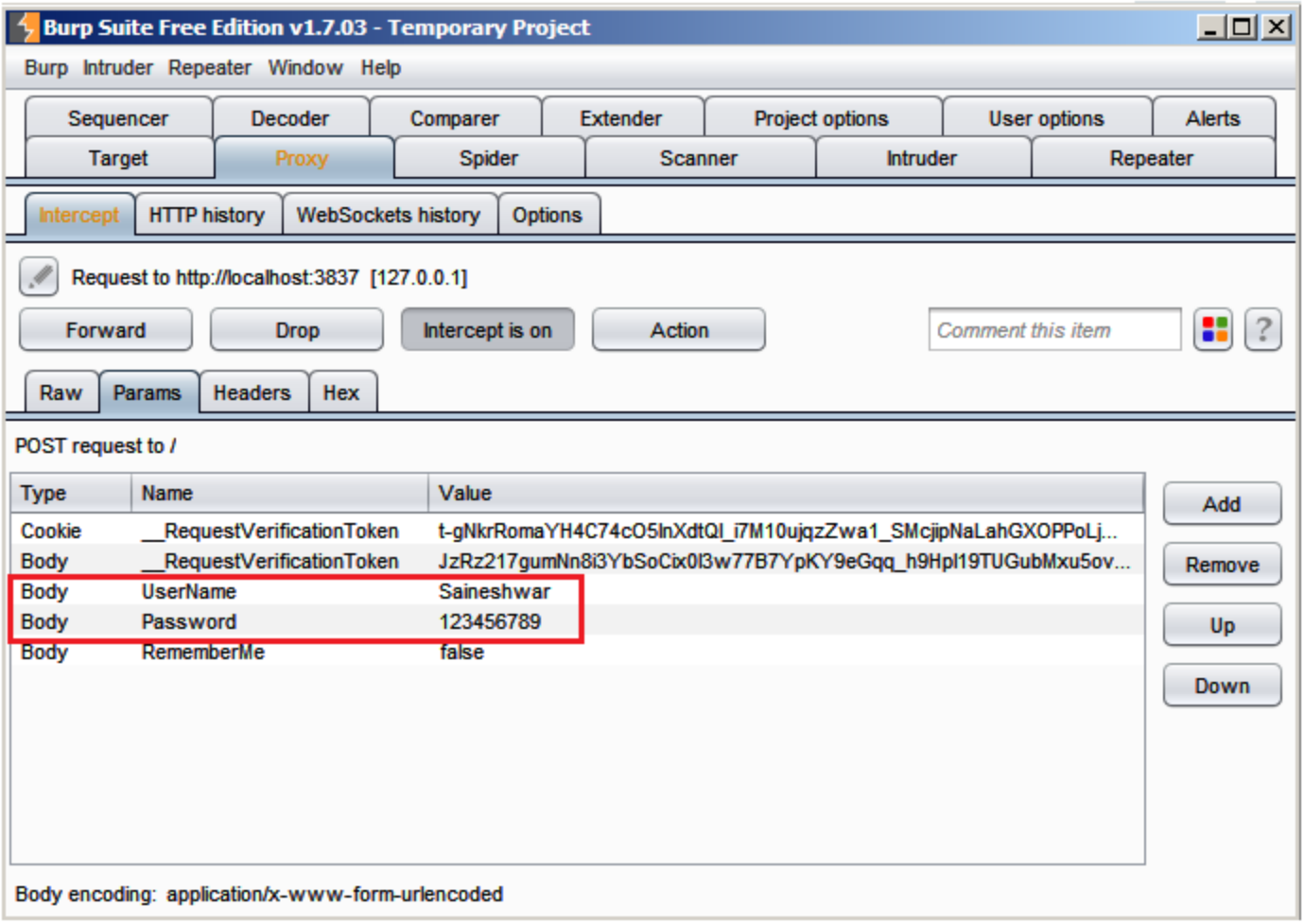


Fig 4. Intercepting Login page showing data in the Clear form.

Solution :-

1) Always Send Sensitive Data to Server in Encrypted format using Strong Hashing with Seed (Random Hash).
2) Always apply SSL to the Web application.
3) Do not store Sensitive data if want to store using Strong Hashing techniques

Always Send Sensitive Data to Server in Encrypted format using Strong Hashing with seed

In this demo, we are going use an **MD5 algorithm with seed** to encrypted data on client side and send to a server such that it cannot be stolen by the attacker.

In below snap not User entering credentials.

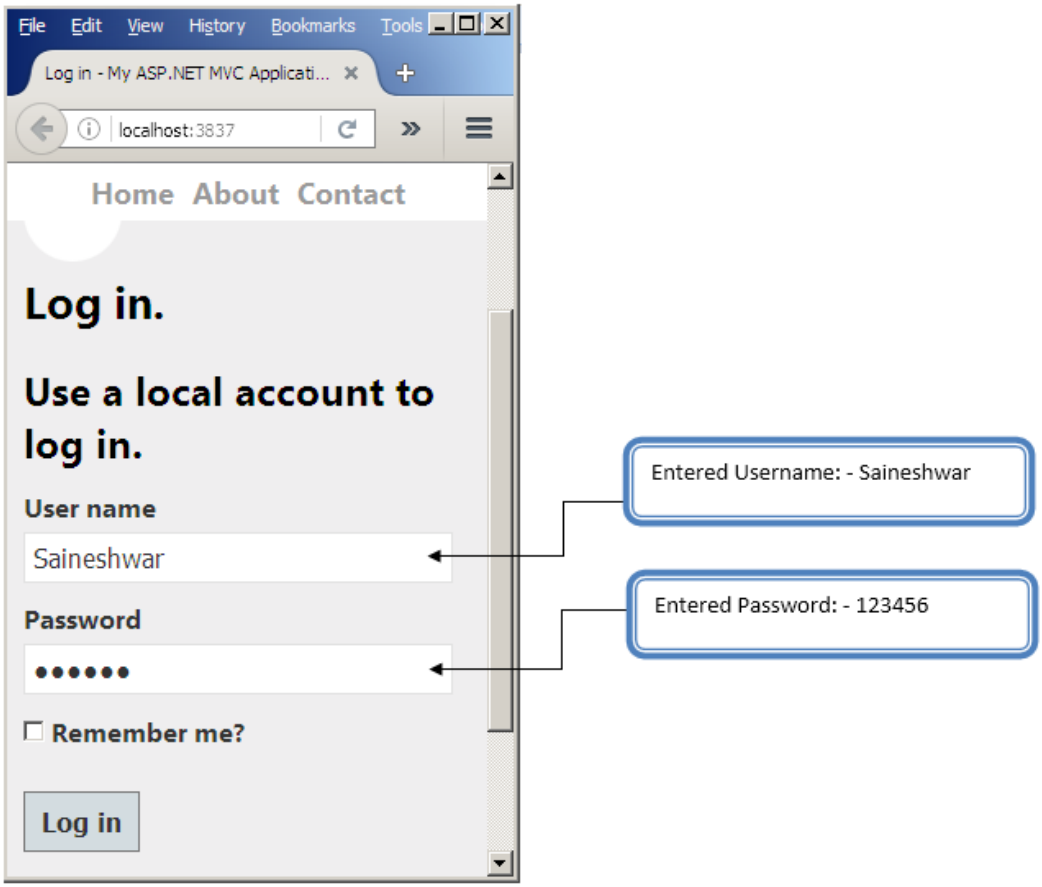


Fig 5. Entering Credentials after Encrypting data.

After entering User credentials when the user clicks on login button, this time, the user entered password get encrypted along with seed on the client side using **MD5** and then it is sent to the server , in this process if an attacker tries to sniff network then encrypted hash is only seen and which cannot be decrypted.

In details, you can see below snapshot where the attacker has sniff network.

Blue line in snapshot indicates **Username**.

The red line in snapshot indicates **Encrypted Password along with the seed**.

The green line in snapshot indicates **seed value which is generated from the server.**

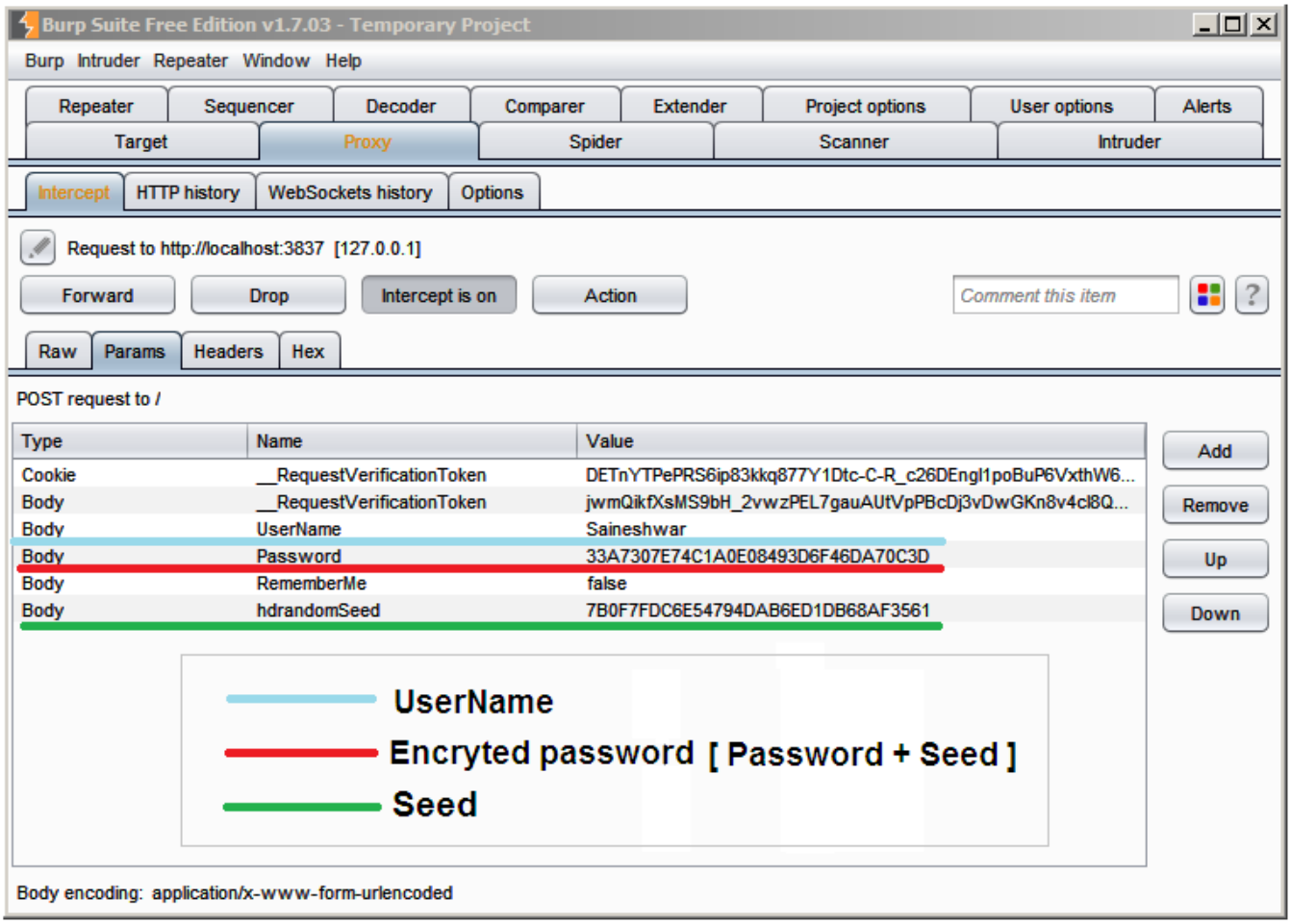


Fig 6. Intercepting login page after Encryption of data.

Till now we have seen snapshot how to do this let's see code snippet of it.

LOGIN MODEL

```
public class LoginModel
{
    [Required]
    [Display(Name = "User name")]
    public string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }

    public string hdrandomSeed { get; set; }
}
```

Fig 7. LoginModel view.

Below is code snippet of [**HttpGet**] Login ActionMethod.

In this method, we are generating random hash [Seed] and then we are going to assign it to LoginModel and then pass it to Login View.

```
[HttpGet]
[AllowAnonymous]
public ActionResult Login(string returnUrl)
{
    LoginModel LM = new LoginModel();
    Random objRandom = new Random();

     #pragma warning disable 618
     var Seed = FormsAuthentication.HashPasswordForStoringInConfigFile(Convert.ToString(objRandom.Next()), "MD5");
     #pragma warning restore 618

    LM.hdrandomSeed = Seed;
    return View(LM);
}
```

Seed "23AF700A5FB28B4BAE1F926D595603FE"

LM.hdrandomSeed "23AF700A5FB28B4BAE1F926D595603FE"

Fig 7. Generating and Assigning Random Hash to [hdrandomSeed] and then sending Model to View.

After assigning value [**hdrandomSeed**] to model in Login ActionMethod now we are going to use it on Login view as a hidden field.

```
@using (Html.BeginForm("Login", "Account", FormMethod.Post, new { id = "frm" }))
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Log in Form</legend>
        <ol>
            <li>
                @Html.LabelFor(m => m.UserName)
                @Html.TextBoxFor(m => m.UserName)
                @Html.ValidationMessageFor(m => m.UserName)
            </li>
            <li>
                @Html.LabelFor(m => m.Password)
                @Html.PasswordFor(m => m.Password)
                @Html.ValidationMessageFor(m => m.Password)
            </li>
            <li>
                @Html.CheckBoxFor(m => m.RememberMe)
                @Html.LabelFor(m => m.RememberMe, new { @class = "checkbox" })
            </li>
        </ol>
        <input type="submit" id="btnLogin" value="Log in" />
    </fieldset>
    <p>
        @Html.ActionLink("Register", "Register") if you don't have an account.
    </p>
    @Html.HiddenFor(m => m.hdrandomSeed)
}
```

Fig 7.Using the [hdrandomSeed] property as a hidden field on View.

Now after adding a hidden field on the page now let's see how to Encrypt data using MD5 javascript along with the seed.

For doing this we are going to use **jquery library 1.8** and **md5.js library** when user enter Credentials and click on Log In button then we take password entered by User [`var password1 = $('#Password');`] and generate

Hash [**calcMD5(password).toUpperCase()**] of it first then along with seed we again generate Hash [**var hash = calcMD5(seed + calcMD5(password).toUpperCase());**] which is unique and then it is sent to server.

SEE BELOW SNAPSHOT FOR DETAILS

```
<script src="~/Scripts/jquery-1.8.2.min.js"></script>
<script src="~/Scripts/md5.js"></script>
<script type="text/javascript">
    $(document).ready(function () {
        $('#btnLogin').click(function ()
        {
            if ($('#Password').val() != "")
            {
                var seed = $('#hdrandomSeed');
                return md5auth(seed.val());
            }
            $("#frm").submit();
        });
    });

    $(function () {
        var controls = $(".disable");
        controls.bind("paste", function () {
            return false;
        });
        controls.bind("drop", function () {
            return false;
        });
        controls.bind("cut", function () {
            return false;
        });
        controls.bind("copy", function () {
            return false;
        });
    });
</script>
```

```
<script type="text/javascript">
    function md5auth(seed)
    {
        var password1 = $('#Password');
        var password = password1.val();
        var hash = calcMD5(seed + calcMD5(password).toUpperCase());
        password1.val(hash.toUpperCase());
        return true;
    }
</script>
```

Seed: -
23AF700A5FB28B4BAE1F926D595603FE

Encrypted Password: -
AC109BF72E05D9539B03FFB0B2EA7F4C

Entered Password: - 123456

Fig 8.Code Snippet of Client Side Encryption of MD5 along with Seed for generating Hash.

In the variable session, you can see real time values which are generated when user click on login button.

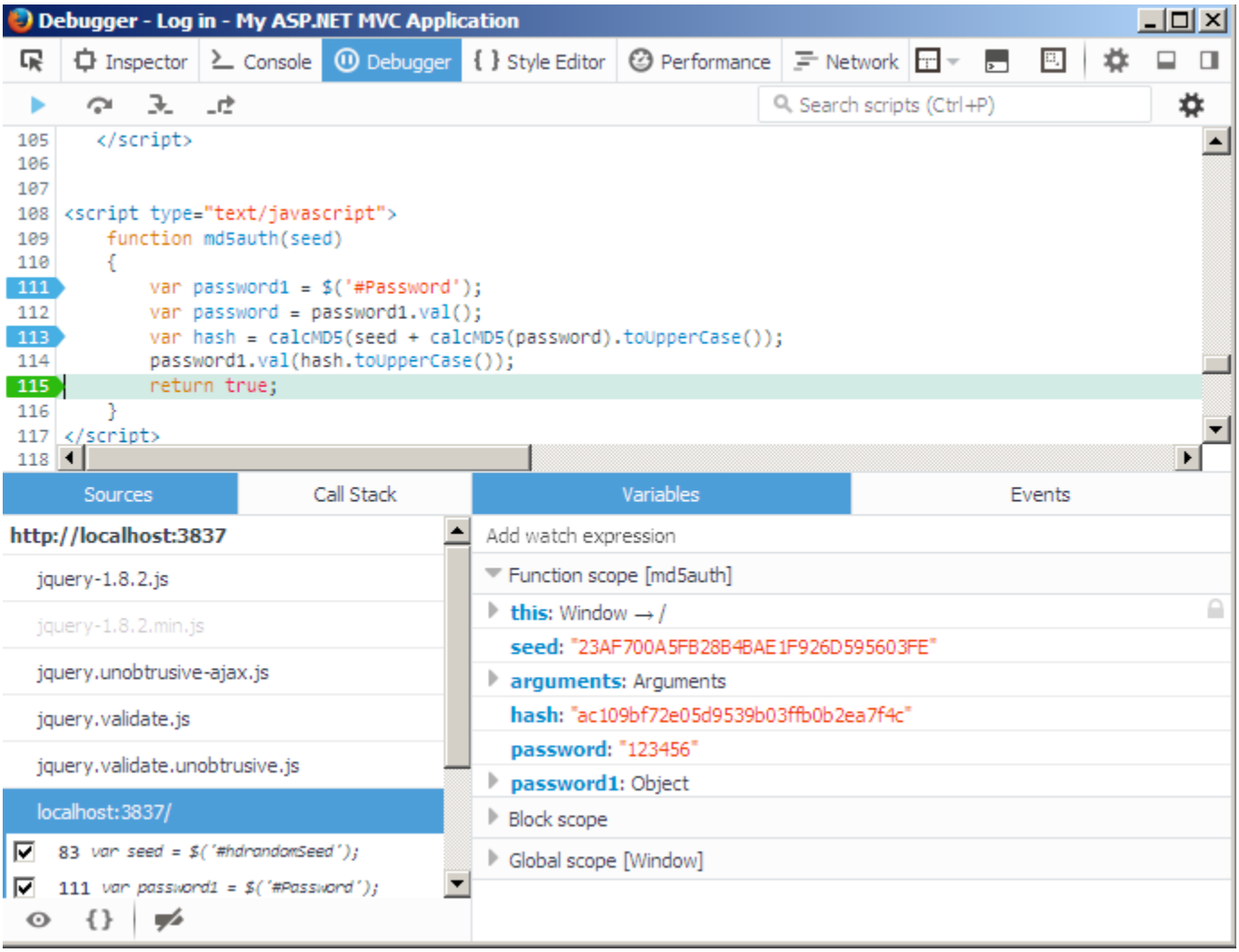


Fig 8. Debug mode of Client-side Encryption.

After client side encryption now let's see attacker is able to intercept and see the password in clear text.

INTERCEPTING LOGIN PAGE

In below snapshot, you can see password which is entered by the user is in encrypted form and it cannot be understood by an attacker because it is combination hash of Seed + Password which is entered by User.

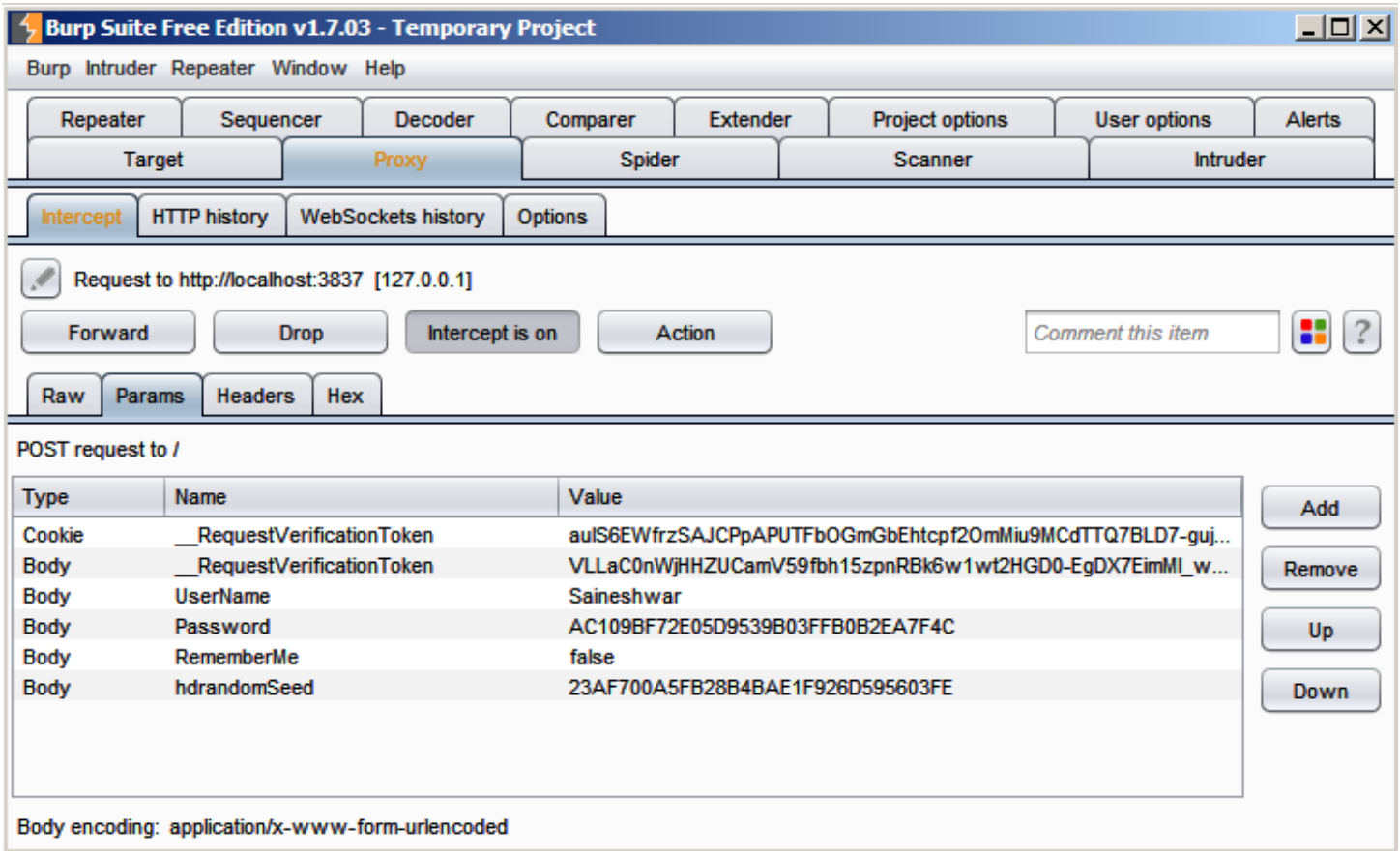


Fig 9. Interception of Login page.

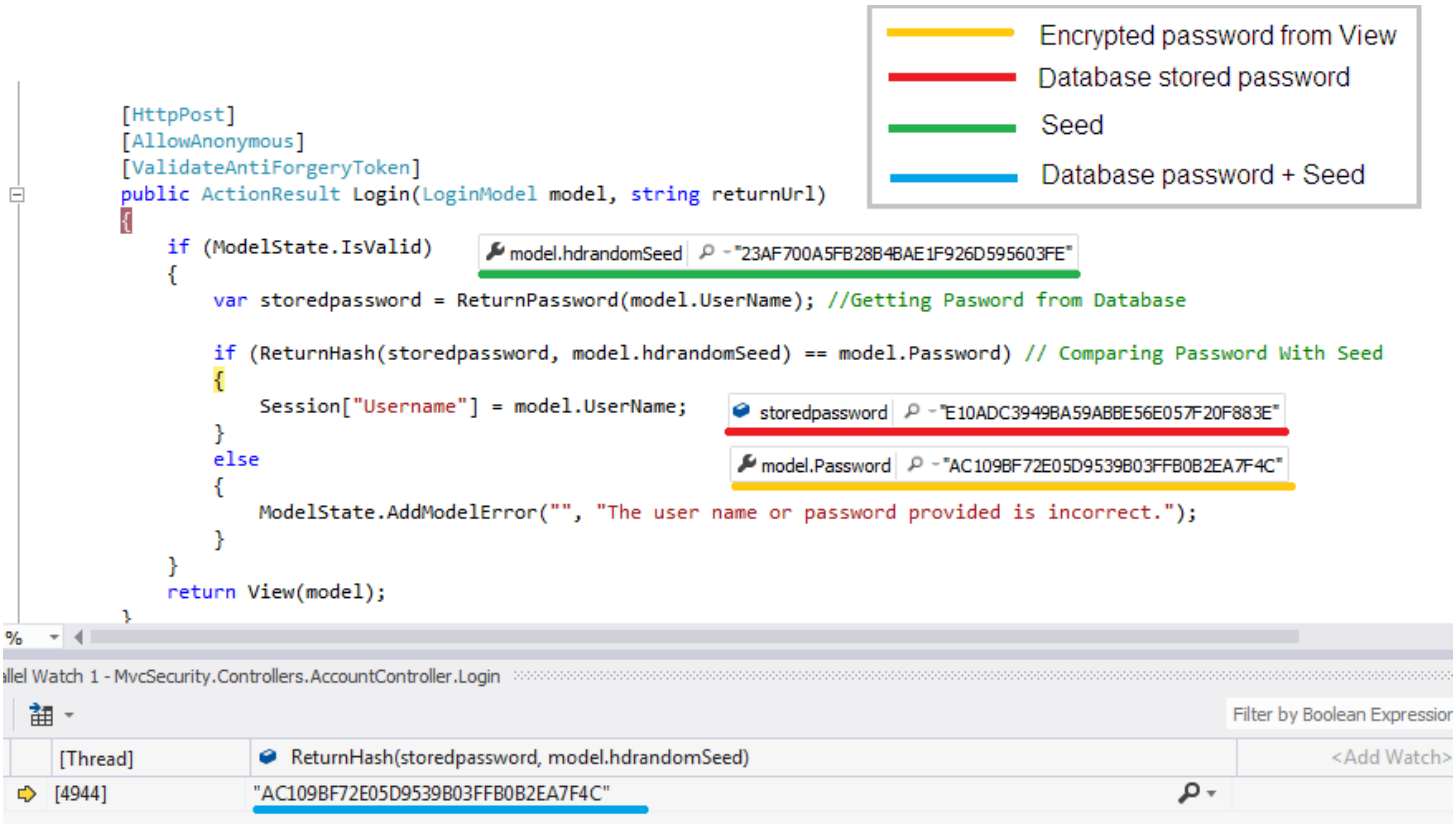
After interception next we are going to see values how they are Posted to Login Action Method.

LOGIN ACTION METHOD AFTER POSTING VALUES FROM LOGIN VIEW

Here we can clearly see password is in encrypted form .

Fig 10. Login Action Method after posting Values from Login View.

In next step, we are going to compare password which is stored in a database for doing that first we are going to get the password from username which user has entered and then we are going to compare it against which is stored in the database. The line which is marked **green** is Seed value along with that you can see Database stored password is marked as **red** which we get by passing Username and then yellow line which indicates that this password in posted from Login View and finally blue line which is combination of Database password and Seed we compare it with password which is posted.



Finally, Sensitive data entered by User is Secure.

Fig 11.Code snippet of Login Action Method with pining real time values.

2) Use SSL for Securing Web application

SSL (Secure Sockets Layer) is Layer which Secure (Encrypted) communication between client and server such that any data [Banking details, Password, and another financial transaction] passed from client and server is Secure (Encrypted) .

Fig 1.SSL (Secure Sockets Layer).

SSL is mainly Applied on the Login page and payment gateways if you want to apply on the entire application then also you can apply.

Fig 2. Browser View of SSL.

If you want to know in details how to Enable SSL on IIS Server there is a good article from Scott Guthrie Sir blog check URL below.

http://weblogs.asp.net/scottgu/tip-trick-enabling-ssl-on-iis7-using-self-signed-certificates

3) Do not store Sensitive data in Database in a clear form

Always Try not to store Credit Card, Debit Card and financial details and other Sensitive details in the database in Clear form . Always Use Strong Hashing techniques to encrypted data and then store in the database. if an attacker gets direct access to the database then all data in clear form can be Breached.

Below is a list of Algorithm which can be Used according to need .

Hash Algorithm

if someone wants just **Hash** then they can use **Hash Algorithm** we mostly use Hash function for Encrypting Password.

Symmetric Algorithm

If someone wants just one key for encryption and decryption then they can use **Symmetric Algorithm.**

Asymmetric Algorithm

If someone wants just one key for encryption (**Public key**) and another key decryption (**Private key**) then they can use **Asymmetric Algorithm. E.g** we can use this when we are sharing Web Services and WebAPI with clients when the user.

Hash Algorithm

1) MD5
2) SHA256
3) SHA384
4) SHA512

E.g

METHOD TO GENERATE MD5 HASH

```
private string Generate_MD5_Hash(string data_To_Encrypted)
{
    using (MD5 encryptor = MD5.Create())
    {
        MD5 md5 = System.Security.Cryptography.MD5.Create();
        byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(data_To_Encrypted);
        byte[] hash = md5.ComputeHash(inputBytes);

        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < hash.Length; i++)
        {
            sb.Append(hash[i].ToString());
        }

        return sb.ToString();
    }
}
```

PASS TEXT TO GENERATE HASH

```
string Hash = Generate_MD5_Hash("Hello");
```

OUTPUT:-

Symmetric Algorithm

1) Aes
2) DES
3) RC2
4) Rijndael
5) TripleDES

E.g

METHOD OF AES FOR ENCRYPTION

```
private string Encrypt_AES(string clearText)
        {
            string EncryptionKey = "##SAI##1990##"; //Encryption Key
            byte[] clearBytes = Encoding.Unicode.GetBytes(clearText);
            byte[] array = Encoding.ASCII.GetBytes("##100SAINESHWAR99##"); //salt

            using (Aes encryptor = Aes.Create())
            {
                Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, array);
                encryptor.Key = pdb.GetBytes(32);
                encryptor.IV = pdb.GetBytes(16);
                using (MemoryStream ms = new MemoryStream())
                {
                    using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(),
CryptoStreamMode.Write))
                    {
                        cs.Write(clearBytes, 0, clearBytes.Length);
                        cs.Close();
                    }
                    clearText = Convert.ToBase64String(ms.ToArray());
                }
            }
            return clearText;
        }
```

METHOD OF AES FOR DECRYPTION

```
        private string Decrypt_AES(string cipherText)
        {
            string EncryptionKey = "##SAI##1990##"; //Encryption Key
            byte[] cipherBytes = Convert.FromBase64String(cipherText);
            byte[] array = Encoding.ASCII.GetBytes("##100SAINESHWAR99##"); //salt

            using (Aes encryptor = Aes.Create())
            {
                Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, array);
                encryptor.Key = pdb.GetBytes(32);
                encryptor.IV = pdb.GetBytes(16);
                using (MemoryStream ms = new MemoryStream())
                {
                    using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateDecryptor(),
CryptoStreamMode.Write))
                    {
                        cs.Write(cipherBytes, 0, cipherBytes.Length);
                        cs.Close();
                    }
                    cipherText = Encoding.Unicode.GetString(ms.ToArray());
                }
            }
            return cipherText;
        }
```

PASS TEXT TO ENCRYPT VALUE

```
string DataEncrypt = Encrypt_AES("Hello");   // Encrypting Data (Pass text to Encrypt)
```

PASS TEXT TO DECRYPT VALUE

```
string DataDecrypt = Decrypt_AES(DataEncrypt);  // Decrypt data (Pass Encrypt text to Decrypt)
```

OUTPUT:-

```
public ActionResult Index()  DataEncrypt - "bzlwVgMGAkZPErCNO5OBrA=="
{
    string DataEncrypt = Encrypt_AES("Hello");    // Encrypting Data (Pass text to Encrypt)

    string DataDecrypt = Decrypt_AES(DataEncrypt);  // Decrypt data (Pass Encrypt text to Decrypt)

    return View();          DataDecrypt - "Hello"
}
```

Asymmetric Algorithm

1) DSA
2) ECDiffieHellman
3) ECDsa
4) RSA

E.g

METHOD OF RSA FOR ENCRYPTION

```
public byte[] Encrypt(string publicKeyXML, string dataToDycript)
{
    RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
    rsa.FromXmlString(publicKeyXML);
    return rsa.Encrypt(ASCIIEncoding.ASCII.GetBytes(dataToDycript), true);
}
```

METHOD OF RSA FOR DECRYPTION

```
public string Decrypt(string publicPrivateKeyXML, byte[] encryptedData)
{
    RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
    rsa.FromXmlString(publicPrivateKeyXML);
    return ASCIIEncoding.ASCII.GetString(rsa.Decrypt(encryptedData, true));
}
```

OUTPUT:-

```
public ActionResult Index()
{
    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();
    string publicPrivateKeyXML = RSA.ToXmlString(true); //true is Private key
    string publicOnlyKeyXML = RSA.ToXmlString(false);  //false is public key
    byte[] Encryptbytes = Encrypt(publicOnlyKeyXML, "Hello");  // Use Public Key for Encryption
    string DecryptData = Decrypt(publicPrivateKeyXML, Encryptbytes); // Use Private Key for Decryption
    // For Decryption required Private Key + Encryptbytes
    return View();
}
```

publicPrivateKeyXML "<RSAKeyValue><Modulus>lHtq5bpjSfBliljwPAfBqC0c3TyYE+ShXfvRjvuM6

publicOnlyKeyXML "<RSAKeyValue><Modulus>lHtq5bpjSfBliljwPAfBqC0c3TyYE+ShXfvRjvuM6

Encryptbytes {byte[128]}

DecryptData "Hello"

- **Audit trail**

Audit Trail in IT World is used to keep track of User activity on a Web application which he using , it is where important in detecting security problems, performance problems, and Applications Level Error problems. It also helps us to easily track where the problem actually is and resolve it.

Fig 1.Audit.

Solution:-

1) Keep Audit Trail of all User activity on Web Application and always Monitor it.

KEEP AUDIT TRAIL OF ALL USER ACTIVITY ON WEB APPLICATION AND ALWAYS MONITOR IT.

For Maintaining Audit Trail first we are going to create a table in the database for storing Audit data with table name [**AuditTB**] After that we going create an ActionFilterAttribute with Name [**UserAuditFilter**] inside this on

Action Executing we are going to write code for inserting data in the database of the user who are accessing our Application.

AUDITTB TABLE VIEW

In this table, we have taken common fields which we required to recognize User and his Activity.

Column Name	Data Type	Allow Nulls
UsersAuditID	int	☐
UserID	int	☐
SessionID	varchar(50)	☑
IPAddress	varchar(50)	☑
PageAccessed	varchar(300)	☑
LoggedInAt	datetime	☑
LoggedOutAt	datetime	☑
LoginStatus	nvarchar(50)	☑
ControllerName	varchar(50)	☑
ActionName	varchar(50)	☑
		☐

Fig 2. AuditTB Table View.

After displaying table view now let's have look on Model which is Creating according to Table .

AUDITTB MODEL

```
public partial class AuditTB
{
    public int UsersAuditID { get; set; }
    public int UserID { get; set; }
    public string SessionID { get; set; }
    public string IPAddress { get; set; }
    public string PageAccessed { get; set; }
    public Nullable<System.DateTime> LoggedInAt { get; set; }
    public Nullable<System.DateTime> LoggedOutAt { get; set; }
    public string LoginStatus { get; set; }
    public string ControllerName { get; set; }
    public string ActionName { get; set; }
}
```

Fig 3. AuditTB Model.

After having look on Model which is generated now let's create an ActionFilter with **UserAuditFilter**

UserAuditFilter Action filter Code Snippet

UserAuditFilter is a Custom ActionFilter which we have created in this filter we insert data of User Activity into AuditTB table along with this we also check is User Logged in or not into application in the same way we also insert IP address of User who is accessing the application , and time stamp of User logged in and out. For inserting this data we use ORM entity Framework.

```
public class UserAuditFilter : ActionFilterAttribute
    {
        public override void OnActionExecuting(ActionExecutingContext filterContext)
        {
            AllSampleCodeEntities appcontext = new AllSampleCodeEntities();
            AuditTB objaudit = new AuditTB();
            //Getting Action Name
            string actionName = filterContext.ActionDescriptor.ActionName;
            //Getting Controller Name
            string controllerName = filterContext.ActionDescriptor.ControllerDescriptor.ControllerName;
            var request = filterContext.HttpContext.Request;
            if (HttpContext.Current.Session["UserID"] == null) // For Checking User is Logged in or Not
            {
            objaudit.UserID = 0;
            }
            else
            {
            objaudit.UserID = Convert.ToInt32(HttpContext.Current.Session["UserID"]);
            }
            objaudit.UsersAuditID = 0;
            objaudit.SessionID = HttpContext.Current.Session.SessionID; // Application SessionID
```

```
            // User IPAddress
            objaudit.IPAddress =
            request.ServerVariables["HTTP_X_FORWARDED_FOR"] ?? request.UserHostAddress;
            objaudit.PageAccessed = request.RawUrl;  // URL User Requested
            objaudit.LoggedInAt = DateTime.Now;       // Time User Logged In || And time User Request
Method
            if (actionName == "LogOff")
            {
                objaudit.LoggedOutAt = DateTime.Now; // Time User Logged OUT
            }
            objaudit.LoginStatus = "A";
            objaudit.ControllerName = controllerName; // ControllerName
            objaudit.ActionName = actionName;         // ActionName
            appcontext.AuditTBs.Add(objaudit);
            appcontext.SaveChanges();                    // Saving in database using Entity Framework
            base.OnActionExecuting(filterContext);
        }
    }
```

Fig 4. Code Snippet of UserAuditFilter.

REGISTERING USERAUDITFILTER IN GLOBAL ACTION FILTER

Global action filters are mainly used for error handling and logging.

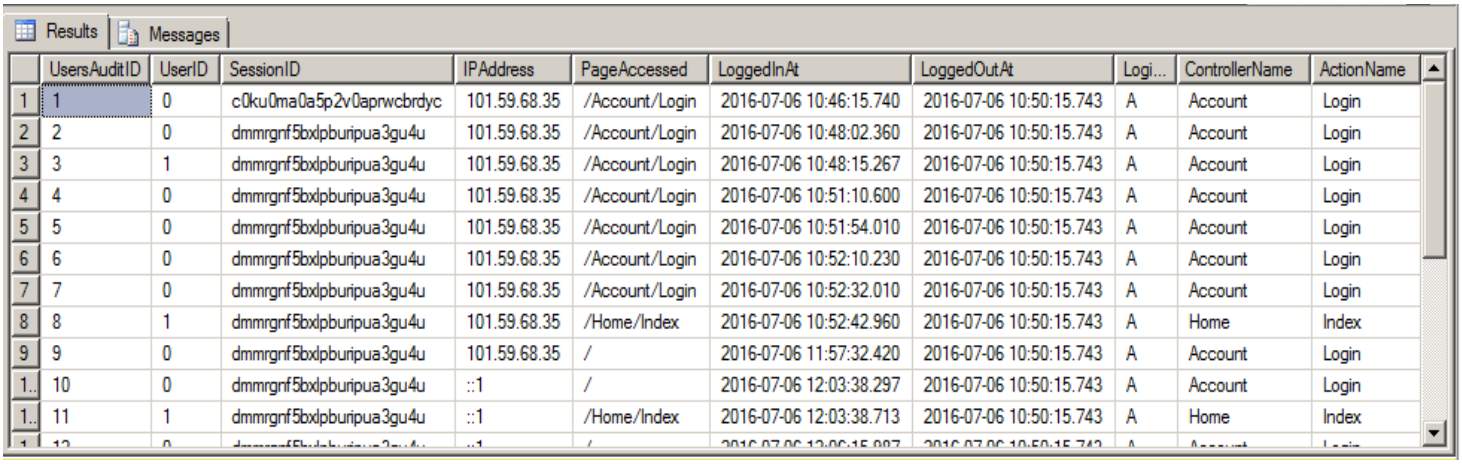
If you have a condition in which you want to apply Action Filter on all action Method in the project then you can use global action filters. Here we need global action filters because we want to track every request of the user for Audit trail hence we have used.

```
using MvcSecurity.Filters;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace MvcSecurity
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801

    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            WebApiConfig.Register(GlobalConfiguration.Configuration);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
            MvcHandler.DisableMvcResponseHeader = true;
            GlobalFilters.Filters.Add(new UserAuditFilter()); // Register UserAuditFilter
        }

        protected void Application_PreSendRequestHeaders()...
    }
}
```

Fig 5.Adding UserAuditFilter to Global filter Collection.

OUTPUT:-

The data which got inserted while User Request pages and do some activity in Audit Table.



Results | Messages

	UsersAuditID	UserID	SessionID	IPAddress	PageAccessed	LoggedInAt	LoggedOutAt	Logi...	ControllerName	ActionName
1	1	0	c0ku0ma0a5p2v0aprwcbrdyc	101.59.68.35	/Account/Login	2016-07-06 10:46:15.740	2016-07-06 10:50:15.743	A	Account	Login
2	2	0	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Account/Login	2016-07-06 10:48:02.360	2016-07-06 10:50:15.743	A	Account	Login
3	3	1	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Account/Login	2016-07-06 10:48:15.267	2016-07-06 10:50:15.743	A	Account	Login
4	4	0	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Account/Login	2016-07-06 10:51:10.600	2016-07-06 10:50:15.743	A	Account	Login
5	5	0	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Account/Login	2016-07-06 10:51:54.010	2016-07-06 10:50:15.743	A	Account	Login
6	6	0	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Account/Login	2016-07-06 10:52:10.230	2016-07-06 10:50:15.743	A	Account	Login
7	7	0	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Account/Login	2016-07-06 10:52:32.010	2016-07-06 10:50:15.743	A	Account	Login
8	8	1	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/Home/Index	2016-07-06 10:52:42.960	2016-07-06 10:50:15.743	A	Home	Index
9	9	0	dmmrgnf5bxlpburipua3gu4u	101.59.68.35	/	2016-07-06 11:57:32.420	2016-07-06 10:50:15.743	A	Account	Login
1..	10	0	dmmrgnf5bxlpburipua3gu4u	::1	/	2016-07-06 12:03:38.297	2016-07-06 10:50:15.743	A	Account	Login
1..	11	1	dmmrgnf5bxlpburipua3gu4u	::1	/Home/Index	2016-07-06 12:03:38.713	2016-07-06 10:50:15.743	A	Home	Index

Fig 6.Audit table View after Insertion of data.

- **Broken authentication and session management**

If Authentication and Session management are not properly Implemented in a web application which may allow an attacker to **steal a password , session tokens , cookies** and such issue may also allow an attacker to get access to the entire application and breach all user credentials.

Ways attacker can steal data

1) Not Secure connection (Not Using SSL)
2) Predictable login credentials
3) Not storing credentials Encrypted form.
4) Improper Application logout.

ATTACKS POSSIBLE

1) Session Fixation

Before finding a solution how to prevent this attack lets have a look on small demo how Session Fixation attack occurs.

Whenever a user sends first request to server the login page is loaded then User enter valid Login credits to login in web application after successful login we assign some values in session to recognize Unique User, meanwhile a [**"ASP.NET_SessionId"**] cookie is Added to the browser for recognizing specific User who has sent a request and [**"ASP.NET_SessionId"**] cookie value will be always sent to the server on every request till you are not logout from the application and on logout, we basically write code for removing session values which are created but we do not remove [**"ASP.NET_SessionId"**] cookie which is created on Login . this value helps an attacker to perform Session Fixation attack.

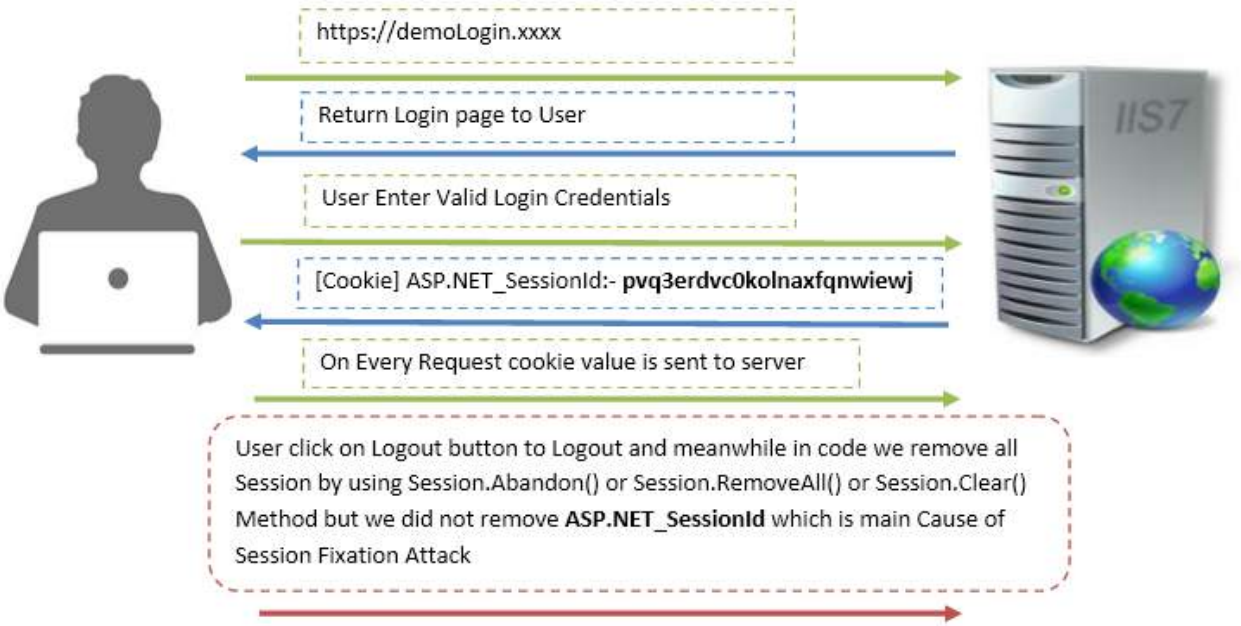
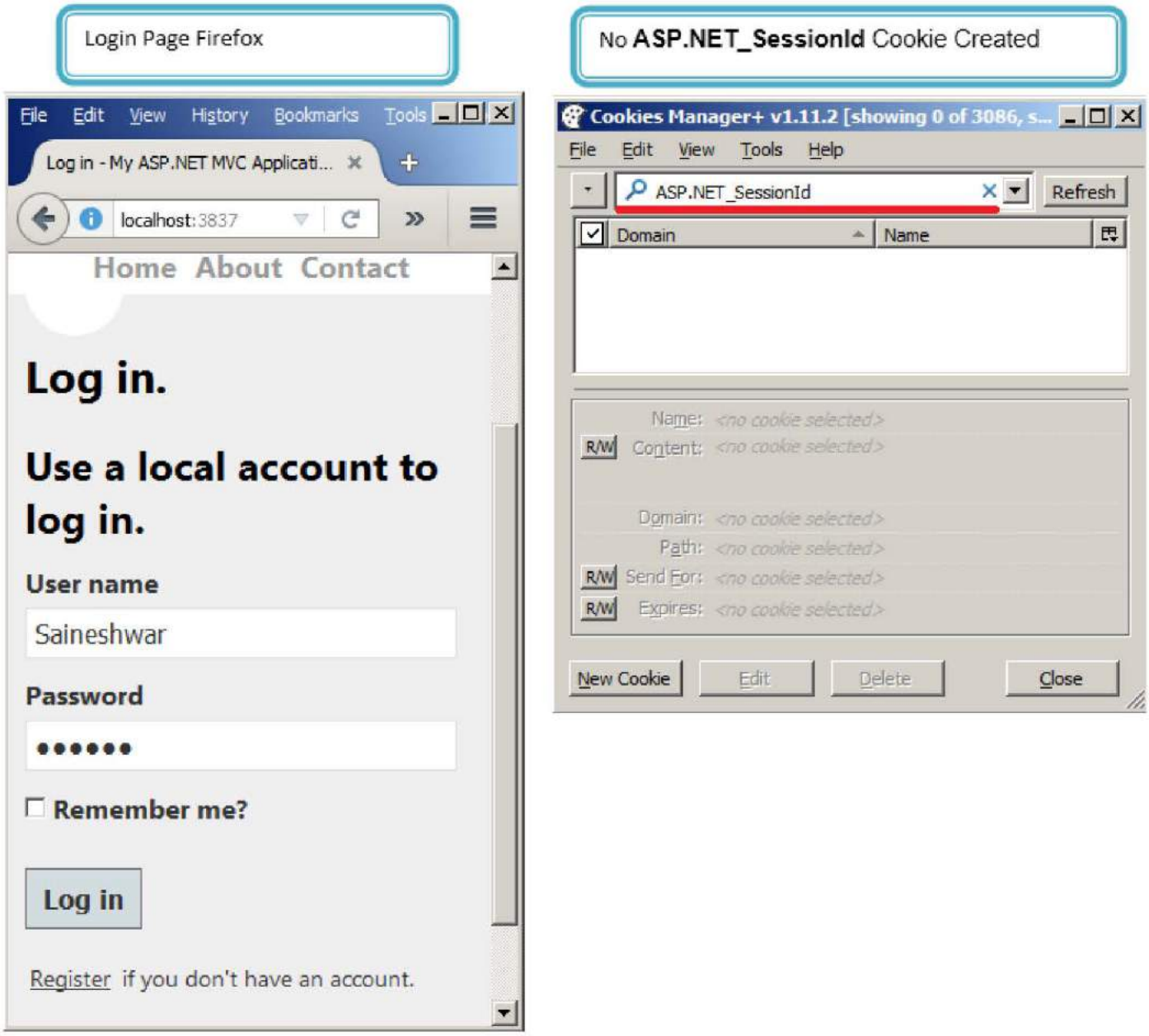


Fig 1.Session Fixation.

DEMO OF SESSION FIXATION

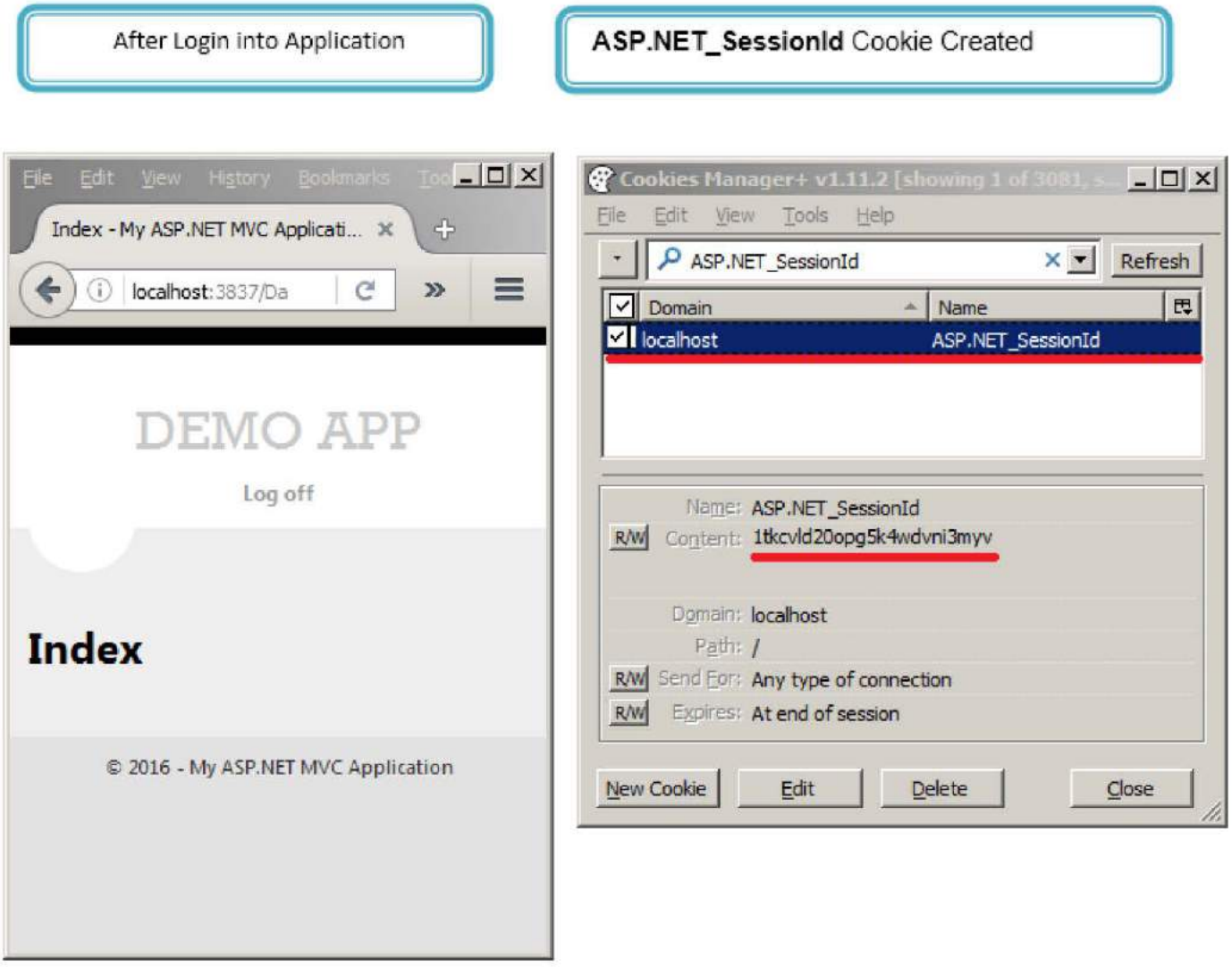
When we access login page we do not have any [**"ASP.NET_SessionId"**] cookie in the browser as we can see in cookie manager.



AFTER USER ENTER VALID CREDENTIALS

After entering Valid Login Credentials [**"ASP.NET_SessionId"**] Cookie is added to the browser.

Note :- When any data is saved into the Session [**"ASP.NET_SessionId"**] cookie is created and added to the user browser.

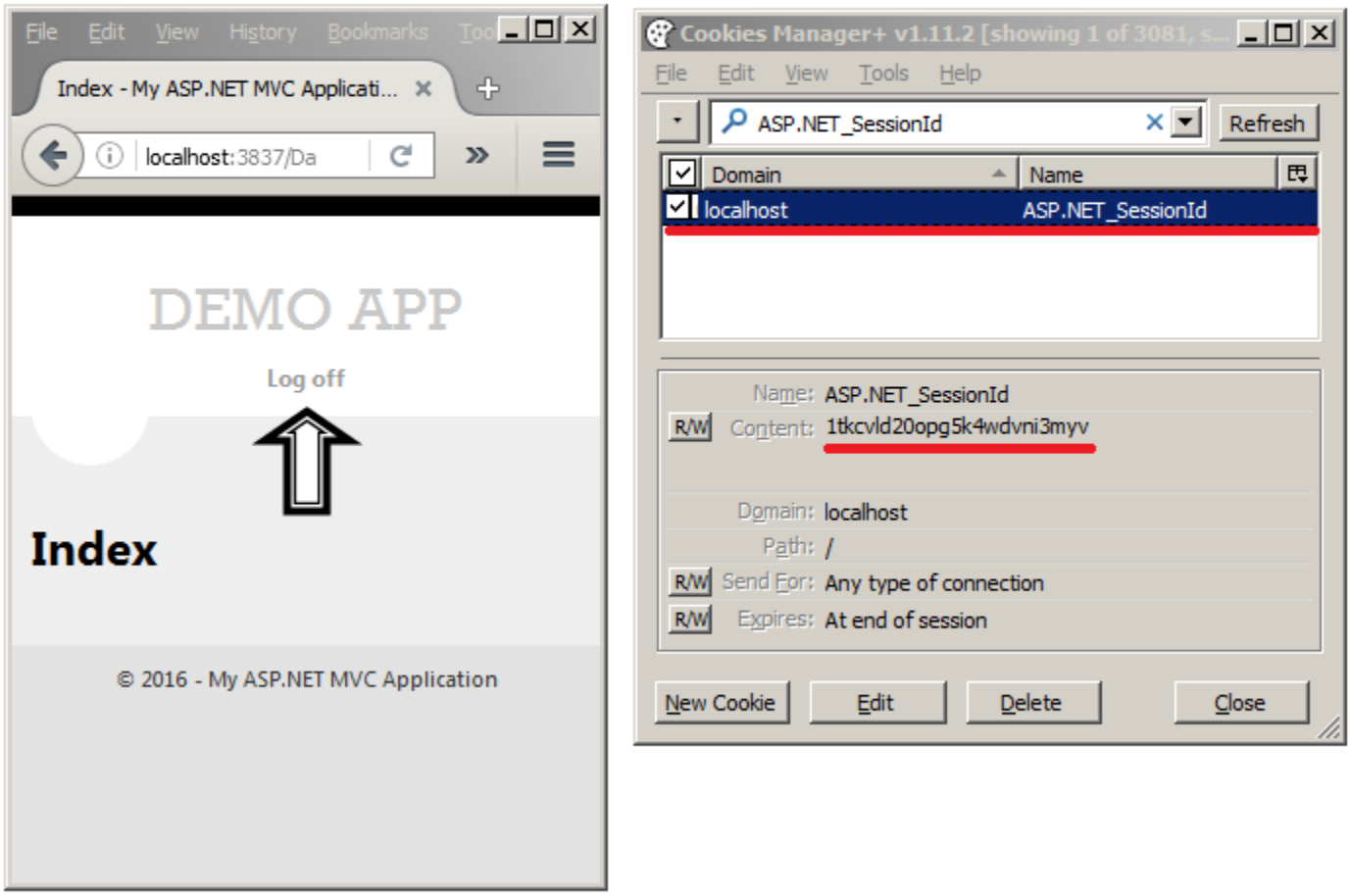


AFTER LOGGING OUT FROM APPLICATION COOKIE STILL EXISTS IN BROWSER

After Logout out from application Still [**"ASP.NET_SessionId"**] Cookie Exits.

After Logout

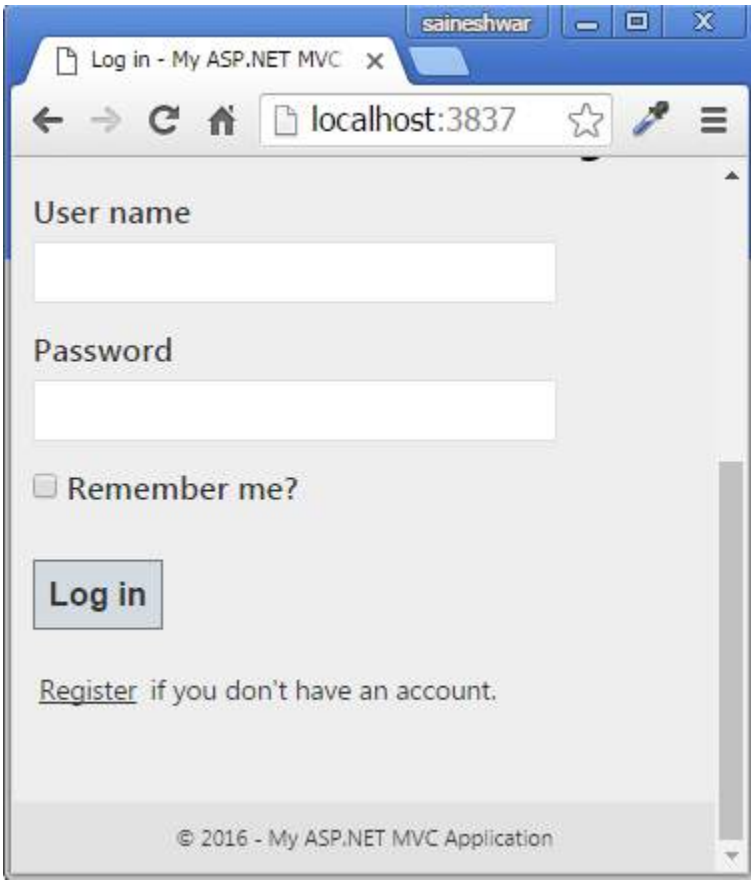
ASP.NET_SessionId Cookie Still Not Deleted



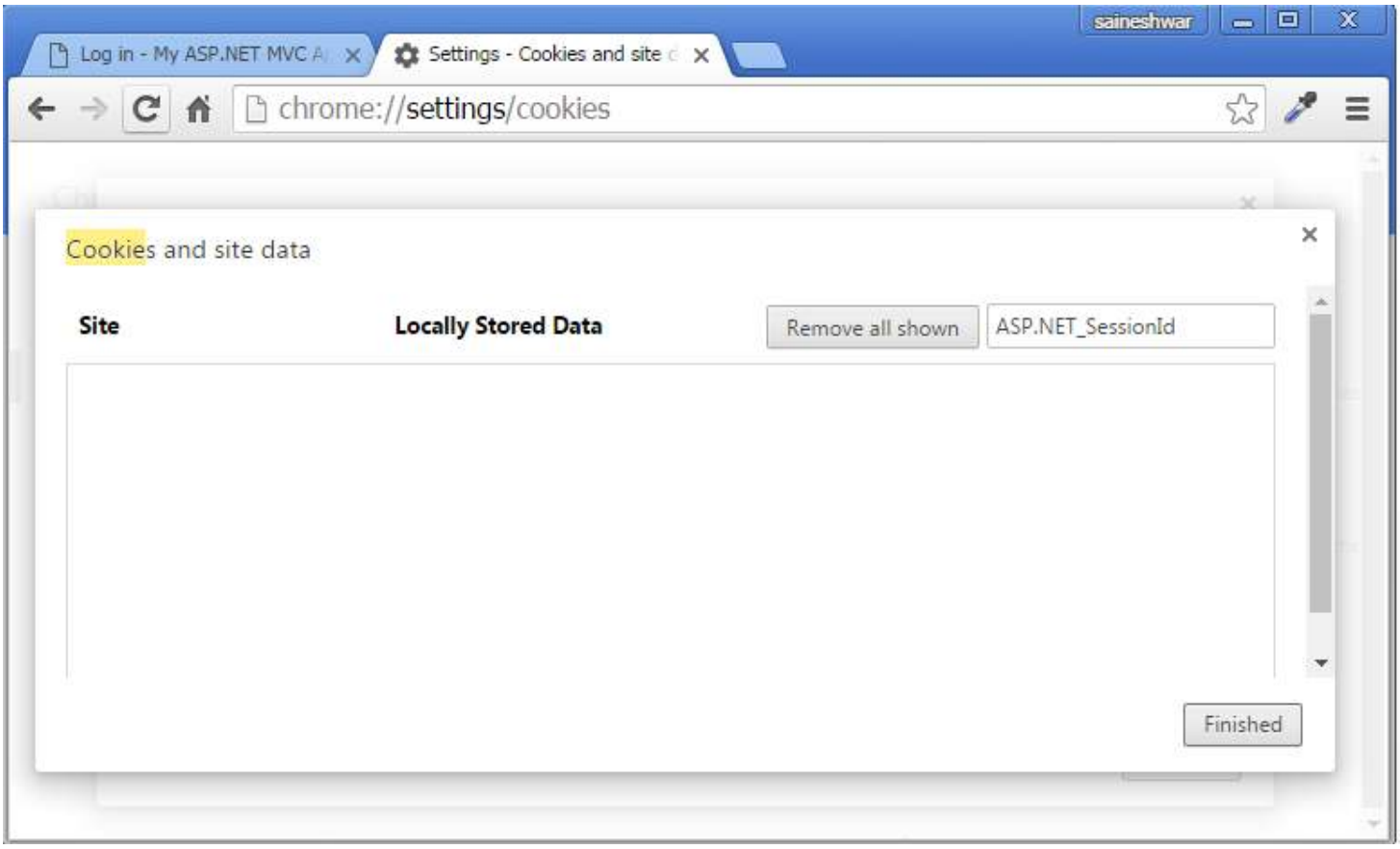
Note: Pre cookie and post cookie are same which can lead to the session fixation.

LET'S DO SOME SESSION FIXATION

The [**"ASP.NET_SessionId"**] Cookie which is not removed after logout which helps attacker for doing session fixation I will open a browser (chrome) in that I will enter URL [**http://localhost:3837/**] of application in which we are going to do session fixation.



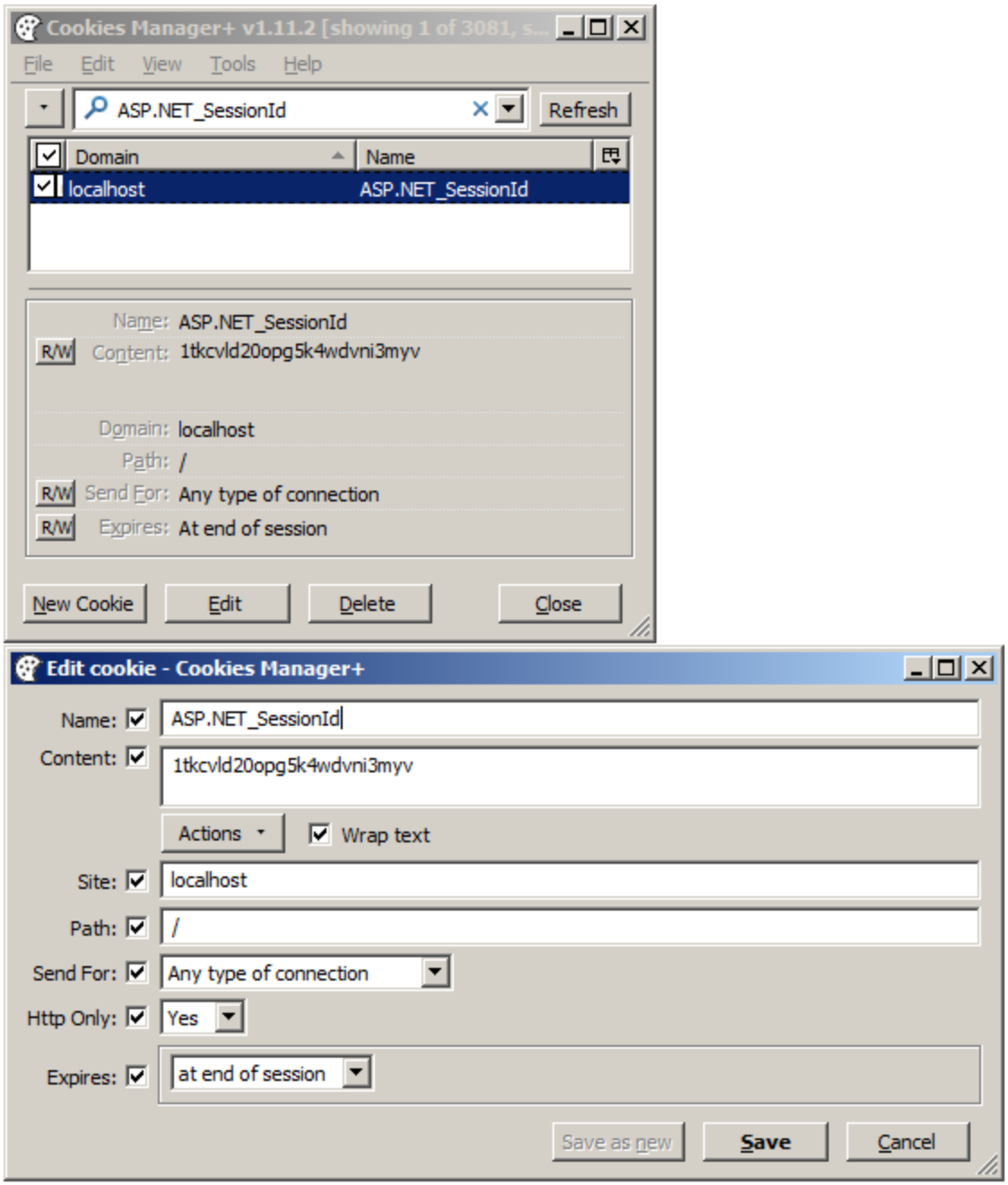
After Entering URL in the browser now lets check we have any [**"ASP.NET_SessionId"**] Cookie Created here oh we don't have any Cookie.



COOKIE WHICH IS ALREADY CREATED IN FIREFOX BROWSER

In this view, I have shown [**"ASP.NET_SessionId"**] cookie which is created in firefox browser when the user logged in.

Note :- For managing Cookie, I have installed **Cookie Manager+** addon in Chrome browser.

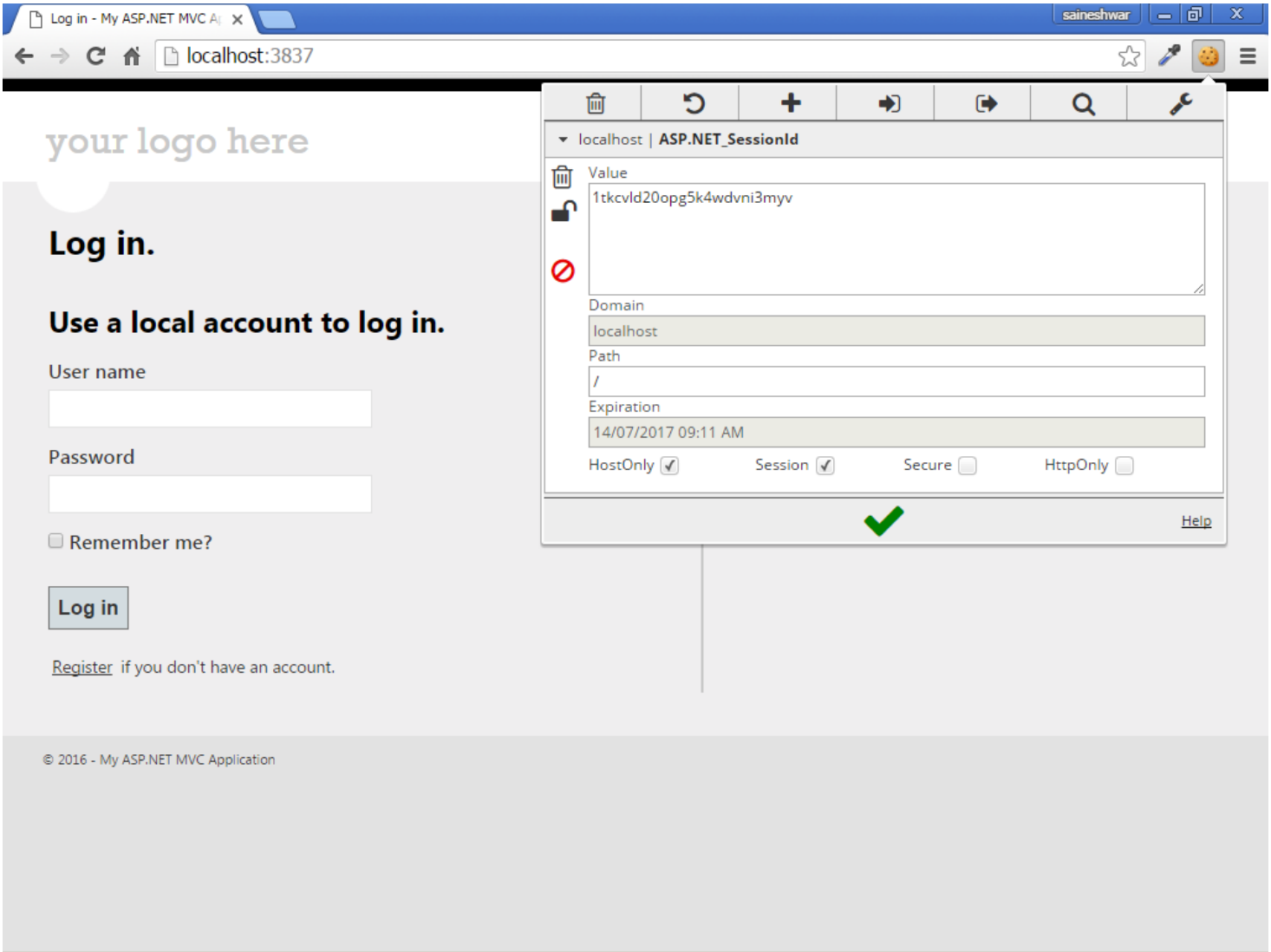


After having a view on [**"ASP.NET_SessionId"**] cookie in firefox now lets Create [**"ASP.NET_SessionId"**] cookie in Chrome browser similar to Cookie which is in Firefox browser with same [**"ASP.NET_SessionId"**] Cookie Name and Values.

CREATED NEW ["ASP.NET_SESSIONID"] COOKIE IN CHROME BROWSER SIMILAR TO COOKIE CREATED IN FIREFOX BROWSER.

This is a step where we have Fixed Session this session is live on another browser [firefox] we have copied values similar to that and created a [**"ASP.NET_SessionId"**] Cookie and assign similar values of SessionID to this Cookie.

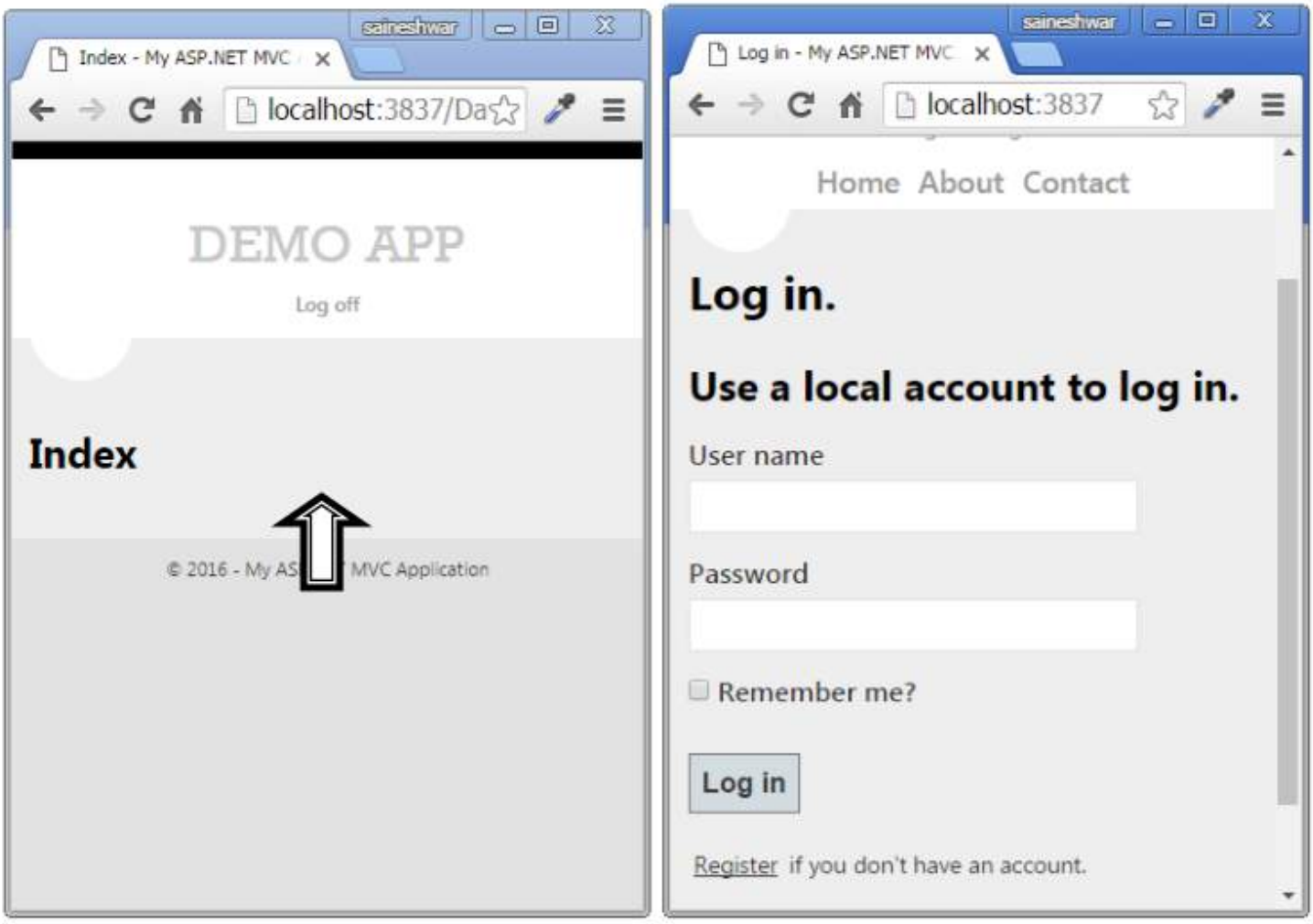
Note :- For Adding Cookie I have installed **Edit this Cookie** addon in Chrome browser



After fixing cookie now we no more require any login to the application if we just enter Inner URL of application we get direct access because this session is created after authentication .

After fixing [**"ASP.NET_SessionId"**] Cookie we not more Required to login because we already have set [**"ASP.NET_SessionId"**] cookie Now we just to enter Inner Application URL for direct Access to all pages of Application .

Think that if this Session ID is of Application Admin than what attacker we try to do.



Solution :-

1) Remove **["ASP.NET_SessionId"]** after logout.
2) Securing Cookies
3) Use SSL for Securing Cookies and Session

REMOVE ["ASP.NET_SESSIONID"] AFTER LOGOUT

On logout we are removing Session values long with that we are removing [**"ASP.NET_SessionId"**] Cookie from browser.

```
//
// POST: /Account/LogOff
public ActionResult LogOff()
{
    //Removing Session
    Session.Abandon();
    Session.Clear();
    Session.RemoveAll();

    //Removing ASP.NET_SessionId Cookie
    if (Request.Cookies["ASP.NET_SessionId"] != null)
    {
        Response.Cookies["ASP.NET_SessionId"].Value = string.Empty;
        Response.Cookies["ASP.NET_SessionId"].Expires = DateTime.Now.AddMonths(-10);
    }

    if (Request.Cookies["AuthenticationToken"] != null)
    {
        Response.Cookies["AuthenticationToken"].Value = string.Empty;
        Response.Cookies["AuthenticationToken"].Expires = DateTime.Now.AddMonths(-10);
    }

    return RedirectToAction("Login", "Account");
}
```

SECURING COOKIE

For securing cookies On Login [HttpPost] Action Method we are going to Create a New Session in that Session [`Session["AuthenticationToken"]`] we are going save New Guid along with that we are going to add a cookie with Name [`"AuthenticationToken"`] and it will also have same Value [Guid] which we have stored in Session.

CODE SNIPPET

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public ActionResult Login(LoginModel model, string returnUrl)
{
    if (ModelState.IsValid)
    {   //Getting Pasword from Database
        var storedpassword = ReturnPassword(model.UserName);
        // Comparing Password With Seed
        if (ReturnHash(storedpassword, model.hdrandomSeed) == model.Password)
        {
            Session["Username"] = model.UserName;
            Session["UserID"] = 1;

            // Getting New Guid
            string guid = Convert.ToString(Guid.NewGuid());
            //Storing new Guid in Session
            Session["AuthenticationToken"] = guid;
            //Adding Cookie in Browser
            Response.Cookies.Add(new HttpCookie("AuthenticationToken", guid));

            return RedirectToAction("Index", "Dashboard");
        }
        else
        {
            ModelState.AddModelError("", "The user name or password provided is incorrect.");
        }
    }
    return View(model);
}
```

CODE SNIPPET DESCRIPTION

Creating a new Guid.

```
// Getting New Guid
string guid = Convert.ToString(Guid.NewGuid());
```

Saving a new Guid in Session.

```
//Storing new Guid in Session
Session["AuthenticationToken"] = guid;
```

Saving a new Guid in Cookie and adding.

```
//Adding Cookie in Browser
Response.Cookies.Add(new HttpCookie("AuthenticationToken", guid));
```

After storing data in session and adding a cookie in the browser now let's match this values on every request and check these values are similar if not then we are going to redirect to Login page.

For doing this part I am going to add an **Authorization Filter** in the project and inside that we are going to write logic for checking Session and cookie values are similar or not.

AUTHENTICATEUSER ACTIONFILTER

If you check below code snippet I have created an **Authorization Filter** with name AuthenticateUser this filter we are inheriting IAuthorizationFilter Interface and FilterAttribute class with this we are implementing method inside interface [**OnAuthorization**] and write whole logic in this method.

```
using System;
using System.Web.Mvc;
```

```
namespace MvcSecurity.Filters
{
    public class AuthenticateUser : FilterAttribute, IAuthorizationFilter
    {
        public void OnAuthorization(AuthorizationContext filterContext)
        {
            string TempSession =
                Convert.ToString(filterContext.HttpContext.Session["AuthenticationToken"]);
            string TempAuthCookie =

Convert.ToString(filterContext.HttpContext.Request.Cookies["AuthenticationToken"].Value);

            if (TempSession != null && TempAuthCookie != null)
            {
                if (!TempSession.Equals(TempAuthCookie))
                {
                    ViewResult result = new ViewResult();
                    result.ViewName = "Login";
                    filterContext.Result = result;
                }
            }
            else
            {
                ViewResult result = new ViewResult();
                result.ViewName = "Login";
                filterContext.Result = result;
            }
        }
    }
```

```
}
```

CODE SNIPPET DESCRIPTION

In this method first we are going get session and cookie values .

```
string TempSession =
Convert.ToString(filterContext.HttpContext.Session["AuthenticationToken"]);

string TempAuthCookie =
Convert.ToString(filterContext.HttpContext.Request.Cookies["AuthenticationToken"].Value);
```

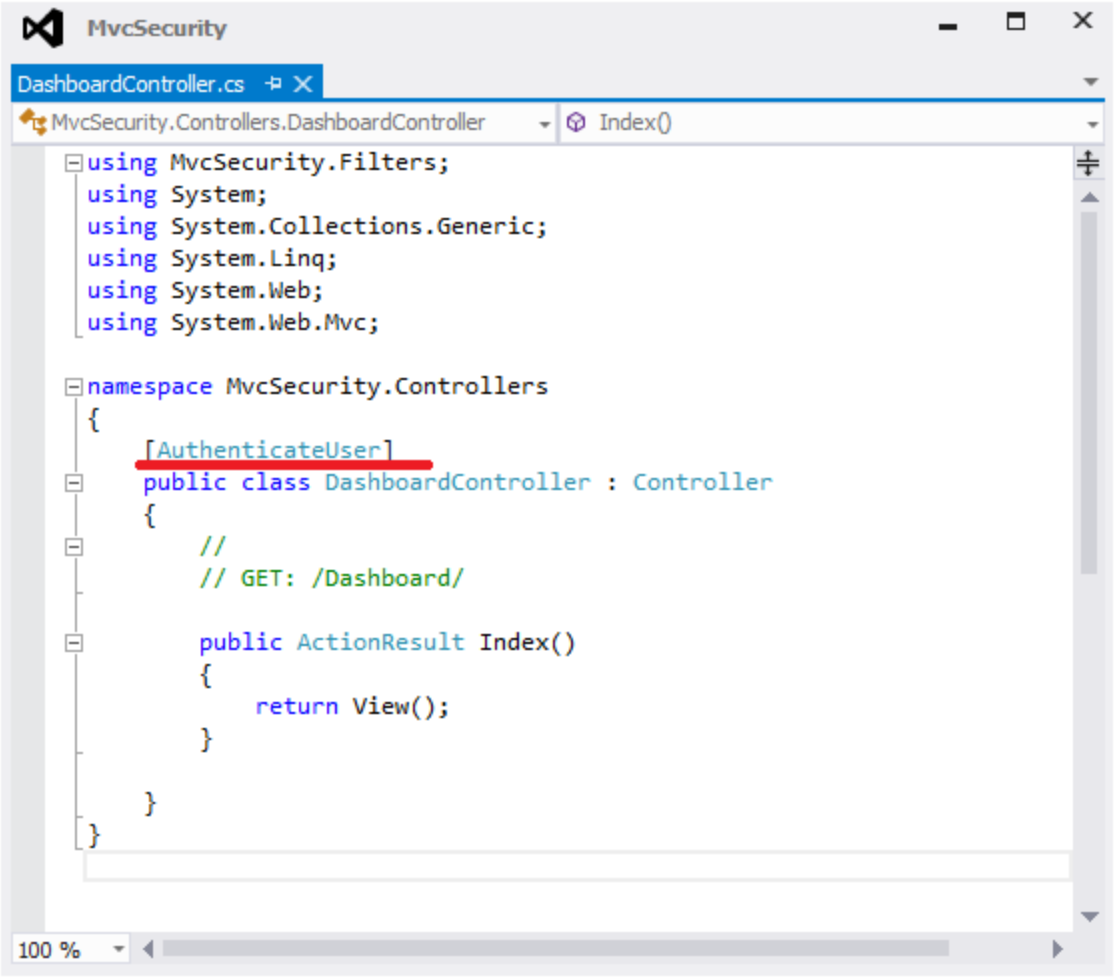
After getting values from session and cookie now we are going check that both session and cookie values are not null after that we are going see values are equal of both session and a cookie if they are not then we are going to redirect to login page.

```
if (TempSession != null && TempAuthCookie != null)
{
    if (!TempSession.Equals(TempAuthCookie))
    {
        ViewResult result = new ViewResult();
        result.ViewName = "Login";
        filterContext.Result = result;
    }
}
else
{
    ViewResult result = new ViewResult();
    result.ViewName = "Login";
    filterContext.Result = result;
}
```

After understanding codesnippet now we are going apply this filter on every controller which user access when he is logged in to application.

Applying AuthenticateUser Filter

Apply this filter on every controller which user access when he is logged into the application.



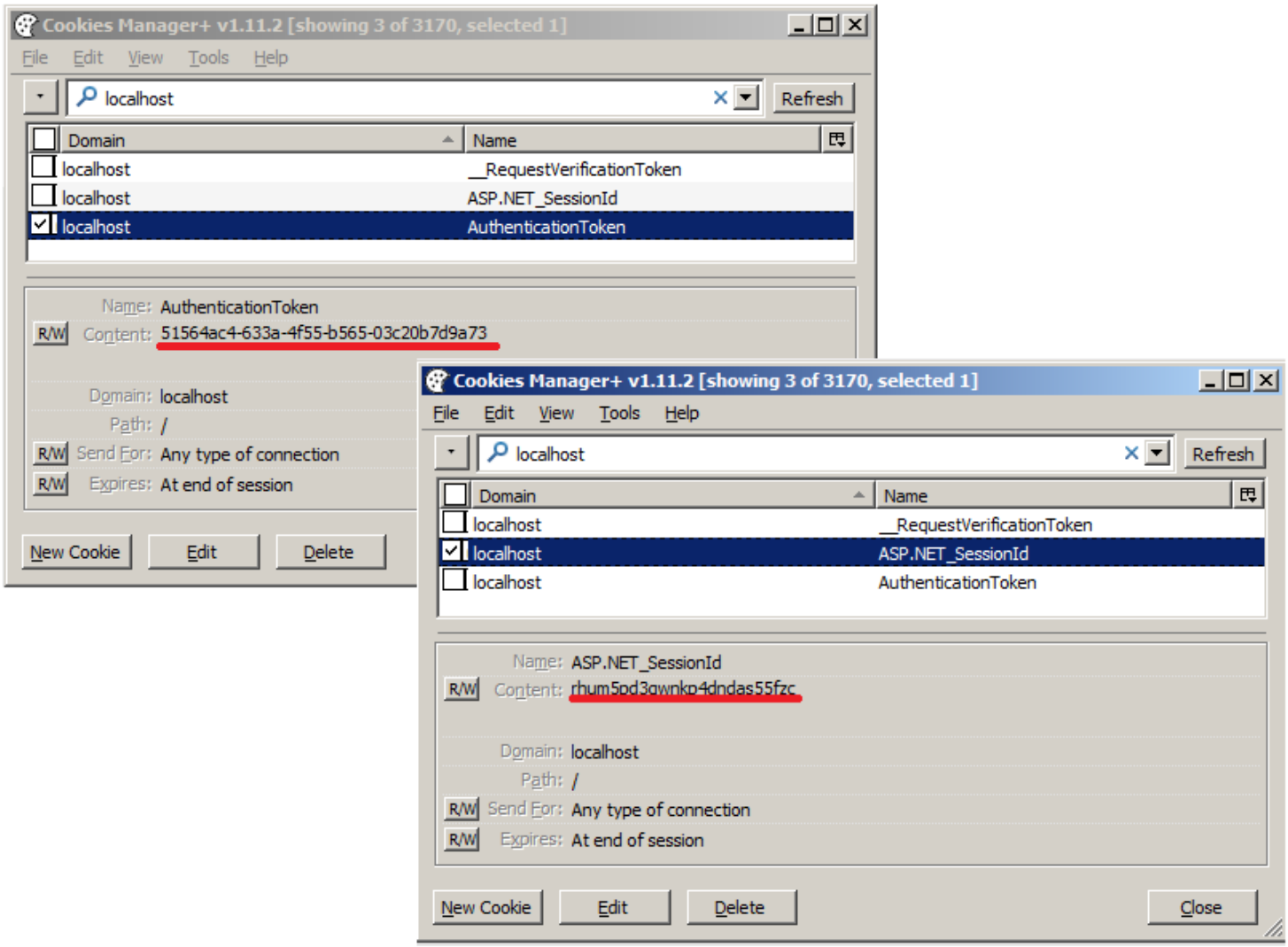
After applying Action filter on all the controller which user access after Logged into the application.

Now if the attacker knows [“**ASP.NET_SessionId**”] cookie value and a new cookie [`Cookies["AuthenticationToken"]`] value he will still not able to Session Fixation attack here because the new [`Cookies["AuthenticationToken"]`] contains GUID which is unique and same values is stored in Session [`Session["AuthenticationToken"]`] on Web Server but attacker can’t know the Session value that is stored on

the web server and this values keep changing every time when user is logging to application and the old session values which attacker used to do attack will not work in this scenarios.

Finally, if we will allow those User access to the application who has valid `Session["AuthenticationToken"]` value and `Cookies["AuthenticationToken"]` value.

REALTIME VALUES OF BOTH COOKIES.



USE SSL FOR SECURING COOKIES AND SESSION VALUES

SSL (Secure Sockets Layer) is Layer which Secure (Encrypted) communication between client and server such that any data [Banking details, Password, Session ,Cookie and another financial transaction] passed from client and server is Secure (Encrypted) .

Fig 1.SSL (Secure Sockets Layer).

- **Unvalidated Redirects and Forwards**

In all web application, we do redirect from one page to another page and sometimes we redirect to another application too but while redirect we won't validate URL which we are going redirect which causes Unvalidated Redirects and Forwards Attack.

This attack mostly uses to phish User to get Valuable details (User Credentials) or to install malicious malware

to the User computer.

Examples

In below snapshot, you will see Simple MVC application URL along with that a created malicious URL by an attacker that redirects users to a malicious site that performs phishing and installs malware into user computers.

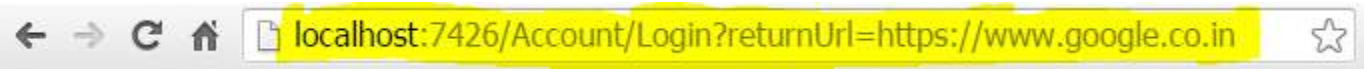


Fig 1.Crafted URL by an attacker.

Original URL :- http://localhost:7426/Account/Login

Crafter URL by Attacker :- ?returnUrl=https://www.google.co.in

ATTACK SCENARIO

In this attack User gets Email from attacker which contains offer related to Ecommerce shopping when user click on link given below he is redirected to shopping site [**http://demotop.com**] but if you see URL closely you will see that this URL contains redirect [**http://demotop.com/Login/Login?url=http://mailicious.com**] now after entering valid Username and Password the User will be redirected to Malicious site [**http://mailicious.com**] which is similar to [**http://demotop.com**] shopping site . On the Malicious site it will show Message “Invalid Username and Password” then again User will enter Username and Password and he will be redirected back to Original shopping site but meanwhile User Credentials are stolen in this attack.

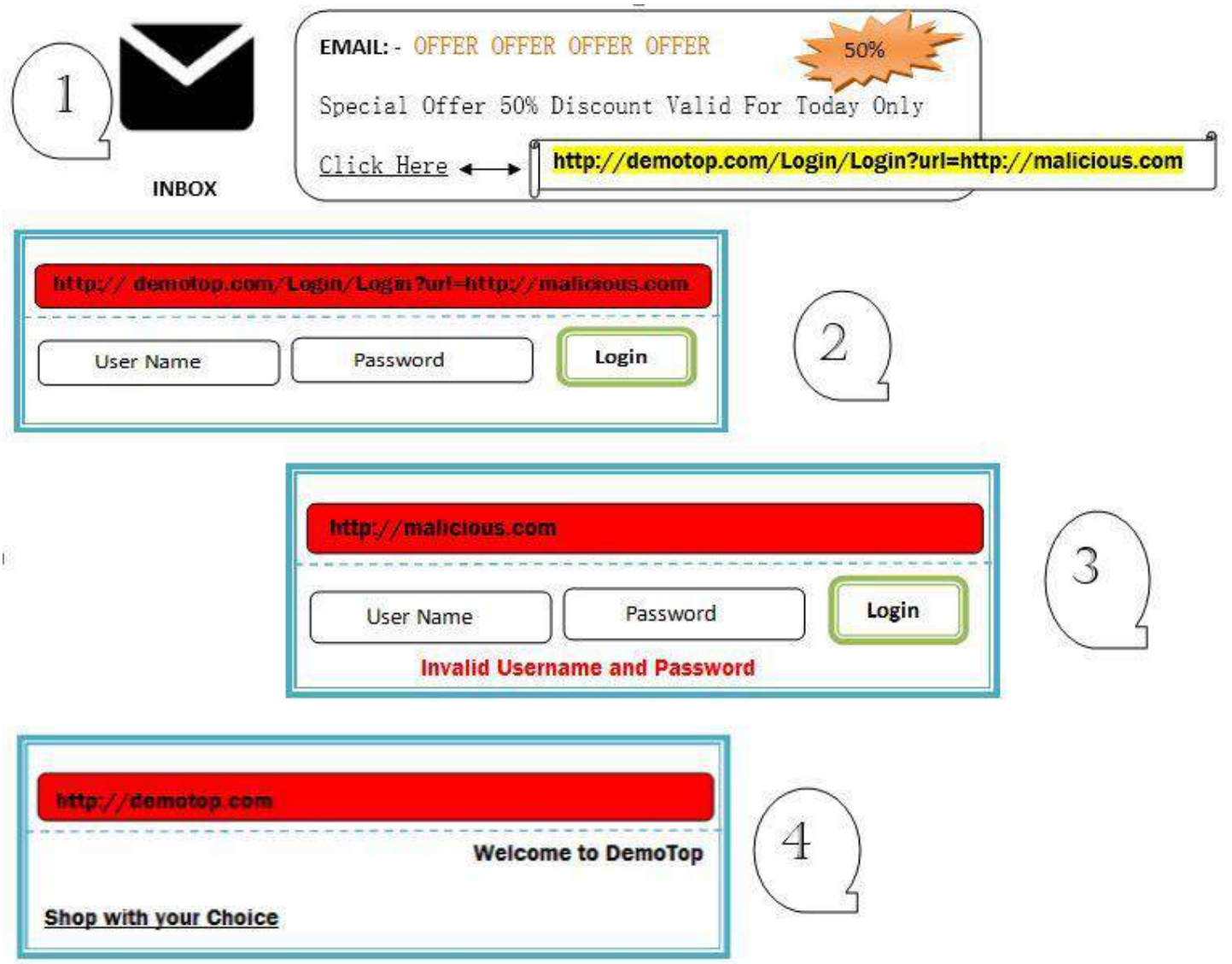


Fig 2. Unvalidated Redirects and Forwards Attack Scenario.

Solution

1. Simply avoid using redirects and forwards.

2. If you still want to use redirects and forwards then validate URL first.

3. Use **Url.IsLocalUrl** to Preventing redirects and forwards in MVC.

USING URL.ISLOCALURL IN MVC

Below is a snapshot of the login page in MVC which contains redirect URL to www.google.com when user enter Username and password then he will be redirected to www.google.com which is invalid to prevent this in MVC4 we have built in method called as **Url.IsLocalUrl** which checks that redirect URL which is Crafted is Local or not if not then it will not redirect it.

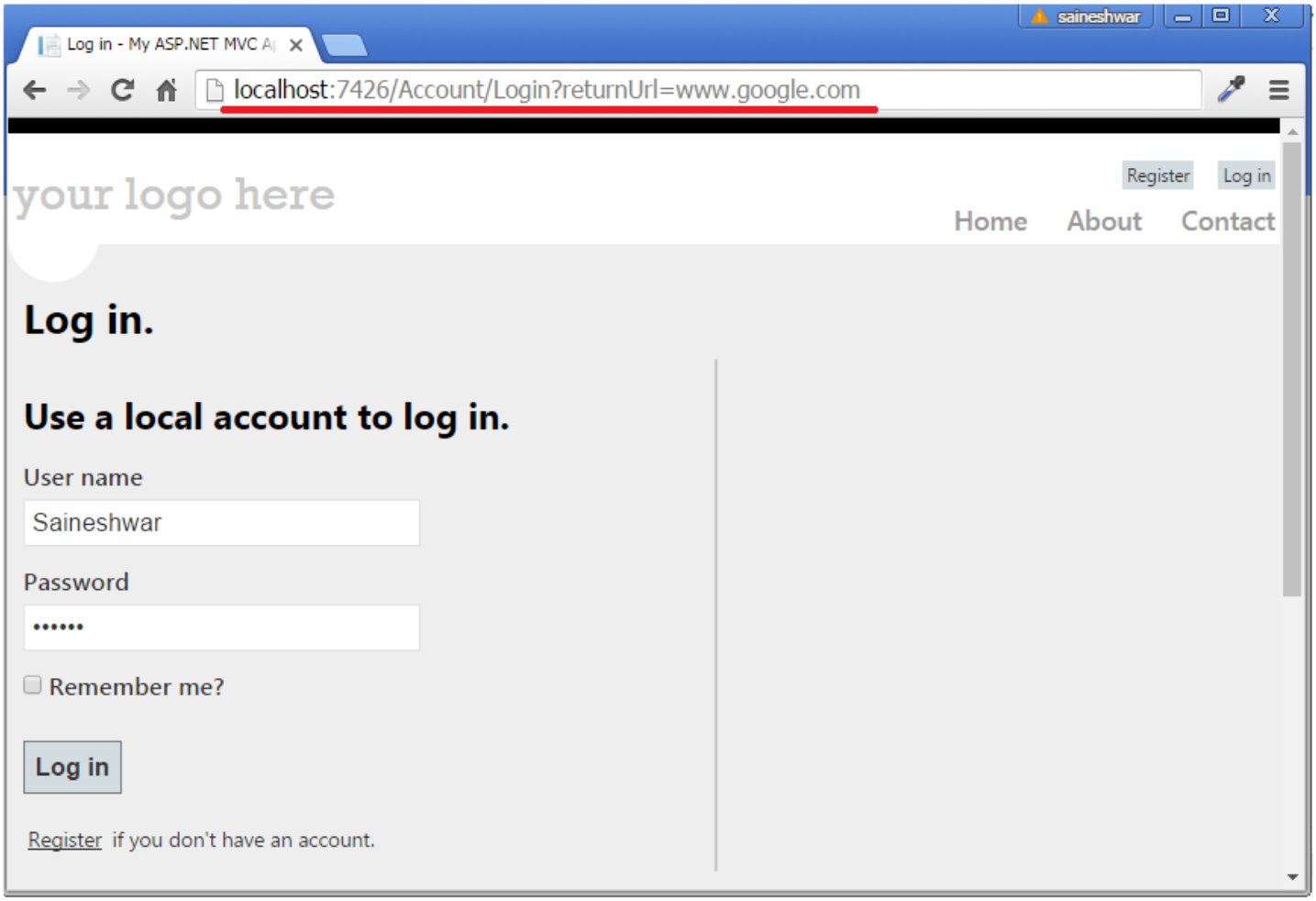


Fig 3. Login page in MVC with Redirect URL.

After understanding how redirect is passed not let's check how this URL gets checked and executed.

The below [HttpPost] Login method get called when user enter Credentials and submit the form along with that redirect URL is also get posted which may contain malicious URL. For showing demo I have just checked that Username and password is not null after that we are calling **RedirectToLocal** Action Method and to this

method we are going to pass redirect URL (**returnUrl**) .

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public ActionResult Login(LoginModel model, string returnUrl)
{
    if (model.UserName!=null && model.Password !=null)
    {
        return RedirectToLocal(returnUrl);
    }

    // If we got this far, something failed, redisplay form
    ModelState.AddModelError("", "The user name or password provided is incorrect.");
    return View(model);
}
```

returnUrl "www.google.com"

Fig 4. Login Post Action Method with returnURL.

After passing URL (**returnUrl**) to **RedirectToLocal** Action Method method then it is going pass URL to IsLocalUrl Method [**Url.IsLocalUrl(returnUrl)**] which is going check URL is local or not and return a boolean value if it is not then it will redirect to Home page else will redirect to returnUrl which is passed.

[True if the URL is local]

[False if URL is not local]

```
private ActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }
    else
    {
        return RedirectToAction("Index", "Home");
    }
}
```

returnUrl "www.google.com"

Fig 5. RedirectToLocal Action Method which checks returnUrl is Local or Not.

Thanks and Hope that we all will get Secure.