SECURITY FEATURES I ADDED TO MY APPLICATION.

Some of the main securities measures I have taking into consideration while build my ASP.NET MVC web application is listed below:

Authentication: This is the process of obtaining identification credentials from users of the application, such identification credentials are username and password. And validating those credentials. If the information supplied are valid, then the user can continue their purchase on the side. For my web application, Windows Authentication was used. The Windows Authentication utilizes the authentication capabilities of ISS.

Few steps taking for the authentication process:

1. Passwords are hashed before storing in the database.
2. Session ID is modifying on login and after logout. This was done using the "System.Web.SessionState.SessionIDManager"

Authorization: Authorization process determines whose identity or role should be granted access to a specific resource or part of the application. I used the URL authorization to map users and roles to some specific URLS. First, you must be 21 years and over to be authorize to you my web application. You must be granted the administrator role or privilege before you can access the Admin area of my website.

Cross Site Scripting (Anti XSS): XSS is the number-one security vulnerability on the Web according to OWASP. XSS can be carried out by attackers entering wrong script commands into a web application that allows unsanitized user input or by user input being directly shown on a page.

To prevent XSS attack on my web application I took the below preventive steps:

1. The ASP.NET request validation which is enabled by default was not disabled. The request validation prevents the server from accepting content containing un-encoded HTML. This will prevent some script injection attacks.
2. I ensured that all URL query string is encoded.
3. I ensured content sanitization is enabled to prevent adding scripts and untrusted attributes in the HTML content. The NuGet package AntiXSS library can be used for this.

Cross Site Request Forgery (CSRF): CSRF is an attack where a malicious website sends a request to a vulnerable site where the user is currently logged in.

To prevent CSRF on my website, I used the MVC AntiForgeryToken.

Cookie Stealing: Cookies are one of the things that makes the web reusable. Most websites make use of cookies to identify users after login. Cookies can be very dangerous if it gets to the wrong hand. Website users can be impersonated if an attacker can get a hold of their cookies.

To prevent cookie stealing I applied the following:

1. I used the SSL certificate, only HTTPS requests are allowed in my web application.
2. I applied the secure and HttpOnly flags in the web.config. This will make sure that cookie should be sent over an encrpted channel, not the HTTP which is a mere plain text format. The HttpOnly ensures that cookie should only be sent to the server with a request not by client-side script like JavaScript. This will also prevent cookies from being stolen through cross-site scripting or cross-frame scripting.

Blocking Brute Force Attacks: A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers and symbols until you can get a correct combination that works.

To prevent this type of attacks on my web application, I implemented the following steps.

1. User account is lockout after a specific number of failed login attempts
2. I enabled the Google reCAPTCHA on the login page.

SQL Injection in Entity Framework: SQL injection attack enables a malicious user to execute commands in website's database by using privileges granted to the website's login.

To prevent the SQL injection attacks, in implemented the following steps

1. Inputs are validated. Input data and validated by checking the type and format of the data to make sure they are correct.
2. I avoided disclosing database error information. The actual error information is not displayed, a customize error page is rather shown.

Cross frame scripting Attack (XFS): To prevent the XFS attack in my application, I implemented the following.

1. X-Frame-Options. The X-Frame-Options header ensures that hackers don't iframe my website. An attacker can use iframe to trick users into clicking links which they were not intended to click. The header was added to the web.config file.

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="X-Frame-Options" value="DENY" />

</customHeaders>

</httpProtocol>

</system.webServer>

In the above example, I am denying any of iframing my website. But I am I using iframes on the same domain then I will change the DENY to SAMEORIGIN. This will allow iframe from the same domain.

1. Content-Security-Policy. The Content-Security-Policy header is a HTTP header, its header works in helping to prevent code injection attacks like Cross-Frame Scripting attacks, clickjacking and Cross-Site Scripting attacks. This is done by the header telling the browser which dynamic resources can load.

A simple example of The Content-Security-Policy header is shown below

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="Content-Security-Policy" value="default-src 'self'" />

</customHeaders>

</httpProtocol>

</system.webServer>

The value in the above example represent the value of the Content-Security-Policy header. It is made up of one or more segments and it's separated by a semicolon. In the illustration, we only have one segment. And the ‘self’ means, you can only load resources from self on the web site. The self means the same origin as the HTML resource.

## Other Security features enabled on my web application is listed below:

X-Xss-Protection. This feature will prevent the page from loading when an XSS attack is detected.

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="X-Xss-Protection" value="1; mode=block" />

</customHeaders>

</httpProtocol>

</system.webServer>

value=1 means the protection level is enabled.

X-Content-Type-Options. This header is to prevent hackers to guess the right mime type (Multi-purpose Internet Mail Extension). Attackers can guess the mime type by inspecting the content. The X-Content-Type-Options avoid sniffing the MIME type.

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="X-Content-Type-Options" value="nosniff" />

</customHeaders>

</httpProtocol>

</system.webServer>

Referrer-Policy: The Referrer header is added automatically by the browsers when a website user clicks a link on the website. That means a website that is linked can see where a user is coming from. This is not good as private information in URL might be forward to other domains which you might not want to. The referrer is removed by with the following

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="Referrer-Policy" value="no-referrer" />

</customHeaders>

</httpProtocol>

</system.webServer>

X-Permitted-Cross-Domain-Policies: I added this to restrict flash components to make cross-origin requests.

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="X-Permitted-Cross-Domain-Policies" value="none" />

</customHeaders>

</httpProtocol>

</system.webServer>

Strict-Transport-Security: I added this to prevent any communication going through or happening through HTTP.

<system.webServer>

<httpProtocol>

<customHeaders>

<add name="Strict-Transport-Security" value="max-age=31536000; includeSubDomains" />

</customHeaders>

</httpProtocol>

</system.webServer>

X-Powered-By: This header was removed so that it won't be obvious which technology I’m using to host my web application. This is to try to reduce as much as possible the information an attacker can get.

<system.webServer>

<httpProtocol>

<customHeaders>

<remove name="X-Powered-By" />

</customHeaders>

</httpProtocol>

</system.webServer>

X-AspNetMvc-Version: Just is also to reduce the amount of information available to attackers, this header was removed to avoid telling attackers the framework and version that was used for development. This header is added automatically. The below code is added to the Global.asax.cs file to disable or remove the header.

MvcHandler.DisableMvcResponseHeader = true;

Server: The server header is removed from the code, so that attackers cannot know that I am using IIS. This is to reduce the vulnerability level of the application.