Trailmax Tech

Implementing HTTPS Everywhere in ASP.Net MVC application.

Posted on February 23, 2014 | by trailmax

HTTPS everywhere is a common theme of the modern infosys topics. Despite of that when I google for implementation of HTTPS in ASP.Net MVC applications, I find only a handful of horrible questions on StackOverflow, about how to implement HTTPS only on certain pages (i.e. login page). There have been numerous rants about security holes awaiting for you down that path. And Troy Hunt will whack you over your had for doing that!

See that link above? Go and read it! Seriously. I'll wait.

Have you read it? Troy there explains why you want to have HTTPS Everywhere on your site, not just on a login page. Listen to this guy, he knows what he is talking about.

Problem I faced when I wanted to implement complete "HTTPS Everywhere" in my MVC applications is lack of implementation instructions. I had a rough idea of what I needed to do, and now that I've done it a few times on different apps, my process is now ironed-out and I can share that with you here.

1. Redirect to HTTPS

Redirecting to HTTPS schema is pretty simple in modern MVC. All you need to know about is `RequireHttpsAttribute` . This is named as Attribute and can be used as an attribute on separate MVC controllers and even actions. And I hate it for that – it encourages for bad practices. But luckily this class is also implements `IAuthorizationFilter` interface, which means this can be used globally on the entire app as a filter.

Problem with this filter – once you add it to your app, you need to configure SSL on your development machine. If you work in team, all dev machines must be configured with SSL. If you allow people to work from home, their home machines must be configured to work with SSL. And configuring SSL on dev machines is a waste of time. Maybe there is a script that can do that automatically, but I could not find one quickly.

Instead of configuring SSL on local IIS, I decided to be a smart-ass and work around it. Quick study of source code highlighted that the class is not sealed and I can just inherit this class. So I inherited `RequireHttpsAttribute` and added logic to ignore all local requests:

```
public class RequreSecureConnectionFilter : RequireHttpsAttribute
{
    public override void OnAuthorization(AuthorizationContext filterContext)
    {
        if (filterContext == null)
        {
            throw new ArgumentNullException("filterContext");
        }

        if (filterContext.HttpContext.Request.IsLocal)
        {
            // when connection to the application is local, don't do any HTTPS stuff
            return;
        }

        base.OnAuthorization(filterContext);
    }
}
```

If you are lazy enough to follow the link to the source code, I'll tell you all this attribute does is check if incoming request schema used is `https` (that is what `Request.IsSecureConnection` does), if not, redirect all `GET` request to `https` . And if request comes that is not secured and not `GET` , throw exception. I think this is a good-aggressive implementation.

One might argue that I'm creating a security hole by not redirecting to https on local requests. But if an intruder managed to do local requests on your server, you are toast anyway and SSl is not your priority at the moment.

I looked up what `filterContext.HttpContext.Request.IsLocal` does and how it can have an impact on security. Here is the source code:

```
    public bool IsLocal {
        get {
            String remoteAddress = UserHostAddress;

            // if unknown, assume not local
            if (String.IsNullOrEmpty(remoteAddress))
                return false;

            // check if localhost
            if (remoteAddress == "127.0.0.1" || remoteAddress == "::1")
                return true;

            // compare with local address
            if (remoteAddress == LocalAddress)
                return true;

            return false;
        }
    }
```

This is decompiled implementation of `System.Web.HttpRequest` . `UserHostAddress` get client's IP address. If IP is localhost (IPv4 or IPv6), return true. `LocalAddress` property returns servers IP address. So basically `.IsLocal()` does what it says on the tin. If request comes from the same IP the application is hosted on, return true. I see no issues here.

By the way, here are the unit tests for my implementation of the secure filter. Can't go without unit testing on this one!

And don't forget to add this filter to list of your global filters

```
public static class FilterConfig
{
    public void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new RequreSecureConnectionFilter());
        // other filters to follow...
    }
}
```

2. Cookies

If you think that redirecting to https is enough, you are very wrong. You must take care of your cookies. And set all of them by default to be `HttpOnly` and `SslOnly` . Read Troy Hunt's excellent blog post why you need your cookies to be secured.

You can secure your cookies in web.config pretty simple:

```
<system.web>
    <httpCookies httpOnlyCookies="true" requireSSL="true"/>
</system.web>
```

The only issue with that is development stage. Again, if you developing locally you won't be able to login to your application without https running locally. Solution to that is `web.config` transformation.

So in your `web.config` you should always have

```
<system.web>
    <httpCookies httpOnlyCookies="true" />
</system.web>
```

and in your `web.Release.config` file add

```
<system.web>
    <httpCookies httpOnlyCookies="true" requireSSL="true" lockItem="true"
xdt:Transform="Replace" />
</system.web>
```

This secures your cookies when you publish your application. Simples!

3. Secure authentication cookie

Apart from all your cookies to be secure, you need to specifically require authentication cookie to be `SslOnly`. For that you need to add `requireSSL="true"` to your `authentication/forms` part of `web.config`. Again, this will require you to run your local IIS with https configured. Or you can do `web.config` transformation only for release. In your `web.Release.config` file add this into `system.web` section

```
<authentication mode="Forms">
    <forms loginUrl="~/Logon/LogOn" timeout="2880" requireSSL="true"
xdt:Transform="Replace"/>
</authentication>
```

4. Strict Transport Security Header

Strict Transport Security Header is http header that tells web-browsers only to use HTTPS when dealing with your web-application. This reduces the risks of SSL Strip attack. To add this header by default to your application you can add add this section to your `web.config`:

```
<system.webServer>
    <httpProtocol>
      <customHeaders>
        <add name="Strict-Transport-Security" value="max-age=16070400;
includeSubDomains" />
      </customHeaders>
    </httpProtocol>
</system.webServer>
```

Again, the same issue as before, developers will have to have SSL configured on their local machines. Or you can do that via `web.config` transformation. Add the following code to your `web.Release.config`:

```
<system.webServer>
    <httpProtocol>
        <customHeaders>
           <add name="Strict-Transport-Security" value="max-age=16070400;
includeSubDomains" xdt:Transform="Insert" />
        </customHeaders>
    </httpProtocol>
</system.webServer>
```

5. Secure your WebApi

WebApi is very cool and default template for MVC application now comes with WebApi activated. Redirecting all MVC requests to HTTPS does not redirect WebApi requests. So even if you secured your MVC pipeline, your WebApi requests are still available via HTTP.

Unfortunately redirecting WebApi requests to HTTPS is not as simple as it is with MVC. There is no `[RequireHttps]` available, so you'll have to make one yourself. Or copy the code below:

```
using System;
using System.Net;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using System.Web;

public class EnforceHttpsHandler : DelegatingHandler
{
    protected override Task<HttpResponseMessage> SendAsync(HttpRequestMessage
request, CancellationToken cancellationToken)
    {
        // if request is local, just serve it without https
        object httpContextBaseObject;
        if (request.Properties.TryGetValue("MS_HttpContext", out
httpContextBaseObject))
        {
            var httpContextBase = httpContextBaseObject as HttpContextBase;

            if (httpContextBase != null && httpContextBase.Request.IsLocal)
            {
                return base.SendAsync(request, cancellationToken);
            }
        }

        // if request is remote, enforce https
        if (request.RequestUri.Scheme != Uri.UriSchemeHttps)
        {
            return Task<HttpResponseMessage>.Factory.StartNew(
                () =>
                {
                    var response = new HttpResponseMessage(HttpStatusCode.Forbidden)
                    {
                        Content = new StringContent("HTTPS Required")
                    };

                    return response;
                });
        }

        return base.SendAsync(request, cancellationToken);
    }
}
```

This is a global handler that rejects all non https requests to WebApi. I did not do any redirection (not sure this term is applicable to WebApi) because there is no excuse for clients to use HTTP first.

WARNING This approach couples WebApi to `System.Web` libraries and you won't be able to use this code in self-hosed WebApi applications. But there is a better way to implement detection if request is local. I have not used it because my unit tests have been written before I learned about better way. And I'm too lazy to fix this -)

Don't forget to add this handler as a global:

```
namespace MyApp.Web.App_Start
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // other configurations...

            // make all web-api requests to be sent over https
            config.MessageHandlers.Add(new EnforceHttpsHandler());
        }
    }
}
```

6. Set up automatic security scanner for your site

ASafaWeb is a great tool that checks for a basic security issues on your application. The best feature is scheduled scan. I've set all my applications to be scanned on weekly basis and if something fails, it emails me. So far this helped me once, when error pages on one of the apps were messed up. If not for this automated scan, the issue could have stayed there forever. So go and sign-up!

Conclusion

This is no way a complete guide on securing your application. But this will help you with one of the few steps you need to take to lock down your application.

In this Gist you can copy my `web.Release.config` transformation file, in case you got confused with my explanation.

| Tagged .Net, c#, https, mvc, recipe

←Resharper vs Visual Studio

Attempt to do View Compilation for Azure Web Role →

24 Comments **Trailmax Tech** 1 **Login**

Recommend 4 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Melissa Snell • 3 years ago

In the App_Start FilterConfig.cs, if you add the folllowing lines:

if(!HttpContext.Current.IsDebuggingEnabled)
filters.Add(new RequireHttpsAttribute());

Then the site will force SSL on every page. Am just about to test it for API as well!
This will only happen if Debug is not enabled, so this means you don't require an SSL cert for local development, but it will apply for staging and production environments (providing that your config transforms remove the debug attribute

6 • Reply • Share ›

Arthur Salgado → Melissa Snell • 2 years ago

Nice. Thanks a lot.

1 • Reply • Share ›

Juan Arango → Melissa Snell • 9 months ago

Thank you! You got it very easy.

• Reply • Share ›

TVSlob → Melissa Snell • 2 years ago

Hey Melissa, thanks! I get worried trying to follow all the gobbly gook. I never debug on the production server and never run development website in release mode. Never needed to. So thanks. Now, lets see what happens when I upload everything to production that I just bought an SSL cert for.

• Reply • Share ›

trailmax Mod → Melissa Snell • 3 years ago

I did this initially, but switch to checking for local request. Problem with DEBUG is you rob yourself from executing your application in non-debug mode. And sometimes you need to test for RELEASE build locally. Also if you are debugging the application deployed on a server, you'll need to configure ssl to work locally and that is not always available.

• Reply • Share ›

beton • 2 years ago

Very useful.
Thank you!

1 ^ | v • Reply • Share ›

Kevin • 3 years ago

Thank you. I have a rather different situation. We have a web application that currently does not use SSL and we would like to start using SSL. We would like the current users of the application to continue to use http and they would be redirected to https only if the https URL is available. In other words if the certificate has been installed and IIS can use it. So I would like for the use of SSL to be dependent on whether the certificate is installed or not. That way we can make the code change now and when the certificate is available we can trust that the site is "secured". Ideas for detecting https availability?

1 ^ | v • Reply • Share ›

trailmax Mod → Kevin • 3 years ago

Sorry, took my time to reply - been on holidays.

I have not done what you need to do, but it seems possible to hook into IIS binding configuration and check if binding on SSL port is available. Something like this: http://blogs.msdn.com/b/car...

Alternatively you can do a redirect to HTTPS via web.config - configuration time. When you install SSL cert on a server, change web.config to redirect to HTTPS: http://azure.microsoft.com/...

^ | v • Reply • Share ›

Woroud fayyad • 8 months ago

how to add option to enable/disable the ssL on my MVC5 application in web.config?

^ | v • Reply • Share ›

trailmax Mod → Woroud fayyad • 8 months ago

Why would you ever disable it?

^ | v • Reply • Share ›

James Mikel • a year ago

Getting __RequestVerificationToken is not present when user is attempting to login while local. (same domain). the project is setup following your OWIN and AD Authentication... And now this article. Any idea where I should look? Remote access is successful.

^ | v • Reply • Share ›

trailmax Mod → James Mikel • a year ago

Do you have @Html.AntiForgeryToken() in your form?

^ | v • Reply • Share ›

James Mikel → trailmax • a year ago

Yes. Login.cshtml. the httppost in the controller has the [ValidateAntiForgeryToken]

^ | v • Reply • Share ›

trailmax Mod → James Mikel • a year ago

Cleaned cookies, tried in browser Private mode?

Reply • Share ›

James Mikel ➦ trailmax • a year ago

Cleared history. Login failed. Switched to private. Login failed.
what should my web config authentication mode be? currently set to "None" Reviewing your OWIN /AD project to see if I got the setting from there.
-- web config looks like the issue. Thanks for sharing your knowledge!

Reply • Share ›

trailmax Mod ➦ James Mikel • a year ago

web.config authentication should be "none", but AD/OWIN has nothing to do with your verification token missing.
Actually every time I had a problem with this token it was either dirty cookies or incorrectly applied [ValidateAntiforgeryToken] and @Html.AntiForgeryToken(). So I'm out of ideas and it is hard to debug without looking on the code.

Reply • Share ›

Lerry Mashaba • 2 years ago

Very resourceful.

Reply • Share ›

Mani Raj • 2 years ago

nice article...

Reply • Share ›

Chris Nevill • 4 years ago

Very useful much appreciated.
Chris, Cardiff, Wales

Reply • Share ›

trailmax Mod ➦ Chris Nevill • 4 years ago

Glad it helps!

Reply • Share ›

Angry_Red_Barber • 4 years ago

I'm not sure that adding the STS header to the web config is entirely correct - then I believe your website will respond with the STS header even when the page request is made over HTTP which conflicts with the HSTS specification.

Reply • Share ›

trailmax Mod ➦ Angry_Red_Barber • 4 years ago

You are correct here, this part goes against HSTS spec. If you care about these things (I clearly don't), you'll have to implement either a IIS handler or MVC filter along with WebApi delegating handler.

Reply • Share ›

Angry_Red_Barber ➦ trailmax • 4 years ago

I just found that the following code added to Global.ascx.cs is sufficient:

/// <summary> Handles the BeginRequest event of the Application control. </summary>

protected void Application_BeginRequest(object sender, EventArgs e)
{
if (Request.IsSecureConnection)
{
Response.AddHeader("Strict-Transport-Security", "max-age=31536000");
}
}

• Reply • Share ›

trailmax Mod ➦ Angry_Red_Barber • 4 years ago

Or you can do this like this. Completely forgot about this option...
Not sure if WebApi requests are handled through this, need to try

• Reply • Share ›

ALSO ON **TRAILMAX TECH**

How to export data from Excel into SQL Server

4 comments • 3 years ago

trailmax — As I said, it is not as good for reading as LinqToExcel, but awesome for writing reports!

Web-site development metaphors for non-tech people.

2 comments • 2 years ago

trailmax — Problem with references to data - somebody need to fetch this data to show on pages. Now we need to explain to folk that …

Working With Attachments in DocumentDB

8 comments • 9 months ago

Tim Scarfe — Good article, thanks (for my small app, I just used byte[] properties on the

Log all data in ASP.Net MVC POST Requests

1 comment • 2 years ago

Nathan Tino — Not sure if the IsPost() method

Search ...

Tags

.Net asp.net Autofixture automation azure bat BuildServer build server C# cakebuild commands Dependency Injection di drivers eclipse EntityFramework firefox html https Identity iis java javascript jquery Linux microsoft mocking mvc OpenXml recipe regexp Registry script sql sql server stackoverflow testing tfs unit-testing Vista visual studio VSTS Windows Wireless XP

Blogroll

- HR.Net Consaltancy and blog
- My Git Cheat-Sheet

- My Twitter