**How To Enable HTTPS In ASP.NET Web API**

**Introduction**

**In this article, you will see how we can enable HTTPS in ASP.NET Web API. We will start by discussing all the steps required to enable HTTPS in ASP.NET web API. And then we will discuss all the steps in detail. Also, you will see how we can enable HTTPS support for the development server.**

**Steps to enable HTTPS in ASP.NET Web API**

* **Write a custom class which is inherited from AuthorizationFilterAttribute**
* **Register that class in ASP.NET Web API Config**
* **Apply [RequireHttps] attribute on API controller actions.**
* **Create a temporary certificate for SSL.**
* **Install the certificate**
* **Enable HTTPS support to the development server in Visual Studio.**

**Write a custom class which is inherited from AuthorizationFilterAttribute**

**Write a custom class as shown below.**

**public class RequireHttpsAttribute : AuthorizationFilterAttribute**

**{**

**public override void OnAuthorization(HttpActionContext actionContext)**

**{**

**if (actionContext.Request.RequestUri.Scheme != Uri.UriSchemeHttps)**

**{**

**actionContext.Response = new HttpResponseMessage(System.Net.HttpStatusCode.Forbidden)**

**{**

**ReasonPhrase = "HTTPS Required for this call"**

**};**

**}**

**else**

**{**

**base.OnAuthorization(actionContext);**

**}**

**}**

**}**

**C#**

**Copy**

**Register that class in ASP.NET Web API Config**

**To register a custom HTTP filter class in web API configuration here are the settings.**

**// Web API configuration and services**

**config.Filters.Add(new RequireHttpsAttribute());**

**C#**

**Copy**

**Remember this is a global setting and will require all controller methods to run on HTTPS.**

**If we want to have a few methods to run on HTTP then in that case, just disable this setting. And use the [Requirehttps] attribute for individual methods.**

**Apply [RequireHttps] attribute on API controller actions.**

**[RequireHttps]**

**public IEnumerable<string> Get()**

**{**

**return new string[] { "value1", "value2" };**

**}**

**C#**

**Copy**

**Note. We need to use this [RequireHttps] attribute only in case we need to enable HTTPS only for selective API controller actions. Otherwise, Web API configuration global settings are enough.**

**But if we are targeting only a few API methods to run on HTTPS then we must disable the global configuration. Otherwise, all method calls will demand HTTPS.**

**Create a temporary certificate for SSL**

**To create a temp certificate run the following command in the command prompt.**

**makecert.exe -n "CN=Development CA" -r -sv TempCA.pvk TempCA.cer**

**Bash**

**Copy**

**Once the certificate is created it will be saved on your machine at the path selected in the command prompt windows.**

**Now, we need to install it.**

**Install the certificate**

**To install the certificate on your local machine, you need to do the following steps.**

* **Open the MMC (Management console) window**
* **Then go to File - > Add or Remove Snap Ins**
* **Then select Certificates from available Snap Ins**
* **Then click on the ADD button**
* **Then select Computer account in the window pane that opens**
* **Then select Local Computer Account**
* **Then click next and OK**

**Now the certificate snap is added to MMC.**

**Now we need to install the certificate by selecting it in a snap.**

**For that,**

* **Go to Certificates; expand it.**
* **Then Select “Trusted root certification Authorities”**
* **Then Select Action - > All Tasks - > Imports**
* **Select the certificate and finish.**

**Now, a temporary certificate is installed on your computer.**

**This certificate will be used for SSL communication on your machine, but apart from installation, you don't need to do anything with respect to certificates.**

**Now, the next step is to enable HTTPS for the development server.**

**Enable HTTPS support to the development server in Visual Studio**

**For that do the following.**

* **Open your web API solution in Visual Studio,**
* **Then select the web API project in Solution Explorer.**
* **Select View Menu in Visual Studio**
* **Now select “Properties window” or click F4.**
* **A window pane will open.**
* **There select the “SSL Enabled” property and set it to true**

**Now, the development server is ready to work with HTTPS too.**

**Summary**

**So, in this article, we discussed how we can make a web API run on HTTPS. For that we discussed all the steps required in detail, like writing a custom class, using the RequiredHttps filter class, and registering this class in API configuration, then we also provided details on installing a temp certificate and using it and finally enabling HTTPS support for development in Visual Studio.**

**How to configure SSL in ASP.NET**

In this article, we will learn how to configure SSL in ASP.NET step-by-step. Securing your website with SSL (Secure Sockets Layer) is crucial for protecting sensitive information, such as login credentials and credit card information. In this article, we will go over the steps to configure SSL in an ASP.NET Core website.

**How to setup HTTPS in ASP.NET applications**

**1- Obtain an SSL certificate**

The first step in configuring SSL is to obtain a certificate from a trusted certificate authority (CA). This certificate will be used to encrypt the connection between the client and server.

There are several ways to obtain an SSL certificate, including:

* **Purchase from a Certificate Authority (CA):**One of the most common ways to obtain an SSL certificate is to purchase one from a reputable CA. Some popular CAs include DigiCert, GlobalSign, and Comodo. When purchasing a certificate, you will need to provide your domain name and some information about your organization. The CA will then verify your identity and issue the certificate.
* **Generate a self-signed certificate:** Another option is to generate a self-signed certificate. This is useful for testing purposes, but not recommended for production use as a self-signed certificate is not trusted by browsers.
* **Use a free SSL service like Let’s Encrypt:** Another option is to obtain a free SSL certificate from Let’s Encrypt, which is a free, automated, and open certificate authority that provides SSL/TLS certificates to enable HTTPS on the web.
* **Use Cloud Providers:** Some cloud providers like AWS, Azure, and GCP have their own certificate manager that allows you to generate and manage SSL certificates for your applications.

I’ll create a new web application and choose to generate a self-signed certificate for this article to enable you to follow along without purchasing CAs. So run the following command to generate a certificate file for you.

dotnet dev-certs https -ep certificate.pfx -p password

You can replace the `certificate.pfx` with a custom path and use a different password.

To avoid privacy errors like this:

You can run the following command afterward:

dotnet dev-certs https –trust

This command only works for Windows if you are a Linux user, visit [this link](https://aka.ms/dev-certs-trust).

Once you have obtained an SSL certificate, you will need to install it on your server and configure your website to use it. By the way, did you know that we offer a unique and powerful online course that boosts your C# career? Check it out [here](https://academy-tutorials.eu/csharp-progress-academy-organic?utm_source=tutorialseuwebsite)!

**2- Configure your website to use SSL**

In the `Program.cs` file of your ASP.NET Core project, you will need to configure the app to use SSL. You can do this by adding the following code in the Configure method:

app.UseHttpsRedirection();

This will automatically **redirect all HTTP requests to HTTPS**.

**3- Provide the SSL certificate**

In the same `Program.cs` file, you will need to provide the SSL certificate to the app. This can be done by adding the following code after `var builder = WebApplication.CreateBuilder(args);` and before `var app = builder.Build();`:

var httpsConnectionAdapterOptions = new HttpsConnectionAdapterOptions

{

    SslProtocols = System.Security.Authentication.SslProtocols.Tls12,

    ClientCertificateMode = ClientCertificateMode.AllowCertificate,

    ServerCertificate = new X509Certificate2("./certificate.pfx", "password")

};

builder.WebHost.ConfigureKestrel(options =>

    options.ConfigureEndpointDefaults(listenOptions =>

        listenOptions.UseHttps(httpsConnectionAdapterOptions)));

Make sure to replace the placeholder `./certificate.pfx` with the actual path to your SSL certificate and `password` with the certificate’s password.

**4- Test your SSL configuration**

After configuring your website to use SSL, it is important to test the configuration to ensure that it is working properly. You can do this by accessing your website using the `https://` protocol and checking for the padlock icon in the browser’s address bar.

**5- Keep your SSL certificate up to date**

SSL certificates have expiration dates; make sure to renew the certificate before it expires to avoid any interruption in service.

If you want to skyrocket your C# career, [check out](https://academy-tutorials.eu/csharp-progress-academy-organic?utm_source=tutorialseuwebsite) our powerful ASP.NET full-stack web development course that also covers test-driven development and C# software architecture.

**How Does HTTPS Work?**

HTTPS (HTTP Secure) protocol establishes a secure connection between a web server and a web browser. HTTPS is built on top of the standard HTTP protocol and uses SSL/TLS (Secure Sockets Layer/Transport Layer Security) to encrypt the data that is being transmitted between the server and browser.

Here’s a brief overview of the HTTPS process:

* The web browser (client) sends a request to the webserver to establish a secure connection.
* The web server sends back its SSL/TLS certificate, which includes the server’s public key.
* The web browser verifies the authenticity of the SSL/TLS certificate by checking it against a list of trusted certificate authorities (CA).
* Once the certificate is verified, the web browser and server initiate the SSL/TLS handshake process.
* The web browser and server agree on a set of encryption methods and then create a unique symmetric key that will be used to encrypt all data transmitted between them.
* Once the SSL/TLS session is established, the web browser will send an HTTP request over the secure SSL/TLS connection.
* The web server will process the request and send back an HTTP response, which is also encrypted using the symmetric key.
* The web browser will then decrypt the response and display the content to the user.

**Conclusion**

By following these steps, you can configure SSL on your ASP.NET Core website and ensure that sensitive information is protected while in transit. It is important to keep in mind that maintaining the security of your website is an ongoing process and requires regular monitoring and updates.

With HTTPS, all data transmitted between the browser and server is encrypted, ensuring that it cannot be intercepted or read by any third party, making it much more secure than regular HTTP. Websites that use HTTPS are also identified by a padlock icon in the browser’s address bar and the prefix “https” in the URL.