Shield Your ASP.NET MVC Web Applications with Content Security Policy (CSP)

Karthik Anandan • 8 min read • Apr 24, 2024 • Updated • 3 Comments

> **TL;DR:** *Fortify your ASP.NET MVC web app against security threats by implementing Content Security Policy (CSP). Mitigate vulnerabilities like XSS and data injection attacks. Learn CSP directives, application methods, and best practices for testing and troubleshooting to secure your app today!*

"**One single vulnerability is all an attacker needs.**"

– Window Snyder

Hackers are everywhere today. The world wide web is also a place for worldwide vulnerabilities. In order to safeguard your application, you need a powerful mechanism. In that case, Content Security Policy (CSP) is at your service with some excellent features.

In this blog post, we will see how to implement CSP in ASP.NET MVC web applications!

Overview

CSP is used to protect your web application. It safeguards it by identifying some types of attacks like cross-site scripting (XSS) and SQL or data injection attacks.

> **Note:** In CSP, some browser features are disabled by default.

If we want to apply CSP to our application, we have to define some CSP *content security directives* in the desired Content-Security-Policy headers or in the <meta> tags.

When a resource doesn't match with the defined policy directives, then it won't be loaded by the browser (scripts and styles from a third-party).

So if a policy restricts images means, then the browser will prevent images from loading when a page contains an <img> tag with a third-party origin in the img *src* attribute.

All latest versions of browsers Chrome, Edge, Firefox, Opera, and Safari provide support for CSP.

Topics to be covered

- Policy directives.
- Source List Reference.
- Apply the policy.
- Test your policy.
- Browser supports.
- Threats
- Troubleshoot

Policy directives

The CSP is used to restrict unauthorized third-party content resources. There are many directives available for a source (application). Once Content-Security-Policy headers are included in your application, the browser will reject any other content from sources that are not explicitly included or pre-approved using any of the directives.

You can add the directives in your ASP.NET web application's HTTP response header GUI in the IIS manager or add the following to your **Web.config** file.

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

The following are some of the policy directives:

default-src

default-src is used as a fallback if the directives (**object-src, img-src, etc.**) are not specified. Then, the **default-src** content policy will be applied for source directives.

- Use **default-src 'self'** to allow content from the current origin.
- Use **default-src 'none'** to block everything that's not added (pre-approved).

Example:

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

script-src

script-src is used to pre-approve script sources.

1. Use **script-src 'self'** to allow scripts from the current origin.
2. You can pre-approve your scripts using **script-src 'https://www.example.com/scripts/*'**. It will allow domain scripts in this URL.
3. **script-src '*.googleapis.com www.example.*'** is used to allow all domain scripts.
4. Specify **unsafe-eval** to use eval () methods for creating code from strings.

```
<add name="Content-Security-Policy" value="script-src 'self' 'unsafe-inline
```

5. **script-src 'unsafe-inline'** is used to allow inline scripts. In this, you can write <script> </script> directly in the view. Don't use inline script in your application directly. If you want to use inline script, you should use **nonce** to avoid security vulnerabilities.

```
<script nonce="r@nd0m">
        doWhatever();
</script>
<add name="Content-Security-Policy" value=" script-src 'r@nd0m'" />
```

Nonce browser support

The nonce directive is supported from CSP Level 2. It is supported by Chrome and Firefox after the version published in 2015, Safari 10+ or Edge 15+. It's not supported in all Internet Explorer versions; you need to use the Edge browser for nonce support instead of Internet Explorer.

style-src

style-src is used to pre-approve the CSS stylesheet sources.

1. Use **style-src 'self'** to allow stylesheets from the current origin.
2. You can pre-approve your styles using **style-src 'https://www.example.com/styles/*'**. It will allow domain styles in this URL.

```
<add name="Content-Security-Policy" value=" style-src 'self' 'https://www.e
```

object-src

object-src allows sources for the **<object>, <embed>,** and **<applet>** tags. You can specify **object-src 'none'** to prevent loading all URL sources.

```
<add name="Content-Security-Policy" value="object-src 'none'" / >
```

img-src

img-src is used to restrict image sources. You can pre-approve third-party images in CSP by specifying the domain.

```
<add name="Content-Security-Policy" value="img-src 'none'" / >
```

font-src

font-src is used to mention sources for loading fonts.

```
<add name="Content-Security-Policy" value="font-src 'none'" / >
```

media-src

media-src is used to restrict sources from loading sound and video resources.

```
<add name="Content-Security-Policy" value="media-src 'none'" / >
```

frame-ancestors

frame-ancestors is used to restrict URLs that can embed the current source in <iframe>, <object>.

```
<add name="Content-Security-Policy" value="frame-ancestors 'self'"/ >
```

upgrade-insecure-requests

upgrade-insecure-requests indicates that the content URL from insecure (HTTP) sources should be acquired securely over HTTPS.

```
<add name="Content-Security-Policy" value="upgrade-insecure-requests">
```

Source List Reference

Source Value	Example	Description
*	script-src '*'	Allows any URL except data: blob: filesystem: schemes
'none'	font-src 'none'	Doesn't allow loading resources from any source.
'self'	script-src 'self'	Allows loading resources from the same origin.
https:	style-src https:	Allows loading resources only over HTTPS on any domain.
data:	img-src 'self' data:	Allows loading resources via the data scheme (Base64 encoded images).
*.example.com	script-src '*.example.com'	Allows loading resources from any subdomain under

		*.example.com
https://cdn.com	script-src ‘https://cdn.com’	Allows loading resources only over HTTPS that matches the given domain.
‘unsafe-inline’	script-src ‘unsafe-inline’	Allows inline source elements such as style attribute, onclick, or script tag bodies. <script> </script> and <style> </style>
‘unsafe-eval’	script-src ‘unsafe-eval’	Allows unsafe dynamic code evaluation such as JavaScript **eval ().**
‘nonce-‘	script-src ‘nonce-r@nd0m’	Allows inline script or CSS to execute if the script (<script nonce = “r@nd0m ” >) tag contains a nonce attribute matching the nonce specified in the CSP header.

		The nonce should be a secure random string and should not be reused. It is applicable from **CSP Level 2.**
‘strict-dynamic’	script-src ‘strict-dynamic’	Allows script to load additional scripts via non-“parser-inserted” script elements (document.createElement(‘script’) ; is allowed). It is applicable from **CSP Level 3.**

Apply the policy

Use <system.webServer> tag to apply the CSP.

Place the directives in the name attribute value. Separate directives with a semicolon (;). Refer to the following code example.

Copy

```
<system.webServer>
    <httpProtocol>
        <customHeaders>
            <add name="Content-Security-Policy" value="upgrade-insecure-requests
        </customHeaders>
    </httpProtocol>
</system.webServer>
```

Test your policy

After implementing the CSP in your application, you can validate your CSP directives. It helps to confirm that third-party scripts are not inadvertently blocked. If any of the CSP have failed in your application, you can generate a report to rectify them.

For more information, see MDN web docs: Content-Security-Policy-Report-Only and Google CSP evaluator.

Browser supports

CSP supports all latest versions of modern browsers but not Internet Explorer at all. Refer to the following links.

- Browser support list
- CSP level

Threats

There are many open tools available to scan your application. OWASP is one of the finest applications for finding security vulnerabilities. Once you implement the CSP, you can scan your complete application. This will not affect your application's performance. You can also scan your production site for:

- Cross-site scripting
- SQL/data injection

See more about security vulnerabilities here.

Troubleshoot

If CSP is not implemented properly in your application, the errors will appear in your browser console. The browser will provide the details about the scripts that are blocked by your webpage. It will also provide the details like:

- How to change the policy to allow a blocked item.
- Elements that don’t accept the policy.

> **Note:** CSP is only effective when the client’s browser supports all the included directives. For the latest browser support matrix, check Can I use: Content-Security-Policy.

Conclusion

In this blog, we have seen the steps to implement Content Security Policy (CSP) in your ASP.NET MVC web applications. I hope this blog post was helpful to you.

Syncfusion provides 80+ ASP.NET Core, and ASP.NET MVC UI controls for web application development. We encourage you to take a moment to learn about our products and browse our interactive demos.

For existing customers, the latest version of our products is available for download from the License and Downloads page. If you are not yet a Syncfusion customer, you can try our 30-day free trial to check out our available features. Also, try our samples from this GitHub location.

For questions, you can contact us through our support forum, support portal, or feedback portal. We are always happy to assist you!

Related blogs

- Easily Publish an ASP.NET Core App in Linux Docker that Compresses PDF Documents
- How to Migrate ASP.NET HTTP Handlers and Modules to ASP.NET Core Middleware
- Create Report Viewer Component in Angular app with ASP.NET Core
- Easily Perform LINQ Mocking to Unit Test ASP.NET Core Application

MEET THE AUTHOR

Karthik Anandan

Karthik Anandan has been a web developer at Syncfusion Software since 2015. He has experience in web development and website security. He is interested in learning new web technologies.