CONTENT SECURITY POLICY VALIDATION

L.Petkova. PhD
University of Library and Information technology (UNIBIT)

lilyanapetkova92@gmail.com

Abstract: *Due to the unstoppable growth of security flaws, the developers need to pay attention and be patient in the integration of security defence in the application development! In most of the cases, they are either uninformed or unqualified of implementing it which cause some huge breaks in the application! There are a lot of documentations/guidelines/tools free for use to help the developers in their work! For the last few years browsers have integrated certain security header controls to support the web application security! In the present research we will present, in our opinion, one of the most important http security response header - the one responsible for the security of the main base of a web application namely the content! Content Security Policy may help in preventing the some of the most vulnerable security attacks (XSS), but in the hand of an unexperienced developer it can breaks the entire application!*

Keywords: HTTP HEADERS, SECURITY, CONTENT POLICY, VALIDATION

1. Introduction

The present article presents a mechanism for web applications to mitigate injection attacks and vulnerabilities, such as cross-site scripting (XSS). That type of attacks allows malicious scripts to be send to the end user to access sessions tokens, perform unauthorized activities, sensitive information, or even rewrite the content of the HTML page.[1][13]

Flaws that allow these attacks to succeed are quite widespread and can be occured on application with no data input validation. [1]

To add another level of security, here comes the Content Security Policy Http header. It declares from where the web application will load scripts, content, circumstances which will block any not declared script injected into the application.

Content Security Policy (CSP) is not intended as a first line of defense against content injection vulnerabilities. Instead, CSP is best used as defense-in-depth, to reduce the harm caused by content injection attacks.

2. Directives

CSP has over 20 different directives which can be used to restrict the load of content on the site! OWASP Secure Headers Project provides a full list of options with description for its purpose.

In our earlier research we have presented how to integrate the http security headers into ан ASP.NET project!

This specific header, topic of the current article, can be added to the configuration by describing each directive one after another, separated with ";"! For every directive we should add a list of all content/scripts allowed to be loaded on the site! Anything other than the presented in the list will not be loaded!

Table 1: *CSP Directives [2][3][12]*

Directive	Description
base-uri	Define the base uri for relative uri
default-src	**Define loading policy for all resources type in case of a resource type dedicated directive is not defined (fallback)**
script-src	**Define which scripts the protected resource can execute**
object-src	**Define from where the protected resource can load plugins**
style-src	**Define which styles (CSS) the user applies to the protected resource**
img-src	**Define from where the protected resource can load images**
media-src	**Define from where the protected resource can load video and audio**
frame-src	**Define from where the protected resource can embed frames**
child-src	Define from where the protected resource can embed frames
frame-ancestors	Define from where the protected resource can be embedded in frames.
font-src	**Define from where the protected resource can load fonts**
connect-src	**Define which URIs the protected resource can load using script interfaces**
manifest-src	Define from where the protected resource can load manifest.
form-action	**Define which URIs can be used as the action of HTML form elements**
sandbox	**Specifies an HTML sandbox policy that the user agent applies to the protected resource**
script-nonce	**Define script execution by requiring the presence of the specified nonce on script elements**
plugin-types	**Define the set of plugins that can be invoked by the protected resource by limiting the types of resources that can be embedded**
reflected-xss	**Instructs a user agent to activate or deactivate any heuristics used to filter or block reflected cross-site scripting attacks, equivalent to the effects of the non-standard X-XSS-Protection header**
block-all-mixed-content	Prevent user agent from loading mixed content.
upgrade-insecure-requests	Instructs user agent to download insecure resources using HTTPS.
referrer	Define information user agent must send in Referer header.
report-uri	**Specifies a URI to which the user agent sends reports about policy violation**
report-to	Specifies a group (defined in Report-To header) to which the user agent sends reports about policy violation.

In our personal experience we have been influenced by Scott Helme - a security researcher and security tools developer. The table above provides information for all directives with the description to each one of them and the ones in bold are those which we have used in our workflow.

Further in the research we present idea of having different versions of the content security policy! Basically some versions have added some new directive which are not backwards compatible.

In the following table we present compatibility of some of the directives with the browsers versions and the CSP version it comes from.

Table 2: *CSP Directives Values Browser Compatibilities [2][11][12][14]*

Value	CSP version	Compatibility
base-uri	CSP Level 2	Chrome 40+; Firefox 35+; Safari 10+
default-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+
script-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+

object-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+
style-src	CSP Level 1	Chrome 25+; Firefox 23+ ; Safari 7+; Edge 12+
img-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+
media-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+
frame-src	Deprecated	
child-src	CSP Level 2	Chrome 40+; Firefox 45+; Edge 15+
frame-ancestors	CSP Level 2	Chrome 39+; Firefox 33+; Edge 15+
font-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+
connect-src	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+
form-action	CSP Level 2	Chrome 40+; Firefox 36+; Edge 15+
sandbox	CSP Level 1	Chrome 25+; Firefox 50+; Safari 7+; Edge 12+
script-nonce	CSP Level 2	Chrome 40+; Firefox 35+; Safari 10+
plugin-types	CSP Level 2	Chrome 40+; Safari 10+; Edge 15+
report-uri	CSP Level 1	Chrome 25+; Firefox 23+; Safari 7+; Edge 12+

3. Source List

The source list provides requirements for each directive what source can be loaded. Each directive can have multiple source values separated with a space. The one exception thought is the 'none' value which should be the only value declared.

Table 3: *CSP directives values[2][3]*

Value	Description
*	Wildcard, allows any URL except data: blob: filesystem: schemes
'none'	Prevents loading resources from any source
'self'	Allows loading resources from the same origin, but not subdomains (same scheme, host and port).
data:	Allows loading resources via the data scheme (eg Base64 encoded images).
'unsafe-inline'	Allows the use of inline JavaScript and CSS source elements
'unsafe-eval'	Allows unsafe dynamic code evaluation such as JavaScript eval()
'nonce-'	Allows script or style tag to execute if the nonce attribute value matches the header value. *(investigated further in the article)*
'sha256-'	Allow a specific script or style to execute if it matches the hash. *(investigated further in the article)*
https:	Allows loading resources only over HTTPS on any domain.
https://cdn.com	Allows loading resources only over HTTPS matching the given domain.
***.example.com**	Allows loading resources from any subdomain under example.com.
domain.example.com	Allows loading resources from the specified domain name.

3.1. Nonce

The use of nonce value is more suitable for dynamically generated pages which requires the use of a sufficiently random and long nonce to prevent an attacker predicting the value for their own scripts. The use of the same nonce value on every script on the page is allowed but it needs to be re-generated and inserted into the header and all script elements on each response. [3]

The CSP directive which uses the nonce value would look like this:

Content-Security-Policy: script-src 'self' 'nonce-rand1234'

And any inline script on the page would use that value like the following one:

<script nonce="rand1234">...</script>

3.2. Hash

The Hash method is more suitable for static content. The browser hashes any inline JavaScript or CSS and executes its if the value matches the one in the HTTP header. [3]

Content-Security-Policy: script-src 'sha256-646840DCAA5….'

3.3. CSP configuration rule

As we have already investigated our past research titled Http Security Headers, there are two ways of configure http header rule in an ASP.NET project: through IIS platform and coding it in the web.config file of the web project!

Here is an example of CSP configuration written in the web.config of the project:

```
<remove name="Content-Security-Policy" />
<add name="Content-Security-Policy" value="default-src 'self'
https://site.com/ *.googleapis.com www.gravatar.com player.vimeo.com
google-analytics.com googletagmanager.com blob: data:;
script-src 'self' 'unsafe-eval'  *.googleapis.com www.google-
analytics.com 'unsafe-inline';
style-src 'self' 'unsafe-inline' *.googleapis.com;
img-src 'self' data: blob: www.gravatar.com *.gstatic.com umbraco.tv;
font-src 'self' blob: data: *.googleapis.com *.gstatic.com;
object-src 'self';
base-uri 'none';
frame-src 'self' site.com;
worker-src 'self' blob: data: filesystem:;
connect-src 'self' *.googleapis.com blob: file: data: filesystem:;"/>
```

4. CSP Versions

Over the years due to the evolution of vulnerabilities and attacks the CSP has the need to evolve too. Some of the changes are related to the need of new directives, other to the need of more detailed violation reports. This leads to the separation of the CSP in several versions.

Initially CSP is designed to be fully backward compatible. But CSP version 2 has some explicitly-mentioned inconsistencies in backward compatibility. Browsers that don't support it still work with servers that implement it, and vice-versa: browsers that don't support CSP simply ignore it, functioning as usual, defaulting to the standard same-origin policy for web content. If the site doesn't offer the CSP header, browsers likewise use the standard same-origin policy.[11]

4.1. CSP Level 1.0

CSP Level 1.0 was defined November 2012. The initial directives were: ***default-src; script-src; object-src; style-src; img-src; media-src; frame-src; font-src; connect-src; sandbox; report-uri***. [8]

From now on every other version lays on all rules and directives from this one!

4.2. CSP Level 2.0

This version is proposed November 2016. Here are some of the changes applied on version 2.0: [9]

- The following directives are brand new in this revision: ***base-uri; child-src; form-action; frame-ancestors; plugin-types***;
- If the loaded resource is a result of a redirect, the path component of the source is now ignored;
- Individual inline scripts and stylesheets may be whitelisted via **nonces** and **hashes** *(presented in the article in point 3)*;
- A **SecurityPolicyViolationEvent** is fired upon violations;

- A number of new fields were added to violation reports, such as **effectiveDirective**, **statusCode**, **sourceFile**, **lineNumber**, and **columnNumber**.

4.3. CSP Level 3.0

CSP version 3.0 is barely new. It was proposed October 2018 which in our personal experience still makes it inappropriate to be used.

Some of the changes applied on version 3.0 are:[10]

- The frame-src directive, which was deprecated in CSP Level 2, has been undeprecated, but continues to defer to child-src if not present (which defers to default-src in turn);
- New directives: worker-src; manifest-src;
- The URL matching algorithm now treats insecure schemes and ports as matching their secure variants. That is, the source expression http://example.com:80 will match both http://example.com:80 and https://example.com:443;
- Reports from inline violations will now report "inline" as the blocked resource; a sample attribute if the relevant directive contains the 'report-sample' expression;
- The report-uri directive is deprecated in favor of the new report-to directive;
- The 'strict-dynamic' source expression will now allow script which executes on a page to load more script via non-"parser-inserted" script elements;
- The 'unsafe-hashes' source expression will now allow event handlers, style attributes and javascript: navigation targets to match hashes.

5. Browser Compatibility

The following compatibility table is structured according a global usage statistics from March 2019! The global usage of CSP Level 2 almost 10% over the one of CSP Level 1.0. The statistics show that around 88.69% of the browsers support CSP Level 2 and 7% from it support it partially! For CSP Level 1.0 those numbers are higher - 96.47% of the browsers support it and only 3% from it partially used it! The below table gives more details about different versions of CSP and their supports on 7 different browsers: Internet Explorer, Edge, Mozilla Firefox, Chrome, Safari, Opera and Android. [14]

Table 4: *CSP versions Browsers Compatibilities [2][14]*

	IE	Edge	Firefox	Chrome	Safari	Opera	Android
Content-Security-Policy ***Level 2***	-	18 **partial**	67 **partial**	73	12.1	58	67
Content-Security-Policy ***Level 1.0***	11 **partial**	18	66	73	12.1	58	67
X-Content-Security-Policy ***Deprecated***	10+ **Limited**	12+ **Limited**	4+	-	-	-	-
X-Webkit-CSP ***Deprecated***	-	-	-	14+	6+	-	-

As we see from the table CSP Level 2 is partially supported from version 18 of Edge browser and version 67 of Mozilla Firefox. And its usage on Internet Explorer is unknown! However CSP Level 1.0 usage on IE is partially supported on version 11! X-Content-Security-Policy and X-Webkit-CSP are deprecated and although some browsers support them, experts advised to avoid their use!

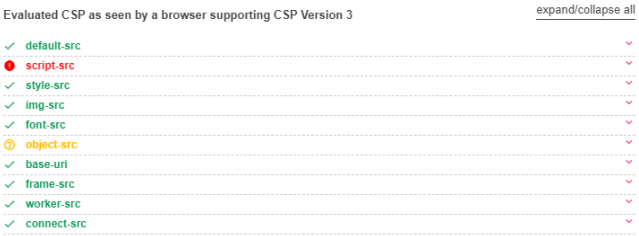
6. CSP validators

There are a lot security validation tools to help the developer in integrating the content security policy (*along with the other HTTP security headers*) and depending the developer's resources, those tools can be free and paid.

In our personal experience we have worked with several free tools which provided the necessary results.

The extension we have used to help us in check the CSP integration on a site is called called CSP Evaluator and is used as an extension to the Chrome browser. It gives a live time report of the CSP integration on the loaded site. If the badge is in green color then the CSP with the supported version is integrated successfully on the site! Otherwise it gives a list of directives with its correct and wrong values to be fixed.

Image 1: *CSP evaluator report*



Another free tool is the one that can be found on the following Internet address: https://securityheaders.io/. It can be also added as an extension to the browser. It gives a full HTTP security headers report!

7. Results

During our research we made several security checks on a few selected web applications. As part of our research we have again used the ALEXA analytical insight to choose 6 website from an excerpt of 50 sites based on a selected Country category - Bulgaria. [7]

The sites selected for this articles were selected on a personal opinion based on the most advertised and the most commonly used website. We are not going to present the name of sites due to law and policies restrictions! [7]

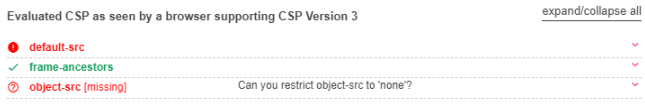
Checking the selected sites with https://securityheaders.io/ we can see that 1 out of 6 websites are using the CSP policy to protect their site from XSS attacks and vulnerabilities.

Table 5: *CSP security statistic* [7]

	site A	**site B**	**site C**	**site D**	**site E**	**site F**
Content-Security-Policy	+	-	-	-	-	-

But if we run the CSP evaluator tool we can see that the CSP protection is not fully integrated which makes it not too secure:

Image 2: *CSP for site A*



The results show that the web applications we have selected are not secure from XSS attacks. And considering that we have selected some of the most commonly used website in Bulgaria, that makes us think how much the content we provide to those sites is safed and how much we are exposed to malicious actions.

8. Conclusion

The need of content security integration becomes larger with the evolution of technology! It does not fully protect our applications but it gives an extra level of security by restricting the attackers actions.

And from the analytics we have provided we have to consider that the poor CSP is not a giving the necessary level of protection. If it is not properly integrated, its purpose stays unachieved.

References

[1] OWASP, Cross-site scripting
[2] Content Security Policy Reference, Founder Inc. 2012-2016
[3] Content Security Policy - An Introduction, Scott Helme, November 2014
[4] ICT Security Trends, Willian Dimitrov, Sofia, 2017, Avangard, ISBN 978-619-160-766-2
[5] Software testing, Willian Dimitrov, Sofia, 2017, Avangard, ISBN 978-619-160-765-5
[6] ICT Security Model, Willian Dimitrov, Sofia, 2018, Avangard, ISBN 978-619-160-950-5
[7] HTTP SECURITY HEADERS, Lilyana Petkova, March 2019
[8] Content Security Policy 1.0, Brandon Sterne, Adam Barth, November 2012, W3C
[9] Content Security Policy 2.0, Mike West, Adam Barth, Dan Veditz, Brandon Sterne, November 2016, W3C
[10] Content Security Policy 3.0, Mike West, October 2018, W3C
[11] Content-Security-Policy, Mozilla Developers Network, April 2019
[12] Content Security Policy, Mozilla Developers Network, April 2019
[13] 5 Practical Scenarios for XSS Attacks, Satyam Singh, October 2018
[14] https://caniuse.com, April 2019