Content Security Policy (CSP) as countermeasure to Cross Site Scripting (XSS) attacks

Ivan DOLNÁK*
* University of Žilina, Faculty of Electrical Engineering, Žilina, Slovak Republic
ivan.dolnak@fel.uniza.sk

***Abstract*—This article presents one of several HTTP Security Headers – Content Security Policy (CSP) header, nowadays preferred countermeasure to Cross Site Scripting (XSS) attacks. It is emphasized that implementation of CSP header is relatively simple method how to improve security level of communication among people and devices over Internet. In the environment where secure web services are vital – web applications and Internet of Things (IoT) networks, this way it is possible to strictly define communication parties and assets used in web services. Simple and effective reporting is native part of the CSP design, what means administrators can be notified about running attacks almost instantly. It is demonstrated on practical examples what benefits of CSP implementation can bring to communication and how easy is to propagate CSP header to web browser.**

I. INTRODUCTION

Secure communication is always current topic and new security challenges are discovered every day. Securing communication between web server and web browser is not totally brand-new area, but there are appearing new security tools – HTTP Security Headers, which have identified new techniques in securing communication between the above mentioned parties.

Communication among people and devices over HTTPS protocol is widely accepted today, but allowing communication only among approved domains or subdomains is relatively new security request. Thanks to web applications and IoT networks, it is becoming important to implement countermeasures to Cross Site Scripting (XSS) attacks. Hardening communication over HTTPS protocol and allowing communication only among allowed domains and subdomains, brings requested results.

Despite the fact HTTPS protocol is relatively secure, it is necessary to think about possible threats in HTML source code resulting to inappropriate use of URIs. For example, it is possible to overlook HTTP protocol used instead of HTTPS even in well-managed code. Second example are dynamically generated HTML pages, where everybody from the Internet can inject links to the 3rd party malicious targets and these malicious assets can lead to unpredictable web application or web service behaviour.

Implementation of Content Security Policy (CSP) as HTTP Security Header is the main topic of this article. CSP is one of several HTTP Response Headers used in last time for advanced protection of communication over HTTP and HTTPS protocols. It brings higher level of security in communication between web server and web browser, and among devices communicating in IoT networks.

CSP header is not mandatory today, but it is recommended for environments, where communicating parties (people and devices) want to be sure they use assets (source code and services) only among approved domains and subdomains.

Nowadays, everybody can evaluate security level of web service thanks to publicly available testing tools like those available at www.securityheaders.io developed by Scott Helme. It is possible to scan web servers and evaluate them on scale from A+ to F [1].

Used methodology is not exact or scientific, but it offers overview on security level web servers can gain. In time of writing this article (October 2017), above mentioned web site published grand total statistics about grades, which have gained domains, subdomains (or fully qualified domains), tested by Internet volunteers for themselves.

From approximately 7 million realized tests, there were published pairs – grade and totals, which have led to the following grand total statistics.

Approximately 24% of web sites (or fully qualified domains) is in very good condition, with score A+ or A. Another 20% of web sites is in average conditions, with score on scale from B to D. In the first case the reason is that administrators already explored new security threats and possible workarounds. In the second case, the reason could be fact, that new HTTP Response Headers (known as HTTP Security Headers) are not so widely adopted yet and it is usually possible to identify only some of them in production environment (configuration).

Unfortunately, approximately 56% of web sites is in poor or even very bad conditions, because according to test results, they have gained score E or F.

Overlooking the fact this statistic is only informational and not scientifically exact, it highlights the necessity of adopting HTTP Security Headers in communication over Internet realized by HTTP or HTTPS protocols.

II. CONTENT SECURITY POLICY

Content Security Policy (CSP) as another kind of HTTP Security Header has purpose, like in case of HTTP Strict Transport Policy, to deliver from web server to web browser security policy in form of HTTP Response Header. In this policy are included assets – sources of content that the web browser may load. It is effective countermeasure to Cross Site Scripting (XSS) attacks and this mechanism is widely accepted among the top web

browsers, like: Chrome, Firefox, Opera, Safari, Internet Explorer and Edge. Unfortunately, support of SCP headers vary among them and among browser versions too [2].

A. *Is Content Security Policy necessary?*

Web browsers not only load HTML code alone, but they load assets mentioned in code, like stylesheets, java scripts, fonts, image, etc., because they were instructed to do so by source code of web page. Web browsers do not have any method to identify, whether they may or may not load those assets. An attacker can place a special snippet of code on web page to instruct browsers download 3rd party malicious code. In traditional way, web browsers have no reason not to trust the assets from malicious domains or subdomains a not to load it.

This is where Content Security Policy comes in and offers elegant solution to this security issue – called Cross Site Scripting (XSS) attack.

B. *Approving secure sources*

CSP header defines allowed sources for web page content that web browser can load. By identifying approved sources that web browser use for page rendering, it is possible to protect web browser from a range of security issues.

CSP Response Header is pretty simple method how to give permission to web browser to load for example script only from domain self. This is illustrated in form of HTTP Response (CSP) header on Fig. 1.

```
Content-Security-Policy: script-src 'self'
```

Figure 1. Example of CSP header [2]

Using before mentioned example of an attacker with malicious code from 3rd party domain, now the web browser prevents to load java script from domain or subdomain different from that for which CSP header was specified.

Like in case of HSTS header, CSP header consists of name-value pair, where name is "Content-Security-Policy" or "Content-Security-Policy-Report-Only" and value is a string of directives, separated by semicolon. For example, Fig. 1 specified directive called "script-src", where approved sources are defined as "self". Keyword "self" specify whole domain and make policy easier to read as it grows. It is even faster to write CSP using "self" keyword, in comparison with using "https://domain.tld" string everywhere. It there is an attempt to load code from, for example www.attackers.com, web browser will now not load the script from this domain.

C. *What can be protected*

CSP defines a wide range of directives that can be used to enforce policy across all circumstances. Brief information about widely available CSP headers can be found at The Open Web Application Security Project [3] and detailed descriptions of directives are written by W3C Consortium [4].

Despite the fact implementation of directives in web browsers vary [2], in literature there are usually mentioned the following directives:

- **default-src**: defines loading policy for all resource type in case of a resource type dedicated directive is not defined (fallback),
- **script-src**: defines which java scripts the protected resource can execute,
- **object-src**: defines from where the protected resource can load plugins,
- **style-src**: defines which styles (shylesheets, or CSS) the web browser applies to the protected resource,
- **img-src**: defines from where the protected resource can load images,
- **media-src**: defines from where the protected resource can load video and audio,
- **frame-src**: defines from where the protected resource can embed frames,
- **font-src**: defines from where the protected resource can load fonts,
- **connect-src**: defines which URIs the protected resource can load using script interfaces,
- **form-action**: defines 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**: defines script execution by requiring the presence of the specified nonce on script elements,
- **plugin-types**: defines 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 web browser 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,
- **report-uri**: specifies a URI to which the user agent sends reports about policy violation.

Precise definitions of CSP headers in time of writing this article (October 2017) is available in form of Editor's Draft by W3C Consortium [4].

D. *Creating a CSP header*

Process of specification of SCP header could be as specific as it meets requirements. The following example policies illustrate what is possible, but none of them is mandatory.

- Example 1: Allowing any assets to be loaded over HTTPS protocol from any domain, is illustrated on Fig. 2.

```
Content-Security-Policy: default-src https:
```

Figure 2. CSP header example #1

- Example 2: Allowing any assets to be loaded from any domain on specific domain using any scheme or port, is illustrated on Fig 3.

```
Content-Security-Policy: default-src domain.tld
```

Figure 3. CSP header example #2

- Example 3: Allowing only assets to be loaded from specific domain over https on any port, is illustrated on Fig. 4.

```
Content-Security-Policy: default-src https://domain.tld:*
```

Figure 4. CSP header example #3

It was proofed by author's praxis, that the most common starting point for creating schema is the example on Fig. 5.

```
Content-Security-Policy: default-src https://domain.tld
```

Figure 5. Starting point for creating SCP schema

Above mentioned schema can also use the following keywords [2] [4]:

- none – blocks the use of this type of resource,
- self – matches the current domain, but not subdomains,
- unsafe-inline – allows the use of inline java scripts and stylesheets,
- unsafe-eval – allows the use of mechanisms like eval().

In a process of creating CSP schema it mandatory to use the following rules:

1. Wildcards character * can be used for the scheme as port and left most part of a domain.
2. It is not allowed to specify directive twice, because the second will be ignored.
3. There is not inheritance from the default source directive.

For testing purposes, it was used author's personal web site available at https://www.dolnak.eu, where were used the most common 3rd party services used today, like services from Google, Cloudflare, jQuery, Bootstrap, etc.

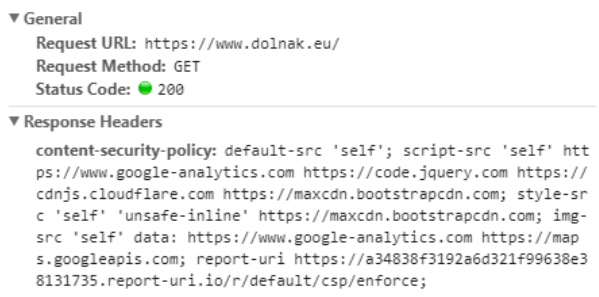

Figure 6. CSP header implemented on author's personal web page

The same configuration options for Apache httpd server are presented in more readable form on the Fig. 7. In the code snippet are used SCP directives:

- Default Source
- Script Source
- Style Source
- Image Source
- Report URI.

```
<IfModule mod_headers.c>

  Header always set Content-Security-Policy "
            default-src 'self'; \
            script-src 'self'
                        https://www.google-analytics.com \
                        https://code.jquery.com
                        https://cdnjs.cloudflare.com \
                         https://maxcdn.bootstrapcdn.com; \
            style-src 'self' 'unsafe-inline'
                        https://maxcdn.bootstrapcdn.com; \
            img-src 'self' data:
                        https://www.google-analytics.com \
                        https://maps.googleapis.com; \
            report-uri https://.report-uri.io/r/default/csp/enforce;"

</IfModule>
```

Figure 7. SCP configuration snippet for Apache httpd

E. Testing a CSP header

After creation of first CSP header version, it is recommended to start in "Report Only" mode, illustrated on Fig. 8.

```
<IfModule mod_headers.c>

  Header always set Content-Security-Policy-Report-Only "default-src 'self'"

</IfModule>
```

Figure 8. Report-Only version of SCP header

This "mode" actually means there are two version of SCP header, with and without 'Report-Only" supplement. Because different web browsers understand to CSP header parameter differently, "Report Only" mode offers possibility to test the policy among the most common web browsers used for specific domain (website) [2].

This usually means the browser receive and act upon the policy according the internal rules, but instead of enforcing the policy, it will give a feedback on what the effects of the policy would have been.

This way it is possible to tune policy without the risk of blocking some assets like java scripts, stylesheets or even some domains and subdomains.

There is also possibility to send from web server to web browser both headers – with and without "Report-Only", what is usable in situation where "Report-Only" header is optimal and can be used as production ready – without "Report-Only" supplement.

Second option is to use production ready header and still use testing header in order to enforce policy and test new parameters at the same time. For this purpose is very helpful report directive.

F. *Reporting policy violation*

Identifying links in HTML code to disallowed asset or attack due to found XSS attack vector is not possible without appropriate reporting. Without reporting, there is no straightforward way how to identify policy violation. For information what is happening on web server accessed by web browser, it is necessary to use "report-uri" directive.

This directive specifies at which location the web browser should send JSON formatted report in the event of CSP policy violation. Example of JSON report is illustrated on Fig. 9 without deeper explanation. In author's opinion, repot format is self-explanatory.

```
{
  "csp-report": {
    "document-uri": "https://www.dolnak.eu/",
    "violated-directive": "img-src",
    "effective-directive": "img-src",
    "original-policy": "default-src 'self'; report-uri https://.report-uri.io/r/
default/csp/reportOnly;",
    "disposition": "report",
    "blocked-uri": "https://img.routerboard.com//mimg/785_m.png",
    "line-number": 81,
    "source-file": "https://www.dolnak.eu/",
    "status-code": 0,
    "script-sample": ""
  }
}
```

Figure 9. JSON formatted report

Today, because of transformation of web sites from HTTP to HTTPS protocol, there are situations where some asssets are to be downloading via HTTP protocol and if it is not allowed by CSP policy, report directive helps to send message about it to specific location.

G. *CSP builder and reporting service*

As in the case of HTTP Security Header testing, there is free and publicly available CSP builder and reporting service (Fig. 10) created by Scott Helme and available at: https://report-uri.io/ [5] [6].

It is really "Real time security reporting tool", as it calls itself, and it helps to implement and monitor implementation of several HTTP Security Headers.

Figure 10. Report-uri.io web site logo

CONCLUSION

This article presented introduction to Content Security Policy header, a new security technology used in web and IoT environment for improving security level of communication over Internet among people and devices. It was illustrated how simple is to adopt Content Security Policy and make communication among people and device over Internet even more secure.

REFERENCES

[1] Scott Helme. *SecurityHeaders.io web site.* October, 2017. Available at: https://securityheaders.io

[2] Scott Helme. *Content Security Policy – An Introduction.* November 2014. Available at: https://scotthelme.co.uk/content-security-policy-an-introduction/

[3] The Open Web Application Security Project. *Content Security Policy.* August, 2013. Available at: https://www.owasp.org/index.php/Content_Security_Policy

[4] W3C Consortium. *Content Security Policy Level 3.* August 2017. Available at: https://w3c.github.io/webappsec-csp/

[5] Scott Helme. *Report-uri.io web site.* October, 2017. Available at: https://report-uri.io/

[6] Scott Helme. *CSP Builder.* October, 2017 Available at: https://report-uri.io/home/generate