

CSP Series

1. Intro

CSP (Content Security Policy) is there / in-place to mitigate some attacks, such as xss, csrf. It behaves as a whitelist mechanism for resources loaded or executed on the website, and is defined by HTTP headers or meta elements.

Although CSP provides strong security protection, it also causes the following problems:

1. Eval and related functions are disabled
2. embedded JavaScript code will not be executed
3. remote scripts can only be loaded through whitelisting

These problems hinder the popularity of CSP. If you want to use CSP technology to protect your website, developers have to spend a lot of time separating the embedded JavaScript code and making some adjustments...

Browsers that support CSP

CSP mainly has three headers:

1. Content-Security-Policy (chrome 25+ Firefox 23+ Opera 19+)
2. X-Content-Security-Policy (Firefox 23+ IE10+)
3. X-WebKit-CSP (Chrome 25+)

The CSPs we often see are similar to this:

```
header("Content-Security-Policy: default-src 'none';
connect-src 'self';
frame-src 'self';
script-src xxxx/js/ 'sha256-KcMxZjpVxhUhzZiwuZ82bc0vAhYbUJsxyCX0DP5ulTo=' 'sha256-
u++5+hMvnsKeoBWohJxx03U9yHQHZU+2damUA6wnikQ=' 'sha256-
zArnh0kTjtE0VDnamf0rI8qSpoiZbXttc6LzqNno8MM=' 'sha256-
3PB3EBmojhuJg8mStgxkyy30EJYJ73ru0F7nRScYnxk=' 'sha256-
bk9UfcsBy+DUFULLU6uX/sJa0q707B8Aal2VVL43aDs=' ;
font-src xxxx/fonts/ fonts.gstatic.com;
style-src xxxx/css/ fonts.googleapis.com;
```

```
img-src 'self'");
```

As you can see it contains a wide variety of wording:

1. none and self, none of what the representative does not match, self representatives of matching homologous source
2. is similar matches such <https://example.com/path/to/file.js> special file, or https://example.com/ This will match everything under the source.
3. The third one is similar to https: and will match all sources that contain this special format.
3. It may also be example.com, which will match all sources of this host, or *.example.com, which will match all subdomains of this host.
4. The fifth is similar to nonce-qwertyu12345, which will match a special node.
5. Of course, there is encrypted similar to sha256-abcd ... It will also match a special node in the page (this value will change every time you modify it).

A detailed example can be found in the documentation:

```
serialized-source-list = ( source-expression *( RWS source-expression ) ) / "'none'"
source-expression      = scheme-source / host-source / keyword-source
                        / nonce-source / hash-source

; Schemes:
scheme-source = scheme ":" ; scheme is defined in section 3.1 of RFC 3986.

; Hosts: "example.com" / ".example.com" / "https://.example.com:12/path/to/file.js"
host-source = [ scheme-part "://" ] host-part [ port-part ] [ path-part ]
scheme-part = scheme
host-part   = "" / [ "." ] 1host-char ( "." 1host-char )
host-char   = ALPHA / DIGIT / "-"
port-part   = ":" ( 1DIGIT / "*" )
path-part   = path
; path is defined in section 3.3 of RFC 3986.

; Keywords:
keyword-source = "'self'" / "'unsafe-inline'" / "'unsafe-eval'"

; Nonces: 'nonce-[nonce goes here]
nonce-source = "'nonce-' base64-value '"
base64-value = 1*( ALPHA / DIGIT / "+" / "/" / "-" / "_" ) * 2( "=" )
```

```
; Digests: 'sha256-[digest goes here]'  
hash-source      = "" hash-algorithm "-" base64-value ""  
hash-algorithm = "sha256" / "sha384" / "sha512"
```

There is a small question about using IP...

Although the use of IP conforms to the above syntax, the security of requests directly to the ip address is itself in doubt, and if it is possible, it is better to use the domain name.

In General

The CSP detection method is to first determine the specific request type, and then return the name of a valid instruction in the following way. Depending on the type of the request, the following different steps will be performed:

To understand the following algorithm, we first need to know what is the originator of the request

A) Initiator: Each request has an initiator, including "download", "imageset", "manifest", or "xslt".

B) Destination: Each request has a corresponding destination, including "document", "embed", "font", "image", "manifest", "media", "object", "report", "script", "Serviceworker", "sharedworker", "style", "worker", or "xslt".

1. If the request's initiator is "fetch", return connect-src.
2. If the request's initiator is "manifest", return manifest-src.
3. If the request's destination is "subresource", return connect-src.
4. If the request's destination is "unknown", return object-src.
5. If the request's destination is "document" and the request's target browsing context is a nested browsing context: return child-src.
6. Audio -> "track" -> "vide, return media-src.
7. font, return font-src.
8. image, return image-src.
9. style, return style-src.

2. Prevention

CSP is especially important for your users: they no longer need to be exposed to any unsolicited script, content or XSS threats on your website.

The most important advantage of a CSP for a website maintainer is perception. If you set strict rules on the source of the picture, a script kid tries to insert an image of an unauthorized source on your website, then the picture will be banned, and you will receive a reminder as soon as possible .

Developers also need to know exactly what their front-end code is doing, and CSP can help them control everything. Will prompt them to refactor parts of their code (avoid inline functions and styles, etc.) and prompt them to follow best practices.

There are a few simple ways to prevent CSP based attacks.

Adding policies through meta tags: The preferred setting method for CSP is the HTTP header, which is very useful, but it is more straightforward to set through tags or scripts. WebKit has implemented the feature of setting permissions through meta elements , so you can now try the following settings in Chrome: add `<meta http-equiv = "X-WebKit-CSP" content = "[POLICY GOES HERE]"` in the header of the document `>`.

DOM API: If this feature is added in the next iteration of CSP, you can query the current security policy of the page through Javascript and adjust it according to different situations. For example, if `eval ()` is available, your code implementation may be slightly different.

Content security policy applies to all common resources

3. `content-src`: limit the type of connection (such as XHR, WebSockets, and EventSource)
4. `font-src`: Controls the source of web fonts. For example, you can use Google's web fonts through `font-src`
5. `img-src`: defines the source of the loadable image.
6. `media-src`: Restrict video and audio sources.
7. `object-src`: Restrict sources of Flash and other plugins.
8. `style-src`: Similar to `Script-src`, but only works on css files.

Under the CSP 1 specification, you can also set the following rules:

1. `img-src` Valid image source
2. `connect-src` Apply to XMLHttpRequest (AJAX), WebSocket or EventSource
3. `font-src` Valid font source
4. `object-src` Effective plug-in source (eg, , ,)
5. `media-src` Valid and source

The CSP 2 specification contains the following rules:

1. child-src Valid web workers and element sources, such as and <iframe> (this directive replaces the obsolete frame-src directive in CSP 1)
2. form-action Can be a valid source of HTML actions
3. frame-ancestors Use , <iframe>, , or useful source embedded resources
4. upgrade-insecure-requests Command the user agent to rewrite the URL protocol and change HTTP to HTTPS (for some websites that need to rewrite a lot of stale URLs).

3. CTF CONTEXT

Common attack surface

script-src : script: only trust the current domain name

object-src : Do not trust any URL, ie do not load any resources

style-src : stylesheet: trust only cdn.example.org and third-party.org

child-src : Must be loaded using the HTTPS protocol. This has been removed from the web standard and may not be supported in newer browsers.

Other resources: no restrictions on other resources

When CSP is enabled, non-CSP-compliant external resources are prevented from loading.

Example 1:

A fair amount of CSP's can be detected purely from source, lets look at the source code:

```
<?php
$headerCSP = "Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-
TmV2ZXIgz29pbmcgdG8gZ2l2ZSB5b3UgdXA='";

header( $headerCSP );

// Disable XSS protections so that inline alert boxes will work
header ( "X-XSS-Protection: 0");
```

```

<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(1)</script>

?>
<?php
if (isset ($_POST['include'])) {
$page[ 'body' ] .= "
" . $_POST['include'] . "
";
}
$page[ 'body' ] .= '
<form name="csp" method="POST">
<p>Whatever you enter here gets dropped directly into the page, see if you can get an alert
box to pop up.</p>
<input size="50" type="text" name="include" value="" id="include" />
<input type="submit" value="Include" />
</form>
';

```

```

① Cache-Control: no-cache, must-revalidate
② Connection: close
③ Content-Length: 4222
④ Content-Security-Policy: script-src 'self' 'unsafe-inli...29pbmcgdG8gZ2l2ZSB5b3UgdXA=';
⑤ Content-Type: text/html; charset=utf-8
⑥ Date: Mon, 13 May 2019 15:22:31 GMT
⑦ Expires: Tue, 23 Jun 2009 12:00:00 GMT

```

You can see there are `nonce` and `unsafe-inline` here I think the inspection point is the understanding of the parameters (special values) in the script-src moreover, The legal source of script-src in the http header has changed.

1. unsafe-inline, which allows the use of inline resources such as inline `<script>` elements, javascript: URLs, inline event handlers (such as onclick), and inline `<style>` elements. Must include single quotes.
2. nonce-source, only specific inline script blocks are allowed, nonce = "TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA"

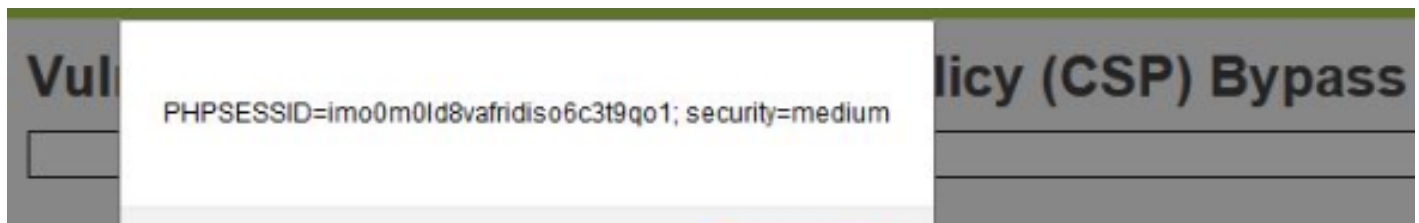
Basically.... It's even easier now, you can enter the following code directly:

```

<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert("document.cookie")</script>

```

The result is a successful injection.



By: Olivier (Boschko) Laflamme

- Twitter: https://twitter.com/olivier_boschko
- LinkedIn: <https://www.linkedin.com/in/olivierlaflammelink/>

Revision #1

Created 24 September 2022 00:48:58 by mxrch

Updated 24 September 2022 00:53:57 by mxrch