CSP Intro (Part 1 CSP Series)

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-KcMxZjpVxhUhzZiwuZ82bc0vAhYbUJsxyCXODP5ulto=' 'sha256-
u++5+hMvnsKeoBWohJxxO3U9yHQHZU+2damUA6wnikQ=' 'sha256-
zArnh0kTjtEOVDnamfOrI8qSpoiZbXttc6LzqNno8MM=' 'sha256-
3PB3EBmojhuJg8mStgxkyy3OEJYJ73ruOF7nRScYnxk=' 'sha256-
bk9UfcsBy+DUFULLU6uX/sJa0q7O7B8Aal2VVl43aDs=';
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   = "*" / [ "*." ] 1*host-char *( "." 1*host-char )
host-char   = ALPHA / DIGIT / "-"
port-part   = ":" ( 1*DIGIT / "*" )
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.

Revision #3
Created 5 May 2020 23:58:48 by Boschko
Updated 6 May 2020 00:09:59 by Boschko