SEMINAR 6
HTTP SECURITY AND CSP

The HTTP protocol as an additional security layer

JOSÉ MANUEL REDONDO LÓPEZ *"S-81 ISAAC PERAL"* PROJECT V3.0

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

INDEX

Introduction

José Manuel
Redondo López

- **The HTTP protocol is not only a mean to transport information between HTTP clients and servers**
- **It has a lot of additional features beyond merely carrying information**
  - Including security features
  - These features can be used **through HTTP headers**
  - But misusing these headers can be a problem too
- **This seminar will show you good and bad usages of HTTP headers**
  - So, you can benefit from this mechanism in your web applications (or in your web server configurations)

< Go to Index

Source: zona-militar.com

"Custom" HTTP headers are a security problem

Giving too much information inadvertedly

Remembering SEW: HTTP Requests

José Manuel
Redondo López

- **You need to remember some parts of the HTTP protocol explained in the SEW course**
- **HTTP requests have these elements**
  - One or more request headers
  - A blank line
  - The body of a message (optional)
  - the first line of an HTTP request has 3 elements separated by spaces
    - **A verb** indicating the HTTP method used
      - The most typical is GET, which reads a resource from the web server
    - The **requested URL**
    - The HTTP version used

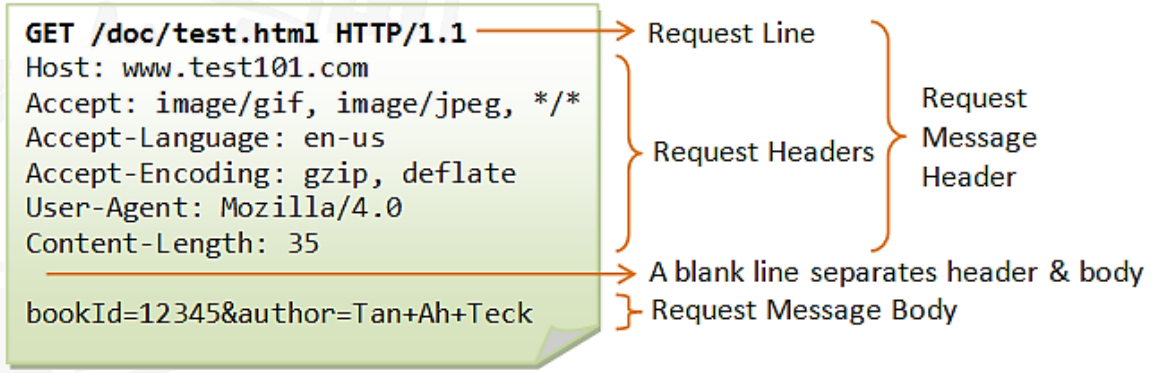
```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

Request headers

General headers

Entity headers

Source: https://www.freecodecamp.org/news/http-and-everything-you-need-to-know-about-it/

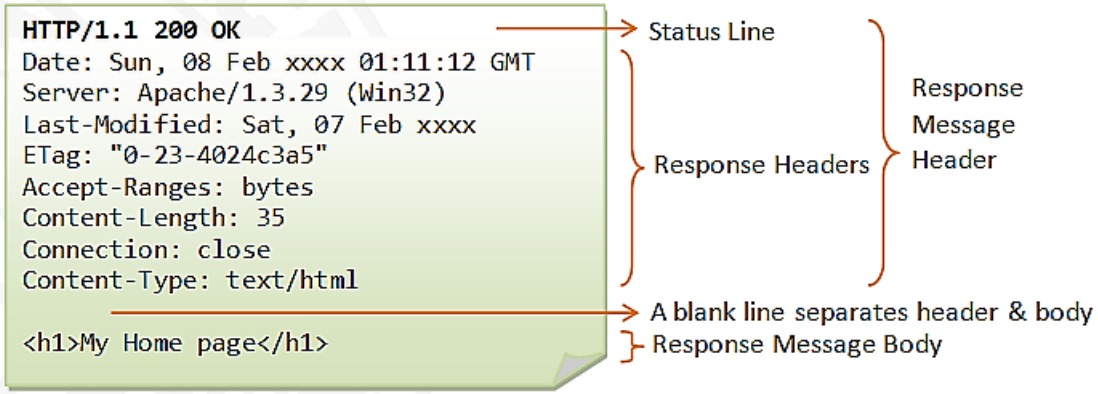


Source: https://aprendiendoarduino.wordpress.com/2017/09/11/protocolo-http-3/

Remembering SEW: HTTP Responses

José Manuel
Redondo López

- **The response to a typical HTTP request has three elements**
  - The **status line**, with
    - The HTTP version
    - A numeric status code
    - The text that describes the response status
  - **Headers**: Which are used to send instructions to the browser, such as
    - HTTP security headers
    - CSP (Content Security Policy)
  - **The body of the response**
    - Separated from the header by a blank line

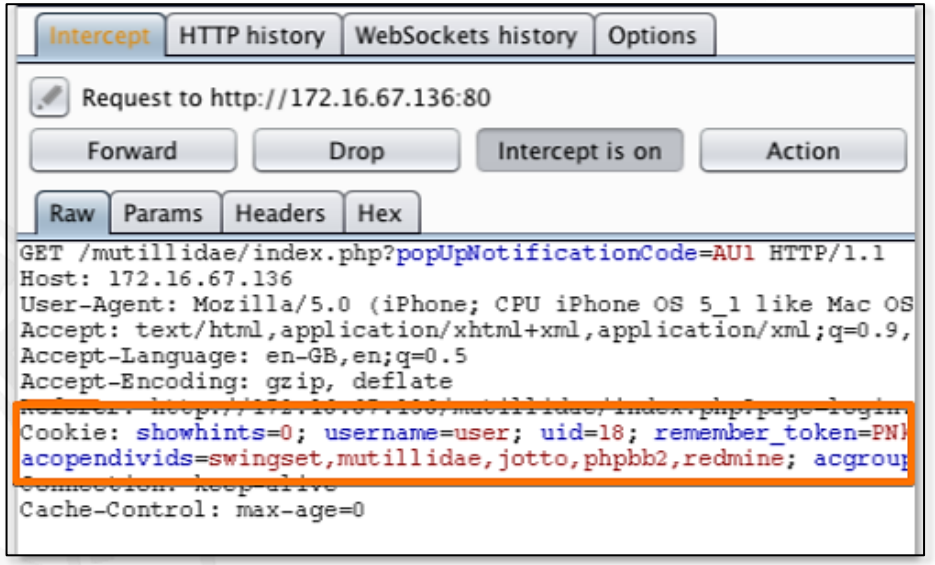


Source:
https://personales.unican.es/corcuerp/ingweb/notes/HTTP_Basics.html

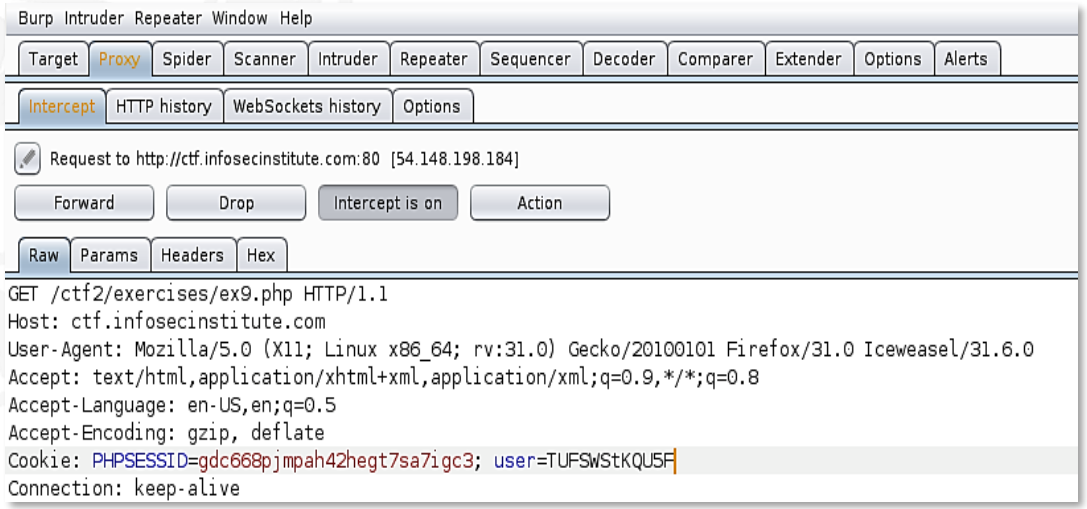
REMEMBERING SEW: COOKIES

José Manuel
Redondo López

- **Cookies are a vital part of HTTP**
  - Used to **provide state** to the protocol
  - Allows the server to send data to the client
  - The customer saves this data, and sends it back when he makes his next request
  - The client sends his current status in each request
- **They typically save a single session ID**
  - The server uses it to associate a state of that client from among all the states it saves
- **The server sends a cookie to the client using the `SetCookie` response header**
  - When the user makes the following requests to the server this cookie is added to the HTTP headers
  - The cookie may also contain more key-value pairs with additional information



Source: https://portswigger.net/support/using-burp-to-hack-cookies-and-manipulate-sessions

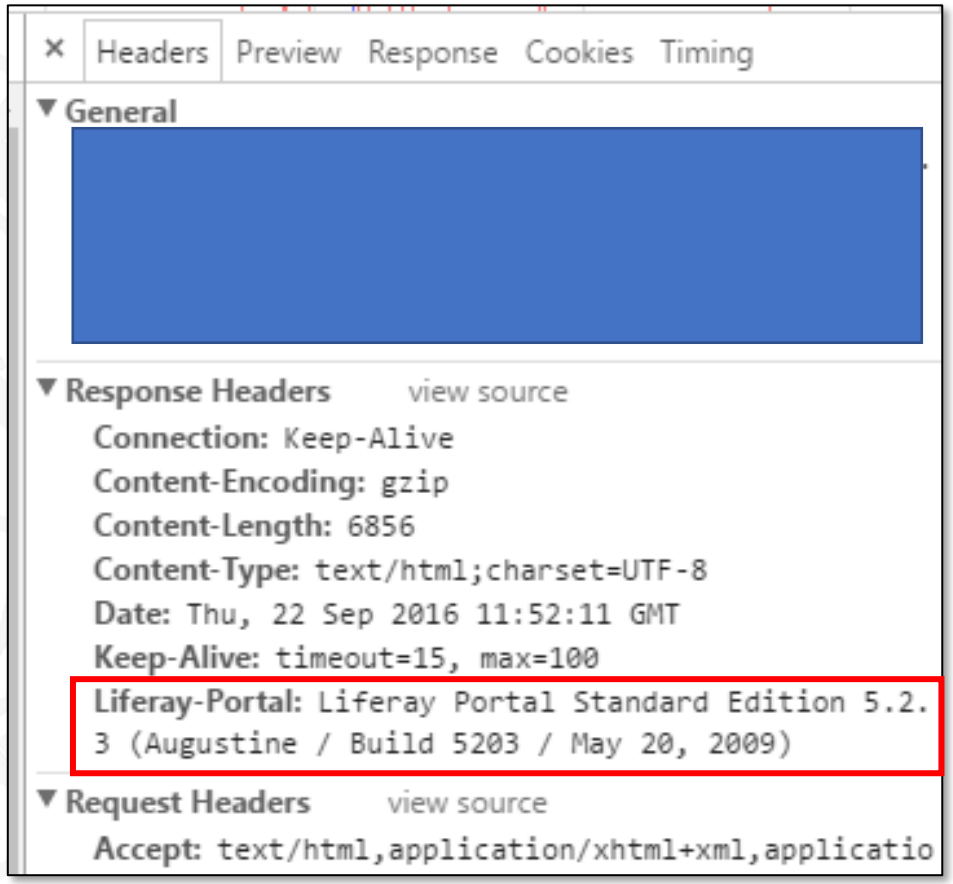


Source: https://uaf.io/web/2015/07/16/infosec-ctf2-level-9.html

HTTP HEADERS THAT ARE SECURITY PROBLEMS

José Manuel
Redondo López

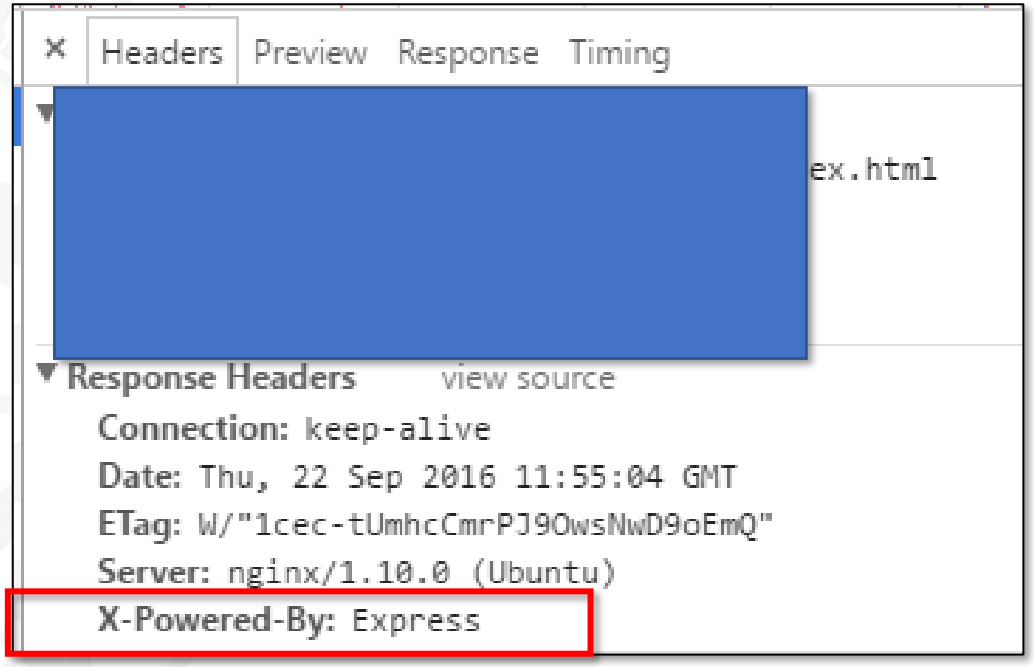
- **There are non-standard HTTP response headers that can cause security problems**
  - **Introduced by using a product / framework**
    - Reveal or declare what products are used, its version or more server details
    - With this information you can go to a vulnerability repository...
    - ... and find out the ones that the version of this product has…
  - **The ones declaring the type of server used**: Introduced by the web server itself!
    - With the same problem as the previous one
- **This example reveals the use of Liferay and its version**
  - https://www.cvedetails.com/vulnerability-list/vendor_id-2114/product_id-12592/version_id-109268/Liferay-Portal-5.2.3.html



HTTP HEADERS THAT ARE SECURITY PROBLEMS

José Manuel
Redondo López

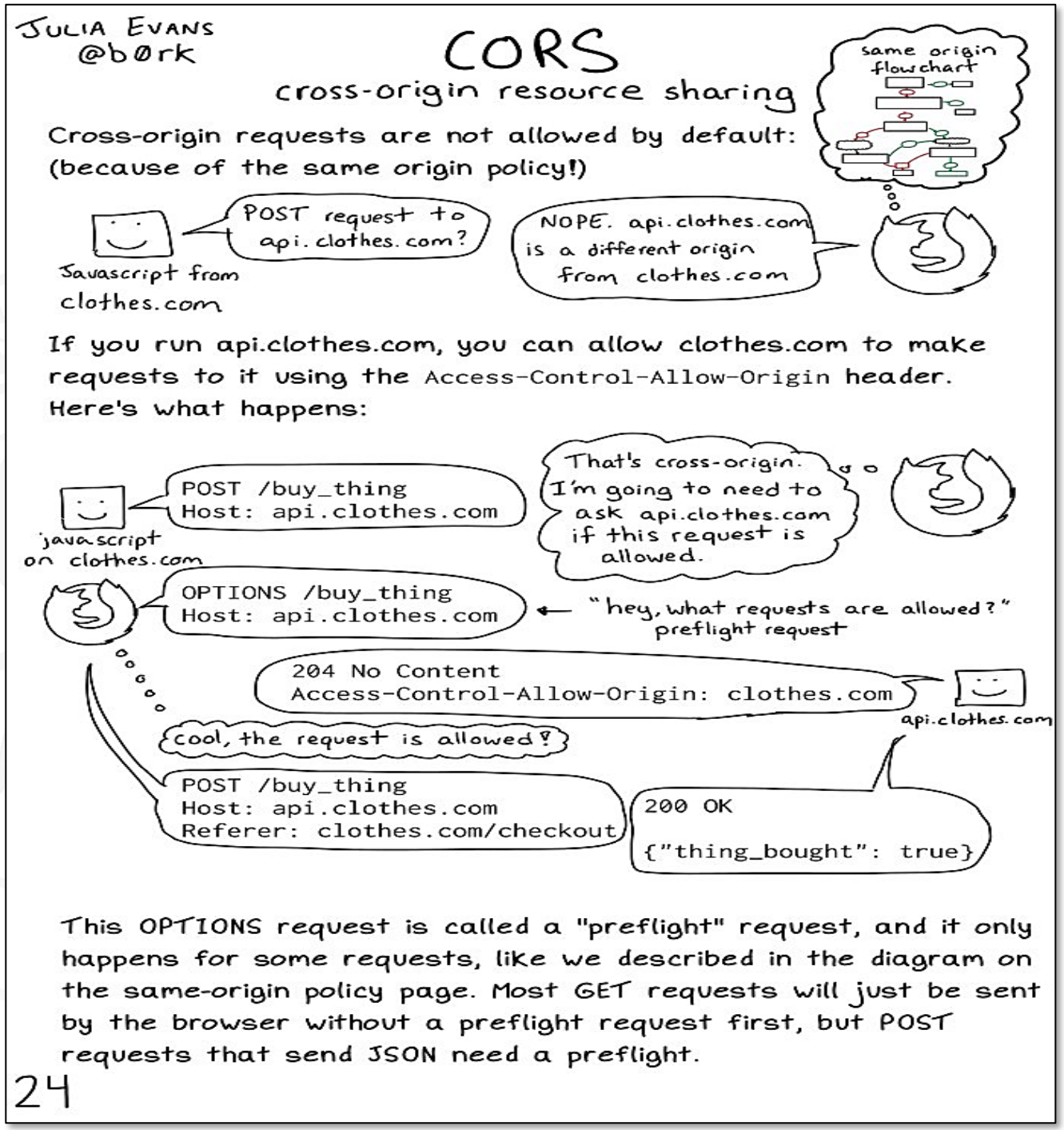
- **Here Nginx 1.10.0 and Ubuntu are revealed**
  - https://www.cvedetails.com/vulnerability-list/vendor_id-10048/product_id-17956/version_id-198730/Nginx-Nginx-1.10.0.html
  - Also, the Express framework
- **These are "advertising" headers, not really needed for serving websites**
- **Each server has its own way of deleting them**
  - Apache: https://httpd.apache.org/docs/current/mod/mod_headers.html
  - IIS (with URL Rewrite options): https://blogs.msdn.microsoft.com/varunm/2013/04/23/remove-unwanted-http-response-headers/
  - Nginx (installing the `nginx-extras` package): https://stackoverflow.com/questions/246227/how-do-you-change-the-server-header-returned-by-nginx
- **Frameworks usually have a removal option**



Cross-Origin Resource Sharing (CORS)

José Manuel Redondo López

- **Normally, a web page is free to make request to different domains than the one it resides**
  - But only for certain element types, like images, CSS, scripts, iframes, videos…
  - Other element types, such as AJAX requests or POST with certain MIME types, are forbidden by default
  - They follow the same origin policy
- **The CORS policy allow to control which domains can access other domains**
  - "Preflight" requests allow asking a server which domains they admit
- **Do not mess with the CORS policy!**



< Go to Index

Strict CSP >

Content Security Policy (CSP)

The most powerful security-related HTTP Header

What is CSP?

José Manuel
Redondo López

- **Cross-site scripting (XSS) is one of the most serious web security problems and thus, many countermeasures have been created (see theory topic 6)**
  - Contextual auto-escaping: https://queue.acm.org/detail.cfm?id=2663760
  - Automatic vulnerability scans to identify vulnerabilities by evaluating the application: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
- **Content Security Policy (CSP) is designed as an additional layer of defense against XSS, clickjacking, and other code injection attacks**
  - Specifies domains authorized to run scripts or load resources on a page
    - If someone injects a malicious script that is read from a domain other than an authorized one…
    - **The browser does not run it**: the page has the vulnerability, but not its effects (stopped by the browser)
  - **An HTTP response tells the browser from where certain resources can be loaded**
    - We need to issue the `Content-Security-Policy: policy` header in HTTP responses
    - `policy` is a string that contains a list of policies that define the CSP policy, separated by `;`
    - Each directive can have different values that describe the policy to follow for a given type of resource
    - Let's see what directives exist and their possible values

CSP: Content Security Policy

@Sec_t0

With Love By INTIGRITI

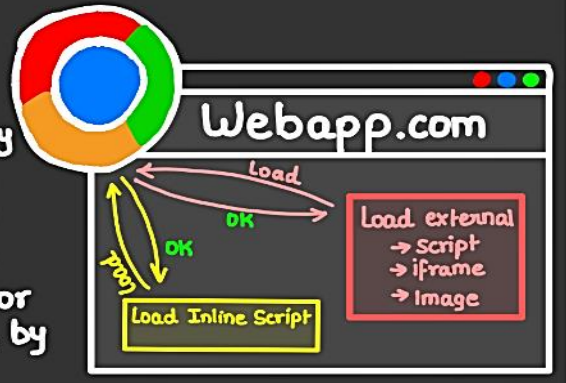
SecurityZines.com

1 Where Is The Problem ?

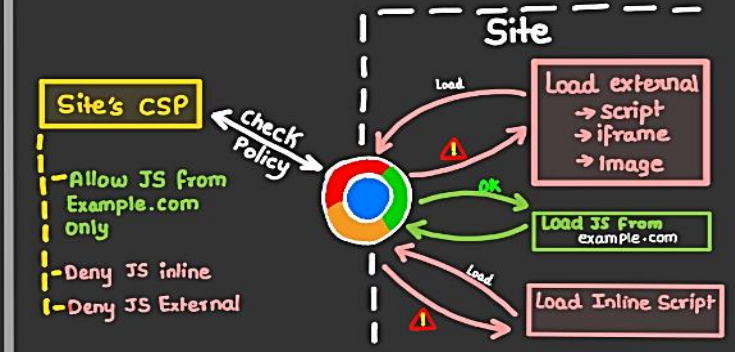
*If XSS is exploited and CSP is not enforced, then can execute malicious scripts.

2 Issue

Browser doesn't stop any resource load request from website, be it from attacker or genuinly made by site.



3 How CSP Helps ?



4 CSP Header

Response header set by Server

Content-Security-Policy : policy

Policy Contains one or more directives and Sources ';' separated

Each directive has set of allowed resource Sources

5 Directives

Tells browser what Content sources can be trusted. Eg

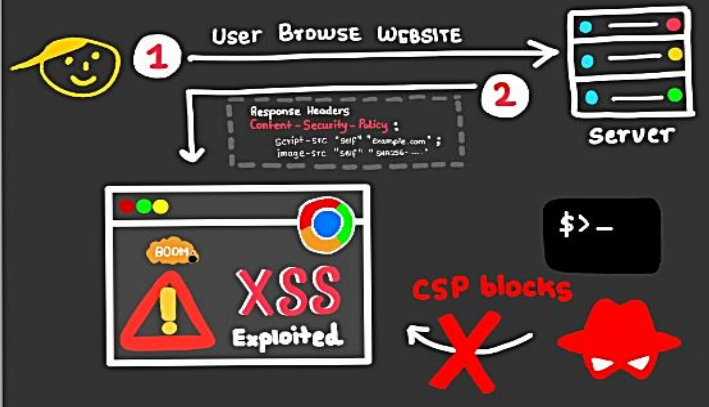
default-src : policy for all resources
Style-src : Policy for style tag
Script-src : Policy for script tag
img-src : Policy for image tag

Each directive takes a source list, Sources separated by Space

Valid Sources can be

"self" — load from current origin only
"*" — load from anywhere
"none" — Prevent loading from anywhere
"SHA256-...." — Load the source file if its hash matches the mentioned hash.
"*.example.com" Load only from subdomains of example.com

6 Complete Picture



Directives and possible values

José Manuel
Redondo López

Directive	Example Value	Description
default-src	'self' cdn.example.com	This is the default policy for loading content such as JavaScript, Images, CSS, Fonts, AJAX requests, Frames, HTML5 Media
script-src	**'self' js.example.com**	**Defines valid sources of JavaScript**
style-src	'self' css.example.com	Defines valid sources of stylesheets
img-src	'self' img.example.com	Defines valid sources of images
connect-src	'self'	Applies to XMLHttpRequest (AJAX), WebSocket or EventSource. If not allowed the browser emulates a 400 HTTP status code.
font-src	font.example.com	Defines valid sources of fonts
object-src	'self'	Defines valid sources of plugins, e.g., <object>, <embed> or <applet>
media-src	media.example.com	Defines valid sources of audio and video, e.g., HTML5 <audio>, <video> elements
frame-src	'self'	Defines valid sources for loading frames. child-src is preferred over this deprecated directive. Deprecated
sandbox	allow-forms allow-scripts	Enables a sandbox for the requested resource like the iframe sandbox attribute. The sandbox applies a same origin policy, prevents popups, plugins and script execution is blocked. You can keep the sandbox value empty to keep all restrictions in place, or add values: allow-forms allow-same-origin allow-scripts allow-popups, allow-modals, allow-orientation-lock, allow-pointer-lock, allow-presentation, allow-popups-to-escape-sandbox, and allow-top-navigation
report-uri	/some-report-uri	Instructs the browser to POST reports of policy failures to this URI. You can also append -Report-Only to the HTTP header name to instruct the browser to only send reports (does not block anything)
child-src	'self'	Defines valid sources for web workers and nested browsing contexts loaded using elements such as <frame> and <iframe>
form-action	'self'	Defines valid sources that can be used as a HTML <form> action
frame-ancestors	'none'	Defines valid sources for embedding the resource using <frame> <iframe> <object> <embed> <applet>. Setting this directive to 'none' should be roughly equivalent to X-Frame-Options: DENY
plugin-types	application/pdf	Defines valid MIME types for plugins via <object> and <embed>

Source Value	Example	Description
*	img-src *	Wildcard, allows any URL except data: blob: filesystem: schemes
'none'	object-src 'none'	Prevents loading resources from any source
'self'	script-src 'self'	Allows loading resources from the same origin (same scheme, host and port)
data:	img-src 'self' data:	Allows loading resources via the data scheme (e.g. Base64 encoded images)
domain.example.com	img-src domain.example.com	Allows loading resources from the specified domain name
*.example.com	img-src *.example.com	Allows loading resources from any subdomain under example.com
https://cdn.com	img-src https://cdn.com	Allows loading resources only over HTTPS matching the given domain
https:	img-src https:	Allows loading resources only over HTTPS on any domain
'unsafe-inline'	script-src 'unsafe-inline'	Allows use of inline source elements such as style attribute, onclick, or script tag bodies (depends on the context of the source it is applied to) and javascript: URIs
'unsafe-eval'	script-src 'unsafe-eval'	Allows unsafe dynamic code evaluation such as JavaScript eval()
'nonce-'	**script-src 'nonce-2726c7f26c'**	**Allows script or style tag to execute if the nonce attribute value matches the header value. I. e.: <script nonce="2726c7f26c">alert("hello"); </script>**
'sha256-'	**script-src 'sha256-qzn...ng='**	**Allow a specific script or style to execute if it matches the hash. Doesn't work for javascript: URIs. For example: sha256-qznLcsROx4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng= will allow alert('Hello, world.');**

How does CSP work? Examples

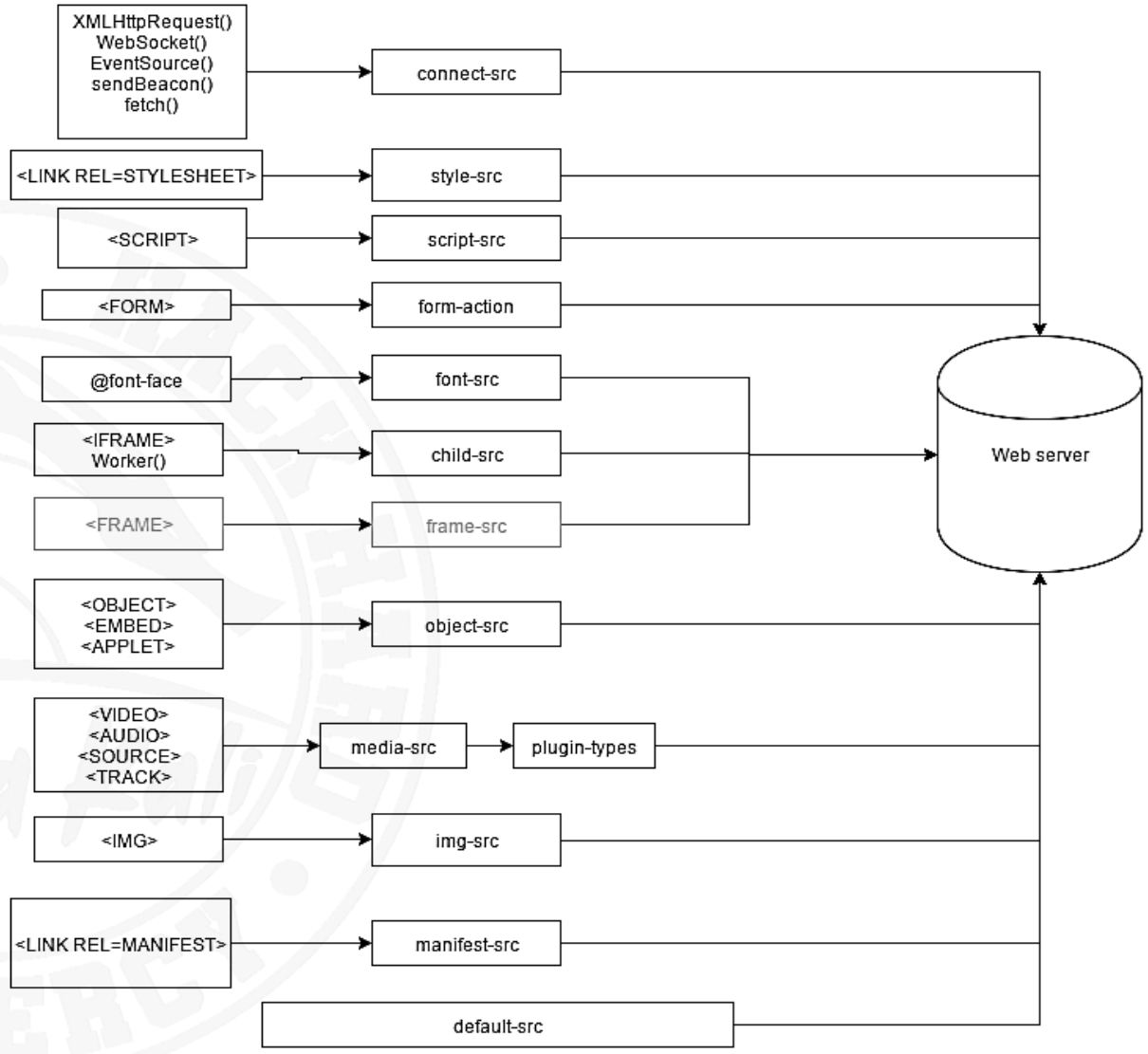
José Manuel
Redondo López

- **The following examples try to clarify it**
  - Only content from the own domain (excludes subdomains): `Content-Security-Policy: default-src 'self'`
  - The previous + the trusted domain `resources.com` and subdomains: `Content-Security-Policy: default-src 'self' *.resources.com`
  - The previous + images from any source: `Content-Security-Policy: default-src 'self' *.resources.com; img-src *`
    - It does not allow loading JavaScript or other content from sites other than the allowed ones: not specifying `script-src` or similar uses the `default-src` value for all other policies that control resources of different types
  - The previous + restricting audio and video to only a few authorized vendors (not including their subdomains), and scripts to a specific domain: `Content-Security-Policy: default-src 'self'; img-src *; media-src imgur.com youtube.com; script-src scripts.resources.com`
- **So, what do we have to do? Place each content type in the correct place and issue the headers with values appropriate to those locations**
  - And deny (`'none'`) any kind of content that we will not use as a precaution: **minimum privilege policy**!

How does CSP work?

José Manuel
Redondo López

- **HTTP headers provides the ability to control the client (browser) from an application**
- **The most important policy is `default-src`, which applies to all resources that do not have their own policy defined**
  - You can limit the execution of scripts with `script-src`, or with `default-src` if you do not define it
  - Similarly, styling from elements `<style>` or `style` attributes can be controlled with `style-src`, or `default-src` if not defined
- **The header can specify any set of policies separated by `;`**
- **The image describes the mapping between HTML 5 elements / JavaScript and CSP**



Mapping between HTML5 and JavaScript features and Content Security Policy controls
Based on CSP Level 3 draft, 2015, http://w3c.github.io/webappsec-csp/
Drawn by Pawel Krawczyk https://ipsec.pl

How does CSP work? Problem management

José Manuel
Redondo López

- **It is possible to make mistakes by specifying a CSP policy**
  - But you can use the `Content-Security-Policy-Report-Only` header to debug it
  - The policy has no effect but, if any content violates its restrictions, a problem report is sent to the URI indicated with the `report-uri` directive in the previous table
  - **Example**: `Content-Security-Policy-Report-Only: default-src 'self'; report-uri http://reports.mycompany.com/report_processing.py`
- **Once verified, change it to the standard `Content-Security-Policy` header**
  - Both have equivalent options
- **If both are present, both are processed**
  - In case of policy violation, some contents will be blocked, and others reported to the specified URI
  - You can also use `report-uri` over a `Content-Security-Policy` header if we want both active blocking and violations reports
  - This way we can identify potential security issues/attacks

How to enable CSP?

José Manuel
Redondo López

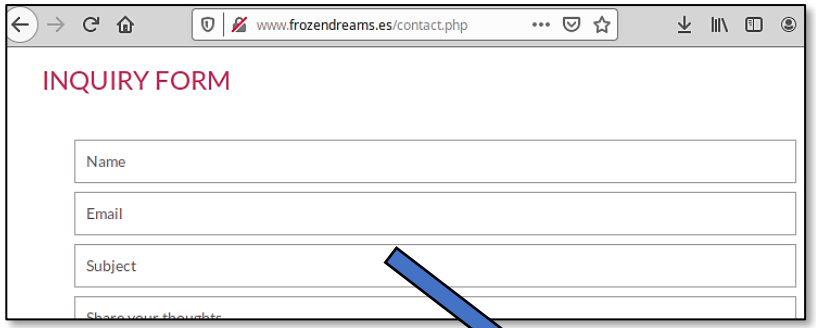
- **By configuring the web server (language / framework independent!)**
  - In Apache, add to `apache2.conf` or to any virtual host
    - `Header set Content-Security-Policy "default-src 'self';"`
  - In Nginx, add this to a `server` block
    - `add_header Content-Security-Policy "default-src 'self';";`
  - In IIS, use the "Reply Headers" module or put this in a `web.config` file

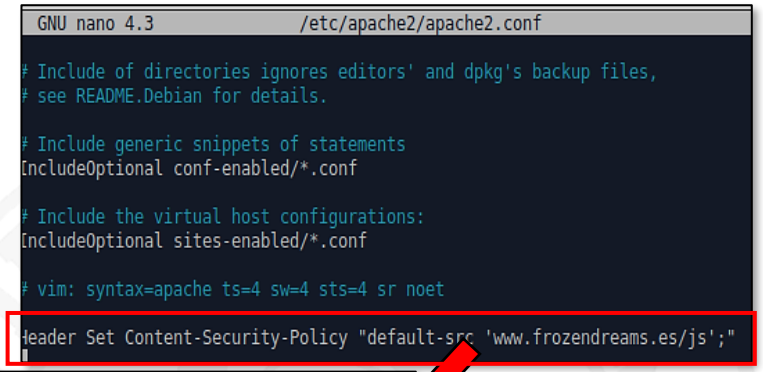
```
<system.webServer>
 <httpProtocol>
   <customHeaders>
     <add name="Content-Security-Policy" value="default-src 'self';" />
   </customHeaders>
 </httpProtocol>
</system.webServer>
```

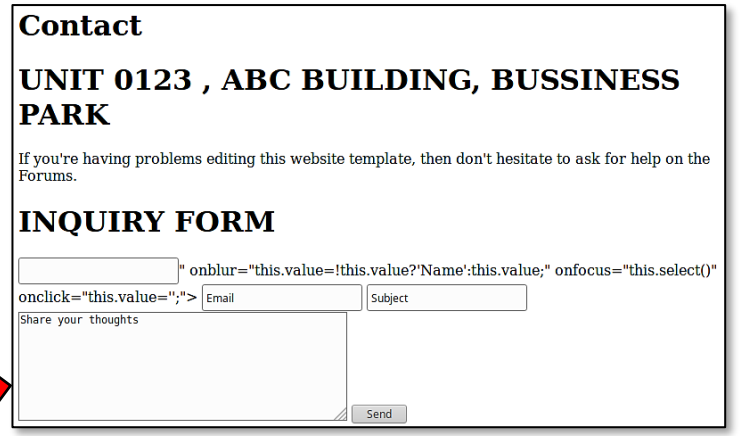
- **By manually emitting the headers in our application, depending on the framework**
  - JEE: https://www.owasp.org/index.php/Content_Security_Policy
  - PHP / Laravel: https://github.com/spatie/laravel-csp
  - Express: https://www.npmjs.com/package/express-csp-header
  - Node.js: https://stackoverflow.com/questions/21048252/nodejs-where-exactly-can-i-put-the-content-security-policy
  - **Others**: look for low-level modification of HTTP responses or CSP management modules

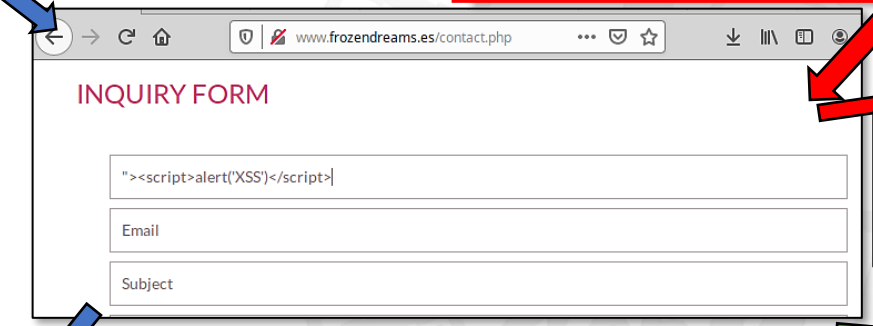
With CSP vs without CSP

José Manuel
Redondo López



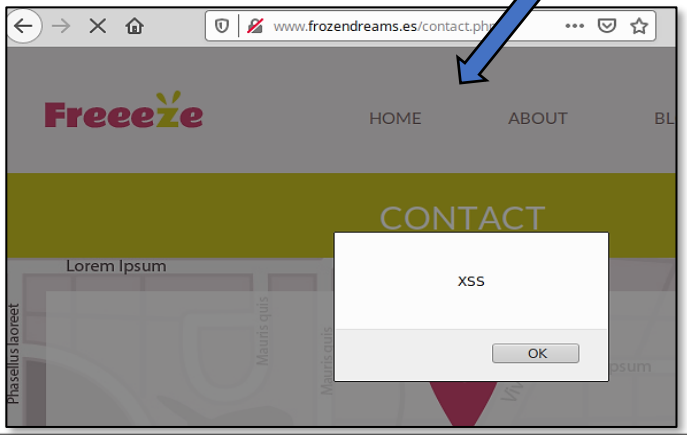


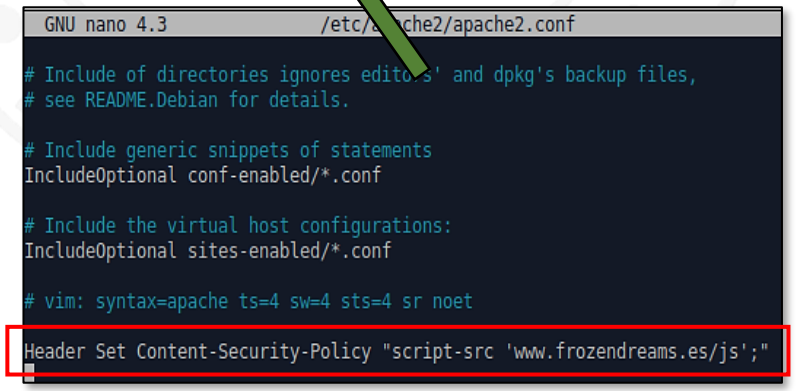


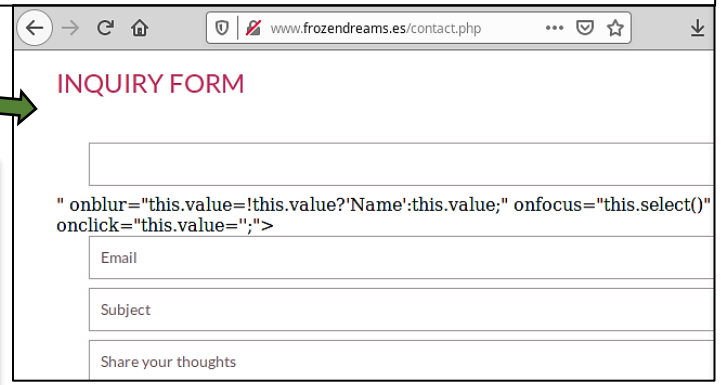


The page is vulnerable to XSS

Setting default-src successfully avoids the attack, but also disrupts the page. CSS, JS, etc. are not loaded from their sources as it applies to every page element without its own CSP directive







Setting script-src successfully avoids the attack, without interfering other elements loading sources

Strict CSP

CSP usage that prevents known bypasses

Is this a definitive method?

José Manuel
Redondo López

- **Most of CSP options authorize places that load resources (whitelists)**
  - But this can't stop all attacks, as there are known ways to bypass whitelist-based CSP restrictions
  - https://ai.google/research/pubs/pub45542
  - https://csp.withgoogle.com/docs/faq.html#problems-with-whitelists
- **All the problems are derived from assuming that an authorized domain will serve only 100% reliable content, which is not possible**
  - **JavaScript with user-controlled callbacks**: used by JSONP interfaces to load API data
    - E.g.: `<script src"/path/jsonp?callback-alert(document.domain)//"></script>`
    - If an authorized domain has a JSONP interface, it can be used to achieve an almost unrestricted XSS attack
  - **Reflection**: If a script in an authorized domain uses reflection to invoke another function in another context, it can do it bypassing the CSP restrictions
    - To be a security issue there must be a previous code injection vulnerability and make the injected malicious function the one that is run by reflection
  - **Dynamic code execution**: Angular uses `eval` to run templates with part of web pages
    - Because of this, only by uploading the library to an authorized domain (even if not used), Angular can be used to completely bypass the CSP policy

Is this a definitive method?

José Manuel
Redondo López

- **Unexpected responses that can be parsed as scripts**: Browsers are permissive with the MIME type of a response: anything parseable as JavaScript with no syntax errors can be used to bypass CSP
  - CSV data whose content is partially controlled by the user: `name,alert(1),34`
  - Error messages that write input parameters
  - Files uploaded by users to authorized domains, even if problematic characters are removed or sanitized
- **Third-party domains that are loaded from an authorized one**
- **Redirects**: Whitelists also support paths within authorized domains
  - But if that path is a redirect to another, then it will also allow you to load scripts from the redirected one

```
Content-Security-Policy: script-src redondo.com scripts.redondo.com/safe/load.js
<script src="//redondo.com?redirect=redondo.com/other/malicious.js"> //Allowed! ☹
```

  - Redirects are very common in modern web applications (OAuth...)

- **So... It doesn't work? Yes! But you must use it very carefully...**
  - We need a more powerful mechanism because these bypasses are common today and we could have one... even without knowing it ☹

So, what can be done?

José Manuel
Redondo López

- **The only secure way to apply CSP is to use nonce type values for the policy**
  - Whose value are single-use alphanumeric random tokens
  - This is Strict CSP
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src
- **This way**
  - The application calculates and injects, in the known scripts of its source code, a nonce on the response of each page request
    - **nonces cannot be predicted and we should use our framework functionalities to calculate them**
    - Existing applications must be modified to implement it
  - As we saw, only scripts that have that nonce injected will be executed by the browser

```
Content-Security-Policy: script-src 'nonce-random666’
…
<script nonce='random666'>alert(‘This runs, the nonce is correct')</script>
<script>alert(‘This does not run, no nonce is attached')</script>
<script nonce=‘iamahacker123'>alert(‘This does not run, incorrect nonce’)</script>
```

Implementing a strict CSP policy

José Manuel
Redondo López

- **Therefore, the safest way to deploy CSPs in modern and complex applications is to use a Strict CSP**
- **This forces us to change our applications as follows**
  - Add a **nonce** attribute to all `<script>` elements
    - Some template-based development technologies allow you to do this automatically
  - Refactor any code with online event handlers (`onclick`,...) and URIs with the `javascript:` scheme
  - For each load of any page, generate a new **nonce**
    - Pass it to the system that injects it into the scripts of the page code (templates, etc.)
    - Finally, pass that value in the `Content-Security-Policy` header in the HTTP response (see table)
- **This guide details and provides examples of how to change existing applications to implement it**
  - https://csp.withgoogle.com/docs/adopting-csp.html
- **We will greatly reduce the possibility of XSS attacks in our applications!**

SCRIPT-SRC DIRECTIVE

José Manuel
Redondo López

- **Given the importance of scripts and their vulnerabilities, the `script-src` directive has several values that we must study to achieve maximum benefit**
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src
- **One is `'strict-dynamic'`: if a script can be executed because it carries a correct nonce or hash, all scripts it loads will also be allowed**
  - It nullifies any whitelists, `self`, or `unsafe-inline` that may present in the policy
    - They will run if the browser does not support `strict-dynamic`: backup mechanisms for older browsers
  - Example: `script-src 'nonce-random123' 'strict-dynamic'; object-src 'none'`
- **With this we achieve more complete protection against XSS**
  - If attackers get a persistent XSS attack, they won't know the nonce to be used in each request
    - A new one is generated on demand only on those `<script>` tags statically known (those in the web code)
    - Any other `<script>` would not be executed
- **This is not suitable for attacks that inject a script into the body of an existing one**
  - We must avoid using data from user input to generate parts of JavaScript code

Real "production-ready" CSP example

José Manuel Redondo López

Let's analyze a real policy

- `object-src 'none'`: Does not run plugins
- `script-src nonce-random1234 'unsafe-inline'`: nonce-based protection for supporting browsers
  - Overrides `unsafe-inline`, unless the browser does not support **nonces**
  - `unsafe-inline` and `unsafe-eval` provide compatibility with existing websites
    - But should be removed if these features are not used
- `script-src 'strict-dynamic' https: http:` : Allows executing dynamically added scripts if they are run by an authorized script
  - If supported, `https` and `http` are ignored
  - If not, allows you to load scripts from anywhere
- `base-uri 'none'`: Disables `<base>`, the locations of scripts loaded with relative URLs can't be changed
  - `base-uri 'self'` is usually also safe
- `report-uri`: Report policy violations to an URL

```
Content-Security-Policy:
  object-src 'none';
  script-src 'nonce-random1234' 'unsafe-inline'
'unsafe-eval' 'strict-dynamic' https: http:;
  base-uri 'none';
  report-uri https://reports.redondo.com/
```

This way

- **Modern browser**: Just run scripts with correct nonces and those scripts loaded by them
- **Legacy browser**: No XSS protection is implemented, buy applications run correctly

This policy sacrifices security for compatibility with existing applications / legacy browsers (realistic case)

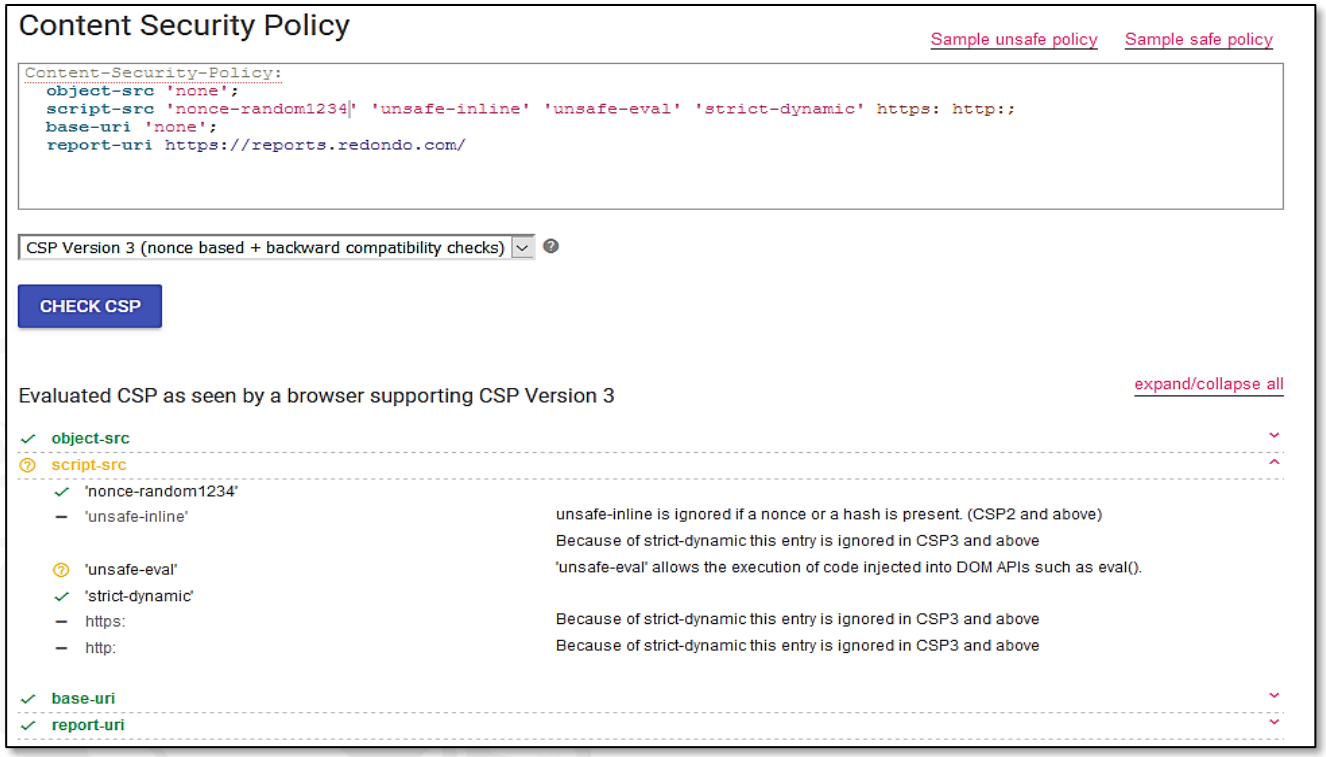
José Manuel
Redondo López

(1) Google CSP Evaluator

EVALUATING CSP POLICIES

José Manuel
Redondo López

- **To check if a policy is correct, we can use Google's CSP Evaluator**
  - https://csp-evaluator.withgoogle.com/
  - The policy "quality“ is reported
  - Notifies possible failures
  - Detects syntax errors
  - Give recommendations



< Go to Index

Examples >

Source: zona-militar.com

Other HTTP headers

Additional security layers

X-Frame-Options (Deprecated, only for compatibility)

https://developer.mozilla.org/es/docs/Web/HTTP/Headers/X-Frame-Options

José Manuel Redondo López

- **Clickjacking is an attack that tricks the user into clicking on an element on the web that triggers an invisible action on another**
  - The HTML tags `<frame>` and `<iframe>` allows you to overlay content on top of other content
    - That's why clickjacking is also known as UI Redressing
  - A user clicks on "Get a discount coupon" but it does on "Delete my account"
    - If the user is authenticated on the target website, they will delete their account if there are no more security measures (Yes/No confirmation, a second step...)
- **Has been deprecated and replaced by the new CSP header `frame-ancestors`:**
  - https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html
  - `Content-Security-Policy: frame-ancestors 'none'`: prevents any domain from framing the content
    - **Recommended setting**
  - `Content-Security-Policy: frame-ancestors 'self'`: This only allows the current site to frame the content
  - `Content-Security-Policy: frame-ancestors 'self' *.trustedsite.com https://allowed.site.com`
    - Allows the current site
    - As well as any page on `trustedsite.com` (using any protocol)
    - And only the page `allowed.site.com` (using HTTPS only on the default port, 443)

X-XSS-Protection (Deprecated, only for compatibility)

https://developer.mozilla.org/es/docs/Web/HTTP/Headers/X-XSS-Protection

José Manuel
Redondo López

- **Most browsers have embedded anti-XSS filters**
  - They are enabled by default, but users or certain settings can turn them off
- **This header allows you to force your activation**
  - This way the client's anti-XSS filter will be used regardless of its current configuration
- **This header has been deprecated and replaced by deactivating the `unsafe-inline` CSP policy**
  - https://developer.mozilla.org/es/docs/Web/HTTP/Headers/X-XSS-Protection
- **There are other measures that we can take in our browsers**
  - The set of NoScript + uBlock (or similar) plugins installed in the browser
  - Get a blacklist of known malicious hosts: https://github.com/StevenBlack/hosts
    - This blacklist can be implemented at the individual computer level (`hosts` file)
    - Or for an entire network (on a router or firewall that has the option to build blacklists)

HPKP

https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning

José Manuel
Redondo López

- **Security measure that asks the browser to associate a cryptographic public key with a web server**
- **Reduces the risk of an AitM attack using stolen or fake certificates**
  - The server's x.509 certificate public key is used for HTTPs
  - If its CA is compromised, AITM attacks on TLS connections made with its certificates can happen
  - HPKP avoids this for HTTPs, setting which key belongs to each server
- **Is a Trust on First Use (TOFU) technique**
  - On first connection to a server, the client saves that information for a time
  - On successive visits, the server is expected to have one of the known public keys for that first visit (otherwise a warning will be displayed)
- **It is more complex to use than the rest, and requires configuration to obtain and supply the public key**
  - Tutorial: https://scotthelme.co.uk/hpkp-http-public-key-pinning/

NOTE: There could be multiple valid public keys associated with multiple pin-sha256 entries

```
Public-Key-Pins:
pin-sha256= "cUPcTAZWKaASuYWhhneDttWpY3oBAkE3h2+soZS7sWs=";
pin-sha256= "M8HztCzM3elUxkcjR2S5P4hhyBNf6lHkmjAHKhpGPWE=";
max-age=5184000; includeSubDomains; report-uri=
"https://www.redondo.com/hpkp-report"
```

X-Content-Type-Options

https://developer.mozilla.org/es/docs/Web/HTTP/Headers/X-Content-Type-Options

José Manuel
Redondo López

- **This header avoids problems caused by an action that some browsers implement: MIME Type sniffing**
  - If the MIME type of a resource declared on the server is unknown or ambiguous, some browsers read it to verify or obtain it
  - Those browsers have a set of rules to identify the MIME type that a resource has
    - And treat it as such…even if its declared MIME type is different of the one identified!
- **The problem is that this can enable certain attacks**
  - For example, GIFARS and other variants: https://en.wikipedia.org/wiki/Gifar
  - Increases the danger of drive-by downloads (unintentional downloads)
- **It needs to be configured as follows:** `x-content-type: nosniff`
  - **Nginx**: `add_header x-content-type-options "nosniff" always;`
  - **Apache**: `header always set x-content-type-options "nosniff"`

HSTS (STRICT-TRANSPORT-SECURITY)

https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Strict-Transport-Security

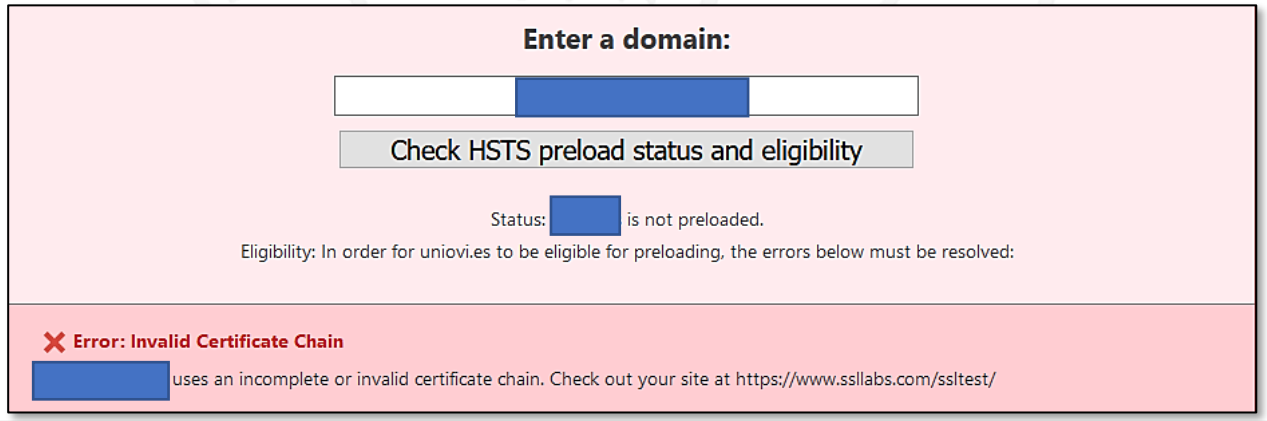
José Manuel
Redondo López

- **Security measure that forces HTTPs access to the webs**
  - Avoids SSL Downgrade attacks or configuration errors where HTTPs pages are mixed with HTTP ones
  - The server must be prepared to allow HTTPs traffic
  - **Example**: `strict-transport-security: max-age=31536000; includeSubDomains; preload`
- **It's another Trust on First Use mechanism**
  - The first time a site is accessed using HTTPs and returns a `Strict-Transport-Security` header, the browser records this information
  - Future attempts to load the site using HTTP will use HTTPS automatically instead
    - If we access the site through a fake Wifi point that tries to change to HTTP, it will not work
  - When the expiration time specified by the `Strict-Transport-Security` header has passed, the next attempt to load the site over HTTP will be processed as normal
    - Each time a browser receives `Strict-Transport-Security` from a site, it can update its expiration time
    - This prevents them from expiring and being unprotected
  - To disable HSTS and allow HTTP accesses you must set `max-age` to `0`
- **To learn more: https://www.keycdn.com/support/http-strict-transport-security**

HSTS PRELOAD

José Manuel
Redondo López

- **As we've seen, the first request is key for HSTS to prevent AiTM attacks**
  - If someone makes a first HTTP request, the attacker can intercept it and redirect to another site
  - The attacker can thus send the header with the `max-age` parameter to `0` and disable HSTS
- **To avoid this, STS preloaded lists are used**
  - They are lists containing well-known websites that support HSTS
  - Used by browsers and servers so HTTP requests to them are automatically changed to HTTPS
  - The problem is that this solution is not scalable
  - Google maintains a site where you can upload and verify domains in preloaded lists for Chrome
    - https://hstspreload.org/
    - It tells us what conditions we must meet if we are to include our own



EXPECT-CT

https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Expect-CT

José Manuel
Redondo López

- **Certificate transparency is a certificate auditing and monitoring system**
  - https://www.certificate-transparency.org/what-is-ct
  - Allows the owner of a domain/CA to check if there are erroneous or maliciously issued certificates
- **So, you can prevent using these certificates server connections**
  - An indication of potential attacks
  - Allows websites to inform and enforce Certificate Transparency requirements
  - And force the browser to verify if the certificate appears in the CT public logs
- **Example**
  - `expect-ct: max-age-604800, enforce, enforce report-uri-https://www.redondo.com/report`
  - Forces the certificate transparency policy for 7 days and reports policy violations
- **To enable it**
  - **Nginx**: `add_header expect-ct "max-age=604800, enforce, report-uri='https://www.redondo.com/report' always;`
  - **Apache**: `header always set expect-ct "max-age=604800, enforce, report-uri=https://www.redondo.com/report`

Referrer-Policy

https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Referrer-Policy

José Manuel
Redondo López

- **The HTTP `Referer` header has the address of the previous page that contains the link that loaded the one being displayed**
  - Lets servers know where they are visited from
  - This data can be used to perform analysis, logs, optimizations, advertising...
- **But we have the right to decide if our servers give that information**
  - The `Referrer-Policy` header determines which data should be included in the HTTP `Referer` header of each request
  - A full `Referer` consists of
    - **Host** (`www.ssi.com`)
    - **Path** (`/clients/profile.html`)
    - **Querystring** (`id=jose`)
- **Example:** `Referrer-Policy: no-referrer`

Value	Sends...
no-referrer	No data is sent
no-referrer-when-downgrade (default)	A full Referer (host + path + querystring) when the HTTP security level does not change (HTTPS → HTTPS). Nothing will be sent to less secure targets (HTTPS → HTTP).
origin	Only the host as Referer on every case. The document https://redondo.com/index.html sends the Referer https://redondo.com/
origin-when-cross-origin	A full Referer (host + path + querystring) if the request is made to the same origin. Just the host in any other case
same-origin	A full Referer (host + path + querystring) if the request is made to the same origin. Nothing in any other case
strict-origin	Only the host when the HTTP security level does not change (HTTPS → HTTPS). Nothing will be sent to less secure targets (HTTPS → HTTP).
strict-origin-when-cross-origin	A full Referer (host + path + querystring) if the request is made to the same origin, only the host when the HTTP security level does not change (HTTPS → HTTPS) and nothing will be sent to less secure targets (HTTPS → HTTP).
unsafe-url	Always a full Referer (host + path + querystring) no matter origin or security level. This directive reveal origins, resource access routers and other potential sensitive information of TLS protected resources to unsecure targets, which may be a security problem

Permissions-Policy (previously named Feature-Policy)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Feature-Policy

José Manuel Redondo López

Experimental header, its specification is not yet stable

- https://www.w3.org/TR/permissions-policy-1/

Allows to enable or disable browser features

- Allows it at the **web level as well as frame-level** (`<iframe>`)
- These are its possible values, which indicate that the associated characteristic...
  - `*`: Allowed on the current web and on all its nested `iframe` regardless of their origin
  - `'self'`: Allowed on the current web and on all its nested `iframe` that have the same origin
  - `'src'`: (for `iframe` only) Allowed only on the current `iframe` if the page that is loaded into it has the same origin as the URL that appears as the value of the iframe's `src` attribute
  - `'none'`: Not allowed on the current website or on any of its nested elements
  - `<URL(s)>`: Allowed only to specific sources, separated by a space (`https://ssi.com https://backup.com`)

Example: `Permissions-Policy: autoplay 'none'; camera 'none'`

- **Nginx**: `add_header Permissions-Policy "autoplay 'none'; camera 'none'" always;`
- **Apache**: `header always set Permissions-Policy "autoplay 'none'; camera 'none'"`

Allows you to manipulate HUGE number of browser features

- Allows you to modify the default browser behavior when using them
- And with it, prevent unauthorized use of them by means of attacks / bad settings

Directives that control if the current document can...

José Manuel Redondo López

Directive	Purpose
ambient-light-sensor	Gather information about the amount of light in the environment around the device through the AmbientLightSensor interface
autoplay	Autoplay media requested through the HTMLMediaElement interface. The autoplay attribute on <audio> and <video> elements will be ignored
accelerometer	Gather information about the acceleration of the device through the Accelerometer interface
Battery	Control if the use of the Battery Status API is allowed
camera	Use video input devices. When enabled, the Promise returned by getUserMedia() will reject with a NotAllowedError
display-capture	Use the getDisplayMedia() method to capture screen contents. When this policy is enabled, the promise returned by getDisplayMedia() will reject with a NotAllowedError
document-domain	Set document.domain
encrypted-media	Use the Encrypted Media Extensions API (EME). When enabled, the Promise returned by Navigator.requestMediaKeySystemAccess() will throw a DOMException
execution-while-not-rendered	Controls whether tasks should execute in frames while they're not being rendered (e.g. if an iframe is hidden or display: none)
execution-while-out-of-viewport	Controls whether tasks should execute in frames while they're outside of the visible viewport
fullscreen	Use Element.requestFullScreen()

Directive	Purpose
Geolocation	Use the Geolocation Interface, the getCurrentPosition() and watchPosition() functions
Gyroscope	Gather information about the orientation of the device through the Gyroscope interface
Magnetometer	Gather information about the orientation of the device through the Magnetometer interface
Microphone	Use audio input devices. When enabled, the Promise returned by MediaDevices.getUserMedia() will reject with a NotAllowedError
Midi	Use the Web MIDI API. When enabled, the Promise returned by Navigator.requestMIDIAccess() will reject with a DOMException
Payment	Use the Payment Request API. When enabled, the PaymentRequest() constructor will throw a SecurityError DOMException
picture-in-picture	Play a video in a Picture-in-Picture mode via the corresponding API
Speaker	Play audio via any methods
sync-xhr	Make synchronous XMLHttpRequest requests
Usb	Use the WebUSB API
wake-lock	Use Wake Lock API to avoid devices entering power-saving mode
Webauthn	Use Web Authentication API to create, store, and retreive public-key credentials
Vr	Use the WebVR API. When enabled, the Promise returned by Navigator.getVRDisplays() will reject with a DOMException
xr-spatial-tracking	Use the WebXR Device API to interact with a WebXR session

HTTP header usage examples

Checking HTTP headers “in the wild”

José Manuel
Redondo López

(1) Viewing Headers of real Websites

(2) Analyzing Headers of real Websites

Cabeceras | Cookies | Parámetros | Respuesta | Tiempos | Traza de la pila | Segu

Filtrar cabeceras

Cabeceras de la respuesta (1,552 KB) — Cabeceras sin procesar

content-encoding: gzip
content-security-policy: default-src 'self' blob: wss: ... 'unsafe-inline' data: https:;
content-type: text/html; charset=utf-8
date: Wed, 23 Oct 2019 15:46:43 GMT
etag: W/"530fa-FZ8yCnQs5KEDtQwvkIhU1U31EAQ"
rlogid: t6klaook%60b0%3D%3C%3Dosuojbnk...%3B(5220344-16df94c5c1e-0x803
server: envoy
set-cookie: nonsession=CgADKACBnFndjZjk0Yz...15:46:43 GMT;Path=/; HttpOnly
set-cookie: s=CgAD4ACBdscdjZjk0YzgyODYxNmQ...=.ebay.co.uk;Path=/; HttpOnly
set-cookie: dp1=bbl/GBen-GB6172dce3^;Domai...15:46:43 GMT;Path=/; HttpOnly
set-cookie: ak_bmsc=9C9542C2EE9179FF8DA51A... domain=.ebay.co.uk; HttpOnly
strict-transport-security: max-age=31536000
vary: Accept-Encoding
x-content-type-options: nosniff
x-ebay-pop-id: UFES2-LHR-caching
x-ebay-pop-id: UFES2-LHR-frontcache
x-ebay-pop-id: UFES2-LHR-dweb-2
x-edgeconnect-midmile-rtt: 23
x-edgeconnect-origin-mex-latency: 12
x-envoy-upstream-service-time: 8
X-Firefox-Spdy: h2
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block

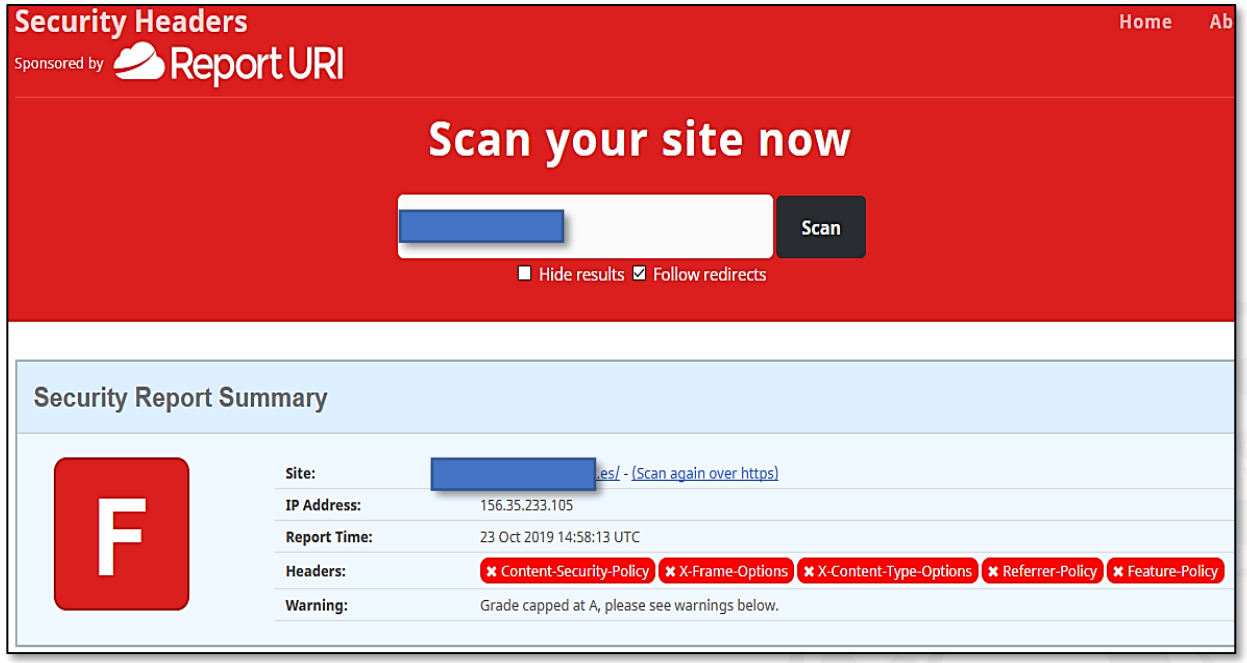
Cabeceras | Cookies | Parámetros | Respuesta | Caché | Tiempos | Seguridad

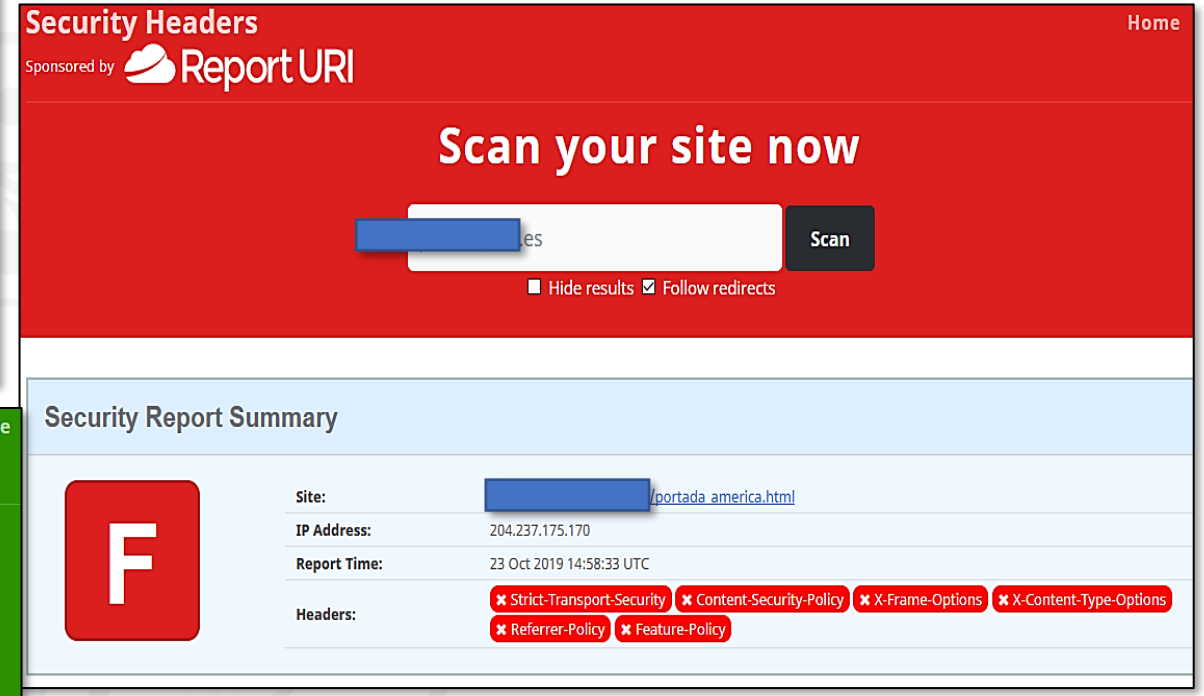
URL solicitada: https://www.lamoncloa.gob.es/Theme/RLaMoncloa/css/normalize.css
Método de la petición: GET
Dirección remota: 212.128.109.1:443
Código de estado: 304 Not Modified
Versión: HTTP/1.1
Política de referencia: no-referrer-when-downgrade — Editar y volver a enviar

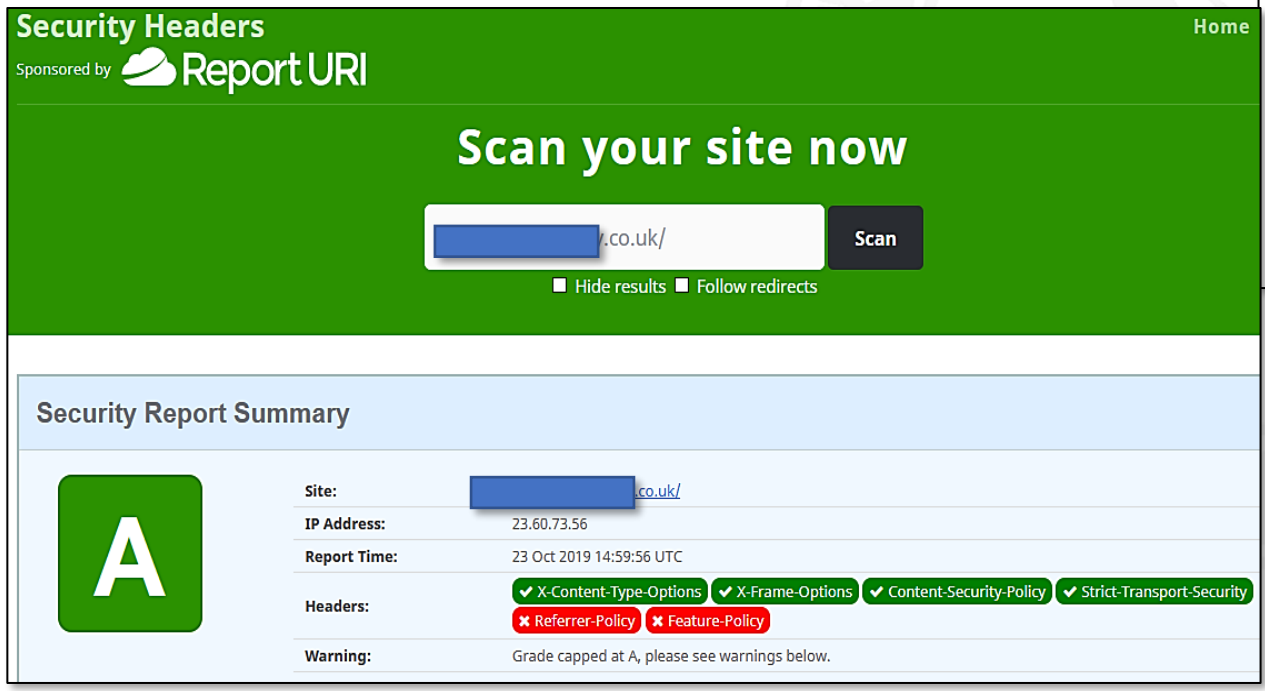
Filtrar cabeceras

Cabeceras de la respuesta (486 B) — Cabeceras sin procesar

Accept-Ranges: bytes
Cache-Control: public, max-age=300
Content-Length: 0
Date: Wed, 23 Oct 2019 15:47:59 GMT
ETag: "{4F54593D-F4C5-41E3-AD57-515C1D12BE14},1pub"
request-id: a8b7109f-8235-b025-e01d-ce3be28757e3
Server: Microsoft-IIS/8.0
SPIisLatency: 0
SPRequestDuration: 1
SPRequestGuid: a8b7109f-8235-b025-e01d-ce3be28757e3
X-Content-Type-Options: nosniff
X-FRAME-OPTIONS: SAMEORIGIN
X-MS-InvokeApp: 1; RequireReadOnly
X-Powered-By: ASP.NET


Security Headers
Sponsored by Report URI
Home
Ab
Scan your site now
Scan
Hide results
Follow redirects
Security Report Summary
F
Site:
es/ - (Scan again over https)
IP Address:
156.35.233.105
Report Time:
23 Oct 2019 14:58:13 UTC
Headers:
Content-Security-Policy
X-Frame-Options
X-Content-Type-Options
Referrer-Policy
Feature-Policy
Warning:
Grade capped at A, please see warnings below.


Security Headers
Sponsored by Report URI
Home
Scan your site now
es
Scan
Hide results
Follow redirects
Security Report Summary
F
Site:
/portada_america.html
IP Address:
204.237.175.170
Report Time:
23 Oct 2019 14:58:33 UTC
Headers:
Strict-Transport-Security
Content-Security-Policy
X-Frame-Options
X-Content-Type-Options
Referrer-Policy
Feature-Policy


Security Headers
Sponsored by Report URI
Home
Scan your site now
.co.uk/
Scan
Hide results
Follow redirects
Security Report Summary
A
Site:
.co.uk/
IP Address:
23.60.73.56
Report Time:
23 Oct 2019 14:59:56 UTC
Headers:
X-Content-Type-Options
X-Frame-Options
Content-Security-Policy
Strict-Transport-Security
Referrer-Policy
Feature-Policy
Warning:
Grade capped at A, please see warnings below.

New advances in this field

José Manuel
Redondo López

- The contents we have just seen belong to a research line on “native defenses on the web ecosystem”
- It is a current trending topic and in constant advance
- Researchers incorporate new solutions frequently
- To review the last additions in this field, you can check
  - https://security.googleblog.com/2020/07/towards-native-security-defenses-for.html

Self-evaluation achievement list: Seminar 6

José Manuel
Redondo López

Rank	Concept
	Understand why extra headers with information in the HTTP protocol are dangerous
	Know about the proper values of simple HTTP headers
	Understand the importance of not giving too much information in the **Referer** header
	Implement an appropriate CSP policy without nonces
	Understand why CSP without nonces is not enough in most cases
	Know how to validate a CSP policy
	Know how to use the **feature-policy** header
	Understand the utility of other HTTP headers related to security
	Implement an appropriate CSP policy with nonces
	**Heading to victory**: Be able to enforce HTTP security by appropriately using its headers

SEMINAR 6
HTTP SECURITY AND CSP

Source: zona-militar.com