



Privacy Audit







for arc42.de

Completed on: August 7, 2021

Expires on: September 7, 2021

Scanned from: United States

Find solutions for privacy website compliance at www.osano.com

	HTTPS by default	Yes
	Content Security Policy	Not implemented
	Referrer Policy	Referrers leaked
0	Cookies	0
6	Third-party requests	6 requests to 2 unique hosts
	Server location	 United States of America
	Server IP address	75.2.60.5

✓ HTTPS by default

www.arc42.de uses HTTPS by default.

Osano's automated web browser reports the following:

State	Title	Summary	Description
✓	Certificate	valid and trusted	The connection to this site is using a valid, trusted server certificate issued by R3.
✓	Connection	secure connection settings	The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_256_GCM.
✓	Resources	all served securely	All resources on this page are served securely.

More information about the site's TLS/SSL configuration:

[Analyze www.arc42.de on SSL Labs](#)

[Observatory by Mozilla](#)

[Mozilla TLS Observatory](#)

[testssl.sh](#)

To enable HTTPS on a website, a **certificate** for the domain needs to be installed on the web server. To get a certificate that browsers will trust, you need one issued by a trusted certificate authority (otherwise a visitor's browser will show a warning).

[Let's Encrypt](#) is a non-profit certificate authority (sponsored by Mozilla, EFF, Cisco, Facebook and others) providing free certificates through an easy, automated process. You can set it up yourself, or use one of the many hosting providers who have built-in support for Let's Encrypt.

[Get started with Let's Encrypt](#)

[Mozilla SSL/TLS Configuration Generator](#) [for advanced users]

For checking the configuration of a server, try [SSL Labs SSL Server Test](#) (web), [testssl.sh](#) (CLI tool), [Mozilla TLS Observatory](#) (CLI tool) or [Observatory by Mozilla](#) (web).

✓ HTTP Strict Transport Security (HSTS)

HSTS policy for <https://www.arc42.de>:

```
max-age=31536000
```

Pass	Test
✓	<code>max-age</code> set to at least 6 months
✗	<code>includeSubDomains</code> – policy also applies to subdomains
–	<code>preload</code> – requests inclusion in preload lists (only relevant for base domain)

Base domain (<https://arc42.de>) HSTS status unknown.

HSTS is just an HTTP header. In its simplest form, the policy tells a browser to enable HSTS for that exact domain or subdomain, and to remember it for a given number of seconds (the policy is refreshed every time browser sees the header again):

```
Strict-Transport-Security: max-age=31536000;
```

In its **strongest and recommended form**, the HSTS policy includes **all subdomains**, and indicates a willingness to be "preloaded" into browsers:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

Note that `includeSubDomains` should be deployed at the base domain, i.e., <https://example.com>, *not* <https://www.example.com>. While we recommend the use of `includeSubDomains`, be **very careful**, as it means that **all subdomains** associated with the parent domain **must** support HTTPS. (They do not have to each have their own HSTS policy.)

For a user to take advantage of HSTS, their browser does have to see the HSTS header at least once. This means that users are not protected until after their first successful secure connection to a given domain.

To solve this problem, the Chrome security team created an "HSTS preload list": a list of domains baked into Chrome that get Strict Transport Security enabled automatically, even for the first visit.

Firefox, Safari, Opera, and Edge also incorporate Chrome's HSTS preload list, making this feature shared across major browsers.

The Chrome security team allows anyone to [submit their domain to the list](#), provided it meets a few requirements.

[HTTP Strict Transport Security](#) [cio.gov]

[HSTS Preload List Submission](#) [hstspreload.org]

[Strict-Transport-Security](#) [mozilla.org]

Text adapted from the CIO Council's [The HTTPS-Only Standard](#) (public domain).

✗ Content Security Policy

CSP header not implemented.

The recommended way to enable Content Security Policy is with the `Content-Security-Policy` HTTP header, e.g.:

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

It can also be enabled with an HTML `<meta>` element:

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self'">
```

CSP is a powerful mechanism that we strongly recommend. It allows for very fine-grained control. However, creating a good policy (or adjusting your site to work with a good policy) can take some time and effort. To make this easier, it's possible to use CSP in report-only mode.

See the following pages for more information:

[Content Security Policy \(CSP\)](#) [developer.mozilla.org]

[Google Web Fundamentals: Content Security Policy](#) [developers.google.com]

[CSP Cheat Sheet](#) [scotthelme.co.uk]

[Report URI: Tools](#) (CSP analyser, CSP builder) [report-uri.com]

[CSP Evaluator](#) [csp-evaluator.withgoogle.com]

[CSP Level 2 specification](#) [w3.org]

[CSP Level 3 specification](#) [w3.org]

[Browser support](#) [caniuse.com]

× Referrer Policy

Referrer Policy not set. This means that the default value `no-referrer-when-downgrade`, leaking referrers in many situations, is used.

A referrer policy can easily be set with a `<meta>` element in your HTML. Simply include this inside the `<head>` section:

```
<meta name="referrer" content="no-referrer">
```

Alternatively, set the `Referrer-Policy` HTTP header, e.g.:

```
Referrer-Policy: no-referrer
```

If a referrer policy is delivered via both Referrer-Policy header and meta element, the meta element's policy takes precedence.

If multiple policy values are specified, the browser will use the last one, ignoring unknown values (fallback values should thus appear first). Multiple values should be separated by commas, e.g.:

```
<meta name="referrer" content="no-referrer, same-origin">
```

Several policies are offered, such as `origin` (strips everything except the origin) and `origin-when-cross-origin` (sends full URL with same-origin requests, otherwise stripped). We recommend `no-referrer`, which kills the referrer header entirely for all requests, no matter the destination; or `same-origin`, which kills the referrer for third-party requests but not for requests to the same origin.

[Referrer-Policy](#) [developer.mozilla.org]

[Referer header: privacy and security concerns](#) [developer.mozilla.org]

[Referrer Policy specification](#) [w3.org]

[Browser support](#) [caniuse.com]

✗ Subresource Integrity (SRI)

Subresource Integrity (SRI) not implemented, but all external resources are loaded over HTTPS

The following third-party resources are not loaded using SRI:

Type	URL
script	https://unpkg.com/@botpoison/browser
css	https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@5/css/all.min.css
css	https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@5/css/all.min.css

SRI can be used with `script` and `link` elements. To enable SRI on an element, you need to add `integrity` and `crossorigin` attributes to it.

`integrity` should contain integrity metadata: a string describing the [cryptographic hash function used](#) (currently sha256, sha384, or sha512), followed by a dash, followed by the [base64](#)-encoded hash of the file.

`crossorigin` must be set to `anonymous` for third-party resources when using SRI. This has to do with [CORS](#).

For example, given the file <https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js>, we can calculate the SHA384 hash:

```
$ openssl dgst -sha384 -binary jquery.min.js | openssl base64 -A  
tsQFqpEReu7ZLhBV2VZ1Au7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT
```

The correct HTML code should then be:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"  
integrity="sha384-  
tsQFqpEReu7ZLhBV2VZ1Au7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT"  
crossorigin="anonymous"></script>
```

When the browser sees this element, it will download `jquery.min.js`, calculate the SHA384 hash, compare it to the hash in the `integrity` attribute, and only run the script if the hashes match. For example, if someone were to modify `jquery.min.js` on the remote server after we calculated the original hash, Firefox would refuse to run the script and you'd see this in the browser console:

```
None of the "sha384" hashes in the integrity attribute match the content of the subresource.
```

To make all this easier, you can use [Mozilla's SRI Hash Generator](#).

[Subresource Integrity](#) [developer.mozilla.org]

[Protecting your embedded content with subresource integrity \(SRI\)](#) [troyhunt.com]

[Subresource Integrity specification](#) [w3.org]

[Browser support](#) [caniuse.com]

HTTP headers

Pass	Header	Value	Result
✗	X-Content-Type-Options		X-Content-Type-Options header not implemented
✗	X-Frame-Options		X-Frame-Options (XFO) header not implemented
✗	X-XSS-Protection		X-XSS-Protection header not implemented

To enable these headers you'll need to add them to your web server configuration. This is a simple change. Exactly how you do it depends on what server you use. [This page](#) [developer.mozilla.org] has configuration examples for Apache, Nginx and IIS.

`X-Content-Type-Options` should be set to `nosniff`, which is the only valid value.

`X-Frame-Options` can be set to `deny` (page can never be loaded in a frame), `sameorigin` (page can only be loaded in a frame only if the origin is the same), or `allow-from <URI>` (page can only be loaded in a frame on a page on the specified origin).

`X-XSS-Protection` should be set to `1` or `1; mode=block`.

Cookies

No cookies detected.

 GDPR: [Rec. 60](#), [Rec. 61](#), [Rec. 69](#), [Rec. 70](#), [Rec. 75](#), [Rec. 78](#), [Art. 5.1.a](#), [Art. 5.1.c](#), [Art. 5.1.e](#), [Art. 21](#), [Art. 22](#), [Art. 32](#).

[e-PD \(2002/58/EC\)](#). [Rec. 24](#), [25](#), [Art. 5.2](#).

[e-PD revised \(2009/136/EC\)](#). [Rec. 65](#), [66](#).

First-party cookies are placed by the web site owner in some register on their visitors' device in order to be able to re-identify the visitor on subsequent page loads. First-party cookies can be related to technical features on a web site (such as remembering language settings or the contents of a shopping basket), or related to commercial features of the web site owners' activities (such as being able to trace a visitors' behaviour over the duration of their visit, or over much longer time periods, often for years, in order to be able to serve advertisements to the users or to get usage statistics to guide later changes to the web site that are envisaged to make the web site more attractive to recurring users). First-party cookies may come from services provided by the web site owner (language settings in a Content Management System) or from services used by the web site owner (analytics tools).

Third-party cookies are placed by a service affiliated with the web site owner on the devices of visitors to the web site in order to be able to re-identify the visitor on subsequent page loads, or across different web sites. Third-party cookies are typically related to commercial features of a web site owners' activities, usually advertising, but may also relate to technical features in scripts used by a web site (such as language settings).

Storing information or gaining access to information stored in the visitors' devices, for instance in the form of cookies, has been subject to sui generis legislation in the European Union ([ePD](#), [Art. 5.3](#)). These sui generis laws have tried to make a distinction between information stored to support technical features and information stored to support commercial features. In practice, poor enforcement of these rules has made the legal landscape unclear. Because there exists no legal duty for citizens to receive better targeted advertisement, nor a legal duty for citizens to assist web developers in improving web sites, it's doubtful that a legal basis exists for storing information to support commercial features without the consent of the web visitor (GDPR [Art. 7](#)). It is argued that the legitimate interests of a web site owner ([Art. 6.1.f](#), [Art. 6.4](#)) may nevertheless enable them to subject a visitor to targeted ads or cause a visitor to assist the web developers. Then there must exist relevant and appropriate relationship between the web visitor and the web site owner in situations (GDPR [Rec. 47](#)), which calls into question the use of third-party service first-party cookies. In either case, if the legitimate interest legal basis for processing is invoked, adequate security measures must be undertaken (GDPR [Art. 32](#)).

Particular care must be taken with regards to the period of storage (GDPR [Art. 5.1.e](#)). While it is technically easy for a web site owner to set the duration of a information stored in the form of cookies to a long period time, the principle of storage limitation implies a balancing act between the interest of tracking a visitors' behaviour and the interest of the visitor to keep their behaviour private. It's been established that a reasonable storage period does not exceed [one year](#).

localStorage

localStorage not used.

Third-party requests

6 requests (6 secure, 0 insecure) to 2 unique hosts.

A third-party request is a request to a domain that's not `arc42.de` or one of its subdomains.

Host	IP	Classification
cdn.jsdelivr.net (3)  https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@5/webfonts/... https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@5/webfonts/... https://cdn.jsdelivr.net/npm/@fortawesome/fontawesome-free@5/css/all.m...	151.101.113.229	-
unpkg.com (3)  https://unpkg.com/@botpoison/browser@0.1.27/dist/index.js https://unpkg.com/@botpoison/browser@0.1.27 https://unpkg.com/@botpoison/browser	104.16.126.175	-

 GDPR: [Rec. 69](#), [Rec. 70](#), [Art. 5.1.b-c](#), [Art. 25](#).

Server location

The server www.arc42.de (75.2.60.5) appears to have been located in **United States of America** during our test.

Raw headers

Header	Value
age	1
cache-control	public, max-age=0, must-revalidate
content-encoding	br
content-type	text/html; charset=UTF-8
date	Sat, 07 Aug 2021 11:11:57 GMT
etag	"b7a1f9b52f5454b3b5fd5bfee5001d10-ssl-df"
server	Netlify
status	200
strict-transport-security	max-age=31536000
vary	Accept-Encoding
x-nf-request-id	01FCG4ZY088BXYVWXV77GZE4PP