Article Memo and tutorial on HTTP headers related to web application security

baykal · 05.08.2021

Trace

baykal
RAM User

05.08.2021 #1

Hello, friends!
In this article, I want to share with you the results of a small study on HTTP headers that are related to the security of web applications (hereinafter referred to as just headers).

First, we will briefly analyze the main types of vulnerabilities of web applications, as well as the main types of attacks based on these vulnerabilities.
Next, we'll look at all the modern headlines, each one individually. This is in the theoretical part of the article.
In the practical part, we will implement a simple Express application, deploy it to Heroku and evaluate security using WebPageTest and Security Headers.
Also, given the great popularity of services for generating static sites, we will configure and deploy an application with similar functionality on Netlify.
source code of the applications can be found here.
A demo of heroku apps can be viewed here,and Netlify apps can be viewed here.

The main sources of truth in the preparation of this article for me were the following resources:

- Security headers quick reference — Google Developers
- OWASP Secure Headers Project
- Web security — MDN

Security headers

All headings can be divided into three groups. Headers *for sites that handle sensitive user data*

- Content Security Policy (CSP);
- Trusted Types.

Headers for all sites

- X-Content-Type-Options;
- X-Frame-Options;
- Cross-Origin Resource Policy (CORP);
- Cross-Origin Opener Policy (COOP);
- HTTP Strict Transport Security (HSTS).

Headers for sites with advanced features

Advanced features in this case refers to the ability to use site resources by other sources (origins) or the ability to embedd (embedding) the site in other applications. The first refers to services like CDN (Content Delivery Network), the second to services like sandboxes - specially allocated (isolated) environments for code execution. Source refers to protocol, host, domain, and port.

- Cross-Origin Resource Sharing (CORS);
- Cross-Origin Embedder Policy (COEP).

Security threats that exist on the web

Protection of the site from injection vulnerabilities

Threats associated with the possibility of embedding code occur when unverified data processed by the application can affect the behavior of the application. In particular, this can lead to the execution of scripts controlled by the attacker (belonging to him). The most common type of code injection attack is cross-site scripting (Cross-Site Scripting, XSS; by the way, the abbreviation XSS was chosen to avoid confusion with CSS)in various forms, including reflected or non-permanent XSS (reflected XSS), stored or persistent

XSS (stored XSS), XSS based on the DOM (DOM XSS), etc. XSS can give the attacker full access to user data that is processed by the application, and to other information within the source.
Traditional methods of protection against XSS are: automatic shielding of HTML templates using special tools, refusal to use unsafe JavaScript APIs (for example, eval() or innerHTML), storing user data in another source and neutralizing or sanitizing data coming from users, for example, through filling in form
fields.

Recommendations

- Use CSP to determine which scripts can run in your application.
- Use Trusted Types to neutralize data passed to insecure APIs.
- Use X-Content-Type-Options to prevent the browser from misinterpreted MIME types of loaded resources.

Site
isolation Web openness allows sites to interact with each other in ways that can lead to security breaches. This includes sending "unexpected" requests to authenticate or load data from the application into the attacker's document, allowing the attacker to read or even modify that data.
The most common vulnerabilities related to the general availability of the application are clickjacking, Cross-Site Request Forgery (XSRF), Cross-Site Script Inclusion (XSSI), and various leaks of information between sources.

Recommendations

- Use X-Frame-Options to prevent your document from being embedded in other applications.
- use CORP to prevent other sources from using your site's resources.
- Use COOP to protect your app's windows from interacting with other apps.
- Use CORS to control access to your site's resources from other sources.

The security of sites with sophisticated

Spectre functionality makes any data uploaded to the same browsing context group potentially public, despite the domain restriction rule. Browsers limit the features that can lead to a security breach by using a code runtime called Cross-Origin Isolation. This, in particular, allows you to safely use such powerful features as SharedArrayBuffer.

Recommendations

- Use COEP in conjunction with COOP to provide cross-market isolation for your application.

Encryption of outgoing traffic

Insufficient encryption of transmitted data can lead to the fact that the attacker, in case of interception of this data, will receive information about the interaction of users with the application.
Ineffective encryption can be caused by the following:

- Using HTTP instead of HTTPS
- mixed content (when some resources are loaded over HTTPS, and others over HTTP);
- cookies without the Secure attribute or the corresponding prefix (it also makes sense to define the HttpOnly configuration);
- weak CORS policy.

Recommendations

- Use HSTS to serve all of your app's content over HTTPS.

Let's move on to the headlines.

Content Security Policy (CSP)

XSS is an attack where a vulnerability existing on a site allows an attacker to embed and execute their scripts. CSP provides an additional layer to repel such attacks by restricting the scripts that can run on the page.
Engineers from Google recommend using a strict CSP
mode. This can be done in one of two ways:

- If HTML pages are rendered on the server, you should use a nonce-based CSP.
- If the markup is static or delivered from a kesha, for example, if the application is a single-page (SPA), a hash-based CSP should be used.

Example of using nonce-based CSP:

Code: Copy to clipboard

```
Content-Security-Policy:
 script-src 'nonce-{RANDOM1}' 'strict-dynamic' https: 'unsafe-inline';
 object-src 'none';
 base-uri 'none';
```

Using CSP

*Note:*CSP is an additional protection against XSS attacks, the main protection is to neutralize the data entered by the user.

1. *Nonce-based CSP* nonce is a random number that is used only once.
If you can't generate such a number for each response, then it's best to use hash-based CSP.
Generate a nonce on the server for the script in response to each request and set the following header:

Code: Copy to clipboard

```
Content-Security-Policy:
 script-src 'nonce-{RANDOM1}' 'strict-dynamic' https: 'unsafe-inline';
 object-src 'none';
 base-uri 'none';
```

Then, in the markup, set each script tag to the nonce attribute with the value of the string {RANDOM1}:

Code: Copy to clipboard

```
<script nonce="{RANDOM1}" src="https://example.com/script1.js"></script>
<script nonce="{RANDOM1}">
 // ...
</script>
```

A good example of using nonce-based CSP is Google Photos.

2. *Hash-based CSP Server:*

Code: Copy to clipboard

```
Content-Security-Policy:
 script-src 'sha256-{HASH1}' 'sha256-{HASH2}' 'strict-dynamic' https: 'unsafe-inline';
 object-src 'none';
 base-uri 'none';
```

In this case, you can only use built-in scripts, since most browsers currently do not support hashing external scripts.

Code: Copy to clipboard

```
<script>
 // встроенный script1
</script>
<script>
 // встроенный script2
</script>
```

CSP Evaluator is a great tool for evaluating CSP.

Notes

- https: is a fallback for Firefox, and unsafe-inline is for very old browsers;
- the frame-ancestors directive protects the site from clickjacking by preventing other sites from using the content of your application. X-Frame-Options is a simpler solution, but frame-ancestors allows you to fine-tune the allowed sources;
- CSP can be used to ensure that all resources are loaded over HTTPS. This is not very relevant as most browsers currently block mixed content;
- CSP can be used in report-only mode;
- CSP can be set in markup as a meta tag.

The following directives can be used in the heading in question:

Directive	Description
base-uri	Specifies the base URI for relative
default-src	Defines a policy for loading resources of all types in the absence of a special directive (default policy)
script-src	Defines scripts that can run on a page
object-src	Specifies where resources can be loaded from – plugins
style-src	Defines the styles that can be applied to the page
img-src	Determines where images can be downloaded from
media-src	Specifies where audio and video files can be downloaded from

child-src	Specifies where frames can be loaded from
frame-ancestors	Specifies where (in which sources) a resource can be loaded into frames
font-src	Specifies where fonts can be loaded from
connect-src	Specifies allowed URIs
manifest-src	Specifies where manifest files can be downloaded from
form-action	Specifies which URIs can be used to submit forms (in the action attribute)
sandbox	Defines the HTML sandbox policy that the user applies to the protected resource
script-nonce	Specifies that a unique value is required to run the script
plugin-types	Defines a set of plug-ins that can be called by a protected resource by restricting the types of embedded resources
reflected-xss	Used to activate/deactivate browser heuristic methods to filter or block reflected XSS attacks
block-all-mixed-content	Prevents mixed content from being downloaded
upgrade-insecure-requests	Specifies that insecure resources (loaded over HTTP) should be loaded over HTTPS
report-to	Defines the group (specified in the Report-To header) to which policy violation reports are sent

Possible directive values for CSP non-strict mode:

- 'self' — resources can be loaded only from this source;
- 'none' — prohibition on loading resources;
- * — resources can be downloaded from any source;
- example.com - resources can only be loaded from example.com.

Code: Copy to clipboard

```
Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; scrip
```

In this case, images can be downloaded from any source, other media files - only with media1.com and media2.com (excluding their subdomains), scripts - only with example.com.

Trusted Types

DOM-based XSS is an attack where malicious code is passed to a receiver that supports dynamic code execution, such as eval() or innerHTML.
Trusted Types provides tools for creating, modifying, and maintaining applications that are fully protected from the XSS DOM.
This mode can be enabled via CSP. It makes JavaScript code secure by default by restricting the values

accepted by unsafe APIs to a special object, the Trusted Type.
To create such objects, you can define policies that verify compliance with security rules (such as escaping and neutralizing) before writing data to the DOM.
These policies are then placed in code that may be of interest to the XSS DOM.

Example of using Enable Trusted Types for dangerous DOM receivers:

Code: Copy to clipboard

```
Content-Security-Policy: require-trusted-types-for 'script'
```

Currently, the only available value for the require-trusted-types-for directive is script.
Of course, Trusted Types can be combined with other CSP directives:

Code: Copy to clipboard

```
Content-Security-Policy:
 script-src 'nonce-{RANDOM1}' 'strict-dynamic' https: 'unsafe-inline';
 object-src 'none';
 base-uri 'none';
 require-trusted-types-for 'script';
```

You can use the trusted-types directive to restrict the namespace for Trusted Types policies, such as trusted-types myPolicy.
Define the
policy:

Code: Copy to clipboard

```
// проверяем поддержку
if (window.trustedTypes && trustedTypes.createPolicy) {
 // создаем политику
 const policy = trustedTypes.createPolicy('escapePolicy', {
   createHTML: (str) => str.replace(/\</g, '&lt;').replace(/>/g, '&gt;')
 })
}
```

Apply the policy:

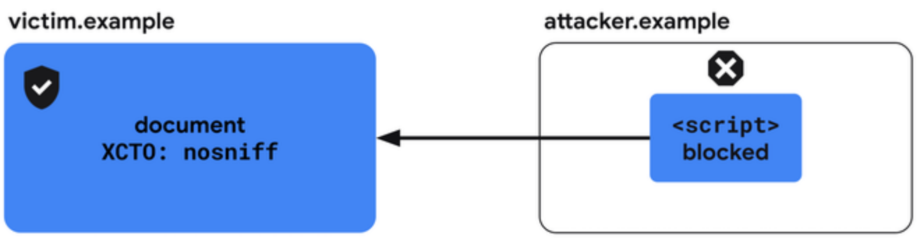
Code: Copy to clipboard

```
// будет выброшено исключение
el.innerHTML = 'some string'
// ок
const escaped = policy.createHTML('<img src=x onerror=alert(1)>')
el.innerHTML = escaped // '&lt;img src=x onerror=alert(1)&gt;'
```

The require-trusted-types-for 'script' directive makes the use of a trusted type mandatory. Any attempt to use a string in an insecure API will fail.
can read more about Trusted Types here.

X-Content-Type-Options

When a malicious HTML document is served by your domain (for example, when an image uploaded to a photo storage service contains valid markup), some browsers may consider it an active document and allow it to execute scripts in the context of the application.
X-Content-Type-Options: nosniff forces the browser to check the correctness of the MIME type in the header of the Received Content-Type
response. We recommend that you set this header for all resources that you download.



Code: Copy to clipboard

```
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
```

X-Frame-Options

If a malicious site is able to embed your app as an iframe, it could provide an attacker with the ability to induce unintentional user actions through clickjacking. In some cases, this also allows the attacker to examine the contents of the document.
X-Frame-Options is an indicator of whether your site should be rendered in <frame>, <iframe>, <embed>or <object>.

In order to allow embedding only certain pages of the site, the frame-ancestors directive of the CSP header is
used.

Examples of use Completely prohibit implementation:

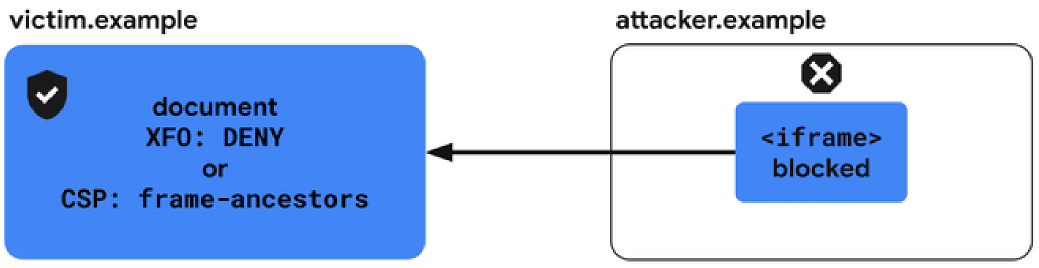
Code: Copy to clipboard

```
X-Frame-Options: DENY
```

Allow the creation of frames only on your own site:

Code: Copy to clipboard

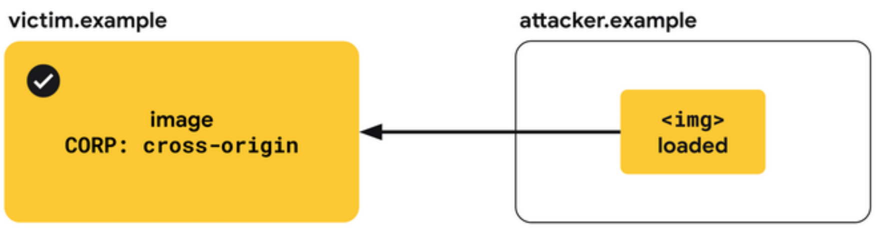
```
X-Frame-Options: SAMEORIGIN
```



*Note:*By default, all documents are embedded.

Cross-Origin-Resource-Policy (CORP)

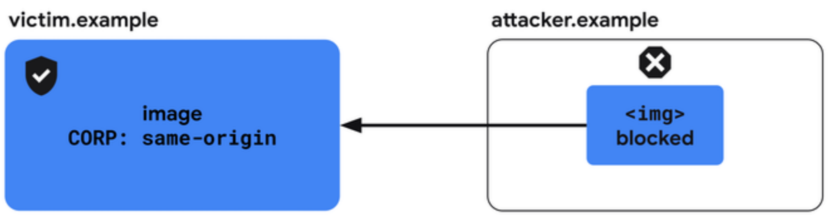
An attacker can inject your site's resources into their app in order to obtain information about your site.
CORP determines which sites can embed your app's resources.
This header takes 1 of 3 possible values: same-origin, same-site, and cross-origin.
For services like CDN, we recommend that you use the cross-origin value if you have not defined the appropriate CORS header for
them.



Code: Copy to clipboard

```
Cross-Origin-Resource-Policy: cross-origin
```

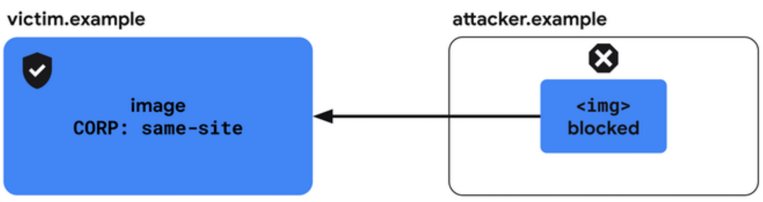
same-origin allows the embedding of resources by pages belonging to the same source. This value applies to sensitive user information or responses from the API that are designed to be used within this source. Note: Resources will still be available for download because CORP restricts only the deployment of these resources to other sources.



Code: Copy to clipboard

```
Cross-Origin-Resource-Policy: same-origin
```

same-site is designed for resources that are used not only by the domain (as in the case of same-origin), but also by its subdomains.

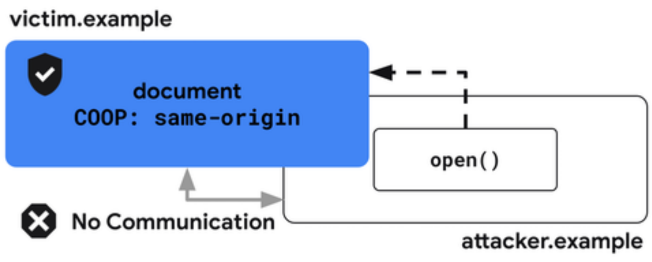


Code: Copy to clipboard

```
Cross-Origin-Resource-Policy: same-site
```

Cross-Origin-Opener-Policy (COOP)

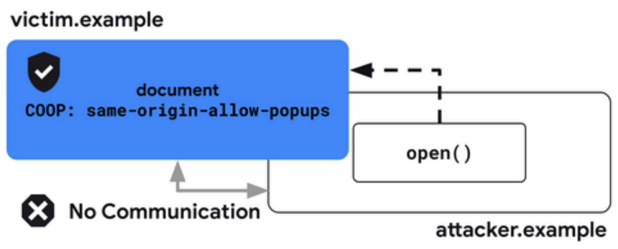
If the attacker's site can open another site in a pop-up (pop-up window),then the attacker has the opportunity to search for intersite sources of information leakage. In some cases, this also allows for the implementation of the side channel attack described in Spectre.
The Cross-Origin-Opener-Policy header allows you to prevent the site from opening using the window.open() method or the target="_blank" link without rel="noopener".
As a result, someone who tries to open the site in this way will not have a link to the site, and he will not be able to interact with it.
The same-origin value of the header in question allows you to completely prohibit the opening of the site in other sources.



Code: Copy to clipboard

```
Cross-Origin-Opener-Policy: same-origin
```

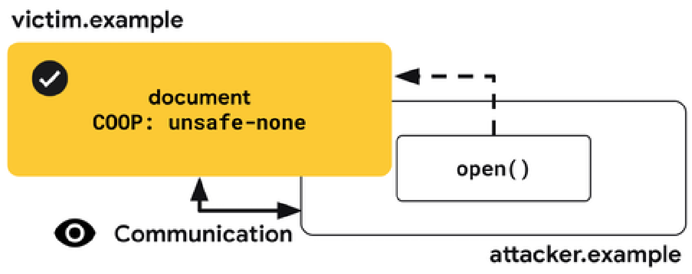
The same-origin-allow-popups value also protects the document from opening other sources in popups, but allows the application to interact with its own popups.



Code: Copy to clipboard

```
Cross-Origin-Opener-Policy: same-origin-allow-popups
```

unsafe-none is the default value, it allows the site to be opened as a pop-up in other sources.



Code: Copy to clipboard

```
Cross-Origin-Opener-Policy: unsafe-none
```

We may receive reports from COOP:

Code: Copy to clipboard

```
Cross-Origin-Opener-Policy: same-origin; report-to="coop"
```

COOP also supports report-only mode, which allows you to receive reports of violations without blocking them.

Code: Copy to clipboard

```
Cross-Origin-Opener-Policy-Report-Only: same-origin; report-to="coop"
```

Cross-Origin Resource Sharing (CORS)

CORS is not a header, but a mechanism used by the browser to provide access to application resources. By default, browsers use a single source or shared origin policy that denies access to such resources from other sources.
For example, if you load an image from another source, even though it appears on the page, the JavaScript code will not have access to it. The resource provider can grant such access through the CORS configuration using two headers:

Code: Copy to clipboard

```
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Credentials: true
```

Using CORS

Let's start with the fact that there are two types of HTTP requests. Depending on the details of the request, it can be classified as simple or complex (a request that requires a preliminary request).
The criteria for a simple query are as follows:

- the request method is GET, HEAD, or POST.
- custom headers can only be Accept, Accept-Language, Content-Language and Content-Type;
- the value of the Content-Type header can only be application/x-www-form-urlencoded, multipart/form-data, or text/plain.

All other queries are considered complex.

Simple query
In this case, the browser sends a request to another source with an Origin header whose value is the source of the request:

Code: Copy to clipboard

```
Get / HTTP/1.1
Origin: https://example.com
```

Answer:

Code: Copy to clipboard

```
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Credentials: true
```

- Access-Control-Allow-Origin: https://example.com means that https://example.com has access to the contents of the response. If the value of this header is *, the resources will be available to any site. In this case, credentials are not required;
- Access-Control-Allow-Credentials: true means that the request to obtain resources must contain privileges (cookies). If you do not have permissions in the request, even if there is a source in the Access-Control-Allow-Origin header, the request will be rejected.

Complex query

A preliminary query is executed before a complex query. It is executed by the OPTIONS method to determine whether the main request can be sent:

Code: Copy to clipboard

```
OPTIONS / HTTP/1.1
Origin: https://example.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type
```

- `Access-Control-Request-Method: POST` — the subsequent request will be sent by the POST method;
- `Access-Control-Request-Headers: X-PINGOTHER, Content-Type` - The subsequent request will be sent with X-PINGOTHER and Content-Type headers.

Answer:

Code: Copy to clipboard

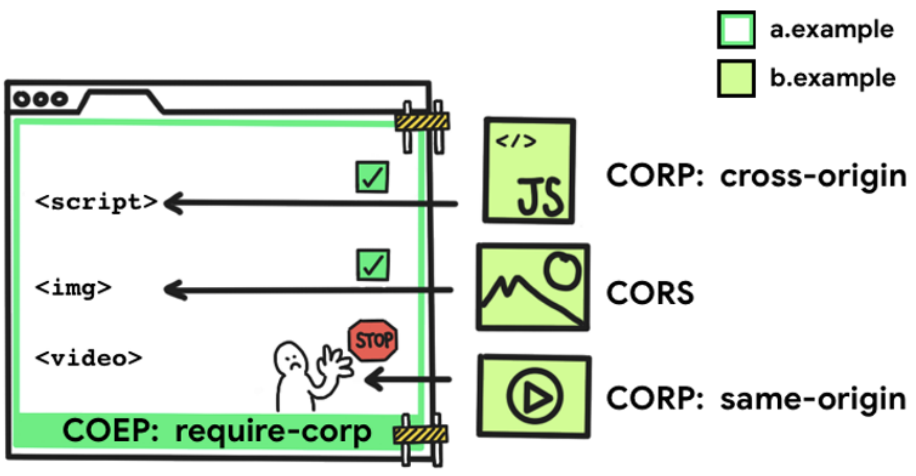
```
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
```

- `Access-Control-Allow-Methods: POST, GET, OPTIONS` — the subsequent request can be executed by the specified methods;
- `Access-Control-Allow-Headers: X-PINGOTHER, Content-Type` — a subsequent request may contain the specified headers;
- `Access-Control-Max-Age: 86400` — the result of a complex query will be written to the cache and will be stored there for 86400 seconds.

Cross-Origin-Embedder-Policy (COEP)

To prevent resources from being stolen from other sources using the attacks described in Spectre, features such as SharedArrayBuffer, performance.measureUserAgentSpecificMemory(), or JS Self Profiling API are disabled by default.

Cross-Origin-Embedder-Policy: Require-corp prohibits documents and workers from uploading images, scripts, styles, frames, and other resources until they are allowed to be accessed using CORS or CORP headers. COEP can be used in conjunction with COOP to configure cross-site document isolation.
At the moment, require-corp is the only available value of the header in question, except unsafe-none, which is the default
value.



Complete cross-site isolation of the application

Code:

Copy to clipboard

```
Cross-Origin-Embedder-Policy: require-corp
Cross-Origin-Opener-Policy: same-origin
```

Isolation with lock reports

Code:

Copy to clipboard

```
Cross-Origin-Embedder-Policy: require-corp; report-to="coep"
```

Reports only

Code:

Copy to clipboard

```
Cross-Origin-Embedder-Policy-Report-Only: require-corp; report-to="coep"
```

HTTP Strict Transport Security (HSTS)

Data transmitted over HTTP is not encrypted, making it available to interceptors at the network level. The Strict-Transport-Security header prohibits the use of
HTTP. If this header is present, the browser will use HTTPS without redirecting to HTTP (if there is no resource over HTTPS) for the specified time (max-age).

Code:

Copy to clipboard

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

Directive

- max-age — time in seconds during which the browser must "remember" that the site is available only via HTTPS;
- includeSubDomains — extends the policy to subdomains.

Other headings

Referrer-Policy

The Referrer-Policy header defines the content of the referrer information specified in the Referer header. The Referer header contains the address of the request, such as the address of the previous page, or the address of the uploaded image, or other resource. It is used for analytics, logging, cache optimization, etc. However, it can also be used to track or steal information, perform side effects leading to leakage of sensitive user data, etc.

Code:

Copy to clipboard

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

Possible values

Meaning	Description
no-referrer	The Referer header is not included in the query
no-referrer-when-downgrade	The default value. A referrer is specified when a request is made between HTTPS and HTTPS, but not when a request is made between HTTPS and HTTP
origin	Only the source of the request is specified (for example, the document referrer will https://example.com/page.html will https://example.com)
origin-when-cross-origin	when you run a query within a single source the full URL is specified, otherwise only the source is specified (as in the previous example)
same-origin	When you run a query within the same source, the source is specified, otherwise the referrer is not specified
strict-origin	Looks like no-referrer-when-downgrade, but only the source is listed
strict-origin-when-cross-origin	Combination of strict-origin and origin-when-cross-origin
unsafe-url	The full URL is always specified

*Note:*This header is not supported by mobile Safari.

Clear-Site-Data

The Clear-Site-Data header starts clearing the data stored in the browser (cookies, storage, cache) associated with the source. This gives developers control over the data stored locally in the user's browser. This header can be used, for example, during the logout process to clear data stored on the client side.

Code:

Copy to clipboard

```
Clear-Site-Data: "*"
```

Possible values:

Meaning	Description
"cache"	Tells the browser that the server wants to clear locally cached data for the source of the response to the request
"cookies"	Tells the browser that the server wants to delete all cookies for the source. The authentication data will also be cleared. this affects both the domain itself and its subdomains
"storage"	Tells the browser that the server wants to clear all browser storage (localStorage, sessionStorage, IndexedDB, service worker registration - for each registered SV the unregister(), AppCache, WebSQL method, FileSystem API data, plugin data is called)
"executionContexts"	Tells the browser that the server wants to reload all browser contexts (currently barely supported)
"*"	Tells the browser that the server wants to delete all data

*Note:*This header is not supported by Safari.

Permissions-Policy

This header is a replacement for the Feature-Policy header and is designed to control access to some advanced features.

Code:

Copy to clipboard

```
Permissions-Policy: camera=(), fullscreen=*, geolocation=(self "https://example.com" "https://
```

In this case, we completely deny access to the camera (video input) of the device, allow access to the requestFullScreen() method (to enable full-screen video playback mode) for everyone, and to information about the location of the device - only for sources of example.com and another.example.com.
directives

Directive	Description
accelerometer	Controls whether the current document can collect information about the acceleration (projection of apparent acceleration) of the device using the Accelerometer interface
ambient-light-sensor	Controls whether the current document can collect information about the amount of light in the device's environment using the AmbientLightSensor interface

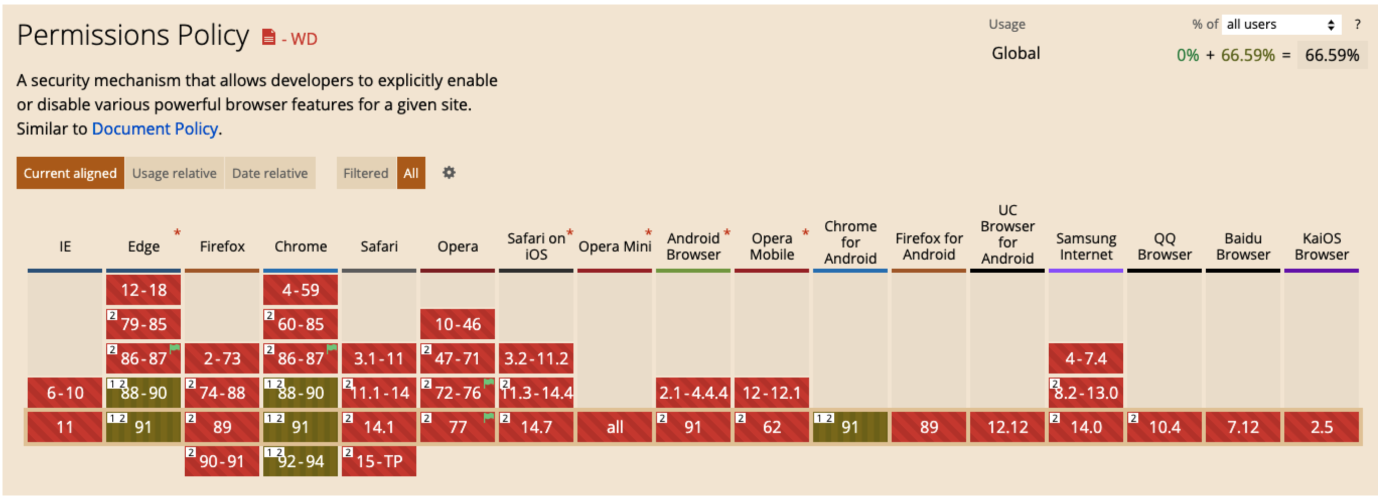
autoplay	Controls whether the current document can automatically play media requested through the HTMLMediaElement interface
battery	Determines whether the Battery Status API can be used
camera	Determines whether the device's video input can be used
display-capture	Specifies whether the screen can be captured using the getDisplayMedia() method
document-domain	Specifies whether document.domain can be installed
encrypted-media	Determines whether to use the Encrypted Media Extensions API (EME)
execution-while-not-rendered	Specifies whether tasks can run in frames without rendering them (for example, when they are hidden or their diplay property is set to none)
execution-while-out-of-viewport	Specifies whether tasks can be performed in frames outside the viewport
fullscreen	Specifies whether to use the requestFullScreen() method
geolocation	Determines whether to use the Geolocation API
gyroscope	Controls whether the current document can collect device orientation information using the Gyroscope API
layout-animations	Specifies whether animations can be shown
legacy-image-formats	Specifies whether legacy images can be displayed
magnetometer	Controls whether the current document can collect device orientation information using the Magnetometer API
microphone	Determines whether the audio input of the device can be used
midi	Determines whether the Web MIDI API can be used
navigation-override	Defines the ability to control spatial navigation by mechanisms developed by the author of the application
oversized-images	Specifies whether large images can be uploaded and displayed
payment	Determines whether the Payment Request API can be used
picture-in-picture	Specifies whether video can be played in picture-in-picture mode
publickey-credentials-get	Specifies whether the Web Authentication API can be used to retrieve public keys, for example, through navigator.credentials.get()

sync-xhr	Determines whether the WebUSB API can be used
vr	Determines whether the WebVR API can be used
wake-lock	Specifies whether the Wake Lock API can be used to prevent the device from switching to power-saving mode
screen-wake-lock	Specifies whether the Screen Wake Lock API can be used to prevent device screen locking
web-share	Specifies whether the Web Share API can be used to transfer text, links, images, and other content
xr-spatial-tracking	Specifies whether the WebXR Device API can be used to interact with a WebXR session

Possible values

- =() — complete prohibition;
- =* — full access;
- (self "https://example.com") — grant permission only to the specified source.

The specification of the header in question is in the status of a working draft, so its support leaves much to be desired:



Permissions Policy - WD

A security mechanism that allows developers to explicitly enable or disable various powerful browser features for a given site. Similar to Document Policy.

Usage: Global — % of all users: 0% + 66.59% = 66.59%

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
	12-18		4-59													
	[2] 79-85		[2] 60-85		10-46											
	[2] 86-87	2-73	[2] 86-87	3.1-11	[2] 47-71	3.2-11.2							4-7.4			
6-10	[1 2] 88-90	[2] 74-88	[1 2] 88-90	[2] 11.1-14	[2] 72-76	[2] 11.3-14.4		2.1-4.4.4	12-12.1				[2] 8.2-13.0			
11	[1 2] 91	[2] 89	[1 2] 91	[2] 14.1	[2] 77	[2] 14.7	all	[2] 91	[2] 62	[1 2] 91	89	12.12	[2] 14.0	[2] 10.4	7.12	2.5
		[2] 90-91	[1 2] 92-94	[2] 15-TP												

Let's move on to the practical part.

Express Application Development

Create a directory for the project, go to it and initialize the project:

Code: Copy to clipboard

```
mkdir secure-app
cd !$

yarn init -yp
# или
npm init -y
```

We form the structure of the project:

Code: Copy to clipboard

```
- public
 - favicon.png
 - index.html
 - style.css
 - script.js
- index.js
- .gitignore
- ...
```

The icon can be found here.
Let's sketch out some simple
code.
In public/index.html, we connect an icon, styles, script, Google fonts, Bootstrap and Boostrap Icons via CDN, create elements for the title, date, time and buttons:

Code: Copy to clipboard

```
<!DOCTYPE html>
<html lang="ru">
 <head>
   <meta charset="UTF-8" />
   <meta http-equiv="X-UA-Compatible" content="IE=edge" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>Secure App</title>
   <link rel="icon" href="favicon.png" />
   <link rel="preconnect" href="https://fonts.googleapis.com" />
   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
   <link
     href="https://fonts.googleapis.com/css2?family=Montserrat&display=swap"
     rel="stylesheet"
   />
   <link
     href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css"
     rel="stylesheet"
     integrity="sha384-+0n0xVW2eSR5OomGNYDnhzAbDsOXxcvSN1TPprVMTNDbiYZCxYbOOl7+AMvyTG2x"
     crossorigin="anonymous"
   />
   <link
     rel="stylesheet"
     href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css"
   />
   <link rel="stylesheet" href="style.css" />
 </head>
 <body>
   <div class="container">
     <h1>Secure App</h1>
     <p>
       <i class="bi bi-calendar"></i>
       Сегодня <time class="date"></time>
     </p>
     <p>
```

```
        <i class="bi bi-clock"></i>
        Сейчас <time class="time"></time>
      </p>
      <div class="buttons">
        <button class="btn btn-danger btn-stop">Остановить таймер</button>
        <button class="btn btn-primary btn-add">Добавить шаблон</button>
```

Add styles to public/style.css

Code: Copy to clipboard

```
* {
 margin: 0;
 padding: 0;
 box-sizing: border-box;
 font-family: 'Montserrat', sans-serif;
}

body {
 min-height: 100vh;
 display: grid;
 place-content: center;
 text-align: center;
}

h1 {
 margin: 0.5em 0;
 text-transform: uppercase;
 font-size: 3rem;
}

p {
 font-size: 1.15rem;
}

.buttons {
 margin: 0.5em 0;
 display: flex;
 flex-direction: column;
 align-items: center;
 gap: 0.5em;
}

button {
 cursor: pointer;
}

pre {
 margin: 0.5em 0;
 white-space: pre-wrap;
 text-align: left;
```

In public/script.js we do the following:

- define a policy of trusted types.
- create utilities to get a reference to the DOM element, format the date and time and register the handler (by default, a one-time and triggering a callback when the click event occurs);
- get references to DOM elements;

- define settings for formatting the date and time.
- add the date and time as the text content of the corresponding elements;
- Define triggers for handlers: to stop the timer, add an HTML template with potentially malicious code, and retrieve HTTP headers.
- register handlers.

Code:

```
// политика доверенных типов
let policy
if (window.trustedTypes && trustedTypes.createPolicy) {
 policy = trustedTypes.createPolicy('escapePolicy', {
   createHTML: (str) => str.replace(/\</g, '&lt').replace(/>/g, '&gt')
 })
}

// утилиты
// для получения ссылки на DOM-элемент
const getEl = (selector, parent = document) => parent.querySelector(selector)
// для форматирования даты и времени
const getDate = (options, locale = 'ru-RU', date = Date.now()) =>
 new Intl.DateTimeFormat(locale, options).format(date)
// для регистрации обработчика (по умолчанию одноразового и запускающего колбэк при возникн
const on = (el, cb, event = 'click', options = { once: true }) =>
 el.addEventListener(event, cb, options)

// DOM-элементы
const containerEl = getEl('.container')
const dateEl = getEl('.date', containerEl)
const timeEl = getEl('.time', containerEl)
const stopBtnEl = getEl('.btn-stop', containerEl)
const addBtnEl = getEl('.btn-add', containerEl)
const getBtnEl = getEl('.btn-get', containerEl)

// настройки для даты
const dateOptions = {
 weekday: 'long',
 day: 'numeric',
 month: 'long',
 year: 'numeric'
}
// настройки для времени
const timeOptions = {
 hour: 'numeric',
 minute: 'numeric',
 second: 'numeric'
}
```

Set dependencies.

production:

Code:

```
yarn add express
```

For development:

Code: Copy to clipboard

```
yarn add -D nodemon open-cli
```

- express — Node.js framework that simplifies server development;
- nodemon — a utility for starting the server for development and its automatic restart when the corresponding file is updated;
- open-cli — a utility for automatically opening the browser tab at the specified address.

Define in package.json commands to start servers:

Code: Copy to clipboard

```
"scripts": {
 "dev": "open-cli http://localhost:3000 && nodemon index.js",
 "start": "node index.js"
}
```

Let's start implementing the server.
fairness, it should be noted that in the Node.js ecosystem there is a special utility for installing HTTP headers related to the security of web applications - Helmet.
Cheat sheet for working with this utility can be found here.
is also a special utility for working with CORS - Cors.
Cheat sheet for working with this utility can be found here.
Most headings can be identified right
away:

Код: Скопировать в буфер обмена

```
// предотвращаем `MIME sniffing`
'X-Content-Type-Options': 'nosniff',

// для старых браузеров, плохо поддерживающих `CSP`
'X-Frame-Options': 'DENY',
'X-XSS-Protection': '1; mode=block',

// по умолчанию браузеры блокируют CORS-запросы
// дополнительные CORS-заголовки
'Cross-Origin-Resource-Policy': 'same-site',
'Cross-Origin-Opener-Policy': 'same-origin-allow-popups',
'Cross-Origin-Embedder-Policy': 'require-corp',

// запрещаем включать информацию о реферере в заголовок `Referer`
'Referrer-Policy': 'no-referrer',

// инструктируем браузер использовать `HTTPS` вместо `HTTP`
// 31536000 секунд — это 365 дней
'Strict-Transport-Security': 'max-age=31536000; includeSubDomains'
```

Также добавим заголовок Expect-CT:

Код: Скопировать в буфер обмена

```
// 86400 секунд — это 1 сутки
'Expect-CT': 'enforce, max-age=86400'
```

Блокируем доступ к камере, микрофону, информации о местонахождении и Payment Request API:

Код: Скопировать в буфер обмена

```
'Permissions-Policy': 'camera=(), microphone=(), geolocation=(), payment=()'
```

Директивы для CSP:

Код: Скопировать в буфер обмена

```
'Content-Security-Policy': `
 // запрещаем загрузку плагинов
 object-src 'none';
 // разрешаем выполнение только собственных скриптов
 script-src 'self';
 // разрешаем загрузку только собственных изображений
 img-src 'self';
 // разрешаем открытие приложения только в собственных фреймах
 frame-ancestors 'self';
 // включаем политику доверенных типов для скриптов
 require-trusted-types-for 'script';
 // блокируем смешанный контент
 block-all-mixed-content;
 // инструктируем браузер использовать `HTTPS` для ресурсов, загружаемых по `HTTP`
 upgrade-insecure-requests
`
```

Обратите внимание: все директивы должны быть указаны в одну строку без переносов. Мы не определяем директивы для стилей и шрифтов, поскольку они загружаются из других источников.

Также *обратите внимание*, что мы не используем nonce для скриптов, поскольку мы не рендерим разметку на стороне сервера, но я приведу соответствующий код.

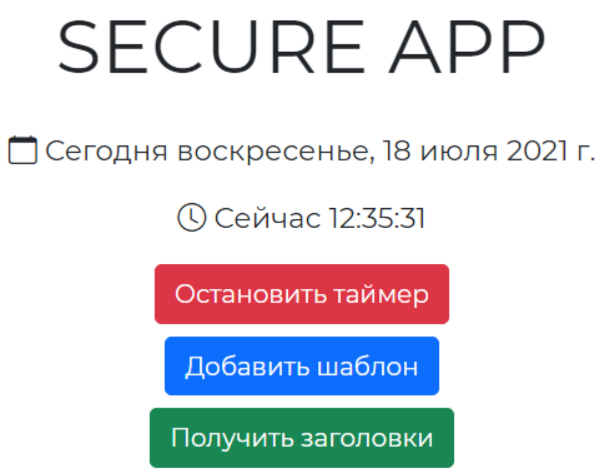
index.js:

Код: Скопировать в буфер обмена

```
const express = require('express')
// утилита для генерации уникальных значений
// const crypto = require('crypto')

// создаем экземпляр Express-приложения
const app = express()
```

```
// посредник для генерации `nonce`
/*
const getNonce = (_, res, next) => {
 res.locals.cspNonce = crypto.randomBytes(16).toString('hex')
 next()
}
*/

// посредник для установки заголовков
// 31536000 – 365 дней
// 86400 – 1 сутки
const setSecurityHeaders = (_, res, next) => {
 res.set({
   'X-Content-Type-Options': 'nosniff',
   'X-Frame-Options': 'DENY',
   'X-XSS-Protection': '1; mode=block',
   'Cross-Origin-Resource-Policy': 'same-site',
   'Cross-Origin-Opener-Policy': 'same-origin-allow-popups',
   'Cross-Origin-Embedder-Policy': 'require-corp',
   'Referrer-Policy': 'no-referrer',
   'Strict-Transport-Security': 'max-age=31536000; includeSubDomains',
   'Expect-CT': 'enforce, max-age=86400',
   'Content-Security-Policy': `object-src 'none'; script-src 'self'; img-src 'self'; frame-
   'Permissions-Policy': 'camera=(), microphone=(), geolocation=(), payment=()'
 })
 next()
}

// удаляем заголовок `X-Powered-By`
app.disable('x-powered-by')
// подключаем посредник для генерации `nonce`
// app.use(getNonce)
```

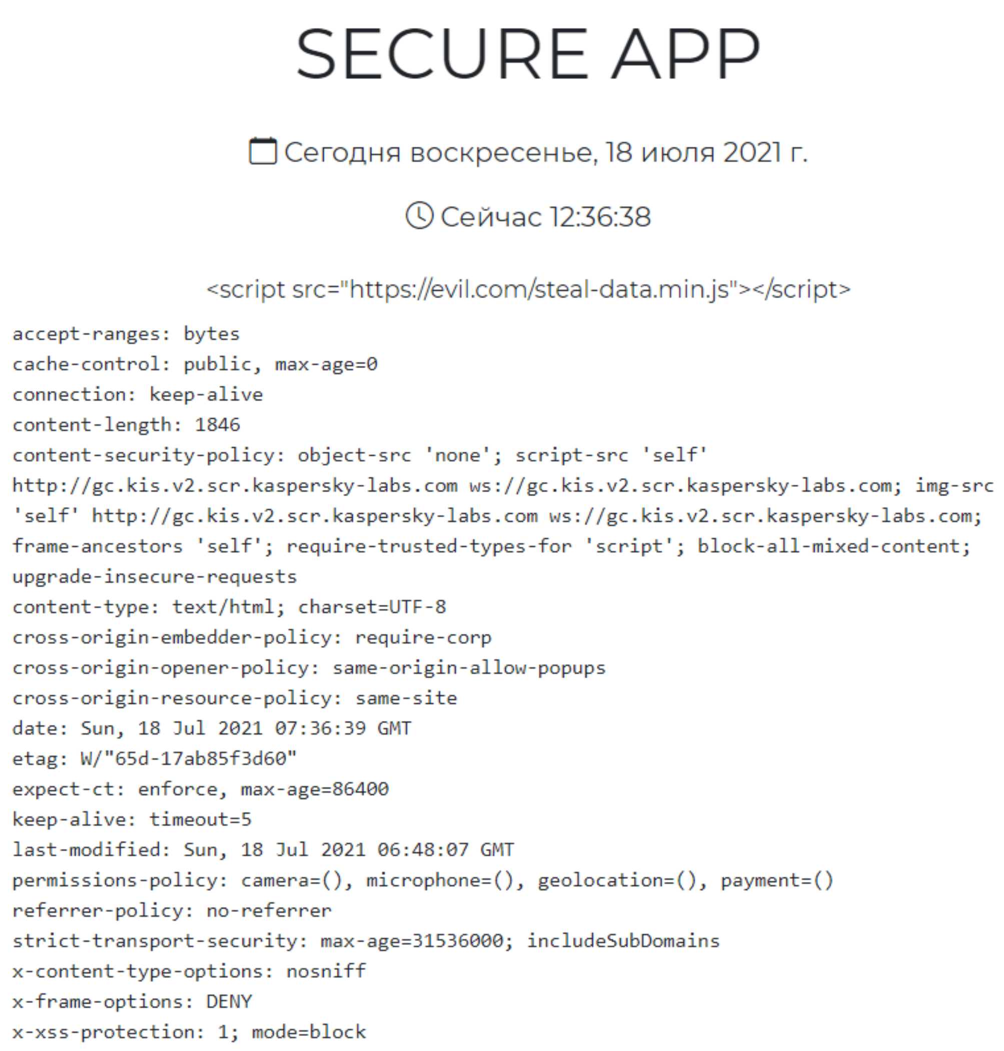
Execute the command yarn dev or npm run dev (of course, node.jsmust be installed on your machine). This command starts the server for development and opens the browser tab at . `http://localhost:3000`



SECURE APP

Сегодня воскресенье, 18 июля 2021 г.

Сейчас 12:35:31



SECURE APP

Сегодня воскресенье, 18 июля 2021 г.

Сейчас 12:36:38

<script src="https://evil.com/steal-data.min.js"></script>

```
accept-ranges: bytes
cache-control: public, max-age=0
connection: keep-alive
content-length: 1846
content-security-policy: object-src 'none'; script-src 'self'
http://gc.kis.v2.scr.kaspersky-labs.com ws://gc.kis.v2.scr.kaspersky-labs.com; img-src
'self' http://gc.kis.v2.scr.kaspersky-labs.com ws://gc.kis.v2.scr.kaspersky-labs.com;
frame-ancestors 'self'; require-trusted-types-for 'script'; block-all-mixed-content;
upgrade-insecure-requests
content-type: text/html; charset=UTF-8
cross-origin-embedder-policy: require-corp
cross-origin-opener-policy: same-origin-allow-popups
cross-origin-resource-policy: same-site
date: Sun, 18 Jul 2021 07:36:39 GMT
etag: W/"65d-17ab85f3d60"
expect-ct: enforce, max-age=86400
keep-alive: timeout=5
last-modified: Sun, 18 Jul 2021 06:48:07 GMT
permissions-policy: camera=(), microphone=(), geolocation=(), payment=()
referrer-policy: no-referrer
strict-transport-security: max-age=31536000; includeSubDomains
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 1; mode=block
```

It's cool! Now let's deploy the application to Heroku and test its security with Security Headers and WebPageTest.

Express App Depot on Heroku

Create an account on Heroku. Globally install Heroku CLI:

Code: Copy to clipboard

```
yarn global add heroku
# или
npm i -g heroku
```

Check the installation:

Code: Copy to clipboard

```
heroku -v
```

Being in the root directory of the project, initialize the Git repository (of course, gitmust be installed on your machine), add and commit changes (do not forget to add node_modules to the .gitignore):

Code: Copy to clipboard

```
git init
git add .
git commit -m "Create secure app"
```

Create a remote repository on Heroku:

Code: Copy to clipboard

```
# авторизация
heroku login
# создание репо
heroku create
# подключение к репо
git remote -v
```

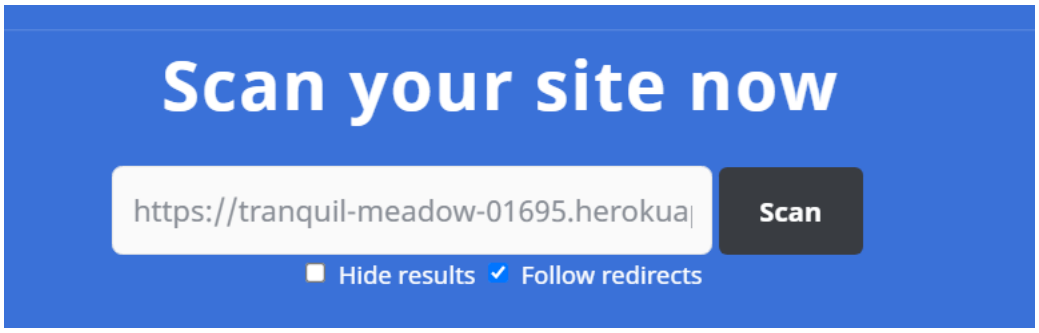
Deploy the application:

Code: Copy to clipboard

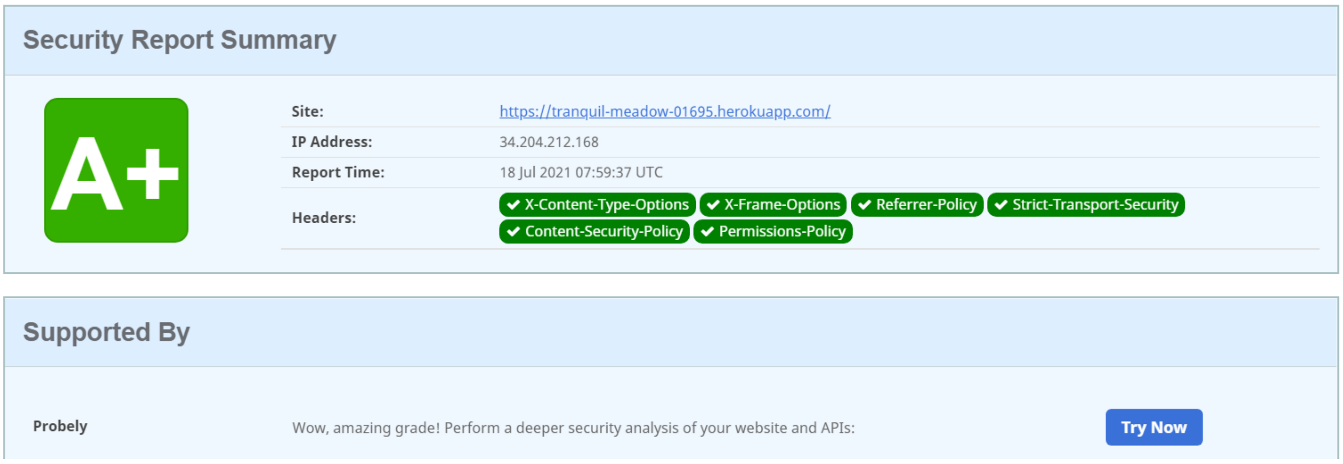
```
git push heroku master
```

Instructions for deploying the application to Heroku can be found here. After executing this command, the URL of your application deployed to Heroku will appear in the terminal, for example, https://tranquil-meadow-01695.herokuapp.com/ .

Go to the specified address and check the functionality of the application. Go to Security Headers,paste the URL of the application into the enter address here field and click on the Scan button:



We get the rating of the application:


Security Report Summary

A+

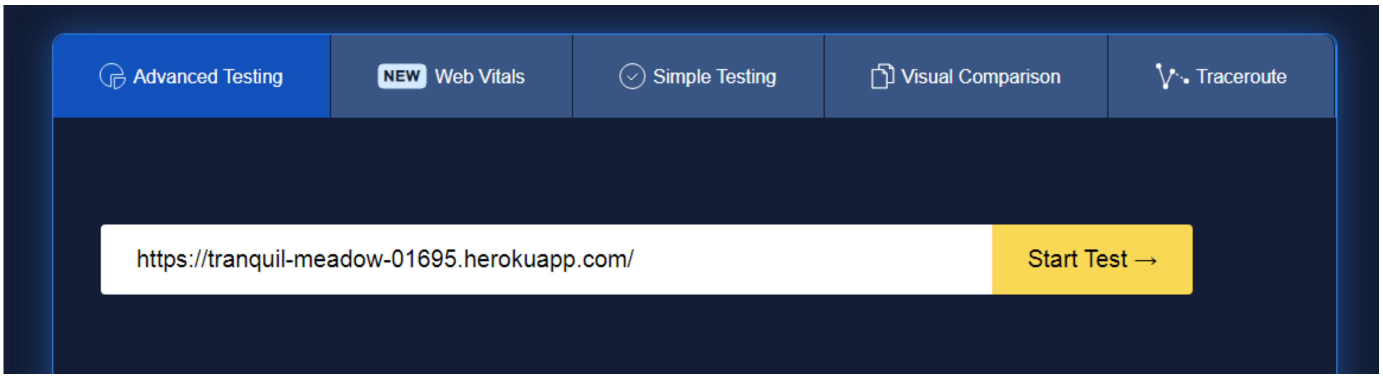
Site:	https://tranquil-meadow-01695.herokuapp.com/
IP Address:	34.204.212.168
Report Time:	18 Jul 2021 07:59:37 UTC
Headers:	✓ X-Content-Type-Options ✓ X-Frame-Options ✓ Referrer-Policy ✓ Strict-Transport-Security ✓ Content-Security-Policy ✓ Permissions-Policy

Supported By

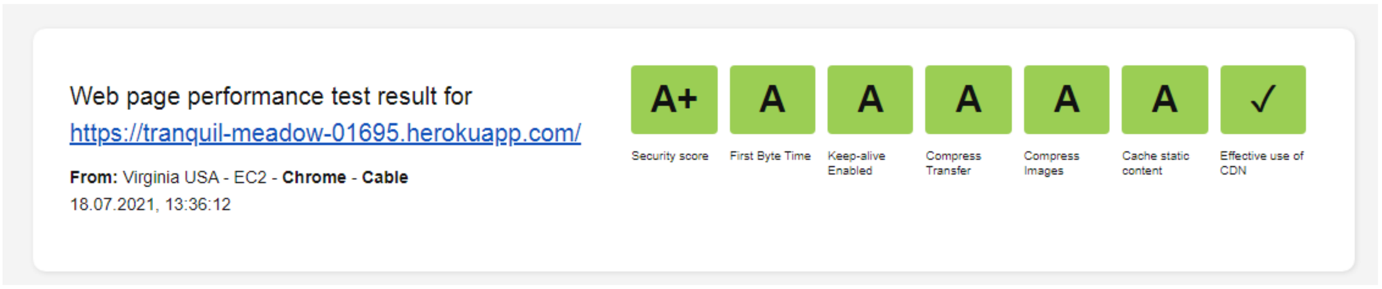
Probely Wow, amazing grade! Perform a deeper security analysis of your website and APIs: Try Now

In Supported By read Wow, great rating.... Go to WebPageTest,paste the URL of the application into the Field Enter a website URL...

and click on the Start Test -> button:



We get the results of the application evaluation (we are interested in the first assessment - Security score):


Web page performance test result for https://tranquil-meadow-01695.herokuapp.com/

From: Virginia USA - EC2 - **Chrome** - **Cable**
18.07.2021, 13:36:12

Security score	First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN
A+	A	A	A	A	A	✓

It looks like we did everything right. Steeply!

Application depot on Netlify

Transfer the files favicon.png, index.html, script.js and style.css from the public folder to a separate directory, for example, netlify. To configure the Netlify server, use the netlify.tomlfile.

Create this file in the project directory. We are only interested in the [[headers]] section:

Code: Copy to clipboard

```
[[headers]]
 for = "/*"
```

```
[headers.values]
  X-Content-Type-Options = "nosniff"
  X-Frame-Options = "DENY"
  X-XSS-Protection = "1; mode=block"
  Cross-Origin-Resource-Policy = "same-site"
  Cross-Origin-Opener-Policy = "same-origin-allow-popups"
  Cross-Origin-Embedder-Policy = "require-corp"
  Referrer-Policy = "no-referrer"
  Strict-Transport-Security = "max-age=31536000; includeSubDomains"
  Expect-CT = "enforce, max-age=86400"
  Content-Security-Policy = "object-src 'none'; script-src 'self'; img-src 'self'; frame-ance
  Permissions-Policy = "camera=(), microphone=(), geolocation=(), payment=()"
```

- `for = "/*"` means for all requests;
- `[header.values]` — headers and their values (just transfer them from the Express server, taking into account the peculiarities of the syntax).

Globally install Netlify CLI:

Code: Copy to clipboard

```
yarn global add netlify-cli
# или
npm i -g netlify-cli
```

Check the installation:

Code: Copy to clipboard

```
netlify -v
```

Log in:

Code: Copy to clipboard

```
netlify login
```

You can start the server for development (optional):

Code: Copy to clipboard

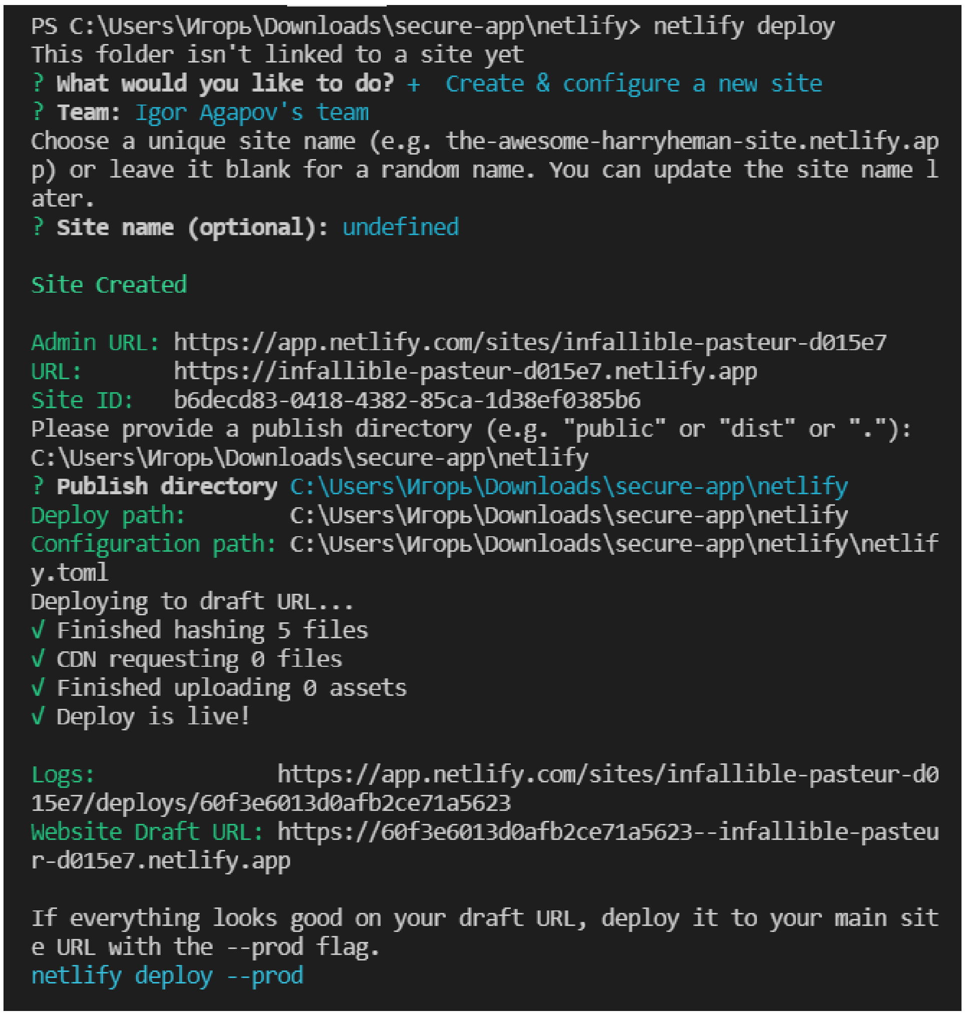
```
netlify dev
```

This command launches the application and opens the browser tab at .
Deploy the application in test
mode: `http://localhost:8888`

Code: Copy to clipboard

```
netlify deploy
```

Select Create & configure a new site, your team (for example, Igor Agapov's team), leave site name empty and select the directory with the application assembly (we do not have such a directory, so we leave the default value - .):



```
PS C:\Users\Игорь\Downloads\secure-app\netlify> netlify deploy
This folder isn't linked to a site yet
? What would you like to do? +  Create & configure a new site
? Team: Igor Agapov's team
Choose a unique site name (e.g. the-awesome-harryheman-site.netlify.ap
p) or leave it blank for a random name. You can update the site name l
ater.
? Site name (optional): undefined

Site Created

Admin URL: https://app.netlify.com/sites/infallible-pasteur-d015e7
URL:       https://infallible-pasteur-d015e7.netlify.app
Site ID:   b6decd83-0418-4382-85ca-1d38ef0385b6
Please provide a publish directory (e.g. "public" or "dist" or "."):
C:\Users\Игорь\Downloads\secure-app\netlify
? Publish directory C:\Users\Игорь\Downloads\secure-app\netlify
Deploy path:        C:\Users\Игорь\Downloads\secure-app\netlify
Configuration path: C:\Users\Игорь\Downloads\secure-app\netlify\netlif
y.toml
Deploying to draft URL...
√ Finished hashing 5 files
√ CDN requesting 0 files
√ Finished uploading 0 assets
√ Deploy is live!

Logs:               https://app.netlify.com/sites/infallible-pasteur-d0
15e7/deploys/60f3e6013d0afb2ce71a5623
Website Draft URL: https://60f3e6013d0afb2ce71a5623--infallible-pasteu
r-d015e7.netlify.app

If everything looks good on your draft URL, deploy it to your main sit
e URL with the --prod flag.
netlify deploy --prod
```

Get the WEBSITE Draft URL, for example, . You can go to the specified address and check the health of the application.
Deploy the application in production mode: `https://60f3e6013d0afb2ce71a5623--infallible-pasteur-d015e7.netlify.app`

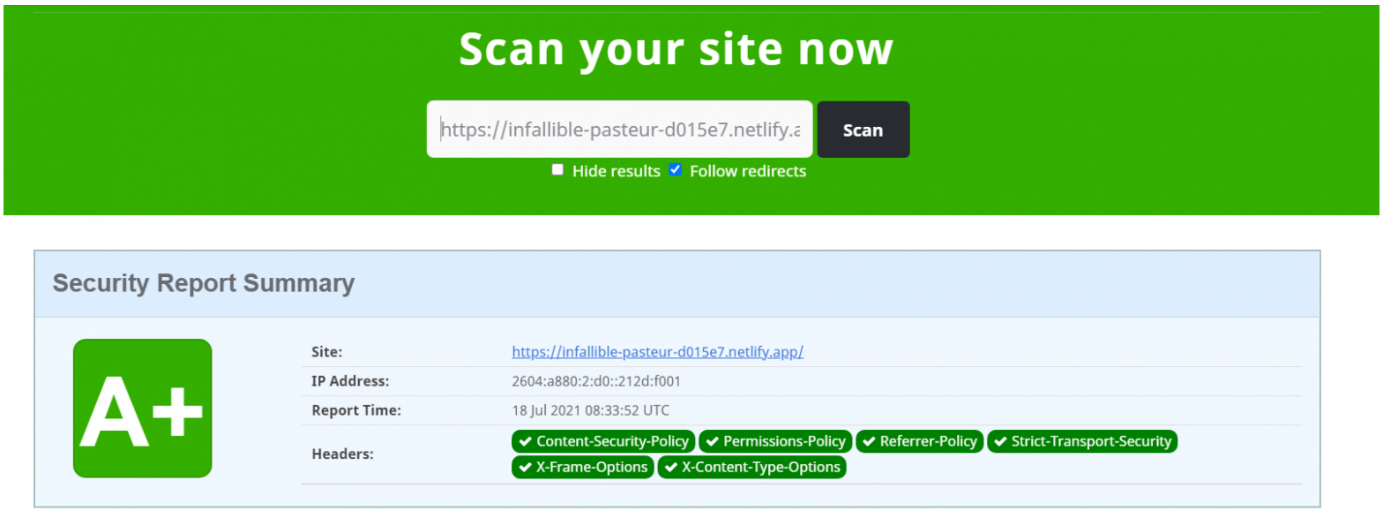
Code: Copy to clipboard

```
netlify deploy -p
```

- `-p` or means production mode. `--prod`

Get the URL of the application (Website URL), for example, . Again, you can go to the specified address and check the functionality of the application.
on how to deploy the application to Netlify can be found here.

Let's go back to Security Headers and WebPageTest and check how secure our Netlify application is: `https://infallible-pasteur-d015e7.netlify.app/`


Scan your site now

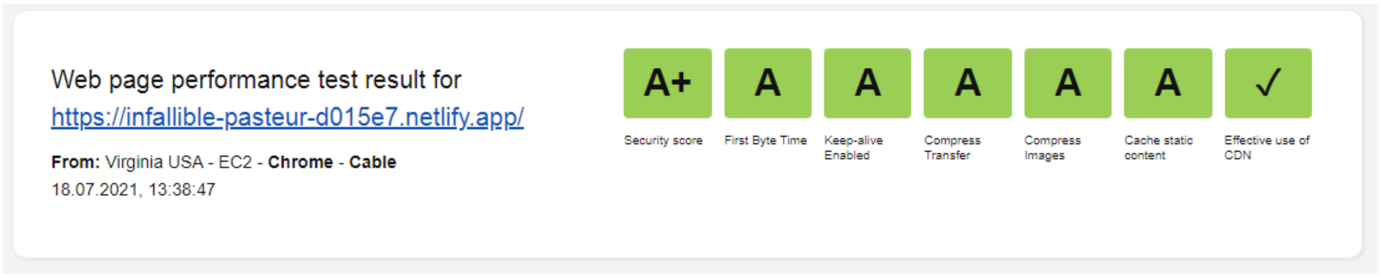
https://infallible-pasteur-d015e7.netlify.a Scan

Hide results ✓ Follow redirects

Security Report Summary

A+

Site:	https://infallible-pasteur-d015e7.netlify.app/
IP Address:	2604:a880:2:d0::212d:f001
Report Time:	18 Jul 2021 08:33:52 UTC
Headers:	✓ Content-Security-Policy ✓ Permissions-Policy ✓ Referrer-Policy ✓ Strict-Transport-Security ✓ X-Frame-Options ✓ X-Content-Type-Options


Web page performance test result for https://infallible-pasteur-d015e7.netlify.app/

From: Virginia USA - EC2 - **Chrome** - **Cable**
18.07.2021, 13:38:47

Security score	First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN
A+	A	A	A	A	A	✓

It seems that we succeeded!

Conclusion

Let's sum it up. We briefly studied all HTTP headers related to the security of web applications, developed server and serverless applications with similar functionality and received the best security ratings for these applications on Security Headers and WebPageTest. I think it's very good for one article. by Igor Agapov @aio350

][0-][0-][0!

Complaint Like + Quotation Answer

Vidmar and wan

Write an answer...