Security Headers

To improve the security of your application, you can use `headers` in `next.config.js` to apply HTTP response headers to all routes in your application.

```
// next.config.js

// You can choose which headers to add to the list
// after learning more below.
const securityHeaders = []

module.exports = {
  async headers() {
    return [
      {
        // Apply these headers to all routes in your application.
        source: '/:path*',
        headers: securityHeaders,
      },
    ]
  },
}
```

Options

X-DNS-Prefetch-Control

This header controls DNS prefetching, allowing browsers to proactively perform domain name resolution on external links, images, CSS, JavaScript, and more. This prefetching is performed in the background, so the DNS is more likely to be resolved by the time the referenced items are needed. This reduces latency when the user clicks a link.

```
{
  key: 'X-DNS-Prefetch-Control',
  value: 'on'
}
```

Strict-Transport-Security

This header informs browsers it should only be accessed using HTTPS, instead of using HTTP. Using the configuration below, all present and future subdomains will use HTTPS for a `max-age` of 2 years. This blocks access to pages or subdomains that can only be served over HTTP.

If you're deploying to Vercel, this header is not necessary as it's automatically added to all deployments unless you declare `headers` in your `next.config.js`.

```
{
  key: 'Strict-Transport-Security',
  value: 'max-age=63072000; includeSubDomains; preload'
}
```

X-XSS-Protection

This header stops pages from loading when they detect reflected cross-site scripting (XSS) attacks. Although this protection is not necessary when sites implement a strong `Content-Security-Policy` disabling the use of inline JavaScript (`'unsafe-inline'`), it can still provide protection for older web browsers that don't support CSP.

```
{
  key: 'X-XSS-Protection',
  value: '1; mode=block'
}
```

X-Frame-Options

This header indicates whether the site should be allowed to be displayed within an `iframe`. This can prevent against clickjacking attacks. This header has been superseded by CSP's `frame-ancestors` option, which has better support in modern browsers.

```
{
  key: 'X-Frame-Options',
  value: 'SAMEORIGIN'
}
```

Permissions-Policy

This header allows you to control which features and APIs can be used in the browser. It was previously named `Feature-Policy`. You can view the full list of permission options here.

```
{
  key: 'Permissions-Policy',
  value: 'camera=(), microphone=(), geolocation=(), interest-cohort=()'
}
```

X-Content-Type-Options

This header prevents the browser from attempting to guess the type of content if the `Content-Type` header is not explicitly set. This can prevent XSS exploits for websites that allow users to upload and share files. For example, a user trying to download an image, but having it treated as a different `Content-Type` like an

executable, which could be malicious. This header also applies to downloading browser extensions. The only valid value for this header is `nosniff`.

```
{
  key: 'X-Content-Type-Options',
  value: 'nosniff'
}
```

Referrer-Policy

This header controls how much information the browser includes when navigating from the current website (origin) to another. You can read about the different options here.

```
{
  key: 'Referrer-Policy',
  value: 'origin-when-cross-origin'
}
```

Content-Security-Policy

This header helps prevent cross-site scripting (XSS), clickjacking and other code injection attacks. Content Security Policy (CSP) can specify allowed origins for content including scripts, stylesheets, images, fonts, objects, media (audio, video), iframes, and more.

You can read about the many different CSP options here.

You can add Content Security Policy directives using a template string.

```
// Before defining your Security Headers
// add Content Security Policy directives using a template string.

const ContentSecurityPolicy = `
  default-src 'self';
  script-src 'self';
  child-src example.com;
  style-src 'self' example.com;
  font-src 'self';
`
```

When a directive uses a keyword such as `self`, wrap it in single quotes `''`.

In the header's value, replace the new line with an empty string.

```
{
  key: 'Content-Security-Policy',
  value: ContentSecurityPolicy.replace(/\s{2,}/g, ' ').trim()
}
```

References

- MDN
- Varun Naik
- Scott Helme
- Mozilla Observatory

Related

For more information, we recommend the following sections:

Headers: Add custom HTTP headers to your Next.js app.