X-Permitted-Cross-Domain-Policies & Referrer-Policy

In this lesson, we'll study a couple of headers.

We'll cover the following

Related to CORS, the `X-Permitted-Cross-Domain-Policies` targets cross-domain policies for Adobe products, namely Flash and Acrobat.

I won't go too much into the details, as this is a header that targets very specific use cases, but, long story short, Adobe products handle cross-domain request through a `crossdomain.xml` file in the root of the domain the request is targeting. The `X-Permitted-Cross-Domain-Policies` defines policies to access this file.

Sounds complicated? I would simply suggest adding an `X-Permitted-Cross-Domain-Policies: none` and ignore clients wanting to make cross-domain requests with Flash.

In 2017, Adobe announced it would discontinue support for Flash, meaning you most likely won't have to deal with this technology in the future. Disabling any flash-related browser-feature is a safe and reasonable choice years into Adobe's original announcement (https://theblog.adobe.com/adobe-flash-update/).

Referrer-Policy

At the beginning of our careers, we all probably made the same mistake, using the `Referer` header to implement a security restriction on our website. If the header contains a specific URL in a whitelist we define, we're going to let users through.

Ok, maybe that wasn't every one of us, but I damn sure made the mistake of trusting the `Referer` header to give us reliable information on the origin the user comes from! The header was useful until we figured that sending this information to sites could pose a potential threat to our users' privacy because it would specify which domains the users came from.

Born at the beginning of 2017 and currently supported by all major browsers, the `Referrer-Policy` header can be used to mitigate these privacy concerns by telling the browser that it should only mask the URL in the `Referrer` header or omit it altogether.

Some of the most common values the `Referrer-Policy` can take are:

- `no-referrer`: the `Referrer` header will be entirely omitted
- `origin`: turns `https://example.com/private-page` to `https://example.com/`
- `same-origin`: send the `Referrer` to same-site origins but omit it for anyone else

It's worth noting that there are many variations of the `Referrer-Policy` (`strict-origin`, `no-referrer-when-downgrade`, etc.) but the ones I mentioned above are going to cover most of your use cases. If you wish to better understand each and every variation you can use, I would recommend heading to the OWASP dedicated page.

> ### ℹ Origin and Referrer
>
> The `Origin` header is very similar to `Referrer`, as it's sent by the browser in cross-domain requests to make sure the caller is allowed to access a resource on a different domain. The `Origin` header is controlled by the browser, so there's no way malicious users can tamper with it. You might be tempted to use it as a firewall for your web application. If the `Origin` is in our whitelist, it lets the request go through.
>
> One thing to consider is that other HTTP clients such as cURL can present their own origin. A simple `curl -H "Origin: example.com" api.example.com` will render all origin-based firewall rules inefficient. That is why you cannot rely on the `Origin` (or the `Referrer`, as we've just seen) to build a firewall to keep malicious clients away.

In the next lesson, we'll study the reporting API.