CROSS-ORIGIN RESOURCE SHARING (CORS)

What is CORS?

Firstly, CORS stands for "Cross-Origin Resource Sharing". It allows you to make requests from one website to another website in the browser, which is normally prohibited by another browser policy called the Same-Origin Policy (SOP).

CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

When we talk about cross origin requests, we're usually talking about requests from one domain or subdomain to a different domain or subdomain. But different protocols (for example http vs https) or different ports can also constitute different origins.

To understand about CORS we need to know about some terms:

SOP: The same-origin policy is a restrictive cross-origin specification that limits the ability for a website to interact with resources outside of the source domain. The same-origin policy was defined many years ago in response to potentially malicious cross-domain interactions, such as one website stealing

private data from another. It generally allows a domain to issue requests to other domains, but not to access the responses.

Origins: Origins can be expressed as a combination of host name, port number and uniform resource Identifier schemes.

Types of CORS misconfigurations:

There are two main types of CORS misconfigurations that can render a web server vulnerable to CORS.

• Access-Control-Allow-Origin (ACAO): This allows for two-way communication with third-party websites. A misconfiguration of the Access-Control-Allow-Origin (ACAO) can be exploited to modify or funnel sensitive data, such as usernames and passwords.

• Access-Control-Allow-Credentials (ACAC): This allows third-party websites to execute privileged actions that only the genuine authenticated user should be able to perform. Examples would be changing your password or your contact information.

Why is CORS implemented?

The CORS protocol is enforced only by the browsers. The browser does this by sending a set of CORS headers to the cross-origin server which returns specific header values in the response. Based on the header values returned in the response from the cross-origin server, the browser provides access to the response or blocks the access by showing a CORS error in the browser console.

CORS was implemented due to the restrictions revolving around the same-origin policy (SOP). This policy limited certain resources to interact only with resources from the parent domain. This came with good reason as AJAX requests can perform advanced requests such as **POST, PUT, DELETE**, etc.

which could put a website's security at risk. Therefore, the same-origin policy increased web security and helped prevent user abuse.

However, in some cases, it is quite beneficial to enable Cross Origin Resource Sharing as it allows for additional freedom and functionality for websites. This is true in many cases these days for web fonts and icons which are often requested from another domain. In this case, with the use of HTTP headers, CORS enables the browser to manage cross-domain content by either allowing or denying it based on the configured security settings.

Most of the time, a script running in the user's browser would only ever need to access resources on the same origin (think about API calls to the same backend that served the JavaScript code in the first place). So the fact that JavaScript can't normally access resources on other origins is a good thing for security. In this context, "other origins" means the URL being accessed differs from the location that the JavaScript is running from, by having: a different scheme (HTTP or HTTPS) a different domain a different port However, there are legitimate scenarios where cross-origin access is desirable or even necessary.

How can we prevent CORS-based attacks?

CORS vulnerabilities are mainly from the misconfiguration. Prevention is therefore a configuration problem. There are some effective defences against CORS attacks:

1. Specify the allowed origins: If a web resource contains sensitive information, the allowed origin(s) should be specified in full in the Access-Control-Allow-Origin header (i.e., no wildcards).

2. Only allow trusted sites: While this one may seem obvious, especially given the previous tip, but origins specified in the Access-Control Allow-Origin header should exclusively be trusted sites. What I mean to convey that you should avoid dynamically reflecting origins from cross-domain request headers without validation unless the website is a public site that doesn’t require any kind of authentication for access, such as an API endpoint.

3. Don’t whitelist “null”: You should avoid using the header Access-Control Allow-Origin: null. While cross-domain resource calls from internal documents and sandboxed requests can specify the “null” origin, you should treat internal cross- origin requests in the same way as external cross-origin requests. You should properly define your CORS headers.

4. Implement proper server-side security policies: Don’t think that properly configuring your CORS headers is enough to secure your web server. It’s one of the pieces, but it isn’t comprehensive. CORS defines browser behaviours and is never a replacement for server-side protection of sensitive data. You should continue protecting sensitive data, such as authentication and session management, in addition to properly configured CORS.

Some of the best practices are:

• Use a firewall

• Only buy well-reviewed and genuine antivirus software.

• Never click on pop-ups.

• If your browser displays a warning about a website you are trying to access, you should pay attention and get the information you need elsewhere.

• Don’t open attachments in emails.

• Don’t click links (URLs) in emails.

References

https://portswigger.net/web-security/cors

https://www.keycdn.com/support/cors

https://medium.com/@electra_chong/cors

https://www.comparitech.com/blogs/cors