Research Report on CORS

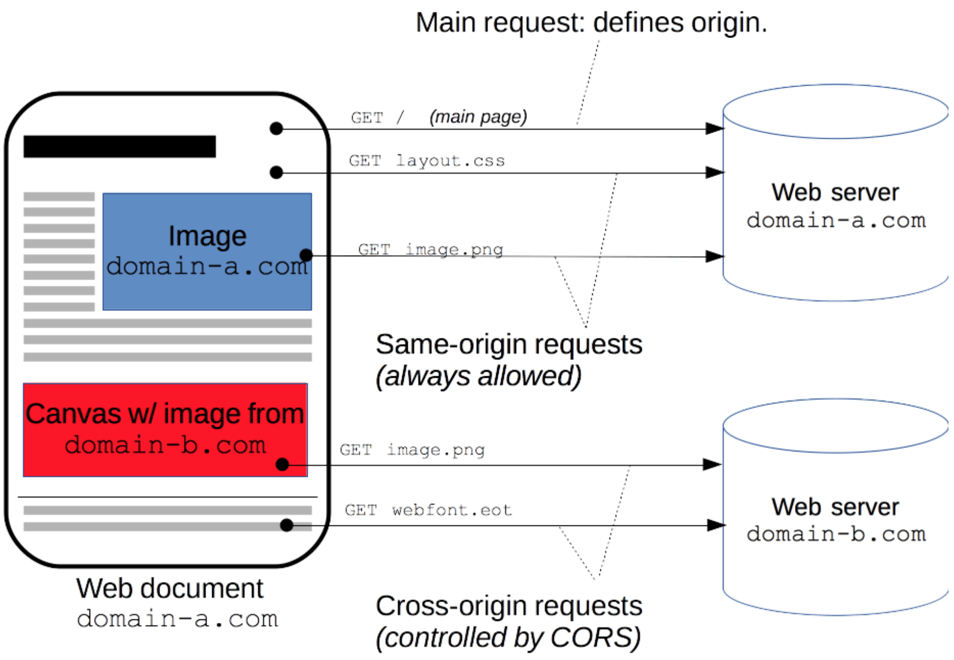
What is CORS/ Cross-Origin Resource Sharing?

“CORS” stands for Cross-Origin Resource Sharing. CORS is a protocol and security standard for browsers that helps to maintain the integrity of a website and secure it from unauthorized access.

It enables JavaScripts running in browsers to connect to APIs and other web resources like fonts, and stylesheets from multiple different providers.

An example of a cross-origin request: the front-end JavaScript code served from https://domain-a.com uses XMLHttpRequest to make a request for https://domain-b.com/data.json.

CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order 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.



Why is CORS implemented?

CORS was implemented due to the restrictions revolving around the same-origin policy. This policy limited certain resources to interact only with resources from the parent domain. This came with good reason as AJAX requests are able to 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.

Here’s what a CORS attack could look like:

1. The victim visits evilwebsite.com while being authenticated to goodwebsite.com.
2. evilwebsite.com dumps a malicious script designed to interact with goodwebsite.com, on the victim’s machine.
3. The victim unwittingly executes the malicious script, and the script issues a cross-origin request to goodwebsite.com. In this example, let’s assume the request is crafted to obtain the credentials necessary to perform a privileged action, such as revealing the user’s password.

4. goodwebsite.com receives the victim’s cross-origin request and the CORS header.
5. The web server will check the CORS header to determine whether or not to send the data to goodwebsite.com. In this example, we’re assuming that CORS is allowed with authentication (Access-Control-Allow-Credentials: true).
6. The request is validated, and the data is sent from the victim’s browser to evilwebsite.com.

How to prevent CORS-based attacks

It’s primarily web server misconfigurations that enable CORS vulnerabilities. The solution is to prevent the vulnerabilities from arising in the first place by properly configuring your web server’s CORS policies. Here are a few simple tips on preventing 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.

As a user, you basically want to be one step ahead of phishing scams and malicious websites and downloads to minimize your chances of falling victim to a CORS attack. The following common-sense tips can help.

These steps are similar for many online attacks such as avoiding fake antivirus so they are generally good practices to follow.

- Use a firewall – All major operating systems have a built-in incoming firewall, and all commercial routers on the market have a built-in NAT firewall. Make sure you enable these as they may protect you in the event that you click a malicious link.
- Only buy well-reviewed and genuine antivirus software from legitimate vendors and configure it to run frequent scans at regular intervals.
- Never click on pop-ups. You never know where they’ll take you next.
- 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 unless you know exactly who sent the attachment and what it is.
- Don’t click links (URLs) in emails unless you know exactly who sent the URL and where it links to. And even then, inspect the link carefully. Is it an HTTP or an HTTPS link? Most legitimate sites use HTTPS today. Does the link contain spelling errors (faceboook instead of facebook)? If you can get to the destination without using the link, do that instead.

References

https://www.comparitech.com/

https://www.keycdn.com/

https://www.tenable.com/