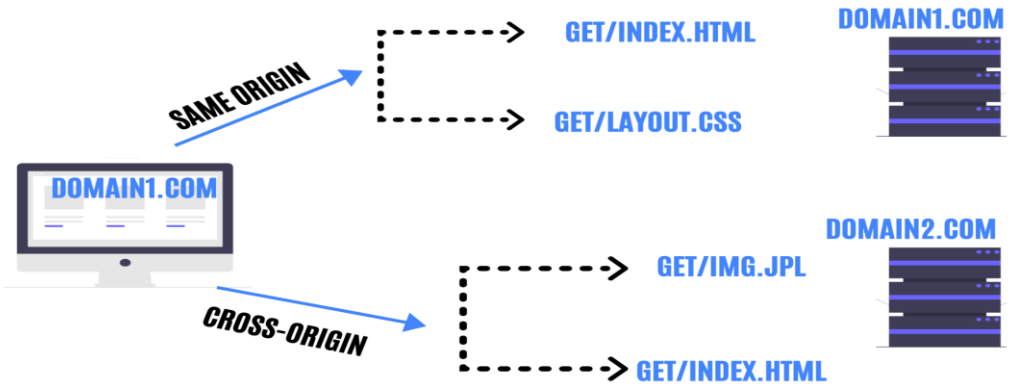
CORS

(Cross-Origin Resource Sharing)

What is CORS?

CORS is a standard for sharing cross-origin resources. It is an HTTP-header based mechanism that allows a server to initiate an origin other than its own from which a browser should permit loading resources. CORS adds new HTTP headers to the list of standard headers. The new CORS headers allow the local server to keep a list of allowed origins.



Types of CORS Requests:

Most requests fall into two major categories:

- **Simple Requests:** These requests do not trigger a preflight check and only "safe listed" CORS headers.
- **Preflight Requests:** These requests send a "preflight" message that outlines what the requester would like to do before the original request. The requested server reviews this preflight message to ensure the request is safe to allow.

Simple CORS example

Here is a simple CORS example of when a browser requests a resource from another domain. Let's say domainx.com makes a request to domainy.com for a particular resource. CORS uses HTTP headers to determine whether or not domainx.com should have access to that resource. The browser automatically sends a request header to domainy.com with

Origin: http://domainx.com
domainy.com receives that request and will respond back with either:

1. Access-Control-Allow-Origin: http://domainx.com
2. Access-Control-Allow-Origin: * (meaning all domains are allowed)
3. An error if the cross-origin requests are not allowed

Preflight CORS example

When certain, more complicated, types of requests are performed, the browser will insert additional preflight requests to validate whether they have the appropriate permissions to perform the action. A request is preflighted if any of the following conditions are met:

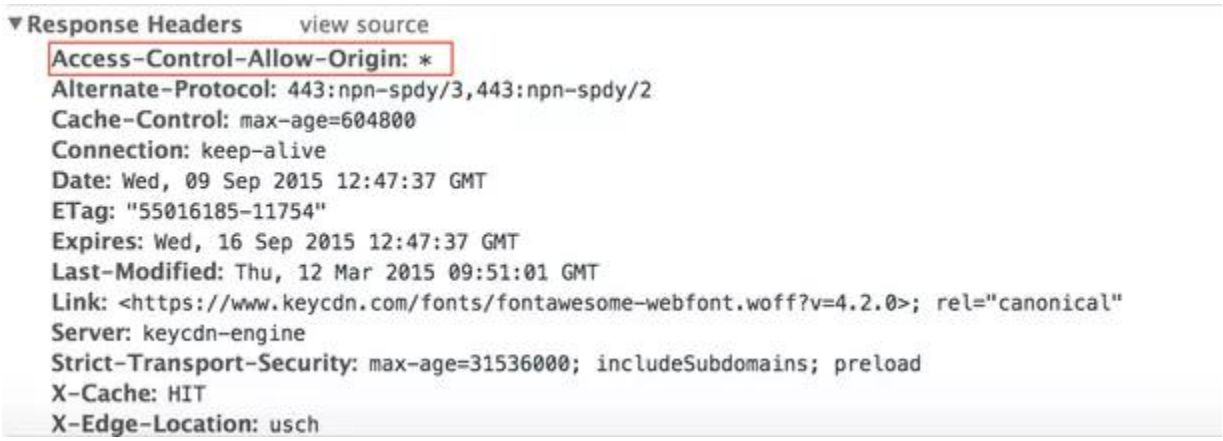
1. It uses an HTTP method other than GET or POST
2. Custom headers are set
3. The request body has a MIME type other than text/plain

Here is an example of a preflight request:
Origin: http://domainx.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-Custom-Header
If domainy.com is willing to accept the action, it may respond with the following headers:
Access-Control-Allow-Origin: http://domainx.com
Access-Control-Allow-Methods: GET, POST
Access-Control-Allow-Headers: X-Custom-Header

A real-world example of how CORS works

CORS works by having the origin domain send HTTP request headers to the host domain that is hosting the resource. The example below shows that https://www.keycdn.com is the origin domain that is requesting a resource from the Host: example.keycdn.com.

Once example.keycdn.com receives the request, it responds by either allowing or denying the origin domain access to the requested resources based on the CORS settings configured. In the example below, it shows that the host responded with the response header of Access-Control-Allow-Origin: *. The * means all domains are allowed to access this resource.



Why use CORS?

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.

HTTP request headers

When a domain is requesting to interact with a resource on another domain, request headers are added from the first domain in order to use the Cross-Origin Resource Sharing feature. These are the HTTP request headers that may be associated with the requesting domain.

- Origin
- Access-Control-Request-Method
- Access-Control-Request-Headers

HTTP response headers

The domain who's resources are being requested can respond to the first domain with the following HTTP response headers based on what configuration options are set.

- Access-Control-Allow-Origin
- Access-Control-Allow-Credentials
- Access-Control-Expose-Headers
- Access-Control-Max-Age
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers

Types of CORS Misconfigurations

CORS contains two main components that when misconfigured can pose a significant risk to any web application:

- **Access-Control-Allow-Origin (ACAO)**: This allows for two-way communication with third-party websites. To modify sensitive data such as usernames or 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. Changing your password or your contact information.

How to find CORS

Request :- Origin: attacker.com	Response :- Access-control-allow-origin: attacker.com Access-control-allow-credentials: true
Request :- Origin: attacker.com	Response :- Access-control-allow-origin: null Access-control-allow-credentials: true
Request :- Origin: attacker.com	Response :- Access-control-allow-origin: * Access-control-allow-credentials: true

Exploiting misconfigured wildcard (*) in CORS Headers

- Consider the below request:

  GET /api/userinfo.php
  Host: www.victim.com
  Origin: www.victim.com

- When you send the above request, you get a response with the Access-Control-Allow-Origin header setting. See the below response code

  HTTP/1.0 200 OK
  Access-Control-Allow-Origin: *
  Access-Control-Allow-Credentials: true

The header is configured with a wildcard (*). It means any domain can access the resources.

- We can modify the REQUEST Origin from victim domain to attacker domain.

```
GET /api/v1/logs HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:58.0) Gecko/20100101 Firefox/58.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Origin: https://testing.aaa.com
Content-Length: 2
```

  Here, https://testing.aaa.com is our attacker's domain

- Below is the response we received. The wildcard is shown in the origin header response with Access-Control-Allow-Origin: *, which means the victim domain allows access to resources from all sites. *Testing.aaa.com* site in our attack case.

```
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Wed, 21 Feb 2018 18:04:56 GMT
Content-Type: application/octet-stream
Content-Length: 101
Connection: keep-alive
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Set-Cookie: io=qG3WyOVLmaWstuLPG2AF
```

This Kind of information you can gather with this attack

How to prevent it?

- Specify the allowed origins
- Only allow trusted sites
- Don't whitelist "null"
- Implement proper server-side security policies
- Enable SOP (Same origin policy)
- Using a firewall

CORS Resources and Cheat Sheets:

https://0xn3va.gitbook.io/cheat-sheets/web-application/cors-misconfiguration
https://github.com/EdOverflow/bugbounty-cheatsheet/blob/master/cheatsheets/cors.md
https://portswigger.net/web-security/cors
https://medium.com/@amangupta566/cors-misconfiguration-leads-to-steal-sensitive-information-disclosure-fdf050b68b66
https://0xn3va.gitbook.io/cheat-sheets/web-application/cors-misconfiguration
https://github.com/carlospolop/hacktricks/blob/master/pentesting-web/cors-bypass.md