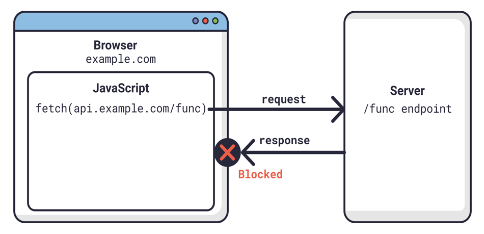
31-10-2022

Muijsenberg, Reno R.F.

FONTYS

CORS

Research



Inhoud

[Aanleiding 2](#_Toc118876796)

[Hoofdvraag 3](#_Toc118876797)

[Deelvragen 3](#_Toc118876798)

[Wat is CORS 4](#_Toc118876799)

[Welke request gebruiken CORS 4](#_Toc118876800)

[Wat is de functionele werking van CORS 4](#_Toc118876801)

[Wat zijn onveiligheden afkomstig van CORS 4](#_Toc118876802)

[Origin Reflection 4](#_Toc118876803)

[Wilcard Origin 5](#_Toc118876804)

[Null Origin 5](#_Toc118876805)

[Expanding Origin 5](#_Toc118876806)

[Conclusie 6](#_Toc118876807)

[Bronnen 7](#_Toc118876808)

# Aanleiding

Dit document bevat een onderzoek over security omdat wij hier een onderzoek moesten schrijven. Ik heb gekozen voor het stukje CORS omdat ik vaak heb wanneer ik aan het programmeren ben dat ik een error hierover krijg. Ik ga daarom dus onderzoeken wat het precies inhoudt en wat ik eraan kan doen.

# Hoofdvraag

* Hoe stel ik CORS correct in in mijn project

# Deelvragen

* Wat is CORS
* Welke request gebruiken CORS
* Wat is de functionele werking van CORS
* Wat zijn onveiligheden afkomstig van CORS

# Wat is CORS

Cors is een HTTP-header gebaseerd mechanisme dat servers toelaat om te specificeren welke afkomsten behalve zichzelf worden toegelaten om resources te laden. CORS is ook afhankelijk van het preflight mechanisme waarbij de browser een request maakt om te checken of de server het toelaat om resources op te halen met het echte request, de browser stuurt een header die aangeeft welke HTTP method het is en de header die in het echte request zit.

Een voorbeeld van CORS is: de front-end JavaScript code die staat op https://domein-a.com gebruikt XMLHttpRequest in een request te maken voor https://domein-b.com/data.json.

Voor veiligheidsredenen beperkt een browser cross-origin HTTP request die vanuit scripts komen. Als voorbeeld: XMLHttpRequest en de Fetch API volgen het same-origin beleid. Dit betekent dat een webapplicatie die gebruikt maakt van deze API’s alleen resources kunnen ophalen op de origin waar ook de applicatie op geladen is tenzij het request de correct HTTP-headers heeft.

# Welke request gebruiken CORS

* Aanroepen van de XMLHttpRequest- of Fetch-API's
* Web Fonts (voor cross-domain font-gebruik in @font-face binnen CSS)
* [WebGL textures](https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Using_textures_in_WebGL)
* Afbeeldingen/videoframes getekend op een canvas met drawImage()
* CSS-vormen van afbeeldingen.

# Wat is de functionele werking van CORS

De cross-origin resource sharing standard werkt door het toe te voegen van een nieuwe HTTP-header die de server uitlegt welke afkomsten toegestaan zijn informatie op te vragen.

CORS-errors resulteren in errors, maar om veiligheidsredenen zijn details over de fout niet beschikbaar voor JavaScript. De code weet alleen dat er een fout is opgetreden.

Wanneer de CORS-policy slecht geconfigureerd is kan dit leiden tot cross-domain aanvallen.

# Wat zijn onveiligheden afkomstig van CORS

Veel moderne browsers gebruiken CORS om toegang te verlenen aan sub domeinen en vertrouwde third parties. Hun implementatie is vaak fout of veel te soepel om ervoor te zorgen dat alles werkt en dit kan leiden tot exploiteerbare kwetsbaarheden.

## Origin Reflection

Wanneer je bijvoorbeeld van `cross-origin.com` content wilt ophalen voor `initial-origin.com` dan moet je dit instellen in de configuratie van CORS op `initial-origin.com` door middel van de `Acces-Control-Allow-Origin` header als volg: `Access-Control-Allow-Origin: cross-origin.com`.

Maar waanneer je een ander domain hebt dat ook bij deze resource moet komen kun je niet een lijst met statische domainen aan CORS geven. Je kunt dit oplossen met het gebruikt van de wildcard `\*` of door de header dynamisch te laten generen. De headers dynamisch genereren kan wanneer niet correct gedaan leiden tot onveiligheden. Veel servers lezen de origin van het request en zetten dit domein vervolgens als de header maar dit zorgt ook dat dit gebeurt bij gevaarlijke domeinen wanneer hun een request maken. Je kunt makkelijk checken of dit het geval is door met een custom domein een request te maken en te kijken of de server aan dit domein een antwoord teruggeeft.

Dit heet `Origin Reflection` omdat de server het request reflecteert in de `Acces-Control-Allow-Origin` header.

Een groter probleem bij deze aanpak is dat wanneer de ` Access-Control-Allow-Credentials` header op true staat een gevaarlijk domein cookies kan sturen naar een de server en bijvoorbeeld een private key kan ophalen waardoor er dus sensitieve informatie gelekt kan worden. Dit kan makkelijk met JavaScript worden gedaan.

## Wilcard Origin

Een oplossing om het probleem van de lijst met origins op te lossen is door het gebruikt van een wildcard. Hiermee geef je toegang tot elk domein en dus ook onveilige domeinen. De wildcard wordt gespecificeerd als: `Access-Control-Allow-Origin: \*`. Dit kan problematisch zijn wanneer de data op de server geheim moet blijven maar deze aanpak is minder serieus dan de `Origin Reflection` maar bij deze oplossing wordt er nooit de `Access-Control-Allow-Credentials` meegestuurd. Wanneer de server geen authenticatie heeft kan er op deze manier nog altijd sensitieve data worden gelekt.

## Null Origin

Sommige domeinen laten het toe om een speciale origin toe toelaten genaamd de Null Origin ` Access-Control-Allow-Origin: null `. Dit is problematisch omdat dit toegang geeft tot alles op de server met credentials. Je kan deze onveiligheid zien in elke response waarin de `Access-Control-Allow-Credentials` zich bevindt.

Hoe kan de null origin gegenereerd worden? Het is duidelijk dat wanneer je een request stuurt vanaf een bepaald domein dat dit domein als de header komt te staan. Een aantal stackoverflows laten zien dar een lokaal HMTL-bestand de `Null Origin` zet. Dit is handig voor aanvallers hun kunnen makkelijk de null header krijgen door middel van een iframe waarmee dus bijvoorbeeld een private key kan worden opgehaald.

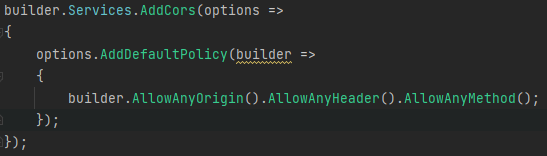
## Expanding Origin

Soms zou je ook zien dat de header gezet wordt door middel van een regex expressie bijvoorbeeld alles wat eindigt met `secure-bank.com` zoals `api.secure-bank.com` en andere sub domeinen. Maar met deze manier kan het ook zo zijn dat `not-so-sercure-bank.com` aan de data van de server kan.

Probeer dus altijd het gebruik van dynamisch gegenereerde headers te voorkomen. Wanneer dit niet mogelijk is test dan zo goed mogelijk of het werkt en of de sensitieve data niet gelekt kan worden.

# Hoe stel ik CORS correct in in mijn project

Toen ik ben begonnen met het ontwikkelen van mijn applicatie had ik al snel het probleem dat ik tegen een CORS-error aanliep terwijl ik een request maakte naar mijn API. Dit komt omdat ik vanuit een javascript applicatie (Angular) naar mijn backend een request wil maken. Omdat dit op te lossen tijdens development heb ik gebruikt gemaakt van de ingebouwde functie van ASP.net namelijk de AllowAnyOrigins() die dus gelijk staat aan de “\*” wildcard.



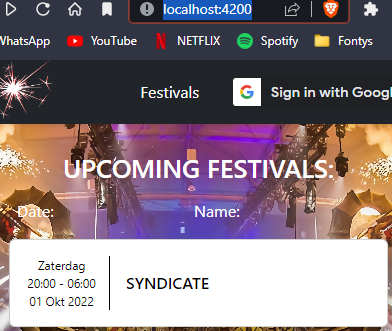
Zoals hierboven gelezen is dit een van de veiligere manieren om alles toe te laten maar nog steeds niet veilig genoeg. Aangezien ik alleen calls maak vanuit een afkomst naar de API kan ik dus makkelijk de specifieke afkomst en de specifieke methode opgeven. Dit zou dan ook zorgen voor het veiligste manier om CORS te gebruiken.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Afbeelding met tekst, zwart, scorebord, schermafbeelding

Automatisch gegenereerde beschrijvingNa het zetten van de CORS heb ik getest of mijn website nog calls maakte naar de API en ik heb getest of andere domeinen nog calls konden maken. In het eerstvolgende screenshot is te zien dat er nog data te zien is op we website die ik gespecificeerd heb en het screen erna kun je zien dat de data niet geladen is en dus ook geblokkeerd.



# Conclusie

In dit onderzoek is gezocht naar het antwoord op de vraag “Hoe stel ik CORS correct in in mijn project”. Hiervoor heb ik onderzocht wat CORS nu eigenlijk is, hoe het precies werkt, welke request CORS gebruiken en wat de nadelen zijn van methodes om CORS in te stellen. Dit heb ik onderzocht door middel van het DOT framework.

Zo is gebleken dat CORS een HTTP-header is die ingesteld kan worden voor welke afkomsten een resource mogen opvragen. Soms gebruiken CORS een pre-flight mechanisme dat van tevoren gaat checken of het request wat de gebruiker/app wil maken ook daadwerkelijk mag op basis van de CORS-instellingen. Met CORS kun je instellen welke afkomsten en HTTP-methode er gemaakt mogen worden naar je backend.

CORS wordt als meeste voorkomende gebruikt bij Fetch API’s en XMLHttpRequest. Verder wordt dit nog gebruikt bij Web Fonts, WebGL textures, afbeeldingen en video frames gemaakt met de drawImage(), en CSS vormen en afbeeldingen.

Er zijn dus een aantal verschillende manieren hoe je CORS kunt instellen in jouw project. Alleen neemt elk van deze problemen onveiligheden met zich mee. Zo kun je instellen dat alle afkomsten je website mogen bereiken maar dit is door logische redenen een slecht idee. De beste manier is dus om zo specifiek mogelijk te zijn met het aangeven van CORS. Wanneer mogelijk stel dan ook maar een afkomst in en zorg dat je alleen de methode toelaat die ook echt gebruikt worden. Wanneer dit niet kan zorg dan dat je zo goed mogelijk je backend beveiligd met een andere beveiliging.

# Bronnen

<https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

<https://www.securityandit.com/security/understanding-cross-domain-attack/>

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

https://medium.com/swlh/exploiting-cors-misconfiguration-vulnerabilities-2a16b5b979#:~:text=All%20CORS%20vulnerabilities%20come%20from,be%20sent%20from%20untrusted%20sources.