Engineer

CORS

hur en server låter sidor från andra ursprung läsa dess svar

Vad det är

CORS, utskrivet Cross-Origin Resource Sharing, är de regler som en server använder för att tala om för webbläsaren att sidor från andra webbplatser får läsa det den skickar tillbaka. Webbläsare håller isär ursprung. Ett ursprung är schema, värd och port tillsammans, så `https://app.example.com`, `https://api.example.com` och `http://app.example.com` är tre olika ursprung, och ett skript på det ena får inte läsa ett svar från ett annat om inte det andra går med på det. CORS är sättet att gå med på det: en liten familj HTTP-headrar, nästan alla med namn som börjar på `Access-Control-`, som servern sätter på sina svar och som webbläsaren kontrollerar innan något alls lämnas vidare till skriptet som frågade.

Protokollet beskrivs i Fetch-standarden från WHATWG, den levande specifikationen av hur webbläsare hämtar resurser. Där står också skälet, och det är enkelt. En sida ska kunna använda data från ett annat ursprung och skicka rikare anrop än ett HTML-formulär klarar, men bara när motparten själv har sagt ja. Utan det jaet skulle vilken sida som helst på nätet kunna läsa servrar som står bakom ett företags brandvägg, eller läsa en användares privata data på en annan webbplats med användarens kakor på köpet.

Själva utbytet är kort. När ett skript ber om en resurs från ett annat ursprung lägger webbläsaren till en `Origin`-header med sidans eget ursprung. Servern svarar med `Access-Control-Allow-Origin`, och värdet är antingen samma ursprung, exakt skrivet, eller `*` för alla. Stämmer det får skriptet svaret. Stämmer det inte får skriptet ett nätverksfel och ingenting av innehållet.

Vissa anrop behöver lov innan de ens skickas. En `GET`, `HEAD` eller `POST` med några få vanliga headrar och en kropp av typen `application/x-www-form-urlencoded`, `multipart/form-data` eller `text/plain` går direkt iväg, eftersom ett HTML-

formulär alltid har kunnat skicka sådant från vilken webbplats som helst. Allt utöver det, till exempel en `PUT`, en `DELETE`, en JSON-kropp eller en `Authorization`-header, föregås av ett förhandsanrop, som alla kallar preflight: ett `OPTIONS`-anrop som namnger metoden i `Access-Control-Request-Method` och de extra headrarna i `Access-Control-Request-Headers`. Det riktiga anropet följer bara om svaret har en lyckad status som 200 eller 204 och tillåter båda delarna.

Använd begreppet i ett beslut

Använd ”Vad är CORS?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Konton, lösenord och behörigheter ordnade så att alla har exakt den åtkomst deras arbete kräver, och de som slutar förlorar den samma dag.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

Nästan varje webbapplikation med ett separat API stöter på CORS första dagen. Frontenden serveras från ett ursprung och API:et från ett annat, och det första `fetch`-anropet från det ena till det andra misslyckas i webbläsarens konsol tills API:et skickar rätt headrar. Team som lär känna CORS på det sättet minns det gärna som ett hinder och konfigurerar det därefter, genom att tillåta det som får felet att försvinna. Det är där risken finns.

Headrarna avgör vem som kan läsa dina användares data genom deras egna webbläsare. En användare som är inloggad i din applikation bär på en sessionskaka. Om API:et låter vilket ursprung som helst skicka anrop med kakor och läsa svaren kan varje sida som användaren besöker anropa API:et i användarens namn och läsa resultatet. Fetch-standarden säger det rakt ut: att dela svar som hämtats med användarens inloggningsuppgifter kräver största försiktighet för att undvika det den kallar confused deputy-problemet. Webbläsaren, som bär användarens kakor, är ställföreträdaren som blir lurad, och en slarvig CORS-inställning låter en främmande sida styra den.

Det är lika viktigt att veta vad CORS inte gör. Det skyddar inte servern. Ett enkelt anrop når fram till servern och körs oavsett om svaret sedan får delas, så en `POST` som ändrar data har ändrat dem innan webbläsaren har tittat på en enda CORS-header. Utanför en webbläsare innehåller `Origin`-headern vad anroparen än skriver, och OWASP:s testguide varnar för att den kan förfalskas. Vem som får

anropa en slutpunkt, och vad de får göra där, avgörs av autentisering och auktorisering på servern, varje gång. CORS avgör bara vilka webbsidor som får läsa svaret inne i en webbläsare.

Så ser det ut i praktiken

Tänk dig en ensidesapplikation på `https://app.example.com` som sparar en anteckning via ett API på `https://api.example.com` och använder den sessionskaka som API:et satte vid inloggningen:

```
await fetch("https://api.example.com/v1/notes/42", {
  method: "PUT",
  credentials: "include",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ text: "Släpp på fredag" }),
});
```

En `PUT` med en JSON-kropp är mer än ett formulär kan skicka, så webbläsaren frågar först. Förhandsanropet har inga kakor med sig; det namnger bara vad det riktiga anropet kommer att använda:

```
OPTIONS /v1/notes/42 HTTP/1.1
Host: api.example.com
Origin: https://app.example.com
Accept: */*
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: content-type
```

API:et jämför ursprunget med sin egen lista och svarar:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://app.example.com
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Headers: content-type
Access-Control-Max-Age: 600
Vary: Origin
```

Nu skickar webbläsaren själva `PUT`-anropet med kakan och en `Origin`-header, och API:ets svar upprepar `Access-Control-Allow-Origin` och `Access-Control-Allow-Credentials: true`. Saknas någon av dem är anteckningen sparad på servern, men skriptet får aldrig se svaret. `Access-Control-Max-Age` låter webbläsaren återanvända svaret på förhandsanropet i upp till 600 sekunder i stället för att fråga före varje anrop; utan headern behåller standarden det i fem sekunder. `Vary: Origin` är med eftersom svaret beror på vem som frågar, och en cache som bortsåg från det skulle kunna lämna ett ursprungs svar vidare till ett annat.

På en plattform med professionella och organisatoriska data har API:et hastighetsbegränsning, CORS, storleksgränser för anrop och validering av indata innan något når databasen. Samma sida förklarar varför CORS är en punkt på den listan och inte hela försvaret: plattformen utgår från att klienten är fientlig, och servern kontrollerar behörigheterna på nytt vid varje anrop, oavsett vad webbläsaren har tillåtit.

Var det går fel

Servern kopierar vilken `Origin` som helst tillbaka till `Access-Control-Allow-Origin`. Det är den snabbaste vägen till att få felet i konsolen att försvinna, och tillsammans med `Access-Control-Allow-Credentials: true` ger det varenda webbplats på internet användarens vy av API:et. Ha en uttrycklig lista över de ursprung som ska ha åtkomst, jämför inkommande `Origin` exakt med den och skicka headrarna bara vid träff.

Listan kontrolleras med ett prefix eller en delsträng. En regel som godtar varje ursprung som börjar med `https://app.example.com` godtar också `https://app.example.com.attacker.example`, och en delsträngsmatchning är ännu sämre. Jämför hela serialiserade ursprung. Ett serialiserat ursprung har varken sökväg eller avslutande snedstreck, så `https://app.example.com/` på listan kommer aldrig att matcha någonting.

`*` förväntas fungera tillsammans med kakor. När ett anrop har inloggningsuppgifter med sig avvisar webbläsaren `*` i `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` och `Access-Control-Expose-Headers`, och `Access-Control-Allow-Credentials` måste vara exakt `true`, med små bokstäver. Ett publikt API som bara läses och inte använder kakor kan tryggt svara `*`; ett API som vilar på en session måste namnge ursprunget.

`null` **står på listan över tillåtna.** Sidor som laddats från en `data:` -URL får ett ogenomskinligt ursprung, och ett anrop som har omdirigerats mellan ursprung rapporterar också `Origin: null`, så många helt olika dokument visar samma värde. Att tillåta `null` är att tillåta alla. Låt det vara ute.

`Vary: Origin` **saknas.** Skickar API:et `Access-Control-Allow-Origin` bara när ett anrop har en `Origin` som stämmer kan en cache spara ett svar utan headern och servera det för ett senare CORS-anrop, eller spara ett ursprungs svar och ge det till ett annat. Fetch-standarden ber om `Vary: Origin` när svaret beror på ursprunget. Är värdet `*` eller ett fast ursprung, skicka det i varje svar och låt bli `Vary`.

`Access-Control-Allow-Headers: *` **antas täcka allt.** Jokertecknet täcker aldrig `Authorization`, som måste namnges, och det täcker ingenting alls för anrop med inloggningsuppgifter. Namnge de headrar som klienten faktiskt skickar.

CORS används som åtkomstkontroll. En server som avvisar anrop för att `Origin` ser fel ut, och annars släpper in dem, har skrivit en kontroll som varje skript utanför en webbläsare går rakt förbi. Autentisera anroparen, auktorisera handlingen på servern och låt CORS sköta sitt enda jobb: att avgöra vad en webbläsare visar för en sida.

Läs mer

- Fetch Standard: the CORS protocol — definitionen: anrops- och svarsheadrarna, förhandsanropet, tabellen över inloggningsuppgifter och rådet om `Vary`.
- RFC 6454: The Web Origin Concept — vad ett ursprung är, hur schema, värd och port avgör det och hur det skrivs i `Origin` -headern.
- OWASP Web Security Testing Guide: Testing Cross Origin Resource Sharing — hur testare letar efter ett jokerteckensursprung eller ett ursprung som skickas tillbaka utan kontroll.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är CORS?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/cors/