Engineer

CORS

hvordan en server lar sider fra andre opphav lese svarene

Hva det er

CORS, skrevet ut Cross-Origin Resource Sharing, er reglene en server bruker for å fortelle nettleseren at sider fra andre nettsteder får lese det den sender tilbake. Nettlesere holder opphav (*origins*) adskilt. Et opphav er skjema, vert og port til sammen, så `https://app.example.com`, `https://api.example.com` og `http://app.example.com` er tre forskjellige opphav, og et skript på det ene får ikke lese et svar fra et annet uten at det andre går med på det. CORS er måten det går med på det: en liten familie HTTP-headere, nesten alle med navn som begynner på `Access-Control-`, som serveren legger på svarene sine, og som nettleseren sjekker før noe som helst slippes videre til skriptet som spurte.

Protokollen står i Fetch-standarden fra WHATWG, den levende spesifikasjonen av hvordan nettlesere henter ressurser. Begrunnelsen står der også, og den er enkel. En side skal kunne bruke data fra et annet opphav og sende rikere forespørsler enn et HTML-skjema kan, men bare når motparten selv har sagt ja. Uten det ja-et kunne en hvilken som helst side på nettet lese servere som står bak en bedrifts brannmur, eller lese en brukers private data på et annet nettsted med brukerens informasjonskapsler (*cookies*) på slep.

Selve utvekslingen er kort. Når et skript ber om en ressurs fra et annet opphav, legger nettleseren til en `Origin`-header med sidens eget opphav. Serveren svarer med `Access-Control-Allow-Origin`, og verdien er enten det samme opphavet, skrevet nøyaktig, eller `*` for alle. Stemmer den, får skriptet svaret. Stemmer den ikke, får skriptet en nettverksfeil og ingenting av innholdet.

Noen forespørsler trenger lov før de i det hele tatt sendes. En `GET`, `HEAD` eller `POST` med noen få vanlige headere og en kropp av typen `application/x-www-form-urlencoded`, `multipart/form-data` eller `text/plain` går rett ut, for det har et HTML-skjema alltid kunnet sende fra et hvilket som helst nettsted. Alt utover det, for

eksempel en `PUT`, en `DELETE`, en JSON-kropp eller en `Authorization`-header, får en forhåndsforespørsel (*preflight*) foran seg: en `OPTIONS`-forespørsel som navngir metoden i `Access-Control-Request-Method` og de ekstra headerne i `Access-Control-Request-Headers`. Den egentlige forespørselen følger bare når svaret har en vellykket status som 200 eller 204 og tillater begge deler.

Bruk begrepet i en beslutning

Bruk «Hva er CORS?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Kontoer, passord og tilganger ordnet slik at folk har nøyaktig den tilgangen jobben krever, og de som slutter, mister den samme dag.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

Nesten alle nettapplikasjoner med et eget API møter CORS den første dagen. Frontenden serveres fra ett opphav og API-et fra et annet, og det første `fetch`-kallet fra det ene til det andre feiler i nettleserens konsoll til API-et sender de riktige headerne. Team som blir kjent med CORS på den måten, husker det gjerne som et hinder og konfigurerer det deretter, ved å tillate det som får feilen til å forsvinne. Det er der risikoen ligger.

Headerne avgjør hvem som kan lese brukernes data gjennom deres egne nettlesere. En bruker som er logget inn i applikasjonen din, har med seg en øktkapsel. Hvis API-et lar et hvilket som helst opphav sende forespørsler med informasjonskapsler og lese svarene, kan hver side brukeren besøker, kalle API-et i brukerens navn og lese resultatet. Fetch-standarden sier det rett ut: å dele svar som er hentet med brukerens påloggingsdata, krever den største forsiktighet for å unngå det den kaller *confused deputy*-problemet. Nettleseren, som bærer brukerens informasjonskapsler, er stedfortrederen som blir lurt, og et slurvete CORS-oppsett lar en fremmed side styre den.

Det er like viktig å vite hva CORS ikke gjør. Det beskytter ikke serveren. En enkel forespørsel når fram til serveren og blir utført uansett om svaret senere får deles, så en `POST` som endrer data, har endret dem før nettleseren har sett på en eneste CORS-header. Utenfor en nettleser er `Origin`-headeren det kalleren skriver inn, og testveiledningen fra OWASP advarer om at den kan forfalskes.

Hvem som får kalle et endepunkt, og hva de får gjøre der, avgjøres av autentisering og autorisasjon på serveren, hver gang. CORS avgjør bare hvilke nettsider som får lese svaret inne i en nettleser.

Slik ser det ut i praksis

Tenk deg en enkeltsideapplikasjon på `https://app.example.com` som lagrer et notat gjennom et API på `https://api.example.com` og bruker øktkapselen API-et satte ved innlogging:

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

En `PUT` med JSON-kropp er mer enn et skjema kan sende, så nettleseren spør først. Forhåndsforespørselen har ingen informasjonskapsler med; den navngir bare det den egentlige forespørselen vil bruke:

```
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 holder opphavet opp mot sin egen liste og svarer:

```
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
```

Nå sender nettleseren selve `PUT`-forespørselen med kapselen og en `Origin`-header, og svaret fra API-et gjentar `Access-Control-Allow-Origin` og `Access-Control-Allow-Credentials: true`. Mangler en av dem, er notatet lagret på serveren, men skriptet får aldri se svaret. `Access-Control-Max-Age` lar nettleseren gjenbruke svaret på forhåndsforespørselen i opptil 600 sekunder i stedet for å spørre før hvert kall; uten headeren holder standarden på det i fem sekunder. `Vary: Origin` er med fordi svaret avhenger av hvem som spør, og en hurtigbuffer (*cache*) som overså det, kunne gi ett opphavs svar videre til et annet.

På en plattform med profesjonelle og organisatoriske data har API-et hastighetsbegrensning, CORS, grenser for forespørselsstørrelse og inndatavalidering før noe rører databasen. Den samme siden forklarer hvorfor CORS er ett punkt på den lista og ikke hele forsvaret: plattformen går ut fra at klienten er fiendtlig, og serveren kontrollerer rettighetene på nytt ved hver forespørsel, uansett hva nettleseren har tillatt.

Hvor det går galt

Serveren kopierer en hvilken som helst `Origin` tilbake i `Access-Control-Allow-Origin`. Det er den raskeste veien til å få feilen i konsollen til å forsvinne, og sammen med `Access-Control-Allow-Credentials: true` gir det hvert eneste nettsted på internett brukerens utsyn i API-et. Ha en eksplisitt liste over opphavene som skal ha tilgang, sammenlign den innkommende `Origin` nøyaktig med den, og send headerne bare ved treff.

Lista sjekkes med et prefiks eller en delstreng. En regel som godtar ethvert opphav som begynner med `https://app.example.com`, godtar også `https://app.example.com.attacker.example`, og et delstrengtreff er verre. Sammenlign hele serialiserte opphav. Et serialisert opphav har verken sti eller avsluttende skråstrek, så `https://app.example.com/` på lista kommer aldri til å treffe noe.

`*` forventes å virke sammen med informasjonskapsler. Når en forespørsel har påloggingsdata med, avviser nettleseren `*` i `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` og `Access-Control-Expose-Headers`, og `Access-Control-Allow-Credentials` må være nøyaktig `true`, med små bokstaver. Et offentlig API som bare leses og ikke bruker informasjonskapsler, kan trygt svare `*`; et API som hviler på en økt, må navngi opphavet.

`null` **står på lista over tillatte.** Sider som er lastet fra en `data:` -URL, får et ugjennomsiktig opphav, og en forespørsel som er blitt omdirigert på tvers av opphav, melder også `Origin: null` , så mange helt ulike dokumenter viser den samme verdien. Å tillate `null` er å tillate alle sammen. La den være ute.

`Vary: Origin` **mangler.** Sender API-et `Access-Control-Allow-Origin` bare når en forespørsel har en `Origin` som stemmer, kan en hurtigbuffer lagre et svar uten headeren og servere det for en senere CORS-forespørsel, eller lagre ett opphavs svar og gi det til et annet. Fetch-standarden ber om `Vary: Origin` når svaret avhenger av opphavet. Er verdien `*` eller ett fast opphav, send den på hvert eneste svar og la `Vary` være.

`Access-Control-Allow-Headers: *` **antas å dekke alt.** Jokertegnet dekker aldri `Authorization` , som må navngis, og det dekker ingenting i det hele tatt for forespørsler med påloggingsdata. Navngi headerne klienten faktisk sender.

CORS brukes som tilgangskontroll. En server som avviser forespørsler fordi `Origin` ser feil ut, og ellers slipper dem inn, har skrevet en sjekk som ethvert skript utenfor en nettleser går rett forbi. Autentiser kalleren, autoriser handlingen på serveren, og la CORS gjøre sin ene jobb: å avgjøre hva en nettleser viser en side.

Les mer

- Fetch Standard: the CORS protocol — definisjonen: forespørsels- og svarheaderne, forhåndsforespørselen, tabellen over påloggingsdata og rådet om `Vary` .
- RFC 6454: The Web Origin Concept — hva et opphav er, hvordan skjema, vert og port avgjør det, og hvordan det skrives i `Origin` -headeren.
- OWASP Web Security Testing Guide: Testing Cross Origin Resource Sharing — hvordan testere ser etter et jokertegnopphav eller et opphav som sendes tilbake uten kontroll.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er CORS?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/cors/