Engineer

CORS

sådan lader en server fremmede oprindelser læse dens svar

Hvad det er

CORS, skrevet helt ud Cross-Origin Resource Sharing, er de regler, en server bruger til at fortælle browseren, at sider fra andre websites må læse det, den sender tilbage. Browsere holder oprindelser adskilt. En oprindelse er skema, vært og port taget under ét, så `https://app.example.com`, `https://api.example.com` og `http://app.example.com` er tre forskellige oprindelser, og et script på den ene må ikke læse et svar fra en anden, medmindre den anden siger god for det. CORS er måden, den siger god for det på: en lille familie af HTTP-headere, som næsten alle begynder med `Access-Control-`, og som serveren sætter på sine svar, mens browseren tjekker dem, før noget som helst når frem til det kaldende script.

Protokollen står i WHATWG's Fetch-standard, den levende specifikation af, hvordan browsere henter ressourcer. Begrundelsen står der også, og den er enkel. En side skal kunne bruge data fra en anden oprindelse og sende mere avancerede requests, end en HTML-formular kan, men kun når modparten selv har sagt ja. Uden det ja kunne en tilfældig side på nettet læse servere, der står bag en virksomheds firewall, eller læse en brugers private data på et andet site med brugerens cookies hængt på.

Selve udvekslingen er kort. Når et script beder om en ressource fra en anden oprindelse, tilføjer browseren en `Origin`-header med sidens egen oprindelse. Serveren svarer med `Access-Control-Allow-Origin`, og værdien er enten den samme oprindelse, skrevet nøjagtigt, eller `*` for alle. Passer den, får scriptet svaret. Passer den ikke, får scriptet en netværksfejl og intet af indholdet.

Nogle requests kræver lov, før de overhovedet bliver sendt. En `GET`, `HEAD` eller `POST` med nogle få almindelige headere og en krop af typen `application/x-www-form-urlencoded`, `multipart/form-data` eller `text/plain` går direkte afsted, for den slags har en HTML-formular altid kunnet sende fra et hvilket som helst site. Alt ud

over det, fx en `PUT`, en `DELETE`, en JSON-krop eller en `Authorization`-header, får en preflight foran sig: en `OPTIONS`-request, der nævner metoden i `Access-Control-Request-Method` og de ekstra headere i `Access-Control-Request-Headers`. Den rigtige request kommer først, når svaret har en ok-status som 200 eller 204 og tillader begge dele.

Brug begrebet i en beslutning

Brug »Hvad er CORS?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Konti, adgangskoder og rettigheder sat op, så folk har præcis den adgang, jobbet kræver, og fratrådte mister den samme dag.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

Har skrevet 1.000 RBAC-regler til Linux-servere Kontakt os →

Hvorfor det betyder noget

Næsten alle webapplikationer med et separat API støder på CORS den første dag. Frontenden serveres fra én oprindelse og API'et fra en anden, og det første `fetch` fra den ene til den anden fejler i browserens konsol, indtil API'et sender de rigtige headere. Teams, der møder CORS på den måde, husker det typisk som en forhindring og konfigurerer det som en, ved at tillade hvad som helst der får fejlen til at gå væk. Det er dér, risikoen ligger.

Headerne afgør, hvem der kan læse dine brugeres data gennem deres egne browsere. En bruger, der er logget ind i din applikation, har en sessionscookie med. Hvis API'et lader en hvilken som helst oprindelse sende requests med cookies og læse svarene, kan enhver side, brugeren besøger, kalde API'et i brugerens navn og læse resultatet. Fetch-standarden siger det ligeud: at dele svar, der er hentet med brugerens loginoplysninger, kræver den yderste omhu for at undgå det, den kalder confused deputy-problemet. Browseren, som bærer brugerens cookies, er den stedfortræder, der bliver narret, og en sjusket CORS-opsætning lader en fremmed side styre den.

Det er lige så vigtigt at vide, hvad CORS ikke gør. Det beskytter ikke serveren. En simpel request når frem til serveren og bliver udført, uanset om svaret bagefter må deles, så en `POST`, der ændrer data, har ændret dem, før browseren har kigget på en eneste CORS-header. Uden for en browser er `Origin`-headeren hvad end kalderen skriver, og OWASP's testvejledning advarer om, at den kan

forfalskes. Hvem der må kalde et endepunkt, og hvad de må gøre der, afgøres af autentificering og autorisation på serveren, hver gang. CORS afgør kun, hvilke websider der må læse svaret inde i en browser.

Sådan ser det ud i praksis

Forestil dig en single-page-applikation på `https://app.example.com`, der gemmer en note gennem et API på `https://api.example.com` og bruger den sessionscookie, API'et satte ved login:

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

En `PUT` med en JSON-krop er mere, end en formular kan sende, så browseren spørger først. Preflighten har ingen cookies med; den nævner bare, hvad den rigtige request vil bruge:

```
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 oprindelsen op mod 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
```

Nu sender browseren selve `PUT`-requesten med cookie og `Origin`-header, og API'ets svar gentager `Access-Control-Allow-Origin` og `Access-Control-Allow-Credentials: true`. Mangler en af dem, er noten gemt på serveren, men scriptet får aldrig svaret at se. `Access-Control-Max-Age` lader browseren genbruge preflight-svaret i op til 600 sekunder i stedet for at spørge før hvert kald; uden headeren gemmer standarden det i fem sekunder. `Vary: Origin` er med, fordi svaret afhænger af, hvem der spørger, og en cache, der oversåt det, kunne give én oprindelses svar videre til en anden.

På en platform med professionelle og organisatoriske data har API'et rate limiting, CORS, grænser for request-størrelse og inputvalidering, før noget når databasen. Den samme side forklarer, hvorfor CORS kun er ét punkt på den liste og ikke hele forsvaret: platformen går ud fra, at klienten er fjendtlig, og serveren kontrollerer rettighederne igen ved hver request, uanset hvad browseren har tilladt.

Hvor det går galt

Serveren kopierer en hvilken som helst `Origin` tilbage i `Access-Control-Allow-Origin`. Det er den hurtigste vej til at få fejlen i konsollen til at forsvinde, og sammen med `Access-Control-Allow-Credentials: true` giver det hvert eneste website på nettet brugerens udsyn til API'et. Hav en eksplicit liste over de oprindelser, der skal have adgang, sammenlign den indkommende `Origin` nøjagtigt med den, og send kun headerne ved et match.

Listen tjekkes med et præfiks eller en delstreng. En regel, der godtager enhver oprindelse, som begynder med `https://app.example.com`, godtager også `https://app.example.com.attacker.example`, og et delstrengsmatch er endnu værre. Sammenlign hele serialiserede oprindelser. En serialiseret oprindelse har hverken sti eller afsluttende skråstreg, så `https://app.example.com/` på listen kommer aldrig til at matche noget.

`*` forventes at virke sammen med cookies. Når en request har loginoplysninger med, afviser browseren `*` i `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` og `Access-Control-Expose-Headers`, og `Access-Control-Allow-Credentials` skal være præcis `true` med små bogstaver. Et offentligt API, der kun læses og ikke bruger cookies, kan trygt svare `*`; et API, der hviler på en session, skal nævne oprindelsen.

`null` **står på listen over tilladte.** Sider, der er indlæst fra en `data:` -URL, får en uigennemsigtig oprindelse, og en request, der er blevet omdirigeret på tværs af oprindelser, melder også `Origin: null`, så mange vidt forskellige dokumenter viser den samme værdi. At tillade `null` er at tillade dem alle. Lad den blive ude.

`Vary: Origin` **mangler.** Sender API'et kun `Access-Control-Allow-Origin`, når en request har en `Origin`, der passer, kan en cache gemme et svar uden headeren og servere det for en senere CORS-request, eller gemme én oprindelses svar og give det til en anden. Fetch-standarden beder om `Vary: Origin`, når svaret afhænger af oprindelsen. Er værdien `*` eller én fast oprindelse, så send den på hvert eneste svar og lad `Vary` være.

`Access-Control-Allow-Headers: *` **regnes for at dække alt.** Jokertegnet dækker aldrig `Authorization`, som skal nævnes ved navn, og det dækker slet ingenting for requests med loginoplysninger. Nævn de headere, klienten faktisk sender.

CORS bliver brugt som adgangskontrol. En server, der afviser requests, fordi `Origin` ser forkert ud, og ellers lukker dem ind, har skrevet et tjek, som ethvert script uden for en browser går lige forbi. Autentificér kalderen, autorisér handlingen på serveren, og lad CORS passe sit ene job: at afgøre, hvad en browser viser en side.

Læs mere

- Fetch Standard: the CORS protocol — definitionen: request- og svarheaderne, preflighten, tabellen over loginoplysninger og rådet om `Vary`.
- RFC 6454: The Web Origin Concept — hvad en oprindelse er, hvordan skema, vært og port afgør den, og hvordan den skrives i `Origin` -headeren.
- OWASP Web Security Testing Guide: Testing Cross Origin Resource Sharing — hvordan testere leder efter en jokertegns-oprindelse eller en oprindelse, der sendes tilbage uden kontrol.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er CORS?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/cors/