Engineer

CORS

die Lesefreigabe für Webseiten anderer Origins im Browser

Was es ist

CORS, ausgeschrieben Cross-Origin Resource Sharing, bezeichnet die Regeln, mit denen ein Server einem Browser mitteilt, dass Seiten anderer Websites seine Antworten lesen dürfen. Browser halten Origins getrennt. Eine Origin besteht aus Schema, Host und Port. `https://app.example.com`, `https://api.example.com` und `http://app.example.com` sind deshalb drei verschiedene Origins. Ein auf einer davon laufendes Skript darf eine Antwort einer anderen nur lesen, wenn diese andere zustimmt. CORS ist diese Zustimmung: eine kleine Familie von HTTP-Headern, die fast alle mit `Access-Control-` beginnen. Der Server fügt sie seinen Antworten hinzu, und der Browser prüft sie, bevor er etwas an das aufrufende Skript übergibt.

Das Protokoll steht im WHATWG Fetch Standard, der fortlaufend gepflegten Spezifikation dafür, wie Browser Ressourcen abrufen. Die Begründung steht dort ausdrücklich. Eine Seite soll Daten einer anderen Origin nutzen und Anfragen senden können, die mehr enthalten als ein HTML-Formular, aber nur mit Zustimmung der anderen Seite. Wäre eine ausdrückliche Zustimmung nicht vorgeschrieben, könnte eine Seite aus dem offenen Web die Antworten von Servern hinter einer Unternehmensfirewall lesen oder mit den angehängten Cookies eines Nutzers dessen private Daten auf einer anderen Website abrufen.

Der grundlegende Austausch ist kurz. Fordert ein Skript eine Ressource einer anderen Origin an, ergänzt der Browser einen `Origin`-Header mit der Origin der eigenen Seite. Der Server antwortet mit `Access-Control-Allow-Origin`. Dessen Wert ist entweder genau diese Origin oder `*` für alle. Stimmt der Wert, erhält das Skript die Antwort. Andernfalls erhält es einen Netzwerkfehler und nichts vom Antwortinhalt.

Manche Anfragen brauchen eine Erlaubnis, bevor sie überhaupt gesendet werden. Ein `GET`, `HEAD` oder `POST` mit nur wenigen gewöhnlichen Headern und einem Body vom Typ `application/x-www-form-urlencoded`, `multipart/form-data` oder `text/plain` wird direkt gesendet. Ein HTML-Formular könnte dasselbe schließlich schon von jeder Website aus senden. Alles darüber hinaus, etwa ein `PUT`, ein `DELETE`, ein JSON-Body oder ein `Authorization`-Header, erhält einen Preflight vorangestellt: eine `OPTIONS`-Anfrage, die die Methode in `Access-Control-Request-Method` und zusätzliche Header in `Access-Control-Request-Headers` nennt. Die eigentliche Anfrage folgt nur, wenn die Antwort einen Erfolgsstatus wie 200 oder 204 hat und beides erlaubt.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist CORS?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Konten, Passwörter und Berechtigungen werden so geordnet, dass alle genau den beruflich erforderlichen Zugriff haben und ihn am Tag ihres Ausscheidens verlieren.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 1.000 RBAC-Regeln für sichere Linux-Server — 1,000 Zugriffsregeln zum Schließen einer Sicherheitslücke

1.000 RBAC-Regeln für sichere Linux-Server Kontakt aufnehmen →

Warum es wichtig ist

Fast jede Webanwendung mit getrennter API begegnet CORS am ersten Tag. Das Frontend kommt von einer Origin, die API von einer anderen. Der erste `fetch` von der einen zur anderen scheitert in der Browserkonsole, bis die API die richtigen Header sendet. Teams, die CORS so kennenlernen, behalten es meist als Hindernis in Erinnerung und konfigurieren es entsprechend: Erlaubt wird, was den Fehler verschwinden lässt. Dort liegt das Risiko.

Die Header entscheiden, wer die Daten Ihrer Nutzer über deren Browser lesen darf. Ein bei Ihrer Anwendung angemeldeter Nutzer trägt ein Sitzungscookie. Erlaubt Ihre API jeder Origin, Anfragen mit Anmeldedaten zu stellen und die Antworten zu lesen, kann jede von diesem Nutzer besuchte Seite Ihre API in seinem Namen aufrufen und das Ergebnis lesen. Der Fetch Standard formuliert die Freigabe von Antworten auf Anfragen mit Anmeldedaten deutlich: „Es ist äußerste Sorgfalt nötig, um das Confused-Deputy-Problem zu vermeiden.“ Der Browser, der die Anmeldedaten des Nutzers hält, ist dabei der Stellvertreter. Eine nachlässige CORS-Richtlinie lässt ihn von einer fremden Seite lenken.

Ebenso wichtig ist zu wissen, was CORS nicht tut. Es schützt den Server nicht. Eine einfache Anfrage erreicht den Server und wird ausgeführt, unabhängig davon, ob die Antwort später freigegeben wird. Ein datenänderndes `POST` hat die Daten deshalb bereits geändert, bevor der Browser einen einzigen CORS-Header betrachtet. Außerhalb eines Browsers enthält der `Origin` -Header, was der Aufrufer hineinschreibt. Der OWASP-Testleitfaden warnt, dass er gefälscht sein kann. Authentifizierung und Autorisierung auf dem Server entscheiden bei jeder Anfrage, wer einen Endpunkt aufrufen und welche Aktionen dort ausführen darf. CORS entscheidet nur, welche Webseiten die Antwort innerhalb eines Browsers lesen dürfen.

Wie es in der Praxis aussieht

Angenommen, eine Single-Page-Anwendung unter `https://app.example.com` speichert eine Notiz über eine API unter `https://api.example.com` . Dazu nutzt sie das Sitzungscookie, das die API bei der Anmeldung gesetzt hat:

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

Ein `PUT` mit JSON-Body ist mehr, als ein Formular senden könnte. Deshalb fragt der Browser zuerst. Der Preflight enthält keine Cookies, sondern nennt nur, was die eigentliche Anfrage verwenden wird:

```
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
```

Die API prüft die Origin gegen ihre eigene Liste und antwortet:

```
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
```

Der Browser sendet nun das `PUT` mit dem Cookie und einem `Origin`-Header. Die API-Antwort wiederholt `Access-Control-Allow-Origin` und `Access-Control-Allow-Credentials: true`. Fehlt einer von beiden, wird die Notiz auf dem Server gespeichert, das Skript erfährt es aber nie. `Access-Control-Max-Age` erlaubt dem Browser, die Preflight-Antwort bis zu 600 Sekunden wiederzuverwenden, statt vor jedem Aufruf nachzufragen. Fehlt der Header, bewahrt der Standard sie fünf Sekunden auf. `Vary: Origin` ist wichtig, weil die Antwort von der Origin der Anfrage abhängt. Ein Cache, der das ignoriert, könnte die Antwort für eine Origin einer anderen geben.

Auf einer Plattform mit beruflichen und organisatorischen Daten führt die API-Schicht Rate Limiting, CORS, Größenbegrenzungen für Anfragen und Eingabevalidierung aus, bevor etwas die Datenbank erreicht. Dieselbe Seite erklärt, warum CORS nur ein Punkt dieser Liste und nicht die gesamte Abwehr ist: Die Plattform geht von einem feindlichen Client aus. Der Server prüft die Berechtigungen jeder Anfrage erneut, unabhängig davon, was der Browser erlaubt hat.

Wo es schiefgeht

Der Server kopiert jede `Origin` in `Access-Control-Allow-Origin` zurück. Das lässt den Konsolenfehler am schnellsten verschwinden. Zusammen mit `Access-Control-Allow-Credentials: true` gibt es jeder Website im Internet dieselbe Sicht auf die API wie dem Nutzer. Führen Sie eine ausdrückliche Liste der Origins, die Zugriff haben sollen, vergleichen Sie die eingehende `Origin` exakt damit und senden Sie die Header nur bei einer Übereinstimmung.

Die Liste wird mit einem Präfix oder einer Teilzeichenfolge geprüft. Eine Regel, die jede mit `https://app.example.com` beginnende Origin akzeptiert, akzeptiert auch `https://app.example.com.attacker.example`. Eine Suche nach Teilzeichenfolgen ist noch schlechter. Vergleichen Sie vollständige serialisierte Origins. Eine serialisierte Origin hat weder Pfad noch abschließenden Schrägstrich. `https://app.example.com/` in der Liste passt deshalb nie.

`*` **soll mit Anmeldedaten funktionieren.** Enthält eine Anfrage Anmeldedaten, lehnt der Browser `*` in `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` und `Access-Control-Expose-Headers` ab. `Access-Control-Allow-Credentials` muss exakt den kleingeschriebenen Wert `true` haben. Eine öffentliche, nur lesende API ohne Cookies kann gefahrlos mit `*` antworten. Eine auf einer Sitzung beruhende API muss die Origin nennen.

`null` **steht auf der Freigabeliste.** Aus einer `data:` -URL geladene Seiten erhalten eine opake Origin. Auch eine über Origins hinweg umgeleitete Anfrage meldet `Origin: null`. Viele voneinander unabhängige Dokumente geben deshalb denselben Wert an. Wer `null` erlaubt, erlaubt sie alle. Lassen Sie den Wert weg.

`Vary: Origin` **fehlt.** Sendet die API `Access-Control-Allow-Origin` nur bei Anfragen mit passender `Origin`, kann ein Cache eine Antwort ohne diesen Header speichern und bei einer späteren CORS-Anfrage ausliefern. Oder er speichert die Antwort einer Origin und liefert sie einer anderen aus. Der Fetch Standard verlangt `Vary: Origin`, sobald die Antwort von der Origin abhängt. Ist der Wert `*` oder eine einzelne feste Origin, senden Sie ihn bei jeder Antwort und lassen Sie `Vary` weg.

`Access-Control-Allow-Headers: *` **soll alles abdecken.** Der Platzhalter deckt `Authorization` nie ab; dieser Header muss namentlich aufgeführt werden. Für Anfragen mit Anmeldedaten deckt er überhaupt nichts ab. Nennen Sie die Header, die der Client tatsächlich sendet.

CORS wird als Zugriffskontrolle behandelt. Ein Server, der Anfragen wegen einer unpassend aussehenden `Origin` ablehnt und andernfalls annimmt, hat eine Prüfung gebaut, an der jedes Skript außerhalb eines Browsers vorbeikommt. Authentifizieren Sie den Aufrufer, autorisieren Sie die Aktion auf dem Server und überlassen Sie CORS seine eine Aufgabe: zu entscheiden, was ein Browser einer Seite zeigt.

Weiterführende Lektüre

- Fetch Standard: das CORS-Protokoll — die Definition: Anfrage- und Antwortheader, Preflight, Anmeldedatentabelle und Hinweise zu `Vary`.
- RFC 6454: Das Konzept der Web-Origin — was eine Origin ist, wie Schema, Host und Port sie bestimmen und wie sie in den `Origin` -Header serialisiert wird.

- OWASP Web Security Testing Guide: Cross-Origin Resource Sharing prüfen — wie Tester nach einer Platzhalter-Origin oder einer ungeprüft zurückgespiegelten Origin suchen.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist CORS?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/cors/