Engineer

CORS

autoriser une page à lire la réponse d'une autre origine

Ce que c'est

CORS, pour Cross-Origin Resource Sharing, est l'ensemble des règles par lesquelles un serveur indique à un navigateur que des pages d'autres sites peuvent lire ses réponses. Les navigateurs séparent les origines. Une origine est l'association d'un schéma, d'un hôte et d'un port : `https://app.example.com`, `https://api.example.com` et `http://app.example.com` sont donc trois origines différentes, et un script exécuté sur l'une ne peut pas lire une réponse d'une autre sans son accord. CORS exprime cet accord : une petite famille d'en-têtes HTTP, presque tous commençant par `Access-Control-`, que le serveur ajoute à ses réponses et que le navigateur vérifie avant de transmettre quoi que ce soit au script appelant.

Le protocole figure dans le Fetch Standard de WHATWG, la spécification évolutive de la récupération des ressources par les navigateurs. Sa justification y est explicite. Une page doit pouvoir utiliser les données d'une autre origine et envoyer des requêtes plus riches qu'un formulaire HTML, mais seulement si l'autre côté y consent. Sans cette exigence d'accord explicite, une page du web ouvert pourrait lire des serveurs derrière un pare-feu d'entreprise ou les données privées d'un utilisateur sur un autre site, avec les cookies de cet utilisateur.

L'échange essentiel est court. Quand un script demande une ressource d'une autre origine, le navigateur ajoute un en-tête `Origin` nommant l'origine de la page. Le serveur répond avec `Access-Control-Allow-Origin`, dont la valeur est soit cette origine écrite exactement, soit `*` pour tout le monde. Si elle correspond, le script reçoit la réponse. Sinon, il reçoit une erreur réseau et aucun contenu du corps.

Certaines requêtes exigent une permission avant même d'être envoyées. Une requête `GET`, `HEAD` ou `POST` avec seulement quelques en-têtes ordinaires et un corps de type `application/x-www-form-urlencoded`, `multipart/form-data` ou `text/plain` part directement, car un formulaire HTML pouvait déjà envoyer la même chose depuis n'importe quel site. Tout ce qui dépasse ce cadre, comme `PUT`, `DELETE`, un corps JSON ou un en-tête `Authorization`, est précédé d'un preflight, une requête préalable : une requête `OPTIONS` qui nomme la méthode dans `Access-Control-Request-Method` et les en-têtes supplémentaires dans `Access-Control-Request-Headers`. La véritable requête ne suit que si la réponse a un statut de réussite, comme 200 ou 204, et autorise les deux.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que CORS? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des comptes, mots de passe et droits attribués selon les besoins exacts de chaque poste, puis retirés le jour du départ.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 1 000 règles RBAC pour sécuriser Linux — 1,000 règles d'accès écrites pour combler une faille de sécurité

1 000 règles RBAC pour sécuriser Linux Contactez-nous →

Pourquoi c'est important

Presque toute application web dotée d'une API séparée rencontre CORS dès le premier jour. Le frontend vient d'une origine et l'API d'une autre ; le premier `fetch` entre les deux échoue dans la console du navigateur jusqu'à ce que l'API envoie les bons en-têtes. Les équipes qui le découvrent ainsi ont tendance à le retenir comme un obstacle et à le configurer comme tel, en autorisant tout ce qui fait disparaître l'erreur. C'est là que se situe le risque.

Les en-têtes décident qui peut lire les données de vos utilisateurs par leurs navigateurs. Une personne connectée à votre application possède un cookie de session. Si votre API autorise toute origine à faire des requêtes accompagnées d'informations d'authentification et à lire leurs réponses, n'importe quelle page visitée par cette personne peut appeler votre API en son nom et lire le résultat. Le Fetch Standard est direct au sujet du partage des réponses à des requêtes accompagnées d'informations d'authentification : « il faut faire preuve d'une extrême prudence pour éviter le problème de l'adjoint confus ». Le navigateur qui détient les informations d'authentification de l'utilisateur est l'adjoint, et une politique CORS négligente permet à la page d'un tiers de le diriger.

Il est tout aussi important de savoir ce que CORS ne fait pas. Il ne protège pas le serveur. Une requête simple arrive au serveur et s'exécute que sa réponse soit ensuite partagée ou non : un `POST` qui modifie des données les a déjà modifiées avant que le navigateur ne regarde le moindre en-tête CORS. En dehors d'un navigateur, l'en-tête `Origin` vaut ce que l'appelant y écrit, et le guide de test OWASP avertit qu'il peut être usurpé. L'authentification et l'autorisation sur le serveur déterminent, à chaque requête, qui peut appeler un point d'accès et ce qu'il peut y faire. CORS décide seulement quelles pages web peuvent lire la réponse dans un navigateur.

À quoi cela ressemble en pratique

Supposons qu'une application monopage à `https://app.example.com` enregistre une note par une API à `https://api.example.com`, avec le cookie de session défini par l'API lors de la connexion :

```
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" }),
});
```

Un `PUT` avec un corps JSON dépasse ce qu'un formulaire peut envoyer ; le navigateur demande donc d'abord la permission. Le preflight ne contient aucun cookie ; il nomme seulement ce qu'utilisera la véritable requête :

```
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
```

L'API compare l'origine à sa propre liste et répond :

```
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
```

Le navigateur envoie alors le `PUT` avec le cookie et un en-tête `Origin`, et la réponse de l'API répète `Access-Control-Allow-Origin` et `Access-Control-Allow-Credentials: true`. Si l'un des deux manque, la note est enregistrée sur le serveur, mais le script ne l'apprend jamais. `Access-Control-Max-Age` permet au navigateur de réutiliser la réponse au preflight pendant au plus 600 secondes au lieu de demander avant chaque appel ; si l'en-tête manque, la norme la conserve cinq secondes. L'en-tête `Vary: Origin` compte parce que la réponse dépend de l'origine de la requête ; un cache qui l'ignorerait pourrait donner la réponse d'une origine à une autre.

Sur une plateforme contenant des données professionnelles et organisationnelles, la couche API exécute la limitation de débit, CORS, les limites de taille des requêtes et la validation des entrées avant que quoi que ce soit n'atteigne la base de données. La même page explique pourquoi CORS est un élément de cette liste et non toute la défense : la plateforme suppose que le client est hostile, et le serveur revérifie les droits de chaque requête, quoi qu'ait autorisé le navigateur.

Où les problèmes surviennent

Le serveur recopie toute valeur `Origin` dans `Access-Control-Allow-Origin`. C'est le moyen le plus rapide de faire disparaître l'erreur de console ; avec `Access-Control-Allow-Credentials: true`, il donne à chaque site d'Internet la vue de l'API dont dispose l'utilisateur. Tenez une liste explicite des origines qui doivent avoir accès, comparez-y exactement l'`Origin` reçu et n'envoyez les en-têtes qu'en cas de correspondance.

La liste est vérifiée par préfixe ou sous-chaîne. Une règle acceptant toute origine qui commence par `https://app.example.com` accepte aussi `https://app.example.com.attacker.example`, et une correspondance par sous-chaîne est pire encore. Comparez les origines sérialisées entières. Une origine sérialisée n'a ni chemin ni barre oblique finale ; `https://app.example.com/` dans la liste ne correspondra donc jamais à rien.

On s'attend à ce que `*` fonctionne avec des informations d'authentification. Lorsqu'une requête inclut des informations d'authentification, le navigateur refuse `*` dans `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods`, `Access-Control-Allow-Headers` et `Access-Control-Expose-Headers`, et `Access-Control-Allow-Credentials` doit valoir exactement `true`, en minuscules. Une API publique en lecture seule, sans cookies, peut répondre `*` sans risque ; une API fondée sur une session doit nommer l'origine.

`null` figure dans la liste autorisée. Les pages chargées depuis une URL `data:` ont une origine opaque, et une requête redirigée entre origines signale aussi `Origin: null` : beaucoup de documents sans lien entre eux présentent donc la même valeur. Autoriser `null` les autorise tous. Ne l'incluez pas.

`Vary: Origin` manque. Si l'API n'envoie `Access-Control-Allow-Origin` que lorsqu'une requête possède une valeur `Origin` correspondante, un cache peut stocker une réponse sans cet en-tête puis la servir à une requête CORS ultérieure, ou stocker la réponse d'une origine et la servir à une autre. Le Fetch Standard demande `Vary: Origin` dès que la réponse dépend de l'origine. Si la valeur autorisée est `*` ou une seule origine fixe, envoyez-la sur chaque réponse et omettez `Vary`.

On suppose que `Access-Control-Allow-Headers: *` couvre tout. Le joker ne couvre jamais `Authorization`, qui doit être nommé explicitement, et il ne couvre rien du tout pour les requêtes accompagnées d'informations d'authentification. Nommez les en-têtes réellement envoyés par le client.

CORS est traité comme un contrôle d'accès. Un serveur qui refuse les requêtes parce que la valeur de `Origin` paraît incorrecte et les accepte sinon a écrit un contrôle que n'importe quel script exécuté hors d'un navigateur contourne. Authentifiez l'appelant, autorisez l'action sur le serveur et laissez CORS accomplir sa seule tâche : décider ce qu'un navigateur montre à une page.

Pour aller plus loin

- Fetch Standard : le protocole CORS — la définition : en-têtes de requête et de réponse, preflight, tableau relatif aux informations d'authentification et conseils sur `Vary`.
- RFC 6454 : le concept d'origine web — ce qu'est une origine, comment schéma, hôte et port la déterminent, et sa sérialisation dans l'en-tête `Origin`.

- Guide OWASP de test de la sécurité web : tester le partage de ressources entre origines — comment les testeurs cherchent une origine joker ou une origine renvoyée sans contrôle.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que CORS ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/cors/