HTTP — sécurité avancée, 40 leçons de cours

Quarante leçons autonomes pour analyser HTTP de bout en bout : sémantique, navigateurs, authentification, politiques de sécurité, caches, mandataires et exploitation. Chaque page contient un laboratoire corrigé et un contrôle original.

HTTP 1.1 / 2 / 3 · ASVS 5.0 · 40 leçons

Parcours indépendant fondé sur les standards HTTP, la documentation navigateur Mozilla et les recommandations OWASP. Les questions sont originales ; la durée et le seuil de l'examen sont pédagogiques. Les pages couvrent les responsabilités de plusieurs couches, sans prétendre qu'un en-tête suffit à sécuriser une application.

Référence consultée le 2026-10-29.

https://asvs.owasp.org/

Support indépendant d'autoformation. Les exemples sont destinés à un laboratoire autorisé ; les fragments ne constituent pas une configuration universelle de production. Les corrections sont incluses dans ce livre. Chaque leçon commence sur une nouvelle page à l'impression.

Programme du cours

1. 01. Reconstituer le trajet HTTP
2. 02. Méthodes et sémantique des opérations
3. 03. Statuts et erreurs sans fuite
4. 04. Idempotence et reprises après incident
5. 05. Champs, valeurs et frontières d'interprétation
6. 06. Content-Length et limites du corps
7. 07. Délimitation et encodage de transfert
8. 08. Désynchronisation et request smuggling
9. 09. HTTP/2 : multiplexage et contraintes partagées
10. 10. HTTP/3 : QUIC, pertes et garanties restantes
11. 11. Origine, site et URL : trois notions distinctes
12. 12. Same-origin policy : lecture et envoi
13. 13. CORS : autoriser une lecture sans ouvrir les droits
14. 14. Preflight : intention annoncée et demande réelle
15. 15. Credentials CORS et frontières de cookies
16. 16. Cookies : transport d'état et confiance
17. 17. Set-Cookie, portée et préfixes
18. 18. Cycle de session, fixation et élévation de privilèges
19. 19. CSRF : déclenchement d'une action sous une identité
20. 20. XSS : données, code et contexte de sortie
21. 21. CSP : politique de chargement et déploiement progressif
22. 22. Nonces CSP : imprévisibilité et cohérence
23. 23. Encadrement et clickjacking
24. 24. HSTS : persistance d'une exigence HTTPS
25. 25. Types de contenu et refus d'interprétation imprévue
26. 26. Referrer-Policy et données dans les URL
27. 27. Caches : fraîcheur, stockage et confidentialité
28. 28. Cache-Control : no-cache, no-store et private
29. 29. Vary et variantes de représentation
30. 30. Revalidation et mises à jour conditionnelles
31. 31. Redirections, méthodes et destinations
32. 32. Host, Forwarded et confiance des mandataires
33. 33. SSRF et sorties HTTP du serveur
34. 34. Uploads, ZIP et budgets par couche
35. 35. Téléchargements et Content-Disposition
36. 36. Limitation de débit et budgets d'abus
37. 37. Authentification HTTP et autorisation métier
38. 38. Clients HTTP serveur : délais, tailles et reprises
39. 39. Journaux HTTP sans fuite et sans faux contexte
40. 40. Revue avancée et dossier de preuves HTTP

01. Reconstituer le trajet HTTP

Identifier quelle couche accepte, transforme ou refuse chaque message.

Comprendre le mécanisme

HTTP organise un échange de requêtes et de réponses. Dans une architecture réelle, le navigateur peut parler à un CDN, qui contacte un reverse proxy puis une application. Chaque intermédiaire possède son propre cache, ses limites et sa journalisation. Le serveur indiqué par un en-tête ne décrit donc pas à lui seul toute la chaîne. Une connexion TLS et une session applicative sont aussi des états différents : la première protège un canal, la seconde établit une continuité d'identité selon une politique.

Raisonner et vérifier

Pour diagnostiquer, séparez observation et déduction. Une erreur reçue dans le navigateur peut être fabriquée avant le framework. Relevez méthode, URL, statut, horodatage, identifiant de corrélation et couche ayant journalisé la requête. Le corps multipart affiché dans un outil de développement peut omettre les octets binaires ; la taille de la réponse n'est pas celle de l'upload. Comparez les indices sur le même trajet plutôt que modifier une limite au hasard.

Refus avant le framework

Un proxy peut refuser avant PHP. Plusieurs couches peuvent produire une réponse avant le framework. Une limite de corps peut interrompre le transfert au reverse proxy. Cherchez d'abord si PHP a reçu la demande ; modifier le contrôleur ne corrige pas une décision prise avant son exécution.

> **Erreur fréquente à comprendre** Toute réponse HTTP provient du contrôleur PHP.
>
> Plusieurs couches peuvent produire une réponse avant le framework.

Localiser un 413

Une réponse 413 peut être émise par nginx. Le statut seul ne désigne pas un contrôle de formulaire. Le champ Server est un indice sur la couche qui répond, pas une cartographie complète. Corrélez le statut avec les journaux nginx et application pour attribuer le refus sans déduire toute l'architecture d'un seul header.

> **Erreur fréquente à comprendre** Un statut 413 prouve un défaut CSRF.
>
> Le statut seul ne désigne pas un contrôle de formulaire.

Longueur de réponse

Content-Length de réponse décrit cette réponse. Requête et réponse sont deux messages distincts. Content-Length: 176 dans la réponse décrit sa propre longueur de corps annoncée. Cela ne mesure ni le fichier ZIP envoyé ni les octets décompressés ; relevez séparément les deux messages.

> **Erreur fréquente à comprendre** Content-Length de réponse mesure le fichier envoyé.
>
> Requête et réponse sont deux messages distincts.

Canal et identité

TLS et session applicative ont des cycles distincts. Le canal et l'identité applicative sont des états différents. Une nouvelle connexion TLS peut porter la même session applicative via un cookie. Inversement, un canal existant peut transporter des demandes pour lesquelles l'utilisateur se déconnecte ; les deux cycles ne se confondent pas.

> **Erreur fréquente à comprendre** Renouveler TLS renouvelle obligatoirement la session métier.
>
> Le canal et l'identité applicative sont des états différents.

Corrélation

Une corrélation aide à rapprocher les journaux. Relier des événements ne prouve pas une identité. Utilisez un identifiant non sensible pour rapprocher événements proxy et application. Contrôlez sa génération et sa validation afin qu'une entrée client ne crée pas un contexte de diagnostic trompeur.

> **Erreur fréquente à comprendre** Un identifiant de corrélation authentifie le client.
>
> Relier des événements ne prouve pas une identité.

Cartographier les sauts

Le trajet peut inclure un CDN et un proxy. Un champ ne décrit pas tous les intermédiaires. Dessinez navigateur, CDN, proxy, serveur web et runtime. Pour chaque saut, notez protocole et terminaison TLS ; cette vue explique pourquoi une limite ou un cache intermédiaire peut décider avant l'origine.

> **Erreur fréquente à comprendre** Server expose nécessairement toute la topologie.
>
> Un champ ne décrit pas tous les intermédiaires.

Enveloppe multipart

Le formulaire multipart ajoute des octets au fichier. Séparateurs et champs participent à la taille transmise. Les séparateurs, en-têtes de parties et champs CSRF ajoutent de la taille au fichier. Prévoyez une marge explicite et mesurez la requête complète plutôt que déduire sa taille du seul explorateur de fichiers.

> **Erreur fréquente à comprendre** La taille du ZIP égale toujours celle de la requête.
>
> Séparateurs et champs participent à la taille transmise.

Non-effet précoce

Une requête peut échouer avant toute écriture métier. Le point de refus détermine les traitements atteints. Un refus avant l'application ne devrait produire aucune écriture métier. Vérifiez base et stockage d'import dans un test de laboratoire ; l'absence de page applicative n'exclut pas à elle seule tout traitement partiel.

> **Erreur fréquente à comprendre** Un refus réseau prouve que la base a été consultée.
>
> Le point de refus détermine les traitements atteints.

Chronologie

Comparer les heures exige une chronologie cohérente. Les observations doivent concerner le même échange. Les horloges et fuseaux des composants doivent permettre une comparaison cohérente. Préférez un format partagé et une corrélation stable, puis vérifiez la dérive avant d'attribuer un événement à la mauvaise tentative.

> **Erreur fréquente à comprendre** Des heures non corrélées suffisent à identifier la requête.
>
> Les observations doivent concerner le même échange.

Observation navigateur

Une capture navigateur constitue une observation partielle. Chaque outil observe une portion du trajet. Les outils peuvent masquer les octets binaires ou agréger certaines informations. Conservez les faits observés et complétez avec mesures serveur ; un champ vide affiché ne prouve pas que le fichier n'a pas été transmis.

> **Erreur fréquente à comprendre** La capture navigateur remplace tous les journaux serveur.
>
> Chaque outil observe une portion du trajet.

Atelier guidé : lire, expliquer et vérifier le code

```
HTTP/1.1 413 Content Too Large
Server: nginx
Content-Type: text/html
Content-Length: 176

[corps de réponse HTML omis]
```

La réponse montre un 413 et un petit corps HTML annoncé de 176 octets. Aucune de ces lignes ne donne la taille du ZIP envoyé. Relevez séparément taille de fichier, taille totale multipart et limite du saut qui refuse, puis confrontez les journaux de ce saut aux traces applicatives.

Le champ appartient à la réponse observée. La requête possède sa propre taille. Aucune extraction n'est mesurée ici. Cette limite n'est pas portée par ce champ.

Travail pratique : Dessinez votre chaîne de laboratoire et attribuez à chaque étape une limite de corps, un budget de temps et un journal. Simulez un refus au proxy.

Critères de correction : Envoyez deux fichiers de tailles connues vers un endpoint de test, puis confrontez les journaux proxy et application. Vérifiez qu'un refus précoce ne déclenche aucune écriture métier. Notez les tailles de requête et de réponse séparément.

Exemple commenté

```
Navigateur → proxy public → nginx → PHP-FPM → application
Requête : POST /admin/import/realisations
Réponse : 413 ; corps HTML court
```

Cette trace est un point de départ, pas la preuve que PHP a ouvert le fichier. Il faut localiser le refus dans les journaux de la couche concernée.

Exercice de mise en pratique

Dessinez votre chaîne de laboratoire et attribuez à chaque étape une limite de corps, un budget de temps et un journal. Simulez un refus au proxy.

Voir une proposition de correction

Envoyez deux fichiers de tailles connues vers un endpoint de test, puis confrontez les journaux proxy et application. Vérifiez qu'un refus précoce ne déclenche aucune écriture métier. Notez les tailles de requête et de réponse séparément.

- Identifier la couche observée.
- Corréler sans enregistrer de secrets.
- Distinguer requête, réponse et interprétation.

Contrôle de connaissances

Une réponse contient Content-Length: 176. Que mesure cette valeur ?

> Réponse expliquée : Le corps de cette réponse — L'en-tête appartient au message réponse observé.

Cours associé : 01. Reconstituer le trajet HTTP ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview

02. Méthodes et sémantique des opérations

Choisir une méthode selon le contrat et non selon la commodité du formulaire.

Comprendre le mécanisme

GET demande une représentation ; il n'est pas conçu pour déclencher une mutation métier. POST soumet un traitement selon le contrat de la ressource. PUT vise un remplacement dans le cadre défini et DELETE une suppression de l'association à la ressource ciblée. La signification d'une méthode n'implique pas que toute réponse soit identique ni que chaque opération soit autorisée. La sécurité sémantique d'une méthode porte sur l'effet demandé ; un journal technique peut évoluer lors d'une lecture.

Raisonner et vérifier

Définissez une table des opérations avant les routes. Un crawler, un préchargement ou un lien partagé peut provoquer un GET sans intention de mutation de l'utilisateur. Restreindre les méthodes au routeur évite certains déclenchements accidentels, mais ne remplace ni l'identité ni l'autorisation sur l'objet. Testez aussi HEAD et OPTIONS selon votre infrastructure, plutôt que supposer que leur comportement est identique à toutes les autres méthodes.

Lecture sûre

GET demande une lecture selon sa sémantique. Un préchargement peut déclencher une lecture sans intention de mutation. Une annulation déclenchée par GET peut être activée par crawler, préchargement ou visite involontaire. Placez les mutations derrière une méthode et un contrat adaptés, avec autorisation et protection CSRF lorsque nécessaire.

> **Erreur fréquente à comprendre** GET convient à une annulation métier demandée.
>
> Un préchargement peut déclencher une lecture sans intention de mutation.

Soumission POST

POST peut soumettre une création. Le verbe ne fournit pas une déduplication durable. POST exprime une soumission au traitement de la ressource. Il ne promet pas automatiquement l'unicité d'un effet ; une création sensible doit définir sa stratégie de déduplication et de reprise.

> **Erreur fréquente à comprendre** POST garantit automatiquement l'absence de doublon.
>
> Le verbe ne fournit pas une déduplication durable.

Idempotence PUT

PUT possède une sémantique idempotente. La propriété concerne l'effet intentionnel des répétitions. L'effet intentionnel d'un PUT répété identique est le même que celui d'un seul. Les journaux peuvent compter plusieurs demandes ; cette sémantique ne signifie pas que chaque réponse sera identique.

> **Erreur fréquente à comprendre** Idempotent signifie réponses toujours identiques.
>
> La propriété concerne l'effet intentionnel des répétitions.

DELETE répété

DELETE répété peut constater une absence. Effet et représentation de réponse sont distincts. Après suppression, un second DELETE peut constater l'absence et retourner un autre statut. L'état final peut rester identique : distinguez effet métier et représentation du résultat de chaque tentative.

> **Erreur fréquente à comprendre** Un second DELETE doit reproduire exactement le premier statut.
>
> Effet et représentation de réponse sont distincts.

HEAD

HEAD ne transporte pas de contenu de réponse. HEAD décrit la réponse sans son contenu. HEAD permet d'obtenir les métadonnées d'une représentation sans son contenu de réponse. Le traitement peut néanmoins avoir un coût serveur ; ne le considérez pas automatiquement comme gratuit ou non journalisé.

> **Erreur fréquente à comprendre** HEAD doit renvoyer intégralement le corps de GET.
>
> HEAD décrit la réponse sans son contenu.

OPTIONS

OPTIONS peut décrire des capacités de communication. Découverte protocolaire et autorisation sont distinctes. Une réponse OPTIONS peut renseigner les capacités ou participer à un preflight. Elle ne doit pas être interprétée comme une permission métier générale pour un utilisateur ou une ressource donnée.

> **Erreur fréquente à comprendre** OPTIONS donne automatiquement des droits métier.
>
> Découverte protocolaire et autorisation sont distinctes.

Restriction de route

Le routeur peut restreindre les méthodes. Le contrôle de méthode ne décide pas des droits. Déclarez les méthodes acceptées et testez une autre méthode sur le même chemin. Ce contrôle réduit les appels hors contrat mais ne prouve pas que l'acteur possède le droit d'annuler une réservation.

> **Erreur fréquente à comprendre** Restreindre à POST prouve la propriété de l'objet.
>
> Le contrôle de méthode ne décide pas des droits.

Effets accessoires

Une méthode sûre peut produire un journal technique. La sûreté concerne l'effet métier demandé. Une méthode sûre peut écrire un journal ou produire des métriques. L'exigence porte sur l'absence de mutation métier demandée implicitement, pas sur l'interdiction de toute activité interne du serveur.

> **Erreur fréquente à comprendre** Une méthode sûre interdit tout journal technique.
>
> La sûreté concerne l'effet métier demandé.

Client libre

Un client direct peut ignorer les boutons affichés. L'interface ne constitue pas la frontière d'accès. Le visiteur n'est pas limité aux boutons de l'interface. Il peut construire POST, DELETE ou un chemin différent ; le serveur doit vérifier chaque demande selon ses propres règles.

> **Erreur fréquente à comprendre** Masquer un bouton interdit la méthode côté serveur.
>
> L'interface ne constitue pas la frontière d'accès.

Rejet et état

Tester une méthode refusée doit vérifier le non-effet. Un contrôle trop tardif peut laisser un effet interdit. Après une méthode refusée, vérifiez statut et état durable. Une réponse 405 produite après une mutation ne satisfait pas la frontière attendue, même si l'interface n'affiche aucun succès.

> **Erreur fréquente à comprendre** Un statut de refus suffit même après mutation.
>
> Un contrôle trop tardif peut laisser un effet interdit.

Atelier guidé : lire, expliquer et vérifier le code

```
GET /lab/reservations/42/cancel HTTP/1.1
Host: booking.example.test

Contrat du laboratoire : cette requête annule réellement la réservation.
```

Le contrat précise que le GET annule réellement une réservation. Le défaut concerne donc sa sémantique, pas seulement son nom de chemin. Déplacez la mutation vers un parcours explicite puis vérifiez méthodes refusées, autorisation sur l'objet et absence d'effet en cas de rejet.

Le verbe est précisément prévu pour une lecture sûre. Le chemin ne prouve pas les droits. Le canal ne supprime pas l'effet métier. Crawlers et préchargements peuvent demander un GET sans cette intention.

Travail pratique : Établissez les méthodes d'un portail de réservation puis essayez une consultation et une annulation avec les méthodes autorisées et refusées.

Critères de correction : Ajoutez des tests de routage pour les méthodes incorrectes et contrôlez l'absence d'effet de bord. Pour une requête valide, testez séparément acteur non autorisé, réservation inexistante et transition métier interdite.

Exemple commenté

```
GET /reservations/42        → consulter
POST /reservations         → créer selon un contrat
POST /reservations/42/cancel → demander une annulation
```

La dernière route est un choix de conception possible. Son serveur doit encore vérifier droits, état métier et répétitions.

Exercice de mise en pratique

Établissez les méthodes d'un portail de réservation puis essayez une consultation et une annulation avec les méthodes autorisées et refusées.

Voir une proposition de correction

Ajoutez des tests de routage pour les méthodes incorrectes et contrôlez l'absence d'effet de bord. Pour une requête valide, testez séparément acteur non autorisé, réservation inexistante et transition métier interdite.

- Aucune mutation métier demandée par GET.
- Méthodes restreintes côté serveur.
- Autorisation indépendante de la méthode.

Contrôle de connaissances

Pourquoi éviter une annulation métier déclenchée par GET ?

> Réponse expliquée : Des lectures automatiques peuvent la déclencher — La méthode de lecture peut être préchargée ou explorée.

Cours associé : 02. Méthodes et sémantique des opérations ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods

03. Statuts et erreurs sans fuite

Exprimer un résultat protocolaire utile sans révéler l'état interne sensible.

Comprendre le mécanisme

Les familles de statuts distinguent notamment succès, redirection, erreur liée à la demande et erreur de traitement serveur. Un code doit être interprété avec le contrat de l'API. Une réponse 202 peut annoncer une acceptation sans fin du traitement ; une réponse 204 ne transporte pas de contenu de représentation. Le statut ne remplace pas un modèle d'erreur ni les règles d'autorisation. Une erreur interne ne doit pas exposer une trace, une requête SQL ou les paramètres de connexion.

Raisonner et vérifier

Séparez messages publics stables et diagnostic privé corrélé. Ne transformez pas toutes les erreurs en 200 : les clients, caches et outils de supervision perdraient un signal important. Inversement, distinguez une donnée invalide d'une panne de dépendance pour permettre une reprise correcte. Une politique de non-divulgation peut harmoniser certaines réponses d'authentification, tout en conservant des événements minimisés utiles à l'investigation.

Accepté mais incomplet

202 peut annoncer un traitement encore incomplet. Acceptation et achèvement ne sont pas équivalents. 202 indique une acceptation sans garantir que le traitement est terminé. Fournissez un identifiant de suivi et des états documentés afin que le client ne confonde pas mise en queue et résultat final.

> **Erreur fréquente à comprendre** 202 garantit la fin des effets métier.
>
> Acceptation et achèvement ne sont pas équivalents.

Réponse sans contenu

204 ne transporte pas de contenu de réponse. Le contrat de ce statut exclut le contenu. 204 exprime un succès sans contenu de réponse. N'ajoutez pas un JSON indispensable que le client devrait lire malgré ce statut ; choisissez un contrat cohérent entre statut et représentation.

> **Erreur fréquente à comprendre** 204 exige un document JSON de résultat.
>
> Le contrat de ce statut exclut le contenu.

Détails privés

Une erreur interne peut être corrélée en privé. Le diagnostic sensible ne doit pas être exposé. Une exception SQL peut contenir schéma, requête et paramètres. Retournez une erreur publique stable et conservez le diagnostic nécessaire dans un périmètre protégé, relié par une corrélation non sensible.

> **Erreur fréquente à comprendre** Toute erreur publique doit afficher la trace SQL.
>
> Le diagnostic sensible ne doit pas être exposé.

Codes stables

Un code d'erreur stable aide les clients. Une interface d'erreur doit fournir un contrat exploitable. Un code métier tel que capacity_exceeded permet un traitement client plus fiable qu'un texte humain variable. Versionnez le schéma d'erreur et évitez d'exposer directement les noms internes d'exceptions.

> **Erreur fréquente à comprendre** Les clients doivent interpréter une trace PHP variable.
>
> Une interface d'erreur doit fournir un contrat exploitable.

Signal de panne

Répondre 200 à toute panne masque un signal. Statuts et supervision doivent refléter le contrat. Retourner 200 avec un texte erreur peut faire considérer une panne comme un succès par cache, client ou supervision. Le statut doit rester cohérent avec le résultat réel du contrat.

> **Erreur fréquente à comprendre** 200 convient systématiquement à toute panne serveur.
>
> Statuts et supervision doivent refléter le contrat.

Catégoriser

Un conflit métier peut différer d'une panne distante. La stratégie dépend de la cause et des effets possibles. Un conflit d'état local et une dépendance indisponible ne demandent pas nécessairement la même reprise. Définissez catégories et possibilités de retry ; ne transformez pas toute exception en réponse identique sans analyse.

> **Erreur fréquente à comprendre** Toutes les erreurs imposent la même reprise.
>
> La stratégie dépend de la cause et des effets possibles.

Authentification uniforme

Une authentification invalide peut avoir un message public uniforme. Une politique de non-divulgation peut harmoniser les réponses. Une erreur publique peut éviter de distinguer compte absent et mauvais secret. Mesurez aussi chemins d'exécution et informations indirectes ; changer seulement le texte ne garantit pas une absence de divulgation.

> **Erreur fréquente à comprendre** Le message public doit toujours révéler si le compte existe.
>
> Une politique de non-divulgation peut harmoniser les réponses.

RequestId

Un requestId public peut être non sensible. La corrélation ne nécessite pas un secret utilisateur. Un identifiant opaque permet de rechercher un incident sans publier la trace complète. Ne placez pas credentials ou données personnelles dans cet identifiant et validez les valeurs externes éventuellement propagées.

> **Erreur fréquente à comprendre** Le requestId doit contenir le mot de passe.
>
> La corrélation ne nécessite pas un secret utilisateur.

Tests de fuite

Le corps d'erreur doit être testé pour les secrets. Le statut ne filtre pas automatiquement le contenu. Provoquez une erreur avec une valeur factice identifiable puis vérifiez qu'elle n'apparaît pas dans le corps public. Contrôlez également les en-têtes et les pages générées par les couches intermédiaires.

> **Erreur fréquente à comprendre** Un statut 500 empêche toute fuite dans le corps.
>
> Le statut ne filtre pas automatiquement le contenu.

Deux contrats

Un schéma public peut différer du diagnostic privé. Les publics et besoins d'accès sont différents. Le support technique peut avoir besoin d'une trace détaillée, tandis que le client reçoit une catégorie stable. Séparez accès et conservation de ces données pour garder un diagnostic utile sans divulgation publique.

> **Erreur fréquente à comprendre** Le journal privé et le corps public doivent être identiques.
>
> Les publics et besoins d'accès sont différents.

Atelier guidé : lire, expliquer et vérifier le code

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{"jobId":"lab-42","state":"queued"}
```

Le job est queued et la réponse est 202. Le client connaît une acceptation et une référence, mais pas la réussite de tous les effets métier. Définissez le mécanisme de suivi et testez les états ultérieurs, y compris échec après acceptation et absence de résultat encore disponible.

Les états sont différents. Le format ne prouve pas la progression métier. Le statut et l'état ne donnent pas la fin du traitement. Ce statut n'apporte pas cette garantie.

Travail pratique : Définissez cinq erreurs de votre portail : format invalide, refus de droit, conflit métier, dépassement de taille et dépendance indisponible.

Critères de correction : Associez chaque scénario à un statut et un schéma public. Simulez les erreurs et vérifiez que le corps ne contient pas de secret. Contrôlez aussi les chemins de log et la politique de reprise du client.

Exemple commenté

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/json
Cache-Control: no-store

{"error":"invalid_quantity","requestId":"lab-42"}
```

Le code d'erreur est contractuel et l'identifiant relie un diagnostic privé. Le message ne contient ni trace ni identifiant de connexion.

Exercice de mise en pratique

Définissez cinq erreurs de votre portail : format invalide, refus de droit, conflit métier, dépassement de taille et dépendance indisponible.

Voir une proposition de correction

Associez chaque scénario à un statut et un schéma public. Simulez les erreurs et vérifiez que le corps ne contient pas de secret. Contrôlez aussi les chemins de log et la politique de reprise du client.

- Statuts cohérents avec le contrat.
- Pas de trace interne dans la réponse.
- Corrélation publique non sensible.

Contrôle de connaissances

Une réponse 202 garantit-elle la fin de tous les effets métier ?

> Réponse expliquée : Non, elle peut annoncer une acceptation — Le traitement peut encore être en cours ou à venir.

Cours associé : 03. Statuts et erreurs sans fuite ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status

04. Idempotence et reprises après incident

Distinguer répétition du message et répétition de son effet.

Comprendre le mécanisme

Une méthode idempotente possède un effet intentionnel équivalent lorsqu'elle est répétée selon son contrat. Cela ne promet pas des codes de réponse identiques : une suppression répétée peut constater un état déjà absent. Un timeout crée souvent une incertitude, car le serveur peut avoir appliqué la modification avant que la réponse ne se perde. Le client ne peut pas conclure à un échec métier uniquement à partir du silence du réseau.

Raisonner et vérifier

Pour une création sensible, définissez un identifiant de demande et un stockage durable du résultat associé. La même clé avec un contenu différent doit être traitée selon une politique explicite, souvent comme un conflit. Une variable PHP en mémoire ne protège pas plusieurs instances ni une reprise après redémarrage. Testez le point où l'effet est validé mais où la réponse est perdue : c'est là que l'idempotence devient observable.

Issue incertaine

Un timeout peut suivre un commit réussi. La réponse peut se perdre après validation de l'effet. Un serveur peut commettre l'écriture puis perdre la réponse. Le timeout décrit l'observation du client, pas l'absence de résultat ; une reprise aveugle peut donc créer un doublon.

> **Erreur fréquente à comprendre** Un timeout prouve l'absence de modification distante.
>
> La réponse peut se perdre après validation de l'effet.

Identité stable

Une clé peut relier demande et résultat durable. La durée de vie mémoire ne fournit pas cette persistance. Une clé de demande identifie une intention métier et son résultat durable. Réutilisez-la lors d'une reprise de cette même intention, dans le délai et le périmètre définis par le contrat.

> **Erreur fréquente à comprendre** Une clé en mémoire protège tous les redémarrages.
>
> La durée de vie mémoire ne fournit pas cette persistance.

Contenu lié

La même clé doit suivre un contrat de contenu. La réutilisation contradictoire doit être définie et contrôlée. Associez clé et empreinte ou représentation canonique du contenu pertinent. Une même clé avec un montant différent doit être refusée ou traitée selon une politique explicite, jamais assimilée silencieusement à la première opération.

> **Erreur fréquente à comprendre** Une même clé autorise toute modification de contenu.
>
> La réutilisation contradictoire doit être définie et contrôlée.

Périmètre acteur

L'acteur fait partie du périmètre de déduplication. Un identifiant de demande n'accorde pas d'autorisation. Incluez tenant ou acteur dans l'unicité lorsque le contrat l'exige. Une clé choisie par un client ne doit pas lui permettre d'obtenir le résultat ou le statut d'une opération appartenant à un autre.

> **Erreur fréquente à comprendre** Partager toutes les clés entre acteurs donne des droits.
>
> Un identifiant de demande n'accorde pas d'autorisation.

Concurrence

Une contrainte unique aide contre la concurrence. Des demandes concurrentes peuvent passer le même précontrôle. Une lecture suivie d'une insertion sans contrainte peut laisser deux demandes concurrentes gagner. Utilisez une unicité durable et traitez son conflit, avec un état en cours défini pour les traitements longs.

> **Erreur fréquente à comprendre** Deux vérifications préalables sans contrainte suffisent toujours.
>
> Des demandes concurrentes peuvent passer le même précontrôle.

Restitution

Le résultat connu peut être restitué à la reprise. La restitution évite un second effet pour la même demande. Après succès déjà connu, retournez le résultat enregistré sans répéter l'effet. Définissez ce qui est conservé et pendant combien de temps, notamment pour les données sensibles ou les réponses devenues volumineuses.

> **Erreur fréquente à comprendre** Une reprise doit toujours créer une nouvelle ressource.
>
> La restitution évite un second effet pour la même demande.

Nouvelle clé

Une nouvelle clé peut représenter une autre demande. La déduplication dépend de l'identité conservée. Changer la clé après un timeout peut annoncer une nouvelle opération au fournisseur. Gardez la référence initiale et recherchez son état plutôt que contourner la déduplication pour obtenir rapidement une réponse.

> **Erreur fréquente à comprendre** Changer de clé garantit la déduplication de l'ancienne demande.
>
> La déduplication dépend de l'identité conservée.

Atomicité locale

Effet et enregistrement du résultat doivent être cohérents. Une panne entre les deux peut laisser un état ambigu. L'effet SQL et son enregistrement de déduplication doivent être cohérents. Pour une frontière externe, utilisez états durables et réconciliation ; une transaction locale ne peut pas annuler magiquement un effet prestataire.

> **Erreur fréquente à comprendre** Le résultat peut être enregistré sans lien avec le commit.
>
> Une panne entre les deux peut laisser un état ambigu.

Effet intentionnel

L'idempotence concerne l'effet intentionnel. Les métadonnées peuvent changer sans nouvel effet métier. Une reprise peut ajouter un log technique tout en conservant un seul débit. Définissez précisément l'effet métier qui doit rester unique, au lieu d'exiger que tout octet de toutes les réponses soit identique.

> **Erreur fréquente à comprendre** L'idempotence exige le même horodatage de réponse.
>
> Les métadonnées peuvent changer sans nouvel effet métier.

Panne critique

Le test important inclut la perte après commit. Ce cas ne couvre pas l'effet réussi dont la réponse est perdue. Simulez un succès durable puis une réponse perdue, et reprenez avec la même clé. Vérifiez nombre d'effets et résultat retrouvé ; un échec avant traitement ne couvre pas cette fenêtre.

> **Erreur fréquente à comprendre** Tester seulement un refus avant traitement suffit aux reprises.
>
> Ce cas ne couvre pas l'effet réussi dont la réponse est perdue.

Atelier guidé : lire, expliquer et vérifier le code

```
Envoi 1 : key=lab-1, amount=100 ; résultat durable connu
Envoi 2 : key=lab-1, amount=200
Politique : une même clé ne peut pas représenter deux contenus différents.
```

Les deux envois utilisent lab-1 mais présentent des montants différents. Le contrat interdit cette réutilisation incompatible : il faut refuser sans second effet. Ajoutez un troisième envoi identique au premier pour démontrer la reprise légitime et la restitution du résultat déjà durable.

Aucune annulation n'est montrée. La même identité de demande est présentée avec un contenu incompatible. Cela violerait la règle donnée. Le résultat existant concerne 100, pas 200.

Travail pratique : Simulez une perte de réponse après commit, puis rejouez la demande. Ajoutez un essai avec la même clé mais une autre quantité.

Critères de correction : Utilisez une contrainte unique et enregistrez résultat et effet de façon cohérente. Vérifiez une seule réservation, la restitution du résultat connu et le refus documenté de la réutilisation contradictoire.

PSP sans réponse : conserver l'incertitude et la référence

Un timeout peut survenir après que le prestataire a accepté le débit. Le client ne sait alors pas si l'effet a eu lieu ; le classer automatiquement en échec puis émettre un autre débit crée un risque de double effet.

Conservez un état inconnu ou en attente avec la référence et la clé initiales. Consultez l'état distant lorsque le contrat le permet. Si une reprise est autorisée, réutilisez la même opération avec la même clé et les mêmes paramètres, dans la portée et la durée d'idempotence documentées. Une nouvelle clé n'est pas un simple détail de transport : elle peut désigner une nouvelle intention.

Testez réponse perdue après succès et traitement encore en cours avec le simulateur PSP. Vérifiez un seul effet et une convergence d'état après rapprochement. Bornez les tentatives et prévoyez une intervention pour l'incertitude persistante ; la politique doit éviter aussi bien le double débit que l'oubli silencieux d'un débit réellement effectué.

Points de vigilance : La reprise doit respecter la portée, la durée et les conditions d'idempotence du PSP ; l'état distant doit être clarifié. Un nouvel identifiant peut être interprété comme une nouvelle opération alors que la première a réussi. Un rollback local ne commande pas le système distant. Le premier acquéreur peut avoir accepté : un basculement aveugle crée un risque de double débit.

Documentation de référence ↗

Exemple commenté

```
Clé : lab-request-001
Corps canonique : {"sku":"A","quantity":2}
Résultat durable : réservation R42
Nouvel envoi même clé + même corps → résultat de R42
```

La clé doit être liée à l'acteur et à l'opération concernée. Son existence seule ne prouve ni autorisation ni unicité globale.

Exercice de mise en pratique

Simulez une perte de réponse après commit, puis rejouez la demande. Ajoutez un essai avec la même clé mais une autre quantité.

Voir une proposition de correction

Utilisez une contrainte unique et enregistrez résultat et effet de façon cohérente. Vérifiez une seule réservation, la restitution du résultat connu et le refus documenté de la réutilisation contradictoire.

- Tester l'incertitude après commit.
- Lier clé et contenu pertinent.
- Éviter une déduplication seulement en mémoire.

Contrôle de connaissances

Un timeout signifie-t-il que le serveur n'a rien modifié ?

> Réponse expliquée : Non, la réponse a pu être perdue après l'effet — L'absence de réponse laisse une incertitude.

Cours associé : 04. Idempotence et reprises après incident ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Glossary/Idempotent

05. Champs, valeurs et frontières d’interprétation

Traiter les en-têtes comme un contrat structuré et non comme du texte libre.

Comprendre le mécanisme

Les champs HTTP transportent des métadonnées avec une syntaxe et une sémantique définies par champ. Certains acceptent des listes ; d’autres ne peuvent pas être combinés arbitrairement. Une application ne doit pas supposer que toutes les valeurs multiples se concatènent avec une virgule. Les noms de champs sont insensibles à la casse, mais les valeurs obéissent à leurs règles propres. Les intermédiaires peuvent aussi supprimer ou reconstruire des informations sur chaque saut.

Raisonner et vérifier

Une valeur reçue du client demeure non fiable, même lorsqu’elle porte un nom évoquant la sécurité. Évitez de construire un champ de réponse en concaténant des entrées contenant des caractères de contrôle. Utilisez les API structurées du framework et une liste fermée pour les choix de politique. Testez les doublons, valeurs absentes, casse et normalisations avec le véritable chemin proxy-application, en observant ce que chaque couche transmet.

Casse des noms

Les noms de champs sont insensibles à la casse. La casse du nom ne change pas sa sémantique HTTP. Content-Type et content-type désignent le même nom de champ. Utilisez les API du serveur et du client pour éviter une logique de sécurité qui ne vérifierait qu’une orthographe particulière.

> **Erreur fréquente à comprendre** Changer Authorization en authorization supprime son sens.
>
> La casse du nom ne change pas sa sémantique HTTP.

Valeurs spécifiques

Les règles des valeurs dépendent du champ. La grammaire des valeurs est propre au champ. Un nom insensible à la casse n’implique pas une règle uniforme sur toutes les valeurs. Suivez la définition de chaque champ avant de normaliser, découper ou comparer ses données.

> **Erreur fréquente à comprendre** Toutes les valeurs sont insensibles à la casse.
>
> La grammaire des valeurs est propre au champ.

Champs multiples

Certains champs ne se combinent pas librement. La combinaison dépend de la définition du champ. Set-Cookie doit conserver ses valeurs selon ses règles propres. Une fusion générale par virgule peut modifier l’interprétation ; testez le résultat à travers les bibliothèques et proxies réellement utilisés.

> **Erreur fréquente à comprendre** Tous les doublons se concatènent avec une virgule.
>
> La combinaison dépend de la définition du champ.

Négociation

Accept exprime une préférence de représentation. Content-Type décrit le contenu du message concerné. Accept décrit les préférences du client sur la représentation reçue. Content-Type décrit le type du contenu envoyé ; inverser ces rôles peut produire un parsing erroné ou une réponse inattendue.

> **Erreur fréquente à comprendre** Accept décrit obligatoirement le corps envoyé.
>
> Content-Type décrit le contenu du message concerné.

Rôle client hostile

Un champ de rôle client reste non fiable. Le nom ne prouve pas l'autorité du producteur. X-User-Role: admin n'établit pas un privilège. Une architecture de headers d'identité exige un intermédiaire authentifié qui retire les valeurs externes et une application qui ne fait confiance qu'à ce chemin.

> **Erreur fréquente à comprendre** X-User-Role admin suffit à accorder admin.
>
> Le nom ne prouve pas l'autorité du producteur.

API structurée

Les API structurées évitent des concaténations dangereuses. Des caractères de contrôle peuvent altérer l'interprétation. Ne concaténez pas une entrée contenant retours de ligne dans une réponse HTTP brute. Utilisez les APIs de champs et leurs validations, puis testez les valeurs malformées avant toute émission.

> **Erreur fréquente à comprendre** Concaténer tout texte client dans une réponse est sûr.
>
> Des caractères de contrôle peuvent altérer l'interprétation.

Transformation proxy

Un proxy peut reconstruire certains champs. Les intermédiaires peuvent transformer le message. Un proxy peut supprimer, ajouter ou réécrire des champs. Définissez lesquels font autorité et observez les deux côtés ; l'application ne voit pas nécessairement les mêmes octets que le navigateur.

> **Erreur fréquente à comprendre** L'application reçoit nécessairement les octets initiaux de chaque champ.
>
> Les intermédiaires peuvent transformer le message.

Trajet réel

Il faut tester les champs multiples sur le trajet réel. Les parseurs successifs peuvent appliquer des traitements différents. Des champs dupliqués peuvent être conservés ou regroupés différemment selon les composants. Une fixture unitaire ne prouve pas le comportement complet ; testez dans un environnement isolé qui reproduit la chaîne.

> **Erreur fréquente à comprendre** Un test de chaîne isolé prouve tous les comportements proxy.
>
> Les parseurs successifs peuvent appliquer des traitements différents.

Nom flatteur

Un identifiant client ne devient pas fiable par renommage. Une convention de nommage n'établit pas une confiance. Renommer une valeur client en Trusted-User ne change pas sa provenance. La confiance dépend de l'authentification du producteur et du contrôle du trajet, pas du vocabulaire choisi dans le nom.

> **Erreur fréquente à comprendre** Un préfixe X- authentifie une valeur.
>
> Une convention de nommage n'établit pas une confiance.

Absence prévue

Une absence peut demander un refus ou une valeur prévue. La politique doit définir les cas manquants. Définissez si un champ requis absent provoque un refus ou un défaut sûr. Une valeur de repli trop permissive peut ouvrir un accès lorsque le proxy ou un client omet une donnée nécessaire.

> **Erreur fréquente à comprendre** Tout champ absent peut être inventé sans contrat.
>
> La politique doit définir les cas manquants.

Atelier guidé : lire, expliquer et vérifier le code

```
Set-Cookie: lab_session=a; Path=/; HttpOnly
Set-Cookie: lab_theme=dark; Path=/
```

Les deux lignes Set-Cookie décrivent deux cookies distincts. Elles ne doivent pas être transformées mécaniquement en un champ fusionné comme n'importe quelle liste. Vérifiez le nombre de cookies effectivement reçus par le navigateur et gardez leur attribut Path et leur protection individuelle.

La combinaison des champs dépend de leur définition, pas d'une règle universelle. Les grammaires et règles de combinaison diffèrent. Un séparateur n'est pas un chiffrement. Plusieurs champs Set-Cookie peuvent être nécessaires.

Travail pratique : Inventoriez les champs utilisés pour langue, format, identité et corrélation. Pour chacun, définissez producteur autorisé, validation et comportement en cas de doublon.

Critères de correction : Envoyez des requêtes de laboratoire avec variantes contrôlées et relevez la valeur effectivement lue par le framework. Refusez les ambiguïtés pertinentes au lieu de laisser deux couches interpréter des identités différentes.

Exemple commenté

```
Accept: application/json
Content-Type: application/json
X-Request-ID: lab-42

Les trois champs n'expriment pas la même propriété.
```

Accept indique une préférence de représentation ; Content-Type décrit le contenu du message concerné ; l'identifiant de corrélation suit un contrat local.

Exercice de mise en pratique

Inventoriez les champs utilisés pour langue, format, identité et corrélation. Pour chacun, définissez producteur autorisé, validation et comportement en cas de doublon.

Voir une proposition de correction

Envoyez des requêtes de laboratoire avec variantes contrôlées et relevez la valeur effectivement lue par le framework. Refusez les ambiguïtés pertinentes au lieu de laisser deux couches interpréter des identités différentes.

- Connaître le contrat par champ.
- Ne pas faire confiance au nom du champ.
- Éviter les concaténations avec contrôles non validés.

Contrôle de connaissances

Peut-on combiner tous les champs HTTP multiples avec une virgule ?

> Réponse expliquée : Non, la règle dépend du champ — La syntaxe et la sémantique sont propres au champ.

Cours associé : 05. Champs, valeurs et frontières d'interprétation ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers

06. Content-Length et limites du corps

Mesurer les octets du bon message et réserver la marge d'encapsulation.

Comprendre le mécanisme

Content-Length décrit une longueur en octets selon le contrat du message, pas un nombre de caractères visibles. Une requête multipart comprend fichiers, séparateurs et champs de formulaire. Une limite au niveau du proxy peut donc refuser un fichier pourtant accepté par la règle métier. La réponse d'erreur possède sa propre longueur, indépendante du fichier envoyé.

Raisonner et vérifier

Rendez cohérentes les limites du proxy, de PHP, du formulaire et du lecteur d'archive. Une limite décompressée répond à un autre risque que la taille reçue sur le réseau. Mesurez chaque frontière avec des fichiers de laboratoire connus ; gardez une marge au-dessus du fichier maximum pour le multipart, sans supprimer toute borne.

Octets

Content-Length se mesure en octets. Un caractère encodé peut occuper plusieurs octets. La taille du corps se mesure en octets et peut différer du nombre de caractères. Testez du texte UTF-8 et un fichier binaire pour éviter une limite applicative basée sur une unité différente de celle du proxy.

> **Erreur fréquente à comprendre** Content-Length compte les caractères visibles.
>
> Un caractère encodé peut occuper plusieurs octets.

Multipart total

Multipart ajoute séparateurs et champs au fichier. L'enveloppe consomme aussi des octets. La requête contient fichier, séparateurs et autres champs. Une limite totale de 601000 octets refuse un fichier de 600000 si son enveloppe ajoute 2000 ; le budget doit inclure tout le message.

> **Erreur fréquente à comprendre** Le fichier peut atteindre exactement toute limite multipart sans marge.
>
> L'enveloppe consomme aussi des octets.

Première limite

Le proxy peut avoir une limite avant PHP. Chaque couche conserve sa propre limite. Inventoriez CDN, proxy, serveur web et PHP. La limite la plus précoce peut refuser avant toute validation métier ; augmenter seulement upload_max_filesize ne corrige pas un plafond nginx plus bas.

> **Erreur fréquente à comprendre** Augmenter PHP garantit le passage de tout proxy.
>
> Chaque couche conserve sa propre limite.

Décompression

La décompression doit avoir ses propres bornes. Le ratio de compression peut être très élevé. Un petit ZIP peut produire une sortie énorme. Bornez taille décompressée cumulée, ratio et écritures pendant extraction ; le seul poids compressé reçu ne protège pas disque et mémoire.

> **Erreur fréquente à comprendre** Une petite archive garantit une petite extraction.
>
> Le ratio de compression peut être très élevé.

Nombre d'entrées

Le nombre d'entrées est une ressource à borner. Beaucoup de petites entrées consomment aussi des ressources. Des milliers de petits fichiers peuvent épuiser descripteurs et temps même sous un plafond d'octets. Limitez nombre de fichiers, profondeur et opérations dans le même budget d'import.

> **Erreur fréquente à comprendre** La taille compressée borne toujours le nombre de fichiers.
>
> Beaucoup de petites entrées consomment aussi des ressources.

Messages distincts

La longueur de réponse ne mesure pas l'upload. Les messages de sens opposés ont des tailles distinctes. Content-Length de la réponse d'erreur indique son propre corps, pas celui de l'upload. Notez les tailles dans des colonnes séparées et vérifiez à quel message chaque observation appartient.

> **Erreur fréquente à comprendre** Un petit corps 413 indique un petit fichier envoyé.
>
> Les messages de sens opposés ont des tailles distinctes.

Lecture bornée

Une lecture du corps doit rester bornée. Une déclaration reçue ne suffit pas à maîtriser le flux. Ne chargez pas sans plafond le corps entier en mémoire avant de décider sa taille acceptable. Une lecture progressive doit arrêter au dépassement et nettoyer les fichiers temporaires créés jusque-là.

> **Erreur fréquente à comprendre** Un champ de taille annoncé dispense de toute borne de lecture.
>
> Une déclaration reçue ne suffit pas à maîtriser le flux.

Temps

Le temps de traitement est une limite indépendante. La complexité et les dépendances influencent la durée. Une taille admissible peut demander trop de temps de décompression ou de validation. Ajoutez une échéance globale et une procédure d'arrêt qui ne laisse pas un import partiellement publié.

> **Erreur fréquente à comprendre** Limiter les octets garantit tout temps de calcul.
>
> La complexité et les dépendances influencent la durée.

Bornes

Des essais aux frontières vérifient les limites. Il faut observer les cas proches des seuils. Essayez juste sous, à et au-dessus du plafond prévu, avec les autres champs présents. Ces cas vérifient l'unité, la marge multipart et le caractère inclusif de la règle appliquée.

> **Erreur fréquente à comprendre** Tester un seul fichier minuscule prouve la limite supérieure.
>
> Il faut observer les cas proches des seuils.

Refus propre

Un refus précoce doit laisser les données métier intactes. La gestion des effets partiels doit respecter le contrat d'import. Après dépassement, contrôlez base, fichiers temporaires et état d'import. Un message d'erreur correct ne suffit pas si des éléments extraits deviennent tout de même consultables dans le répertoire public.

> **Erreur fréquente à comprendre** Un refus de taille autorise une importation partielle silencieuse.
>
> La gestion des effets partiels doit respecter le contrat d'import.

Atelier guidé : lire, expliquer et vérifier le code

```
Fichier : 600 000 octets
Enveloppe multipart et champs : 2 000 octets
Limite totale de requête : 601 000 octets
```

Additionnez 600000 et 2000 : le corps total atteint 602000 octets, donc dépasse 601000. Le fichier seul est sous la limite mais la requête complète ne l'est pas. Reproduisez ce raisonnement aux différentes couches et gardez une validation séparée du contenu décompressé.

Cette mesure oublie les champs et séparateurs. Le fichier participe aussi au corps. La réponse est un autre message. L'encapsulation s'ajoute au fichier.

Travail pratique : Testez une taille juste sous la limite réseau et une juste au-dessus. Recommencez en multipart avec le même contenu de fichier.

Critères de correction : Relevez octets de requête et statut dans le proxy ; confirmez si l'application est appelée. Testez séparément le plafond métier puis le volume décompressé. Consignez la couche qui refuse chaque cas.

Exemple commenté

```
POST /lab/upload HTTP/1.1
Host: lab.example.test
Content-Type: text/plain
Content-Length: 4

PING
```

Le corps contient quatre octets ASCII. L'enveloppe de la requête n'est pas comprise dans cette valeur.

Exercice de mise en pratique

Testez une taille juste sous la limite réseau et une juste au-dessus. Recommencez en multipart avec le même contenu de fichier.

Voir une proposition de correction

Relevez octets de requête et statut dans le proxy ; confirmez si l'application est appelée. Testez séparément le plafond métier puis le volume décompressé. Consignez la couche qui refuse chaque cas.

- Mesurer des octets.
- Prévoir la marge multipart.
- Séparer taille reçue et décompressée.

Contrôle de connaissances

La taille multipart correspond-elle exactement à celle du seul fichier ?

> Réponse expliquée : Non, une enveloppe et des champs s’ajoutent — Le corps comprend aussi la structure multipart.

Cours associé : 06. Content-Length et limites du corps ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Length

07. Délimitation et encodage de transfert

Éviter des interprétations contradictoires entre parseurs.

Comprendre le mécanisme

En HTTP/1.1, les règles de délimitation permettent de savoir où se termine un corps sur une connexion. Transfer-Encoding et Content-Length ont des rôles distincts et des combinaisons peuvent être dangereuses. La compression d'une représentation décrite par Content-Encoding est encore une autre notion. Il faut lire la règle normative du protocole utilisé plutôt que compter sur une heuristique maison.

Raisonner et vérifier

Deux parseurs en chaîne doivent refuser ou normaliser de façon compatible les messages ambigus. Une validation seulement dans le contrôleur peut arriver trop tard si l'intermédiaire a déjà découpé la connexion autrement. Maintenez les serveurs et bibliothèques, vérifiez les règles des proxys et mesurez leur comportement sur une chaîne isolée. Documentez également la traduction entre versions HTTP.

Contenu et transfert

Transfer-Encoding et Content-Encoding ont des rôles distincts. Compression du contenu et délimitation du transfert ne se confondent pas. Content-Encoding décrit un codage de représentation tel que gzip. Transfer-Encoding en HTTP/1.1 intervient dans le transfert et sa délimitation ; les deux champs ne répondent pas à la même question.

> **Erreur fréquente à comprendre** gzip et chunked représentent toujours la même transformation.
>
> Compression du contenu et délimitation du transfert ne se confondent pas.

Parseurs en désaccord

Des parseurs incohérents créent une frontière dangereuse. La désynchronisation exploite précisément des divergences. Si un frontal et un backend ne placent pas la fin du message au même endroit, les octets restants peuvent être interprétés différemment. Le risque se situe dans leur composition, pas seulement dans le contrôleur.

> **Erreur fréquente à comprendre** Deux parseurs interprètent nécessairement toute ambiguïté pareil.
>
> La désynchronisation exploite précisément des divergences.

Chaque saut

La délimitation doit être non ambiguë sur chaque saut. Les messages successifs doivent être séparés selon le protocole. Une connexion peut être terminée puis reconstruite par un proxy. Vérifiez la délimitation et les règles de rejet à chaque saut au lieu de supposer qu'un parsing navigateur correct impose celui du backend.

> **Erreur fréquente à comprendre** Un backend peut deviner librement la fin du message.
>
> Les messages successifs doivent être séparés selon le protocole.

Conversion de version

Une conversion de version demande une politique correcte. La traduction vers un backend peut conserver des risques. Un frontal HTTP/2 peut émettre une requête HTTP/1.1 vers l'origine. Cette traduction doit conserver une délimitation cohérente et retirer ou refuser les champs incompatibles selon le protocole cible.

> **Erreur fréquente à comprendre** Passer par HTTP/2 élimine toute erreur de conversion.
>
> La traduction vers un backend peut conserver des risques.

Longueurs contradictoires

Les champs de longueur contradictoires demandent un traitement strict. Ce n'est pas un mécanisme de choix de taille. Une politique stricte doit éviter les interprétations ambiguës de longueurs et champs de transfert. Appliquez les règles du protocole dans les intermédiaires et vérifiez le refus dans un laboratoire représentatif.

> **Erreur fréquente à comprendre** Des longueurs contradictoires sont une technique de négociation.
>
> Ce n'est pas un mécanisme de choix de taille.

Environnement isolé

Les tests de parseur appartiennent à un laboratoire isolé. Ces essais peuvent perturber des requêtes tierces. Les tests de frontières de messages peuvent perturber des connexions mutualisées. Utilisez une chaîne dédiée avec données factices, sans utilisateurs tiers, et conservez versions et configuration exactes.

> **Erreur fréquente à comprendre** Une cible partagée convient aux essais de désynchronisation sans accord.
>
> Ces essais peuvent perturber des requêtes tierces.

Corps par morceaux

Les budgets s'appliquent aussi aux corps transmis par morceaux. Le mode de transfert ne supprime pas les limites de ressources. Un transfert chunked ne rend pas le corps sans limite. Comptez les octets reçus et appliquez temps, taille et ressources pendant lecture, y compris lorsque la taille totale n'est pas annoncée initialement.

> **Erreur fréquente à comprendre** Un transfert chunked autorise un corps infini.
>
> Le mode de transfert ne supprime pas les limites de ressources.

Avant le framework

Le framework ne contrôle pas tous les parseurs amont. Les composants et leurs versions doivent être examinés séparément. Une vérification PHP arrive après le parsing de ses prédécesseurs. Maintenez et configurez les composants amont ; le contrôleur ne peut pas réparer rétroactivement une interprétation divergente déjà utilisée.

> **Erreur fréquente à comprendre** Corriger une concaténation PHP corrige tous les parseurs réseau.
>
> Les composants et leurs versions doivent être examinés séparément.

Réutilisation

Une connexion réutilisée amplifie certains écarts de délimitation. La persistance peut relier plusieurs messages sur le même canal. Sur une connexion persistante, des octets mal attribués peuvent affecter l'échange suivant. Testez la réutilisation dans la chaîne isolée et vérifiez que le refus ferme ou traite correctement l'état du canal.

> **Erreur fréquente à comprendre** Chaque octet supplémentaire est toujours isolé dans une nouvelle connexion.
>
> La persistance peut relier plusieurs messages sur le même canal.

Traces sûres

Les journaux doivent aider sans enregistrer de secrets. Des traces de laboratoire et métadonnées peuvent suffire. Conservez décisions de parseur, corrélation et tailles nécessaires sans enregistrer cookies ou corps sensibles. Les captures de laboratoire peuvent être détaillées parce que leurs données sont conçues pour être publiques.

> **Erreur fréquente à comprendre** Une investigation impose tous les corps réels en clair.
>
> Des traces de laboratoire et métadonnées peuvent suffire.

Atelier guidé : lire, expliquer et vérifier le code

```
Content-Encoding: gzip
Transfer-Encoding: chunked

Extrait HTTP/1.1 de laboratoire, pas une politique universelle de proxy.
```

L'extrait montre simultanément un contenu gzip et un transfert chunked dans un contexte HTTP/1.1. Expliquez les deux couches sans en déduire une politique universelle applicable à HTTP/2. Pour valider la sécurité, il faut examiner la délimitation et les conversions de toute la chaîne réelle.

Ils ne décident pas des permissions. gzip nomme un codage, pas un nombre. Le contenu compressé peut être transporté selon une délimitation séparée. Aucun de ces champs ne signifie TLS.

Travail pratique : Inventoriez les parseurs traversés par votre requête de laboratoire et leurs versions. Recherchez les règles documentées concernant les délimitations ambiguës.

Critères de correction : Comparez configurations et contrats, puis utilisez des tests de non-régression dans un environnement dédié. Vérifiez le refus avant tout traitement métier et la fermeture de la connexion si la politique l'exige.

Exemple commenté

```
Content-Encoding: gzip

Décrit la représentation compressée.
Transfer-Encoding : mécanisme de transfert distinct en HTTP/1.1.
```

Décompresser une représentation ne résout pas une ambiguïté de délimitation de la connexion. Les deux traitements doivent être identifiés.

Exercice de mise en pratique

Inventoriez les parseurs traversés par votre requête de laboratoire et leurs versions. Recherchez les règles documentées concernant les délimitations ambiguës.

Voir une proposition de correction

Comparez configurations et contrats, puis utilisez des tests de non-régression dans un environnement dédié. Vérifiez le refus avant tout traitement métier et la fermeture de la connexion si la politique l'exige.

- Distinguer transfert et représentation.
- Refuser les ambiguïtés.
- Tester la chaîne complète.

Contrôle de connaissances

Content-Encoding et Transfer-Encoding expriment-ils la même propriété ?

> Réponse expliquée : Non, représentation et transfert sont distincts — Les contrats portent sur des couches différentes.

Cours associé : 07. Délimitation et encodage de transfert ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Transfer-Encoding

08. Désynchronisation et request smuggling

Défendre la frontière entre intermédiaires et serveur applicatif.

Comprendre le mécanisme

Une désynchronisation apparaît lorsque des composants d'une chaîne ne s'accordent pas sur les frontières ou le sens des requêtes. Sur des connexions réutilisées, cette divergence peut affecter une demande ultérieure ou contourner une hypothèse du frontal. Le risque dépend de la chaîne entière, y compris des traductions HTTP/2 vers HTTP/1.1 ; le protocole vu par le navigateur ne décrit pas nécessairement le dernier saut.

Raisonner et vérifier

La défense privilégie des parseurs cohérents, à jour et stricts, ainsi qu'un traitement explicite des messages non conformes. Ne considérez pas une règle WAF ou un mot-clé bloqué comme une preuve complète. Les essais de délimitation doivent rester dans une chaîne de test dédiée, afin d'observer précisément les connexions et les effets sans perturber d'autres utilisateurs.

Frontières divergentes

Le risque dépend d'interprétations différentes des messages. Le problème porte ici sur les frontières des requêtes. Le cœur du problème est un désaccord sur les limites du message. Un frontal qui termine à l'octet 100 et un backend à 140 ne traitent pas le même découpage des échanges.

> **Erreur fréquente à comprendre** La désynchronisation signifie seulement une horloge incorrecte.
>
> Le problème porte ici sur les frontières des requêtes.

Système complet

La chaîne complète proxy/backend compte. L'écart peut se situer entre composants. Reproduisez la version exacte du proxy, sa configuration et le serveur d'origine. Tester seulement une bibliothèque PHP laisse hors périmètre les interprétations déjà faites par CDN et reverse proxy avant son intervention.

> **Erreur fréquente à comprendre** Un seul parseur testé prouve toute la chaîne.
>
> L'écart peut se situer entre composants.

Canal mutualisé

Une connexion mutualisée peut toucher un autre échange. Les frontières incorrectes peuvent affecter des messages suivants. Une connexion partagée peut faire rencontrer les octets restants d'un échange avec une autre demande. Le laboratoire doit donc contrôler l'usage du canal et ne jamais exposer d'autres visiteurs aux tests.

> **Erreur fréquente à comprendre** Le risque est toujours limité à l'émetteur du test.
>
> Les frontières incorrectes peuvent affecter des messages suivants.

Mises à jour

Les mises à jour des intermédiaires comptent. La correction doit viser la couche concernée. Des corrections de parseur et de traduction peuvent changer les refus. Gardez les intermédiaires maintenus et rejouez les fixtures pertinentes lors d'une évolution, sans supposer que l'application seule porte la réparation.

> **Erreur fréquente à comprendre** Changer un bouton du formulaire corrige le parseur proxy.
>
> La correction doit viser la couche concernée.

Ambiguïtés refusées

Le refus d'ambiguïtés réduit le risque. Des interprétations divergentes doivent être évitées. Adoptez des règles strictes compatibles avec les protocoles utilisés plutôt que laisser chaque composant deviner. Vérifiez le comportement réel en présence de données malformées, y compris l'état de connexion après rejet.

> **Erreur fréquente à comprendre** Accepter toutes les longueurs contradictoires améliore la sécurité.
>
> Des interprétations divergentes doivent être évitées.

Versions différentes

Les conversions protocolaires doivent être vérifiées. La traduction peut introduire ou conserver une ambiguïté. L'utilisation de HTTP/2 côté navigateur n'exclut pas HTTP/1.1 derrière un proxy. Dessinez les traductions et testez leurs frontières ; la version du premier saut ne décrit pas tous les autres.

> **Erreur fréquente à comprendre** HTTP/2 en façade garantit tout backend HTTP/1.1 sûr.
>
> La traduction peut introduire ou conserver une ambiguïté.

Isolation

Un laboratoire doit isoler les autres utilisateurs. Le test ne doit pas perturber des tiers. Les expériences doivent utiliser ressources et connexions dédiées avec contenus factices. Prévoyez une remise à zéro entre cas afin qu'une tentative précédente ne fausse pas la lecture de la suivante.

> **Erreur fréquente à comprendre** Une démonstration peut utiliser arbitrairement une file de production.
>
> Le test ne doit pas perturber des tiers.

Reproductibilité

La reproduction doit préciser les versions. Versions et options influencent les comportements. Enregistrez les versions exactes, les options et la topologie avec les résultats. Un refus observé sur une chaîne différente ne démontre pas la même propriété pour celle effectivement déployée chez l'hébergeur.

> **Erreur fréquente à comprendre** Le nom générique nginx décrit toute configuration possible.
>
> Versions et options influencent les comportements.

Rejet observé

Un rejet doit être vérifié à chaque frontière pertinente. L'état de connexion et les couches atteintes demandent des observations. Vérifiez quelle couche refuse et si le backend reçoit quelque chose. Un statut d'erreur final n'exclut pas une interprétation partielle différente ; les journaux corrélés et mesures de laboratoire complètent le résultat client.

> **Erreur fréquente à comprendre** Un refus visuel suffit à prouver toutes les connexions vidées.
>
> L'état de connexion et les couches atteintes demandent des observations.

Politique cohérente

Une politique cohérente est une propriété du système. Les règles ont un périmètre et des limites. La sécurité attendue dépend de l'accord des composants sur les messages. Attribuez un responsable aux changements de proxy et de backend afin que cette propriété reste vérifiée lors des releases.

> **Erreur fréquente à comprendre** Une règle WAF universelle garantit tous les parseurs futurs.
>
> Les règles ont un périmètre et des limites.

Atelier guidé : lire, expliquer et vérifier le code

```
{
  "frontendInterpretation": "message ends at offset 100",
  "backendInterpretation": "message ends at offset 140",
  "sharedConnection": true
}
```

Les offsets différents constituent l'indice principal ; la connexion partagée augmente les conséquences possibles sur les échanges suivants. Il ne suffit pas de répondre que le JSON d'observation est valide. Expliquez la divergence puis définissez une reproduction isolée et les points de mesure nécessaires pour vérifier le refus cohérent.

Les credentials ne corrigent pas cette divergence. Des frontières différentes sur un canal réutilisé peuvent désynchroniser les échanges. Le hasard n'établit pas une règle commune. Le protocole doit définir la décision, pas une heuristique libre.

Travail pratique : Rédigez une procédure de revue des versions, de la normalisation et de la réutilisation des connexions sur votre chaîne de laboratoire.

Critères de correction : Confrontez les journaux de chaque saut sur un corpus contrôlé, vérifiez les refus et ajoutez des tests après mise à jour. Conservez topologie, versions et conditions d'essai : un résultat sur un serveur seul ne couvre pas la chaîne.

Exemple commenté

```
Client de laboratoire → frontal A → connexion réutilisée → serveur B
Invariant : A et B identifient les mêmes frontières de requête.
```

Le schéma décrit l'invariant à vérifier, sans fournir de message ambigu à envoyer vers un service partagé.

Exercice de mise en pratique

Rédigez une procédure de revue des versions, de la normalisation et de la réutilisation des connexions sur votre chaîne de laboratoire.

Voir une proposition de correction

Confrontez les journaux de chaque saut sur un corpus contrôlé, vérifiez les refus et ajoutez des tests après mise à jour. Conservez topologie, versions et conditions d'essai : un résultat sur un serveur seul ne couvre pas la chaîne.

- Inclure les traductions de protocole.
- Préférer cohérence de parseurs aux filtres de texte.
- Isoler le laboratoire.

Contrôle de connaissances

Quel désaccord est au cœur d'une désynchronisation HTTP ?

> Réponse expliquée : L'interprétation des frontières de messages entre composants — Les composants peuvent découper une connexion différemment.

Cours associé : 08. Désynchronisation et request smuggling ↗

Documentation de référence ↗ https://portswigger.net/web-security/request-smuggling

09. HTTP/2 : multiplexage et contraintes partagées

Séparer concurrence logique et ressources de connexion.

Comprendre le mécanisme

HTTP/2 transporte plusieurs flux sur une connexion avec un encodage binaire et une compression des champs. Ce multiplexage évite certaines contraintes du modèle HTTP/1.1, mais les flux partagent encore des ressources. Sur TCP, une perte de segment peut retarder la progression de plusieurs flux. L'application conserve ses propres files, limites et transactions : un nouveau transport ne corrige pas automatiquement les lenteurs métier.

Raisonner et vérifier

Définissez des limites de concurrence et observez annulations, tailles de champs et coût par requête. Une seule connexion peut porter de nombreux travaux ; limiter uniquement le nombre de connexions ne borne donc pas toute la charge. Mesurez les budgets au niveau des flux et du service, en tenant compte des intermédiaires qui terminent ou traduisent HTTP/2.

Streams concurrents

HTTP/2 peut multiplexer des streams sur une connexion. Plusieurs échanges peuvent partager le transport. Une connexion peut transporter plusieurs échanges simultanés. Distinguez identifiant de connexion et identifiant de stream dans l'observation afin d'attribuer latence, annulation et budgets à la bonne opération.

> **Erreur fréquente à comprendre** HTTP/2 impose une connexion TCP par ressource.
>
> Plusieurs échanges peuvent partager le transport.

Compteurs adaptés

Une limite de connexions ne borne pas tous les streams. La concurrence existe aussi dans les streams. Dix connexions autorisées peuvent représenter bien plus de dix demandes actives. Limitez aussi streams et travail applicatif selon la capacité réelle, en particulier pour les opérations lentes ou coûteuses.

> **Erreur fréquente à comprendre** Compter les connexions suffit à compter les requêtes actives.
>
> La concurrence existe aussi dans les streams.

Métier inchangé

HTTP/2 conserve les contrôles métier de l'application. Le transport ne décide pas des permissions. Multiplexer ne remplace ni transaction ni autorisation. Une réservation concurrente demande toujours une règle de capacité et un contrôle de propriétaire, même si ses octets arrivent sur un transport plus efficace.

> **Erreur fréquente à comprendre** Le multiplexage remplace les autorisations.
>
> Le transport ne décide pas des permissions.

Perte TCP

Le TCP partagé peut affecter plusieurs streams lors d'une perte. Les échanges partagent le canal de transport. HTTP/2 partage TCP ; certaines pertes peuvent retarder plusieurs streams au niveau du transport. Ne promettez pas l'absence de toute attente parce que les échanges HTTP sont logiquement multiplexés.

> **Erreur fréquente à comprendre** Chaque stream possède nécessairement son TCP indépendant.
>
> Les échanges partagent le canal de transport.

Ressources

Les limites de ressources restent nécessaires. Le protocole ne crée pas de capacité infinie. Bornez champs, mémoire, opérations actives et temps global. Un client peut créer du travail sans ouvrir beaucoup de connexions ; mesurez aussi le coût d'un stream interrompu après démarrage.

> **Erreur fréquente à comprendre** HTTP/2 rend la mémoire du serveur illimitée.
>
> Le protocole ne crée pas de capacité infinie.

Sauts internes

Le trajet peut changer de version derrière le proxy. La version négociée ne décrit qu'un saut. Le proxy peut traduire vers une autre version. Vérifiez la configuration backend et les budgets de ses connexions ; une optimisation frontale peut déplacer la file d'attente plutôt que la supprimer.

> **Erreur fréquente à comprendre** HTTP/2 côté client prouve HTTP/2 jusqu'à PHP.
>
> La version négociée ne décrit qu'un saut.

Représentation

Les en-têtes ont une représentation différente sur le fil. Le protocole utilise des frames et une gestion spécifique des champs. HTTP/2 utilise des trames et une représentation de champs différente de la syntaxe textuelle HTTP/1.1. Une trace pédagogique en lignes reste une vue logique et ne décrit pas tous les octets du réseau.

> **Erreur fréquente à comprendre** Une capture textuelle HTTP/1.1 décrit directement tout encodage HTTP/2.
>
> Le protocole utilise des frames et une gestion spécifique des champs.

Sémantique conservée

Une requête doit conserver une sémantique de méthode correcte. Le changement de version ne redéfinit pas cette sûreté. GET reste une lecture sûre selon sa sémantique, et une mutation exige toujours un contrat adapté. Le changement de version ne justifie pas une annulation métier cachée dans un lien GET.

> **Erreur fréquente à comprendre** HTTP/2 autorise GET à effectuer toute mutation demandée.
>
> Le changement de version ne redéfinit pas cette sûreté.

Annulations

Les annulations de streams peuvent consommer des ressources. Du travail peut précéder l'annulation. Une remise à zéro de stream peut arriver après consommation de CPU ou début de traitement. Appliquez des protections contre l'abus et vérifiez la propagation d'annulation sans supposer qu'elle annule tout effet durable.

> **Erreur fréquente à comprendre** Un stream annulé coûte nécessairement zéro au serveur.
>
> Du travail peut précéder l'annulation.

Mesures

Les mesures doivent distinguer connexion et opération. Plusieurs étapes et attentes participent à la durée. Rapportez connexions, streams actifs, demandes applicatives et latences séparément. Une baisse du nombre de sockets ne prouve pas une réduction du nombre d'opérations ni de la charge de la base.

> **Erreur fréquente à comprendre** La latence TCP seule prouve le temps métier de chaque requête.
>
> Plusieurs étapes et attentes participent à la durée.

Atelier guidé : lire, expliquer et vérifier le code

```
HTTP version: 2
Active TCP connections: 1
Active streams on this connection: 40
Policy shown: max_connections = 10
```

Une connexion TCP porte quarante streams alors que la politique ne montre qu'un plafond de connexions. Ce plafond ne suffit donc pas à borner le travail simultané. Ajoutez limites adaptées et mesurez le nombre d'opérations réellement actives dans l'application et ses dépendances.

Le compteur de connexions ne borne pas à lui seul tous les streams. Le multiplexage permet le partage de la connexion. Cette affirmation contredit le modèle montré. Aucune règle de streams n'est définie.

Travail pratique : Mesurez trois parcours concurrents dont un rapport lent, avec et sans annulation du client.

Critères de correction : Relevez latence par flux et travail réellement interrompu côté application. Vérifiez les limites de requêtes simultanées et le comportement du frontal sous charge de laboratoire. Distinguez attente réseau et saturation métier.

Exemple commenté

```
Connexion TCP
  flux 1 : catalogue
  flux 3 : réservation
  flux 5 : rapport
Ressources partagées : réseau, mémoire et capacité serveur.
```

La séparation des flux est logique. Elle ne réserve pas un cœur ou une transaction indépendante sans limite pour chaque demande.

Exercice de mise en pratique

Mesurez trois parcours concurrents dont un rapport lent, avec et sans annulation du client.

Voir une proposition de correction

Relevez latence par flux et travail réellement interrompu côté application. Vérifiez les limites de requêtes simultanées et le comportement du frontal sous charge de laboratoire. Distinguez attente réseau et saturation métier.

- Limiter aussi les flux.
- Observer les annulations.
- Ne pas confondre multiplexage et capacité illimitée.

Contrôle de connaissances

HTTP/2 garantit-il que toute attente métier disparaît ?

> Réponse expliquée : Non, l'application garde ses limites et dépendances — Le transport ne supprime pas les files applicatives.

Cours associé : 09. HTTP/2 : multiplexage et contraintes partagées ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Glossary/HTTP_2

10. HTTP/3 : QUIC, pertes et garanties restantes

Identifier les gains du transport sans leur attribuer des garanties applicatives.

Comprendre le mécanisme

HTTP/3 utilise QUIC comme transport au lieu d'un transport TCP classique pour ses échanges. Des flux de transport distincts réduisent certaines formes de blocage entre eux en cas de perte réseau. Ce changement ne supprime pas toutes les dépendances : le contrôle de congestion, le traitement applicatif et les relations entre données peuvent encore retarder des opérations. La sémantique des méthodes et des statuts reste celle de HTTP.

Raisonner et vérifier

Vérifiez les capacités réelles du client, du frontal et des réseaux traversés, ainsi que la stratégie de repli. Le succès d'une connexion de test n'est pas une preuve de performance universelle. Mesurez pertes, latence, consommation et comportement lors d'un changement de chemin. Continuez à appliquer authentification, autorisation, limites de travail et idempotence : QUIC ne remplace pas ces contrôles.

Transport QUIC

HTTP/3 utilise QUIC comme transport. Le transport et son multiplexage diffèrent. HTTP/3 porte les échanges HTTP sur QUIC. La version effectivement négociée doit être observée dans la trace client ; une capacité annoncée par le serveur ne prouve pas son usage par cette demande.

> **Erreur fréquente à comprendre** HTTP/3 consiste en HTTP/2 sur TCP renommé.
>
> Le transport et son multiplexage diffèrent.

Blocages entre streams

Des streams QUIC évitent certains blocages entre streams liés à TCP. Le travail serveur et les dépendances restent des causes d'attente. QUIC évite certains blocages entre streams associés à la livraison ordonnée TCP. Cela ne supprime pas les attentes de congestion, de dépendance applicative ou de verrou en base de données.

> **Erreur fréquente à comprendre** QUIC supprime toutes les attentes dans une application.
>
> Le travail serveur et les dépendances restent des causes d'attente.

Autorisation

Les règles d'autorisation restent applicatives. La protection du transport ne décide pas des droits métier. La demande qui arrive via HTTP/3 doit encore être contrôlée sur acteur et ressource. Une amélioration du transport ne modifie pas les règles de propriété d'une réservation ou d'un document.

> **Erreur fréquente à comprendre** QUIC valide automatiquement les propriétaires d'objets.
>
> La protection du transport ne décide pas des droits métier.

Budgets

Une limite de taille reste nécessaire. Les ressources de calcul et stockage restent finies. Conservez limites de taille et de travail par opération. Le transport ne transforme pas un upload géant ou une archive très coûteuse à extraire en entrée admissible pour l'application.

> **Erreur fréquente à comprendre** HTTP/3 accepte sans coût tout volume.
>
> Les ressources de calcul et stockage restent finies.

Chemin réseau

Le chemin réseau doit supporter le transport choisi. Les politiques et équipements peuvent affecter le passage. Pare-feu, équipements et clients doivent supporter le transport choisi. Vérifiez les conditions réelles de connexion avant d'attribuer une différence de performance au seul code du serveur d'origine.

> **Erreur fréquente à comprendre** Annoncer HTTP/3 garantit sa disponibilité sur tout réseau.
>
> Les politiques et équipements peuvent affecter le passage.

Repli observé

Le repli doit être observé lors des tests. Le client peut avoir utilisé une autre version. Un service peut annoncer h3 mais cette requête négocier h2. Rapportez le protocole mesuré et les causes de repli pertinentes afin que le résultat reste interprétable et reproductible.

> **Erreur fréquente à comprendre** Un affichage réussi prouve toujours HTTP/3.
>
> Le client peut avoir utilisé une autre version.

Canal et donnée

Une connexion sécurisée ne rend pas toute donnée métier fiable. L'identité et les entrées doivent toujours être validées. Un canal protégé peut transporter une entrée malformée ou hostile envoyée par un client authentique. Validation et contrôle métier restent des responsabilités propres à l'application après réception.

> **Erreur fréquente à comprendre** Chiffrer un message garantit que ses affirmations sont vraies.
>
> L'identité et les entrées doivent toujours être validées.

Corrélation

Les erreurs doivent être corrélées au bon saut. Un proxy peut traduire les protocoles. Le proxy public et l'origine peuvent employer des versions différentes. Conservez le saut et l'opération associés à l'erreur pour éviter de diagnostiquer le mauvais transport ou le mauvais composant.

> **Erreur fréquente à comprendre** La version en façade décrit toutes les connexions internes.
>
> Un proxy peut traduire les protocoles.

Performance mesurée

La performance dépend des conditions testées. Topologie et charge influencent les résultats. Comparez avec mêmes données, charge et conditions de perte représentatives. Une mesure locale unique ne démontre pas une amélioration pour tous les réseaux ou tous les profils de client.

> **Erreur fréquente à comprendre** HTTP/3 garantit une meilleure latence dans tous les cas.
>
> Topologie et charge influencent les résultats.

Caches

Les caches conservent un contrat de représentation. Le changement de transport n'élimine pas la sémantique du cache. Fraîcheur, portée et variantes restent définies par la sémantique HTTP. Migrer de transport ne corrige pas une réponse personnelle marquée public ou une clé de cache qui oublie le tenant.

> **Erreur fréquente à comprendre** HTTP/3 interdit tout cache HTTP.
>
> Le changement de transport n'élimine pas la sémantique du cache.

Atelier guidé : lire, expliquer et vérifier le code

```
Advertised capability: h3
Negotiated protocol for this request: h2
Status: 200
```

La capacité annoncée est h3 mais le protocole négocié est h2. Cette demande ne fournit donc aucune mesure directe de HTTP/3. Relevez le protocole effectif, vérifiez le chemin réseau et comparez ensuite des scénarios contrôlés sans attribuer au transport une garantie de reprise métier.

L'annonce ne prouve pas le protocole utilisé. Le statut ne décrit pas le transport négocié. Le contexte peut conduire à une autre version. La négociation mesurée porte sur h2.

Travail pratique : Préparez une comparaison contrôlée d'un parcours en HTTP/2 et HTTP/3 avec une perte réseau simulée et un repli autorisé.

Critères de correction : Conservez le même contenu et la même charge, vérifiez le protocole effectivement négocié et comparez les mesures. Documentez les conditions au lieu de promettre une accélération absolue pour tous les utilisateurs.

Exemple commenté

```
Navigateur → HTTP/3 / QUIC → frontal
Frontal → protocole interne configuré → application
Même contrat métier, transport externe différent.
```

L'activation externe de HTTP/3 ne dit rien, à elle seule, sur le protocole utilisé entre le frontal et le serveur PHP.

Exercice de mise en pratique

Préparez une comparaison contrôlée d'un parcours en HTTP/2 et HTTP/3 avec une perte réseau simulée et un repli autorisé.

Voir une proposition de correction

Conservez le même contenu et la même charge, vérifiez le protocole effectivement négocié et comparez les mesures. Documentez les conditions au lieu de promettre une accélération absolue pour tous les utilisateurs.

- Vérifier la négociation réelle.

- Conserver les contrôles métier.
- Mesurer le repli et les pertes.

Contrôle de connaissances

Quel changement caractérise HTTP/3 ?

> Réponse expliquée : L'usage de QUIC comme transport — La couche de transport évolue.

Cours associé : 10. HTTP/3 : QUIC, pertes et garanties restantes ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Glossary/HTTP_3

11. Origine, site et URL : trois notions distinctes

Calculer la frontière pertinente avant de configurer une politique.

Comprendre le mécanisme

Une origine est définie par schéma, hôte et port selon les règles de l'URL. Deux adresses qui semblent appartenir à la même entreprise peuvent donc être d'origines différentes. La notion de site utilisée par certains contrôles de cookies ne se confond pas avec l'origine ; elle dépend notamment du domaine enregistrable et du schéma. Une même page peut enfin manipuler plusieurs URL sans leur accorder la même confiance.

Raisonner et vérifier

Écrivez les URL complètes de vos interfaces et API. Changer un sous-domaine, un port de développement ou HTTP vers HTTPS peut changer une frontière. Ne comparez pas des hôtes avec une simple recherche de sous-chaîne : une ressemblance visuelle n'établit pas l'appartenance au domaine autorisé. Pour les identifiants et redirections, utilisez les parseurs d'URL et une politique de destinations explicite.

Triplet d'origine

Une origine comporte schéma, hôte et port. Le chemin distingue des ressources, pas l'origine. Comparez schéma, hôte et port après parsing et normalisation appropriée. Le chemin d'une URL ne change pas son origine ; en revanche un port non équivalent ou un autre hôte la change.

> **Erreur fréquente à comprendre** Le chemin fait partie du triplet d'origine.
>
> Le chemin distingue des ressources, pas l'origine.

Ports distincts

Deux ports différents peuvent donner deux origines. Le port participe à la comparaison d'origine. https://api.example.test:443 et le même hôte sur 8443 ont deux origines. Une politique qui ignore le port élargit donc la confiance au-delà de la destination explicitement prévue.

> **Erreur fréquente à comprendre** Même hôte implique toujours même origine.
>
> Le port participe à la comparaison d'origine.

Site distinct

Site et origine ne sont pas des synonymes. Le site possède une granularité différente. Deux sous-domaines peuvent être same-site sans être same-origin. Cette différence compte lorsque vous raisonnez sur SameSite et SOP ; ne transférez pas les garanties de l'un à l'autre.

> **Erreur fréquente à comprendre** Même site garantit même origine.
>
> Le site possède une granularité différente.

Schéma

Le schéma peut changer l'origine. Le schéma appartient au triplet. http et https ne définissent pas la même origine même avec le même nom. Une liste autorisée doit préciser explicitement les schémas attendus plutôt qu'admettre implicitement un trajet non protégé.

> **Erreur fréquente à comprendre** HTTP et HTTPS sur le même hôte ont toujours même origine.
>
> Le schéma appartient au triplet.

Parsing structuré

Une URL doit être parsée avant comparaison structurée. Des hôtes trompeurs peuvent contenir le texte attendu. Une recherche de example.test dans une chaîne peut accepter un hôte hostile ou un nom d'utilisateur trompeur. Parsez l'URL puis comparez les composants et les règles de sous-domaines effectivement admises.

> **Erreur fréquente à comprendre** Une recherche de sous-chaîne suffit pour autoriser un hôte.
>
> Des hôtes trompeurs peuvent contenir le texte attendu.

Sous-domaines

Un sous-domaine n'est pas automatiquement la même origine. L'hôte exact compte pour l'origine. portal.example.test et admin.example.test ont des hôtes différents. Un contrôle d'origine peut donc les distinguer même si leur déploiement ou leurs cookies partagent certains éléments communs de leur infrastructure.

> **Erreur fréquente à comprendre** Tous les sous-domaines partagent nécessairement l'origine.
>
> L'hôte exact compte pour l'origine.

Port implicite

Un port par défaut doit être traité selon la normalisation URL. La représentation textuelle peut différer d'une URL normalisée. Pour HTTPS, un port absent et le port par défaut peuvent être normalisés de manière équivalente. Utilisez un parseur conforme au contexte au lieu de comparer naïvement les chaînes complètes.

> **Erreur fréquente à comprendre** Comparer uniquement les chaînes brutes règle toutes les équivalences.
>
> La représentation textuelle peut différer d'une URL normalisée.

Liste maîtrisée

Le serveur doit maîtriser sa liste d'origines admises. Une apparence textuelle n'est pas une politique exacte. La configuration serveur définit les origines admises. Ne reflétez pas une valeur reçue simplement parce qu'elle semble interne ; testez chaque règle avec un domaine voisin non autorisé.

> **Erreur fréquente à comprendre** Toute origine terminant visuellement par un mot connu est fiable.
>
> Une apparence textuelle n'est pas une politique exacte.

Origine et objet

Une origine admise ne prouve pas la propriété d'un objet. Le contrôle de lecture navigateur reste distinct de l'autorisation. Une demande issue du portail autorisé peut encore viser la facture d'un tiers. Le contrôle d'origine ne remplace pas authentification et autorisation par tenant ou propriétaire sur la ressource.

> **Erreur fréquente à comprendre** CORS donne tous les droits métier à l'origine admise.
>
> Le contrôle de lecture navigateur reste distinct de l'autorisation.

Userinfo

Le nom d'utilisateur d'une URL ne définit pas son hôte. Les composants d'URL ont des positions et sens précis. Dans https://example.test@hostile.test, l'hôte est hostile.test. Une vérification textuelle peut confondre ces composants ; inspectez le résultat parsé et refusez les formes inutiles selon votre contrat.

> **Erreur fréquente à comprendre** Tout texte avant arobase est l'hôte de confiance.
>
> Les composants d'URL ont des positions et sens précis.

Atelier guidé : lire, expliquer et vérifier le code

```
https://api.example.test:443/orders
https://api.example.test:8443/orders
```

Les URL ne diffèrent que par le port, mais ce composant participe à l'origine. Comparez-les après normalisation des ports par défaut, puis ajoutez une URL avec un nom d'utilisateur trompeur. L'objectif est d'expliquer une décision structurée plutôt qu'une simple recherche de sous-chaîne.

L'hôte n'est qu'une composante. Le schéma ne retire pas le port du modèle. Le triplet d'origine comprend le port. Le chemin n'efface pas cette différence.

Travail pratique : Construisez une table de six URL de laboratoire et indiquez lesquelles sont same-origin, puis étudiez séparément la portée de site des cookies.

Critères de correction : Normalisez avec un parseur, comparez les composantes pertinentes et documentez les ports implicites. Ajoutez des cas de sous-domaines, de schémas et de domaines ressemblants ; vérifiez les résultats dans le navigateur.

Exemple commenté

```
https://app.example.test:443
https://api.example.test:443
https://app.example.test:8443
Trois configurations à comparer selon schéma, hôte et port.
```

L'adresse de l'API n'a pas la même origine que celle de l'interface. Un port différent constitue aussi un point de comparaison pertinent.

Exercice de mise en pratique

Construisez une table de six URL de laboratoire et indiquez lesquelles sont same-origin, puis étudiez séparément la portée de site des cookies.

Voir une proposition de correction

Normalisez avec un parseur, comparez les composantes pertinentes et documentez les ports implicites. Ajoutez des cas de sous-domaines, de schémas et de domaines ressemblants ; vérifiez les résultats dans le navigateur.

- Ne pas confondre site et origine.
- Comparer des composantes parsées.
- Inclure schéma et port.

Contrôle de connaissances

Deux sous-domaines différents sont-ils nécessairement same-origin ?

> Réponse expliquée : Non, l'hôte participe à l'origine — L'appartenance organisationnelle ne remplace pas le tuple d'origine.

Cours associé : 11. Origine, site et URL : trois notions distinctes ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Glossary/Origin

12. Same-origin policy : lecture et envoi

Décrire ce que le navigateur limite réellement.

Comprendre le mécanisme

La politique de même origine borne plusieurs accès entre documents ou ressources d'origines différentes. Elle protège notamment certaines lectures et interactions, mais n'interdit pas uniformément tout envoi interorigine. Une image distante, une navigation ou un formulaire peuvent encore produire du trafic dans les conditions prévues par le navigateur. Ce modèle explique pourquoi l'absence d'une permission de lecture CORS ne prouve pas qu'une action serveur n'a pas été déclenchée.

Raisonner et vérifier

Construisez votre modèle de menace avec deux questions distinctes : la demande part-elle, et le script peut-il exploiter la réponse ? Un serveur doit autoriser les mutations même si la page appelante ne pourra pas lire leur résultat. Les mécanismes CSRF, l'authentification et la politique de cookies complètent cette frontière. Testez avec deux origines de laboratoire plutôt qu'une page unique.

Frontière de lecture

La SOP limite certains accès entre origines. Lecture et émission de requête ont des règles distinctes. La SOP limite certains accès entre origines depuis le navigateur. Elle ne constitue pas une authentification du client ni une interdiction universelle pour les demandes qui atteignent le serveur.

> **Erreur fréquente à comprendre** La SOP interdit tout envoi inter-origines.
>
> Lecture et émission de requête ont des règles distinctes.

Ressource intégrée

Une image externe peut être demandée sans lecture libre du contenu. Chargement et exposition au script ne sont pas équivalents. Une image externe peut être chargée sans donner au script une lecture libre de son contenu. Distinguez déclenchement réseau, affichage et accès programmatique aux données de la réponse.

> **Erreur fréquente à comprendre** Une requête d'image prouve l'accès JavaScript à tous ses octets.
>
> Chargement et exposition au script ne sont pas équivalents.

Formulaire externe

Un formulaire peut déclencher une requête inter-origines. Le serveur doit protéger les mutations pertinentes. Un formulaire hébergé ailleurs peut envoyer une demande vers votre service selon les règles navigateur. Si une identité est jointe, l'effet doit être contrôlé même lorsque la réponse reste illisible au site initiateur.

> **Erreur fréquente à comprendre** La SOP suffit à empêcher tout CSRF.
>
> Le serveur doit protéger les mutations pertinentes.

Partage CORS

CORS peut autoriser certaines lectures contrôlées. L'autorisation porte sur un contrat précis de partage. CORS peut autoriser certaines lectures que la politique d'origine limiterait autrement. Cette permission est évaluée par le navigateur et ne remplace pas les contrôles serveur sur acteur et ressource.

> **Erreur fréquente à comprendre** CORS désactive toutes les protections du navigateur.
>
> L'autorisation porte sur un contrat précis de partage.

Client serveur

Un client serveur n'est pas régi comme un script navigateur par la SOP. Les clients directs demandent des contrôles serveur. Un script curl ou un backend peut contacter l'API sans appliquer la SOP du navigateur. Protégez donc les données par une authentification et des règles d'accès réelles côté serveur.

> **Erreur fréquente à comprendre** La SOP est le pare-feu de toutes les API clientes.
>
> Les clients directs demandent des contrôles serveur.

Contrôle API

L'authentification doit être vérifiée côté API. Une politique de lecture n'établit pas l'identité. L'API doit vérifier credentials et droits sur chaque opération sensible. Le fait qu'un frontend soit déployé sur une origine connue ne rend pas toutes les demandes reçues authentiques ou autorisées.

> **Erreur fréquente à comprendre** Bloquer une lecture navigateur authentifie l'émetteur.
>
> Une politique de lecture n'établit pas l'identité.

Frames

Les frames conservent des frontières d'accès. L'intégration ne fusionne pas automatiquement les origines. Un document intégré depuis une autre origine conserve des restrictions d'accès. N'en déduisez pas qu'il ne peut pas afficher une interface trompeuse ; le framing et le clickjacking demandent aussi une politique.

> **Erreur fréquente à comprendre** Toute page intégrée devient du code de la même origine.
>
> L'intégration ne fusionne pas automatiquement les origines.

XSS locale

Une XSS de même origine peut agir dans le contexte de la page. Le script hostile peut bénéficier du contexte compromis. Une XSS s'exécute dans le contexte de l'origine compromise et peut agir avec ses capacités. La SOP ne protège pas cette page contre un code hostile déjà exécuté dans son propre périmètre.

> **Erreur fréquente à comprendre** La SOP neutralise tout script injecté de même origine.
>
> Le script hostile peut bénéficier du contexte compromis.

Mesures séparées

Les tests doivent distinguer réponse envoyée et réponse lisible. Le navigateur peut masquer une réponse déjà produite. Dans un laboratoire, observez le journal serveur puis la tentative de lecture JavaScript. Une erreur CORS dans la console peut coexister avec une demande reçue et un effet déjà produit.

> **Erreur fréquente à comprendre** Une erreur CORS prouve qu'aucune requête n'est partie.
>
> Le navigateur peut masquer une réponse déjà produite.

Autorisation objet

La permission sur l'objet reste une règle serveur. Le modèle du navigateur ne connaît pas ces règles métier. Même une lecture techniquement autorisée entre origines doit respecter les droits de l'acteur. Essayez un identifiant appartenant à un autre compte et vérifiez le refus sans fuite de contenu.

> **Erreur fréquente à comprendre** La SOP valide automatiquement la propriété métier.
>
> Le modèle du navigateur ne connaît pas ces règles métier.

Atelier guidé : lire, expliquer et vérifier le code

```
<form method="post" action="https://booking.example.test/lab/cancel">
  <input name="reservation" value="42">
</form>
<!-- Formulaire sur une autre origine ; le serveur doit appliquer ses contrôles. -->
```

Le formulaire cross-origin peut provoquer une demande sans offrir au site tiers la lecture de sa réponse. Il faut donc protéger l'annulation avant son effet, avec les contrôles adaptés au contexte de session. Testez séparément réception de la demande, décision serveur et visibilité de la réponse.

Le serveur doit vérifier sa défense CSRF. Un effet doit être protégé même si la réponse reste illisible au script tiers. La SOP n'est pas une interdiction universelle d'émission. Les formulaires peuvent soumettre des données.

Travail pratique : Créez deux origines locales et comparez chargement d'image, navigation et lecture par fetch. Sur B, utilisez un endpoint sans mutation.

Critères de correction : Relevez les requêtes reçues puis les accès possibles depuis le script. Classez les résultats par mécanisme et gardez les mutations derrière une politique serveur indépendante de l'accès à la réponse.

Exemple commenté

```
Origine A : page et script
Origine B : ressource distante
Observation 1 : demande reçue par B
Observation 2 : lecture permise ou refusée dans A
```

Les deux observations peuvent diverger. Le refus de lecture dans A n'annule pas automatiquement un effet déjà produit dans B.

Exercice de mise en pratique

Créez deux origines locales et comparez chargement d'image, navigation et lecture par fetch. Sur B, utilisez un endpoint sans mutation.

Voir une proposition de correction

Relevez les requêtes reçues puis les accès possibles depuis le script. Classez les résultats par mécanisme et gardez les mutations derrière une politique serveur indépendante de l'accès à la réponse.

- Séparer émission et lecture.
- Ne pas présenter SOP comme une autorisation.
- Tester avec deux origines distinctes.

Contrôle de connaissances

Un refus de lecture interorigine prouve-t-il l'absence de requête reçue par le serveur ?

> Réponse expliquée : Non, envoi et lecture sont des propriétés distinctes — Certaines requêtes sont envoyées même si la lecture est bloquée.

Cours associé : 12. Same-origin policy : lecture et envoi ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/Security/Defenses/Same-origin_policy

13. CORS : autoriser une lecture sans ouvrir les droits

Définir une liste d'origines autorisées liée à un usage réel.

Comprendre le mécanisme

CORS fournit un protocole par lequel le serveur indique au navigateur quelles lectures interorigine sont permises. Il ne constitue pas une authentification de l'utilisateur ni une barrière universelle contre un client HTTP autonome. Réfléchir automatiquement toute valeur Origin reçue peut accorder une lecture à un appelant non prévu. Les règles doivent distinguer ressources publiques et réponses liées à une identité.

Raisonner et vérifier

Utilisez une liste d'origines complète et exacte pour les interfaces connues. Lorsque la réponse de politique varie selon Origin et peut être mise en cache, prenez en compte cette variation. Évaluez aussi les méthodes et champs permis, sans les élargir pour faire disparaître une erreur de console. Une API non accessible au navigateur d'un tiers peut rester directement joignable : elle garde ses contrôles d'accès.

Partage navigateur

CORS organise un partage de réponse avec le navigateur. La politique de lecture ne vérifie pas l'identité. Access-Control-Allow-Origin décrit un partage de réponse avec une origine de navigateur. Il ne bloque pas automatiquement un appel réseau direct effectué par un autre programme ou un serveur.

> **Erreur fréquente à comprendre** CORS authentifie tous les clients de l'API.
>
> La politique de lecture ne vérifie pas l'identité.

Admission explicite

Une liste d'origines doit être explicite et contrôlée. La valeur reçue n'établit pas sa légitimité. Comparez Origin à une liste maîtrisée et normalisée selon votre contrat. Refléter toute valeur revient à annoncer une confiance très large, particulièrement dangereuse pour des réponses personnelles.

> **Erreur fréquente à comprendre** Réfléchir toute origine reçue constitue une liste de confiance.
>
> La valeur reçue n'établit pas sa légitimité.

Client autonome

Un client direct peut appeler l'API sans appliquer CORS. Les contrôles serveur demeurent indispensables. Un client hors navigateur n'a pas à appliquer CORS comme le ferait fetch dans une page. Exigez credentials et permissions indépendamment des headers CORS renvoyés par l'API.

> **Erreur fréquente à comprendre** Un refus CORS interdit tout appel par curl.
>
> Les contrôles serveur demeurent indispensables.

Méthode et droit

Les méthodes admises ne donnent pas de permissions métier. La déclaration protocolaire ne crée pas ces droits. Autoriser POST dans le partage CORS ne donne pas à Alice le droit de modifier la réservation de Bob. Le serveur doit encore appliquer la décision métier après authentification.

> **Erreur fréquente à comprendre** Access-Control-Allow-Methods accorde la propriété des ressources.
>
> La déclaration protocolaire ne crée pas ces droits.

Origin absent

Un Origin absent nécessite une politique adaptée au type de client. Les clients non navigateur peuvent suivre un autre contrat. Certains clients légitimes peuvent ne pas envoyer Origin. Définissez une politique selon vos types de clients sans transformer son absence en preuve d'authenticité ou en accès privilégié.

> **Erreur fréquente à comprendre** Tout client sans Origin est automatiquement malveillant.
>
> Les clients non navigateur peuvent suivre un autre contrat.

Test navigateur

Une origine refusée doit être testée dans le navigateur. Le trajet complet et le navigateur participent au résultat. Vérifiez une origine admise et une origine refusée avec un vrai navigateur. Un test de headers seul ne démontre pas la lisibilité effective de la réponse dans le contexte considéré.

> **Erreur fréquente à comprendre** Un simple test de chaîne prouve l'exposition finale au script.
>
> Le trajet complet et le navigateur participent au résultat.

Credentials

Les credentials imposent des contraintes de partage spécifiques. L'origine explicite est requise dans ce cas de partage. Une réponse personnelle avec credentials demande une origine explicitement admise et les règles de partage compatibles. Le joker ne doit pas servir de raccourci à une politique d'identités automatiques.

> **Erreur fréquente à comprendre** Le joker d'origine convient à toutes les réponses avec credentials.
>
> L'origine explicite est requise dans ce cas de partage.

Erreurs cohérentes

Les erreurs d'API peuvent aussi nécessiter des en-têtes cohérents. Les réponses d'erreur participent à l'expérience et au contrat. Le frontend doit pouvoir traiter les erreurs que le contrat choisit de partager. Vérifiez les headers sur réponses d'échec aussi, sans ouvrir plus largement les données pour résoudre un simple message console.

> **Erreur fréquente à comprendre** Seules les réponses 200 influencent le parcours CORS.
>
> Les réponses d'erreur participent à l'expérience et au contrat.

Variation et cache

La clé de cache doit respecter les variations pertinentes d'origine. Un cache peut servir la mauvaise politique sans dimension adéquate. Si la réponse varie selon Origin, sa stratégie de cache doit respecter cette dimension. Sinon un intermédiaire peut réutiliser une réponse ou ses permissions pour une autre origine.

> **Erreur fréquente à comprendre** Une réponse différente par origine se partage toujours sans variation.
>
> Un cache peut servir la mauvaise politique sans dimension adéquate.

Droits après identité

L'autorisation sur l'objet doit suivre l'authentification. La politique CORS ne supprime pas l'isolation métier. Un cookie ou bearer token valide identifie un acteur selon son mécanisme. Vérifiez ensuite l'accès à la ressource ; CORS ne porte aucune preuve de propriété métier.

> **Erreur fréquente à comprendre** Une origine admise peut lire tous les comptes.
>
> La politique CORS ne supprime pas l'isolation métier.

Atelier guidé : lire, expliquer et vérifier le code

```
Access-Control-Allow-Origin: https://portal.example.test

Client considéré : script HTTP serveur, hors navigateur.
```

Le client considéré est un script serveur et non une page de navigateur. Le header CORS ne constitue donc pas sa barrière d'admission réseau. Ajoutez une authentification et un contrôle de ressource effectifs, puis utilisez un navigateur pour vérifier séparément la politique de partage annoncée.

CORS définit un partage pour le navigateur, pas toute admission réseau. Un client direct ne suit pas nécessairement cette politique. Un nom d'origine ne définit pas les permissions. Le modèle du navigateur n'est pas celui de tout client serveur.

Travail pratique : Écrivez une matrice CORS pour une ressource publique et une ressource personnelle, avec une origine admise et une origine refusée.

Critères de correction : Vérifiez les réponses réelles et le comportement du navigateur. Répétez avec un client HTTP sans politique navigateur pour confirmer que l'autorisation métier reste appliquée indépendamment des champs CORS.

Exemple commenté

```
Access-Control-Allow-Origin: https://app.example.test
Vary: Origin

Politique illustrative pour une origine explicitement autorisée.
```

Le serveur ne doit émettre cette autorisation que lorsque la demande correspond à sa politique ; elle ne remplace pas le contrôle de ressource.

Exercice de mise en pratique

Écrivez une matrice CORS pour une ressource publique et une ressource personnelle, avec une origine admise et une origine refusée.

Voir une proposition de correction

Vérifiez les réponses réelles et le comportement du navigateur. Répétez avec un client HTTP sans politique navigateur pour confirmer que l'autorisation métier reste appliquée indépendamment des champs CORS.

- Liste d’origines explicite.
- Pas de réflexion libre.
- CORS n’authentifie pas un acteur.

Contrôle de connaissances

CORS empêche-t-il un client HTTP autonome de contacter l’API ?

> Réponse expliquée : Non, ce n’est pas une barrière universelle de transport — Le protocole concerne principalement les permissions du navigateur.

Cours associé : 13. CORS : autoriser une lecture sans ouvrir les droits ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS

14. Preflight : intention annoncée et demande réelle

Tester les deux étapes du protocole CORS.

Comprendre le mécanisme

Une pré-vérification CORS annonce au serveur la méthode et certains champs d'une demande envisagée. Elle utilise OPTIONS dans le cadre prévu par le navigateur. La réponse permet ou refuse cette intention selon la politique. Cette étape ne doit pas exécuter l'opération métier annoncée ni être assimilée à la demande authentifiée finale. Une réponse positive n'empêche pas le serveur de refuser ensuite une demande réelle invalide ou non autorisée.

Raisonner et vérifier

Vérifiez les deux échanges dans les outils réseau. Le cache de pré-vérification suit un mécanisme distinct du cache ordinaire d'une représentation ; une modification de politique peut donc sembler tarder. Documentez la durée et les limites imposées par les clients. N'autorisez pas toutes les méthodes simplement parce que le navigateur réclame une pré-vérification.

OPTIONS préalable

Le navigateur peut envoyer OPTIONS avant la requête réelle. Le précontrôle ne doit pas produire cet effet. Le navigateur peut annoncer méthode et headers envisagés avant la demande réelle. Le serveur répond selon sa politique de partage, mais aucune mutation métier ne devrait être exécutée pendant cette étape de négociation.

> **Erreur fréquente à comprendre** Tout OPTIONS exécute d'abord la mutation métier.
>
> Le précontrôle ne doit pas produire cet effet.

Sans preflight

Certaines requêtes ne déclenchent pas de preflight. Le déclenchement dépend notamment de la méthode et des champs. Certaines demandes remplissent les conditions d'un envoi sans pré-vérification. Ne fondez donc pas la protection d'une mutation sur l'idée que tout appel externe serait bloqué par un OPTIONS préalable.

> **Erreur fréquente à comprendre** Toute requête inter-origines commence nécessairement par OPTIONS.
>
> Le déclenchement dépend notamment de la méthode et des champs.

Contrôles réels

La requête réelle doit conserver ses contrôles. Il ne vérifie pas la propriété métier de chaque objet. Après un preflight accepté, la demande réelle doit encore valider données, identité et autorisation. Une permission de communication n'est pas une permission de modifier une ressource précise.

> **Erreur fréquente à comprendre** Un preflight accepté suffit pour toute autorisation future.
>
> Il ne vérifie pas la propriété métier de chaque objet.

Cache spécifique

Le cache de preflight possède un usage spécifique. Ses informations servent au contrôle de partage navigateur. Le navigateur peut mémoriser une réponse de preflight selon sa politique et la durée annoncée. Une modification serveur peut donc ne pas provoquer immédiatement un nouveau OPTIONS dans le même contexte.

> **Erreur fréquente à comprendre** Le cache OPTIONS est identique à toute réponse métier.
>
> Ses informations servent au contrôle de partage navigateur.

Headers demandés

Le serveur doit examiner les en-têtes demandés selon son contrat. Les capacités doivent rester dans la politique définie. Examinez Access-Control-Request-Headers selon le contrat plutôt que recopier sans contrôle. L'admission d'un header ne prouve pas la fiabilité de sa valeur dans la demande réelle.

> **Erreur fréquente à comprendre** Tout nom demandé doit être accepté automatiquement.
>
> Les capacités doivent rester dans la politique définie.

Refus préalable

Un preflight refusé peut empêcher l'envoi réel par le navigateur. Le refus peut être purement protocolaire. Un preflight refusé peut empêcher le navigateur d'émettre la demande réelle concernée. Vérifiez les journaux pour distinguer ce cas d'une demande envoyée dont seule la lecture de réponse est bloquée.

> **Erreur fréquente à comprendre** Un preflight refusé prouve une panne de base de données.
>
> Le refus peut être purement protocolaire.

OPTIONS réussi

Un succès OPTIONS ne garantit pas le succès métier. Les deux échanges ont des responsabilités différentes. Le POST peut être refusé pour manque de droit après un OPTIONS réussi. Ce comportement est cohérent : les deux réponses portent sur des décisions différentes et doivent être testées séparément.

> **Erreur fréquente à comprendre** Le statut 204 OPTIONS valide le contenu du prochain paiement.
>
> Les deux échanges ont des responsabilités différentes.

Méthode serveur

Les méthodes incorrectes doivent être refusées au serveur. Un client direct n'est pas limité par l'annonce navigateur. Le routeur doit refuser une méthode hors contrat indépendamment du navigateur. Un client autonome peut envoyer directement la demande sans passer par la séquence de preflight attendue.

> **Erreur fréquente à comprendre** La liste annoncée suffit à empêcher tout client direct.
>
> Un client direct n'est pas limité par l'annonce navigateur.

Credentials du flux

Les credentials du flux réel demandent une validation propre. Le token doit encore être validé lors de l'opération. Les règles de cookies et d'authentification du flux réel restent applicables. Un OPTIONS ne démontre ni l'envoi d'un cookie ni la validité d'une session au moment du POST.

> **Erreur fréquente à comprendre** Le précontrôle remplace la vérification du token réel.
>
> Le token doit encore être validé lors de l'opération.

Suite complète

Tester OPTIONS seul ne couvre pas la réponse réelle. Le navigateur applique aussi des règles à la réponse réelle. Testez OPTIONS puis réponse réelle, erreurs comprises. Un test qui vérifie seulement le 204 initial peut manquer un header absent ou une autorisation incorrecte sur l'opération réellement demandée.

> **Erreur fréquente à comprendre** La présence d'un OPTIONS vert prouve la lecture finale du script.
>
> Le navigateur applique aussi des règles à la réponse réelle.

Atelier guidé : lire, expliquer et vérifier le code

```
OPTIONS /lab/orders → 204
POST /lab/orders → 403

Le POST est refusé par une règle d'autorisation métier.
```

Le 204 d'OPTIONS décrit une étape de communication ; le 403 du POST exprime ensuite une décision de droit métier. Les deux résultats peuvent être corrects ensemble. Vérifiez que le POST interdit ne modifie rien et qu'un acteur autorisé peut réussir le même parcours.

Il ne donne pas ce privilège. Le précontrôle ne doit pas produire cet effet. Le contrôle d'objet reste nécessaire. Le partage protocolaire et le droit d'action sont deux décisions.

Travail pratique : Capturez une pré-vérification puis le POST réel de votre laboratoire. Changez ensuite la politique et observez la mise en cache.

Critères de correction : Comparez les journaux : aucun effet métier sur OPTIONS, validation et droits sur POST. Utilisez un contexte de navigateur neuf pour distinguer une règle active d'une permission encore mémorisée.

Exemple commenté

```
OPTIONS /lab/reservations
Origin: https://app.example.test
Access-Control-Request-Method: POST
Access-Control-Request-Headers: content-type
```

Cette requête annonce un futur POST, elle ne porte pas à elle seule la réservation à traiter. Le POST réel reste soumis au contrat complet.

Exercice de mise en pratique

Capturez une pré-vérification puis le POST réel de votre laboratoire. Changez ensuite la politique et observez la mise en cache.

Voir une proposition de correction

Comparez les journaux : aucun effet métier sur OPTIONS, validation et droits sur POST. Utilisez un contexte de navigateur neuf pour distinguer une règle active d'une permission encore mémorisée.

- Aucun effet métier sur le preflight.
- Tester la demande réelle.
- Distinguer les caches.

Contrôle de connaissances

Une pré-vérification positive autorise-t-elle automatiquement la mutation métier ?

> Réponse expliquée : Non, la demande réelle reste à valider et autoriser — La politique CORS ne remplace pas l'autorisation de l'opération.

Cours associé : 14. Preflight : intention annoncée et demande réelle ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request

15. Credentials CORS et frontières de cookies

Ne pas confondre demande d'envoi et permission effective.

Comprendre le mécanisme

Un fetch configuré pour inclure des credentials demande au navigateur d'utiliser les éléments d'authentification applicables. Cela ne neutralise pas les règles de cookies, les restrictions de contexte ou les politiques du navigateur. Pour une lecture CORS avec credentials, le serveur doit fournir une politique compatible, notamment une origine explicite plutôt qu'un joker. Ce mécanisme ne donne toujours aucun droit sur une réservation particulière.

Raisonner et vérifier

Traitez séparément la présence du cookie, l'identité reconnue et la lecture de la réponse. Une panne peut provenir de l'une de ces étapes ; activer tout à la fois empêche le diagnostic et peut ouvrir une fuite. Même pour une origine admise, une mutation par cookie demande une protection adaptée contre les déclenchements intersites et un contrôle d'autorisation.

Include sous conditions

credentials include ne force pas l'envoi de tout cookie. Les politiques de cookies restent applicables. credentials: include demande au navigateur de considérer les credentials du flux. Cette option n'annule ni SameSite, ni portée du cookie, ni les politiques de blocage de cookies tiers du navigateur.

> **Erreur fréquente à comprendre** include contourne toujours SameSite.
>
> Les politiques de cookies restent applicables.

Origine explicite

Une réponse avec credentials exige une origine admise explicite. Le contrat de partage interdit cette généralisation. Pour une lecture CORS avec credentials, le partage doit identifier une origine admise et annoncer les credentials selon le contrat. Le joker associé à true ne fournit pas cette permission utilisable.

> **Erreur fréquente à comprendre** Le joker autorise toutes les lectures avec credentials.
>
> Le contrat de partage interdit cette généralisation.

HttpOnly

HttpOnly contrôle la lecture JavaScript du cookie. Il ne remplace pas Secure ni les autres règles d'envoi. Cet attribut limite la lecture directe du cookie par script, pas son envoi automatique. Une page peut effectuer une demande authentifiée sans pouvoir lire la valeur du cookie lui-même.

> **Erreur fréquente à comprendre** HttpOnly empêche tout envoi du cookie en HTTP.
>
> Il ne remplace pas Secure ni les autres règles d'envoi.

CSRF distinct

CORS et CSRF protègent des propriétés distinctes. Une émission de mutation peut encore demander une protection dédiée. Le partage de réponse et l'intention d'une action sensible ne sont pas la même propriété. Une mutation avec cookie doit conserver sa protection CSRF même lorsque sa configuration CORS semble restrictive.

> **Erreur fréquente à comprendre** Autoriser CORS supprime toute nécessité de contrôle CSRF.
>
> Une émission de mutation peut encore demander une protection dédiée.

Droit métier

Le serveur décide encore des permissions métier. L'identité ne fournit pas une autorisation universelle. Une demande issue d'une origine admise et correctement authentifiée peut viser un objet tiers. Testez cette situation et refusez selon la propriété ou le tenant avant tout effet.

> **Erreur fréquente à comprendre** Des credentials présents donnent tous les droits.
>
> L'identité ne fournit pas une autorisation universelle.

Cookies tiers

Un navigateur peut bloquer certains cookies tiers. Les politiques locales peuvent limiter le stockage ou l'envoi. Les politiques navigateur peuvent limiter certains cookies malgré les headers serveur. Observez le motif de blocage et le contexte réel au lieu de conclure qu'un header CORS force nécessairement leur transmission.

> **Erreur fréquente à comprendre** include garantit le même comportement dans toute politique de confidentialité.
>
> Les politiques locales peuvent limiter le stockage ou l'envoi.

Reflet contrôlé

Une origine reflétée doit être vérifiée avant partage. Cela peut ouvrir la lecture à des origines non fiables. Si Origin est renvoyé dynamiquement, validez-le d'abord contre une liste maîtrisée. Une copie automatique de toute origine reçue retire le sens de cette liste et peut exposer des données personnelles.

> **Erreur fréquente à comprendre** Renvoyer toute origine avec Allow-Credentials est une restriction.
>
> Cela peut ouvrir la lecture à des origines non fiables.

Deux frontières

Les tests doivent couvrir autorisé et refusé. Les cas négatifs vérifient les limites annoncées. Testez lecture admise, lecture refusée et absence de droit sur un objet. Ces cas distinguent partage navigateur et autorisation serveur, qui peuvent échouer indépendamment dans le même parcours.

> **Erreur fréquente à comprendre** Un seul compte nominal démontre toutes les frontières.
>
> Les cas négatifs vérifient les limites annoncées.

Cookie volé

Une valeur de cookie volée reste sensible selon son contrat. Le paramètre ne crée pas cette protection. HttpOnly ne protège pas une copie obtenue par une autre fuite. Appliquez durée, retrait et protection des traces, puis vérifiez qu'un credential retiré n'est plus accepté selon le délai annoncé.

> **Erreur fréquente à comprendre** Le mot credentials empêche tout vol ou rejeu.
>
> Le paramètre ne crée pas cette protection.

Cache privé

Le cache ne doit pas mélanger les réponses privées. La politique de cache doit être conçue séparément. Une réponse personnelle ne doit pas être mélangée entre identités ou origines. Définissez stockage, portée et variantes ; Vary seul ne rend pas admissible une donnée qui ne doit jamais entrer dans un cache partagé.

> **Erreur fréquente à comprendre** Allow-Credentials isole automatiquement tous les caches.
>
> La politique de cache doit être conçue séparément.

Atelier guidé : lire, expliquer et vérifier le code

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

Le couple étoile et Allow-Credentials: true ne satisfait pas une lecture CORS personnelle avec credentials. Remplacez le joker par une origine réellement admise dans ce contrat, puis vérifiez en navigateur l'envoi des cookies et la lisibilité sans oublier les protections CSRF et les droits métier.

L'échec ne signifie pas cette règle générale de cookies. Les contrôles d'identité doivent rester en place. Le contrat exige une origine explicite admise dans ce contexte. Le champ ne chiffre pas une valeur.

Travail pratique : Dans deux origines de laboratoire, testez réponse personnelle avec et sans credentials puis examinez le cookie réellement envoyé.

Critères de correction : Relevez présence du cookie, statut d'authentification et accès au corps séparément. Conservez une origine précise côté serveur et testez aussi une origine hostile, un utilisateur déconnecté et une ressource tierce.

Exemple commenté

```
Access-Control-Allow-Origin: https://app.example.test
Access-Control-Allow-Credentials: true
Vary: Origin
```

Le fragment décrit une permission de lecture limitée. Il ne force pas un navigateur à envoyer un cookie que ses autres politiques interdisent.

Exercice de mise en pratique

Dans deux origines de laboratoire, testez réponse personnelle avec et sans credentials puis examinez le cookie réellement envoyé.

Voir une proposition de correction

Relevez présence du cookie, statut d'authentification et accès au corps séparément. Conservez une origine précise côté serveur et testez aussi une origine hostile, un utilisateur déconnecté et une ressource tierce.

- Origine précise pour une lecture avec credentials.
- Vérifier les règles de cookies.

- Conserver autorisation et CSRF.

Contrôle de connaissances

credentials: include désactive-t-il SameSite et les autres politiques de cookies ?

> Réponse expliquée : Non, ces politiques continuent de s’appliquer — Le mode fetch ne supprime pas les autres contraintes.

Cours associé : 15. Credentials CORS et frontières de cookies ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Access-Control-Allow-Credentials

16. Cookies : transport d’état et confiance

Distinguer identifiant opaque, données client et autorité serveur.

Comprendre le mécanisme

Un cookie est stocké par le navigateur puis envoyé lorsque ses règles d’applicabilité le permettent. Il peut porter un identifiant de session opaque ou une autre donnée, mais son nom et sa présence ne prouvent pas l’autorité de son contenu. Une préférence modifiable et un identifiant authentifié n’ont pas le même rôle. Le serveur décide de l’identité et des droits au moyen d’un mécanisme vérifiable.

Raisonner et vérifier

Inventoriez les cookies, leur finalité et les pages qui les utilisent. Évitez d’y placer inutilement des données personnelles ou des secrets lisibles. Une expiration côté navigateur ne remplace pas la durée d’acceptation de la session côté serveur. Testez déconnexion, changement de privilège et coexistence de cookies portant un même nom avec des portées différentes, afin de repérer les ambiguïtés de lecture.

Identifiant opaque

Un cookie de session peut porter un identifiant opaque. La valeur reçue doit être reliée à un état admis. Un cookie de session peut être une référence aléatoire vers un état serveur. Le nom du cookie ou une valeur role=admin fournie par le client ne constitue pas une preuve de privilège.

> **Erreur fréquente à comprendre** Tout cookie constitue un privilège serveur valide.
>
> La valeur reçue doit être reliée à un état admis.

Acceptation serveur

Le serveur doit vérifier expiration et retrait de session. Les autres détenteurs ne dépendent pas de cette suppression locale. Le serveur doit vérifier que la session existe, reste active et appartient au contexte attendu. La présence d’un cookie syntaxiquement valide ne suffit pas à lui donner une durée illimitée.

> **Erreur fréquente à comprendre** Supprimer le cookie client révoque toutes ses copies.
>
> Les autres détenteurs ne dépendent pas de cette suppression locale.

Envoi automatique

Un cookie peut être envoyé automatiquement selon sa politique. L’envoi peut accompagner un déclenchement tiers. Le navigateur joint les cookies selon destination et attributs applicables. Cette automaticité explique le besoin d’une défense CSRF pour les actions sensibles réalisées avec une identité ambiante.

> **Erreur fréquente à comprendre** Toute requête avec cookie prouve un clic volontaire.
>
> L’envoi peut accompagner un déclenchement tiers.

Deux stockages

Le stockage navigateur et l'état serveur sont distincts. Les cycles de conservation peuvent différer. Effacer un cookie retire une copie côté navigateur ; invalider la session retire une capacité côté serveur selon sa politique. Pour une déconnexion sensible, définissez et testez les deux effets.

> **Erreur fréquente à comprendre** Un cookie expiré visuellement supprime toute donnée serveur.
>
> Les cycles de conservation peuvent différer.

Portée

Le domaine et le chemin participent au périmètre d'envoi. Ce périmètre ne remplace pas une autorisation métier. Domain et Path influencent les destinations d'envoi. Choisissez une portée minimale, sans en déduire une séparation d'autorisation entre utilisateurs ou une défense suffisante contre un script compromis.

> **Erreur fréquente à comprendre** Le chemin du cookie valide les droits sur les objets.
>
> Ce périmètre ne remplace pas une autorisation métier.

Identité maîtrisée

Une session doit être liée à un état d'identité maîtrisé. Une entrée client ne crée pas un rôle de confiance. La session doit référencer un utilisateur selon le mécanisme serveur, pas faire confiance à des rôles envoyés librement. Rechargez ou versionnez l'état nécessaire à la politique de retrait d'accès.

> **Erreur fréquente à comprendre** Le client peut définir son rôle dans un cookie libre.
>
> Une entrée client ne crée pas un rôle de confiance.

Canal protégé

Le canal doit protéger les cookies sensibles. Un identifiant opaque volé peut devenir une capacité. Protégez les cookies sensibles avec HTTPS et les attributs adaptés. Vérifiez les terminaisons proxy et évitez qu'une redirection ou un journal conserve leur valeur en clair.

> **Erreur fréquente à comprendre** Une valeur opaque n'a pas besoin de transport sécurisé.
>
> Un identifiant opaque volé peut devenir une capacité.

Expiration effective

L'expiration doit être testée côté serveur. Le serveur doit appliquer sa propre décision. Conservez une ancienne valeur dans un test puis présentez-la après expiration serveur. Le refus doit venir de la politique d'acceptation, même si un client malveillant ignore l'expiration navigateur.

> **Erreur fréquente à comprendre** La seule date affichée dans l'interface impose le refus serveur.
>
> Le serveur doit appliquer sa propre décision.

Privilège

Les transitions de privilège demandent une politique de session. Le changement d'identité exige une revue de fixation et de rotation. Connexion et élévation de droit sont des transitions qui peuvent exiger rotation et nouvelle authentification. Documentez le devenir des anciennes références afin qu'elles ne gagnent pas silencieusement les nouveaux privilèges.

> **Erreur fréquente à comprendre** Une session anonyme doit rester identique sans analyse après login.
>
> Le changement d'identité exige une revue de fixation et de rotation.

Journaux

La journalisation doit minimiser les valeurs sensibles. Les traces ne doivent pas devenir un magasin de capacités. Enregistrez événements et références non exploitables plutôt que la valeur de session complète. Une fuite de logs contenant les cookies pourrait créer un accès équivalent à celui que la session protège.

> **Erreur fréquente à comprendre** Tous les cookies doivent être copiés dans chaque événement.
>
> Les traces ne doivent pas devenir un magasin de capacités.

Atelier guidé : lire, expliquer et vérifier le code

```
Cookie: role=admin; theme=dark

Hypothèse : aucune preuve serveur ne lie role à une identité autorisée.
```

Le cookie annonce role=admin sans preuve serveur selon l'hypothèse donnée. Cette valeur doit rester une entrée hostile. Remplacez ce modèle par une session ou un token validé puis appliquez une autorisation côté serveur, et vérifiez qu'une modification manuelle du cookie ne donne aucun privilège.

La préférence visuelle ne crée pas de droit. Un client peut présenter une affirmation non fiable. Le nom ne prouve pas l'autorité. Le séparateur ne signe rien.

Travail pratique : Créez une fiche par cookie de laboratoire : finalité, producteur, portée, protections, durée côté navigateur et durée côté serveur.

Critères de correction : Testez une valeur absente, inconnue et révoquée ; vérifiez qu'aucune de ces entrées ne donne un rôle par simple lecture du cookie. Contrôlez ensuite que la déconnexion invalide le contexte selon le contrat prévu.

Exemple commenté

```
Set-Cookie: lab_session=opaque-value; Path=/; Secure; HttpOnly; SameSite=Lax
```

opaque-value est une valeur illustrative, pas un secret utilisable. Le serveur conserve la politique de durée et l'état d'authentification correspondant.

Exercice de mise en pratique

Créez une fiche par cookie de laboratoire : finalité, producteur, portée, protections, durée côté navigateur et durée côté serveur.

Voir une proposition de correction

Testez une valeur absente, inconnue et révoquée ; vérifiez qu'aucune de ces entrées ne donne un rôle par simple lecture du cookie. Contrôlez ensuite que la déconnexion invalide le contexte selon le contrat prévu.

- Cookie reçu non fiable par défaut.
- Durée serveur explicite.
- Aucune permission déduite du nom.

Contrôle de connaissances

Un cookie nommé admin prouve-t-il un privilège ?

> Réponse expliquée : Non, le serveur doit vérifier le contexte et les droits — La donnée reçue n'est pas une autorité en soi.

Cours associé : 16. Cookies : transport d'état et confiance ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies

17. Set-Cookie, portée et préfixes

Comprendre ce que protège chaque attribut et ce qu'il laisse possible.

Comprendre le mécanisme

Secure limite l'envoi du cookie aux contextes de transport prévus comme sûrs ; HttpOnly réduit sa lecture directe via les API JavaScript concernées. SameSite contrôle certains contextes intersites. Ces attributs sont complémentaires, sans assurer l'autorisation métier ni éliminer toutes les conséquences d'une XSS. Domain et Path règlent la portée d'envoi ; un chemin ne doit pas être présenté comme une séparation de confiance absolue entre applications.

Raisonner et vérifier

Le préfixe __Host- impose des conditions utiles sur un cookie accepté par un navigateur conforme : Secure, Path=/ et absence de Domain. Pour supprimer un cookie, faites correspondre sa portée à celle de la valeur créée. Examinez les cookies dupliqués plutôt que conclure à une déconnexion à partir d'un seul champ. Testez les attributs effectivement vus dans le navigateur derrière le proxy HTTPS.

Secure

Secure restreint l'envoi du cookie aux canaux prévus comme sécurisés. La lecture du script relève notamment de HttpOnly. Le navigateur limite l'envoi selon les conditions sécurisées de cet attribut. Vérifiez HTTPS sur le parcours exposé et la perception du schéma derrière les proxies ; un attribut seul ne remplace pas toute la politique de transport.

> **Erreur fréquente à comprendre** Secure empêche toute lecture JavaScript.
>
> La lecture du script relève notamment de HttpOnly.

HttpOnly

HttpOnly limite l'accès direct par JavaScript. Le script peut encore agir dans le contexte compromis. Le script ne peut pas lire directement le cookie par les API ordinaires concernées. Une XSS peut néanmoins déclencher des actions dans la page ; gardez encodage de sortie et prévention d'injection.

> **Erreur fréquente à comprendre** HttpOnly empêche une XSS d'effectuer toute action.
>
> Le script peut encore agir dans le contexte compromis.

SameSite

SameSite concerne des conditions d'envoi inter-sites. Il ne chiffre pas la valeur. Cette politique porte sur des conditions d'envoi inter-sites. Site et origine diffèrent ; testez vos sous-domaines et parcours de navigation au lieu d'assimiler SameSite à un contrôle universel d'origine.

> **Erreur fréquente à comprendre** SameSite est un chiffrement du cookie.
>
> Il ne chiffre pas la valeur.

None avec Secure

SameSite=None demande Secure dans les navigateurs concernés. Les navigateurs imposent des contraintes à ce mode. Les navigateurs concernés exigent Secure pour SameSite=None. Observez l'acceptation réelle et les avertissements ; une ligne Set-Cookie présente dans la réponse peut être rejetée par le client.

> **Erreur fréquente à comprendre** None signifie aucune exigence de transport.
>
> Les navigateurs imposent des contraintes à ce mode.

Préfixe __Host-

__Host- exige notamment Secure, Path=/ et absence de Domain. Le préfixe réduit le périmètre selon des règles précises. Le préfixe exige notamment Secure, Path=/ et aucune directive Domain. Cette forme limite certaines portées, mais ne constitue pas à elle seule une authentification ou une politique complète de session.

> **Erreur fréquente à comprendre** __Host- autorise librement un Domain parent.
>
> Le préfixe réduit le périmètre selon des règles précises.

Path

Path ne remplace pas une frontière d'autorisation métier. Le périmètre d'envoi n'établit pas le rôle. Un chemin d'envoi réduit où le navigateur joint le cookie. Il ne prouve pas qu'un acteur est autorisé sur une ressource de ce chemin et ne remplace pas les contrôles applicatifs.

> **Erreur fréquente à comprendre** Path=/admin prouve que le détenteur est administrateur.
>
> Le périmètre d'envoi n'établit pas le rôle.

Construction

Set-Cookie doit suivre les API de construction adaptées. Les valeurs non contrôlées peuvent altérer le champ. Utilisez les API de cookie pour encoder valeur et attributs correctement. Une concaténation de données non fiables dans une ligne brute peut créer un résultat ambigu ou être refusée par les composants.

> **Erreur fréquente à comprendre** Concaténer toute entrée client dans Set-Cookie est sûr.
>
> Les valeurs non contrôlées peuvent altérer le champ.

Deux expirations

L'expiration du cookie ne définit pas seule l'expiration serveur. Le backend doit gérer son propre cycle de session. Max-Age ou Expires guide la conservation navigateur. Le serveur doit encore limiter l'acceptation de la session ; un client peut conserver une ancienne valeur et la présenter explicitement.

> **Erreur fréquente à comprendre** Max-Age révoque automatiquement l'état serveur distant.
>
> Le backend doit gérer son propre cycle de session.

Périmètre minimal

Un cookie sensible doit avoir un périmètre minimal. Un périmètre élargi peut exposer plus de contextes. Évitez Domain trop large et valeurs inutiles dans le cookie. La portée doit correspondre aux composants qui ont réellement besoin de recevoir cette capacité, avec conservation et retrait définis.

> **Erreur fréquente à comprendre** Un Domain très large réduit toujours l'exposition.
>
> Un périmètre élargi peut exposer plus de contextes.

Observation déployée

Les attributs doivent être observés dans la réponse déployée. Proxy et déploiement peuvent influencer la réponse. Inspectez la réponse et le stockage navigateur après passage par le proxy. Une configuration locale correcte peut être transformée ou rejetée sur le trajet effectif ; testez le résultat reçu.

> **Erreur fréquente à comprendre** Une configuration locale prouve tous les attributs réellement servis.
>
> Proxy et déploiement peuvent influencer la réponse.

Atelier guidé : lire, expliquer et vérifier le code

```
Set-Cookie: __Host-lab=abc; Secure; Path=/; Domain=example.test; HttpOnly
```

Le cookie commence par __Host- mais contient Domain=example.test, en contradiction avec les exigences du préfixe. Identifiez cette directive puis vérifiez le stockage dans un navigateur. La présence textuelle du header ne garantit pas que le client accepte effectivement le cookie.

Ce préfixe exige notamment l'absence de Domain. Secure fait partie des conditions attendues. La racine est le chemin attendu pour ce préfixe. HttpOnly ne contredit pas ce contrat.

Travail pratique : Vérifiez les attributs d'un cookie de laboratoire derrière un proxy et testez sa suppression avec le bon chemin.

Critères de correction : Comparez le Set-Cookie reçu et le magasin navigateur. Vérifiez le transport HTTPS reconnu par l'application, puis la disparition effective de la valeur visée et l'invalidation serveur. Ne vous contentez pas d'un message visuel de déconnexion.

Exemple commenté

```
Set-Cookie: __Host-lab_session=opaque-value; Secure; HttpOnly; Path=/; SameSite=Lax
```

Le préfixe borne la configuration du cookie ; il ne valide pas l'utilisateur. Aucune directive Domain n'est ajoutée à cet exemple.

Exercice de mise en pratique

Vérifiez les attributs d'un cookie de laboratoire derrière un proxy et testez sa suppression avec le bon chemin.

Voir une proposition de correction

Comparez le Set-Cookie reçu et le magasin navigateur. Vérifiez le transport HTTPS reconnu par l'application, puis la disparition effective de la valeur visée et l'invalidation serveur. Ne vous contentez pas d'un message visuel de déconnexion.

- Vérifier chaque attribut.
- Respecter la portée lors de la suppression.
- Ne pas attribuer une autorisation au préfixe.

Contrôle de connaissances

HttpOnly élimine-t-il tous les effets possibles d’une XSS ?

> Réponse expliquée : Non, un script peut encore agir dans le contexte de la page — Empêcher la lecture du cookie ne supprime pas tout usage de la session.

Cours associé : 17. Set-Cookie, portée et préfixes ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie

18. Cycle de session, fixation et élévation de privilèges

Gérer une identité dans le temps sans conserver un état ambigu.

Comprendre le mécanisme

La session associe un identifiant à un contexte accepté par le serveur. Sa création, son renouvellement, son expiration et sa révocation forment un cycle de vie. Une fixation exploite l'acceptation d'un identifiant connu ou contrôlé avant que la victime obtienne un contexte authentifié. Après authentification ou changement de privilège, une stratégie de renouvellement évite de prolonger une ancienne association de confiance.

Raisonner et vérifier

La rotation doit tenir compte des requêtes concurrentes et des erreurs réseau. Une suppression précipitée peut perdre un état légitime ; conserver indéfiniment l'ancien contexte peut laisser un accès indésirable. Définissez durées d'inactivité et absolues, comportement de l'ancien identifiant et événements d'audit minimisés. La déconnexion de la page courante n'est pas automatiquement la révocation de tous les appareils.

Transition de connexion

La connexion constitue une transition sensible de session. La politique doit empêcher une fixation utilisable. Un identifiant connu avant connexion ne doit pas devenir silencieusement une capacité privilégiée pour un tiers. Définissez rotation et reprise des données au moment où l'identité change.

> **Erreur fréquente à comprendre** Une identité anonyme impose le même identifiant après login.
>
> La politique doit empêcher une fixation utilisable.

Continuité contrôlée

Une rotation doit préserver les données nécessaires selon le contrat. Le changement d'identifiant doit rester cohérent avec le parcours. La rotation peut devoir préserver panier ou préférences tout en changeant la capacité d'authentification. Listez ce qui doit suivre la transition et ce qui doit être réinitialisé, puis testez les deux.

> **Erreur fréquente à comprendre** La rotation signifie supprimer sans analyse tout état métier.
>
> Le changement d'identifiant doit rester cohérent avec le parcours.

Instances

La révocation doit être testée sur les instances concernées. La propagation et les caches font partie du système. Un retrait stocké sur une seule instance peut laisser une autre accepter l'ancienne session. Vérifiez le backend partagé, caches et délais de propagation sur tous les nœuds inclus dans le contrat.

> **Erreur fréquente à comprendre** Une suppression locale invalide automatiquement tous les validateurs.
>
> La propagation et les caches font partie du système.

Demandes simultanées

La concurrence peut affecter la rotation. Des requêtes parallèles peuvent observer des transitions. Une requête ancienne peut arriver pendant la rotation. Définissez le traitement de ce chevauchement et vérifiez qu'il ne restaure pas une référence retirée ou ne fait pas perdre un état nécessaire.

> **Erreur fréquente à comprendre** Deux requêtes simultanées ne peuvent jamais toucher une session.
>
> Des requêtes parallèles peuvent observer des transitions.

Durées serveur

Le serveur impose des limites de durée. La durée doit suivre une politique de confiance serveur. Fixez durée maximale et éventuelle durée d'inactivité selon le risque. L'expiration du cookie ne suffit pas : le backend doit refuser une valeur conservée au-delà de la politique d'acceptation.

> **Erreur fréquente à comprendre** Le client décide librement de prolonger une session admin.
>
> La durée doit suivre une politique de confiance serveur.

Élévation

Une élévation de privilège demande une politique explicite. Le changement de contexte peut exiger une vérification adaptée. Passer d'un compte ordinaire à une opération sensible peut demander une réauthentification et une nouvelle capacité. Une simple mise à jour de rôle ne traite pas automatiquement le risque des anciennes références.

> **Erreur fréquente à comprendre** Un login ancien justifie toute future élévation sans contrôle.
>
> Le changement de contexte peut exiger une vérification adaptée.

Logout

Un logout doit avoir un contrat de retrait défini. Les effets dépendent de l'implémentation et du périmètre. Annoncez si la déconnexion retire seulement cette session ou toutes les sessions du compte. Vérifiez une copie conservée après l'action ; l'absence du cookie dans l'interface ne prouve pas le refus serveur.

> **Erreur fréquente à comprendre** Le mot logout garantit partout un retrait instantané.
>
> Les effets dépendent de l'implémentation et du périmètre.

Ancienne référence

Les anciens identifiants doivent suivre une politique de refus. Une compatibilité sans limite peut conserver le risque. Le contrat doit dire quand l'ancien identifiant cesse d'autoriser les demandes privilégiées. Testez cette borne et évitez une période de grâce indéfinie introduite pour contourner des problèmes de concurrence.

> **Erreur fréquente à comprendre** Tout identifiant ancien reste valide pour faciliter la migration.
>
> Une compatibilité sans limite peut conserver le risque.

Stockage protégé

Le stockage de session doit être protégé. Le contenu et les identifiants restent sensibles. Le store de sessions contient des capacités ou des données sensibles. Limitez accès, sauvegardes et visibilité des outils d'exploitation ; une protection de cookie ne répare pas un stockage exposé.

> **Erreur fréquente à comprendre** Une valeur opaque dispense de protéger le stockage serveur.
>
> Le contenu et les identifiants restent sensibles.

Avant et après

Un test doit essayer l'identifiant avant et après transition. Il faut vérifier les capacités de l'ancienne référence. Présentez l'identifiant initial avant connexion, après connexion puis après retrait. Cette chronologie démontre les transitions, au lieu de vérifier seulement qu'un nouvel identifiant a été généré.

> **Erreur fréquente à comprendre** Cliquer une seule fois sur Connexion prouve l'absence de fixation.
>
> Il faut vérifier les capacités de l'ancienne référence.

Atelier guidé : lire, expliquer et vérifier le code

```
before_login: session_id=lab-old
login: success
privileged_request_with_lab_old: accepted
rotation_policy: old_id_must_stop_authorizing_privileged_requests
```

La politique exige que lab-old cesse d'autoriser les demandes privilégiées, mais la trace montre encore une acceptation. La transition est donc incorrecte dans ce contrat. Vérifiez rotation, invalidation et propagation, puis ajoutez un test concurrent qui conserve la continuité des données autorisées.

L'interface ne retire pas une capacité serveur. Changer la représentation ne résout pas l'état de confiance. La taille du corps ne démontre pas le retrait. Le comportement contredit la politique de transition annoncée.

Travail pratique : Simulez deux requêtes parallèles lors d'une connexion puis lors d'une déconnexion. Définissez le sort de chaque identifiant.

Critères de correction : Utilisez un environnement isolé et des comptes fictifs. Vérifiez qu'un identifiant obsolète ne récupère pas les nouveaux privilèges ; contrôlez les faux refus liés à la concurrence et documentez la stratégie de reprise.

Exemple commenté

```
Anonyme → authentification vérifiée → nouvel état de session
Ancien identifiant → politique d'obsolescence explicite
Déconnexion → invalidation selon le périmètre annoncé
```

Le schéma distingue la transition de confiance du simple affichage d'un écran de connexion. La concurrence fait partie du contrat.

Exercice de mise en pratique

Simulez deux requêtes parallèles lors d'une connexion puis lors d'une déconnexion. Définissez le sort de chaque identifiant.

Voir une proposition de correction

Utilisez un environnement isolé et des comptes fictifs. Vérifiez qu'un identifiant obsolète ne récupère pas les nouveaux privilèges ; contrôlez les faux refus liés à la concurrence et documentez la stratégie de reprise.

- Rotation aux transitions de confiance.
- Durées serveur vérifiées.
- Périmètre de révocation annoncé.

Contrôle de connaissances

À quelle transition faut-il notamment revoir l’identifiant et l’état de session ?

> Réponse expliquée : Lors d’une authentification ou élévation de privilège — La transition augmente la confiance associée au contexte.

Cours associé : 18. Cycle de session, fixation et élévation de privilèges ↗

Documentation de référence ↗ https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

19. CSRF : déclenchement d'une action sous une identité

Protéger les mutations auxquelles le navigateur joint un contexte d'authentification.

Comprendre le mécanisme

Une attaque CSRF cherche à faire exécuter une action par le navigateur d'un utilisateur dans un contexte que le serveur reconnaît. Un cookie peut être joint automatiquement selon sa politique. L'attaquant n'a pas nécessairement besoin de lire la réponse pour produire l'effet. Les défenses courantes combinent un jeton lié au contexte pertinent, une vérification de provenance lorsque adaptée et une politique de cookies cohérente.

Raisonner et vérifier

Le contrôle CSRF ne remplace ni validation du contenu ni autorisation de la ressource. Un utilisateur légitime muni d'un jeton ne gagne pas le droit d'annuler la réservation d'un autre acteur. SameSite réduit certains chemins mais ne doit pas être présenté comme une solution universelle à tous les scénarios. Une XSS dans l'origine de confiance peut aussi compromettre les hypothèses de la protection.

Identité ambiante

CSRF exploite notamment l'envoi automatique d'une identité. Un effet peut être déclenché sans lecture du résultat. Une page hostile peut provoquer une demande que le navigateur accompagne de cookies selon leur politique. L'attaquant n'a pas nécessairement besoin de lire le cookie ni la réponse pour produire un effet.

> **Erreur fréquente à comprendre** CSRF nécessite toujours de lire la réponse protégée.
>
> Un effet peut être déclenché sans lecture du résultat.

Jeton contextualisé

Le serveur doit vérifier un jeton selon son contexte. Le contrôle exige une relation valide avec la session et l'usage. Le serveur vérifie un token lié au contexte prévu avant la mutation. Définissez sa liaison, son transport et sa validation ; la seule présence d'un champ nommé _token ne démontre aucun contrôle.

> **Erreur fréquente à comprendre** Un champ nommé token suffit même sans comparaison.
>
> Le contrôle exige une relation valide avec la session et l'usage.

Défense complémentaire

SameSite peut compléter une défense CSRF. Ses garanties dépendent du contexte et du parcours. SameSite réduit certains envois inter-sites mais ne remplace pas automatiquement le mécanisme CSRF de votre parcours. Les contextes same-site et les règles de navigation doivent être analysés concrètement.

> **Erreur fréquente à comprendre** SameSite remplace universellement tous les autres contrôles.
>
> Ses garanties dépendent du contexte et du parcours.

Méthode sûre

Une mutation ne doit pas être demandée par GET. La lecture peut être déclenchée automatiquement. GET ne doit pas annuler ou modifier une réservation. Les lectures automatiques et liens peuvent être déclenchés sans intention de mutation ; utilisez un parcours explicite et les contrôles serveur nécessaires.

> **Erreur fréquente à comprendre** Un GET de suppression évite tout risque CSRF.
>
> La lecture peut être déclenchée automatiquement.

CORS

CORS n'est pas un substitut général au contrôle CSRF. Le navigateur peut masquer une réponse après l'effet. Bloquer une lecture cross-origin n'empêche pas toute demande ni tout effet. Conservez une protection d'intention pour les opérations avec identité automatique au lieu d'attribuer cette propriété à CORS.

> **Erreur fréquente à comprendre** Un refus CORS garantit l'absence d'effet serveur.
>
> Le navigateur peut masquer une réponse après l'effet.

Ordre du refus

Un token invalide doit être refusé avant mutation. Le refus tardif ne prévient pas l'effet interdit. Validez le token avant save, appel prestataire ou émission de message. Un 403 après annulation est trop tardif ; les tests doivent vérifier la base et les frontières externes.

> **Erreur fréquente à comprendre** Le serveur peut écrire puis vérifier CSRF.
>
> Le refus tardif ne prévient pas l'effet interdit.

Droit sur l'objet

Le contrôle de droit sur l'objet reste nécessaire. Il ne prouve pas la propriété de la ressource. Un token CSRF correct peut accompagner la réservation d'un autre utilisateur. Appliquez également l'autorisation sur le sujet ; les deux contrôles répondent à des scénarios de menace différents.

> **Erreur fréquente à comprendre** Un token CSRF valide donne accès à toutes les réservations.
>
> Il ne prouve pas la propriété de la ressource.

Hypothèse XSS

Une XSS peut compromettre les hypothèses du modèle CSRF. Le script peut agir dans un contexte de confiance compromis. Un script hostile déjà exécuté dans la page peut obtenir des tokens ou déclencher les actions. Prévenir XSS reste nécessaire ; la défense CSRF ne constitue pas une protection générale contre ce script.

> **Erreur fréquente à comprendre** Un token CSRF neutralise tout script injecté.
>
> Le script peut agir dans un contexte de confiance compromis.

Cas négatifs

Les tests doivent couvrir absence et mauvaise valeur. Les cas négatifs vérifient réellement le contrôle. Testez token absent, incorrect et provenant d'un autre contexte selon le contrat. Vérifiez refus et non-effet dans chacun, puis un cas nominal afin de ne pas valider une protection qui bloque tout.

> **Erreur fréquente à comprendre** Le test nominal du formulaire suffit à prouver les refus.
>
> Les cas négatifs vérifient réellement le contrôle.

Valeurs sensibles

Les jetons ne doivent pas être publiés dans les logs ordinaires. La corrélation peut être obtenue sans exposer ces valeurs. Évitez de copier les jetons dans les logs et URLs non nécessaires. Les traces peuvent élargir leur distribution ; conservez plutôt catégorie de refus et corrélation non exploitable.

> **Erreur fréquente à comprendre** Un diagnostic exige tous les jetons réels.
>
> La corrélation peut être obtenue sans exposer ces valeurs.

Atelier guidé : lire, expliquer et vérifier le code

```
// Pseudocode volontairement incorrect :
cancelReservation(id)
if (!validCsrfToken(request.token)) {
    return response(403)
}
```

Le pseudocode annule avant de vérifier le token. Déplacez la vérification et l'autorisation avant l'effet, puis placez une assertion sur l'état de réservation et les appels aval. Le statut de refus seul ne suffit pas à détecter ce défaut d'ordre.

Retirer la défense ne protège pas la mutation. Le nom du champ ne change pas la validation. Le refus arrive trop tard pour éviter l'effet montré. Le statut ne corrige pas une action interdite.

Travail pratique : Préparez quatre tests : jeton absent, jeton d'un autre contexte, jeton correct sans permission, et parcours légitime complet.

Critères de correction : Vérifiez l'absence de mutation dans les trois refus. Testez depuis une seconde origine de laboratoire et incluez la politique SameSite réelle. Gardez une preuve des contrôles serveur au lieu de seulement observer la console.

Changement d'IBAN : protéger intention et autorisation

Un formulaire cross-site peut envoyer une demande application/x-www-form-urlencoded dans des conditions où le navigateur joint une identité ambiante. Le site hostile n'a pas besoin de lire la réponse pour que le changement d'IBAN produise un effet. SOP et CORS ne constituent donc pas une preuve d'intention légitime.

Vérifiez un token anti-CSRF lié au contexte prévu, ainsi que la provenance selon une politique stricte adaptée au parcours. Les cookies sécurisés et SameSite complètent cette défense, sans être assimilés à une autorisation générale. Une opération aussi sensible peut aussi demander une confirmation ou une réauthentification selon les exigences métier.

Le contrôle doit précéder l'écriture et tout appel externe. Testez token absent, incorrect et bon token sur l'objet d'un tiers : ce dernier cas exige encore un refus d'autorisation. Après chaque rejet, vérifiez que l'IBAN enregistré et les événements aval n'ont pas changé. Un statut d'erreur seul peut cacher une mutation trop tôt exécutée.

Points de vigilance : HttpOnly limite l'accès JavaScript au cookie, pas son envoi automatique. CORS n'empêche pas à lui seul l'envoi d'un formulaire cross-origin simple. Le jeton et les contrôles de provenance visent l'intention cross-site non

autorisée ; ils complètent la défense. Connaître un identifiant ne prouve pas l'intention du titulaire de la session.

Documentation de référence ↗

Exemple commenté

```
POST /lab/reservations/42/cancel
Cookie: lab_session=...
Champ de formulaire : csrf_token lié au contexte de la session
Contrôles : origine selon politique, token, droits et transition métier
```

Les points de suspension représentent un contexte fictif. Le contrôle d'accès intervient même si le jeton est correct.

Exercice de mise en pratique

Préparez quatre tests : jeton absent, jeton d'un autre contexte, jeton correct sans permission, et parcours légitime complet.

Voir une proposition de correction

Vérifiez l'absence de mutation dans les trois refus. Testez depuis une seconde origine de laboratoire et incluez la politique SameSite réelle. Gardez une preuve des contrôles serveur au lieu de seulement observer la console.

- Tester le refus sans effet.
- Lier le jeton au contexte voulu.
- Distinguer CSRF et autorisation.

Contrôle de connaissances

Un jeton CSRF correct donne-t-il tous les droits métier ?

> Réponse expliquée : Non, l'autorisation de l'action reste nécessaire — Le jeton répond à une autre menace.

Cours associé : 19. CSRF : déclenchement d'une action sous une identité ↗

Documentation de référence ↗ https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

20. XSS : données, code et contexte de sortie

Choisir une API de rendu adaptée à chaque destination.

Comprendre le mécanisme

Une XSS apparaît lorsqu'une donnée non fiable est interprétée comme du code dans un contexte de navigateur. Un encodage HTML ne sécurise pas automatiquement une chaîne placée dans JavaScript, une URL ou une feuille de style. La bonne défense commence par éviter les destinations interprétatives inutiles, puis par employer un mécanisme de rendu et une validation adaptés au contexte. La validation d'entrée seule n'exprime pas tous ces contextes de sortie.

Raisonner et vérifier

Pour afficher du texte dans le DOM, textContent préserve son rôle de donnée, contrairement à une insertion HTML libre. Les URL exigent aussi un contrôle de schéma et de destination selon le besoin. Les templates à échappement automatique réduisent le risque tant que leurs sorties sûres ne sont pas abusivement désactivées. Testez les chemins de rendu initiaux et les mises à jour dynamiques, y compris le corrigé et les exports affichés.

Contexte de sortie

L'échappement dépend du contexte de sortie. Chaque contexte a sa syntaxe et ses risques. Le texte HTML, un attribut, une URL et JavaScript ont des règles distinctes. Évitez de réutiliser une chaîne déjà encodée dans un autre contexte sans définir la protection réellement attendue.

> **Erreur fréquente à comprendre** Un échappement HTML est universel pour JavaScript et URL.
>
> Chaque contexte a sa syntaxe et ses risques.

Texte DOM

textContent traite une chaîne comme du texte. Cette API évite le parsing HTML de cette valeur. textContent place la valeur comme texte sans parser les balises qu'elle contient. C'est adapté à un libellé ; testez le DOM pour vérifier que des chevrons restent visibles et ne créent pas un élément actif.

> **Erreur fréquente à comprendre** textContent interprète la chaîne comme du HTML actif.
>
> Cette API évite le parsing HTML de cette valeur.

HTML interprété

innerHTML peut être un point d'injection. Cette API demande une politique de contenu adaptée. innerHTML demande au navigateur d'interpréter du balisage. Une donnée non fiable à cette destination exige une politique adaptée ; remplacer des caractères au hasard ne constitue pas une sanitisation complète.

> **Erreur fréquente à comprendre** innerHTML est sûr pour toute entrée utilisateur.
>
> Cette API demande une politique de contenu adaptée.

CSP complémentaire

Une CSP complète les défenses d'injection. Elle ne corrige pas automatiquement les points d'injection. Une politique CSP peut limiter certaines exécutions ou ressources, mais n'efface pas toutes les conséquences d'une injection. Gardez la séparation données/code et l'encodage adapté comme protections de base.

> **Erreur fréquente à comprendre** Une CSP remplace toute validation de sortie.
>
> Elle ne corrige pas automatiquement les points d'injection.

Capacités d'origine

Une XSS peut agir sous l'origine compromise. La frontière d'origine ne distingue pas ce script hostile. Une XSS peut lire des données accessibles et agir sous l'origine de la page. Le risque dépasse le seul vol de cookie : actions, accès API et modification d'interface peuvent rester possibles.

> **Erreur fréquente à comprendre** La SOP neutralise tout script de même origine.
>
> La frontière d'origine ne distingue pas ce script hostile.

HttpOnly limité

HttpOnly limite la lecture du cookie. Le script peut encore déclencher des actions. Même sans lire la valeur du cookie, un script peut envoyer des demandes authentifiées depuis la page. Cet attribut reste utile mais ne rend pas acceptable un point d'injection HTML.

> **Erreur fréquente à comprendre** HttpOnly supprime toute conséquence d'une XSS.
>
> Le script peut encore déclencher des actions.

Schémas URL

Une URL doit suivre une politique de schémas admis. Le contexte URL demande aussi un contrôle de destination et de schéma. Un attribut href échappé peut encore porter un schéma dangereux ou une destination interdite. Parsez et admettez les schémas et hôtes selon le besoin avant de rendre le lien.

> **Erreur fréquente à comprendre** Échapper une URL suffit à rendre tout schéma acceptable.
>
> Le contexte URL demande aussi un contrôle de destination et de schéma.

HTML autorisé

Un HTML autorisé demande une sanitisation adaptée. Le stockage ne rend pas sûre une entrée hostile. Si le produit exige du HTML utilisateur, employez un nettoyage maintenu avec une liste d'éléments et attributs admis. Vérifiez le résultat dans le contexte final plutôt qu'un simple retrait de script.

> **Erreur fréquente à comprendre** Tout HTML stocké en base est automatiquement fiable.
>
> Le stockage ne rend pas sûre une entrée hostile.

Rendu effectif

Les tests doivent vérifier le rendu effectif. L'interprétation par le navigateur détermine le comportement. Des tests sur la chaîne serveur peuvent manquer une interprétation du navigateur. Contrôlez éléments, attributs et comportement dans le DOM avec données factices contenant guillemets, chevrons et URL non admise.

> **Erreur fréquente à comprendre** Une chaîne présente dans le code suffit à prouver une exécution.
>
> L'interprétation par le navigateur détermine le comportement.

Données et code

Les données et le code doivent être séparés. La construction de code avec des données non fiables crée un risque. Préférez des API structurées et des valeurs passées comme données. Construire du JavaScript ou des handlers HTML par concaténation multiplie les contextes d'échappement et rend la preuve de sécurité fragile.

> **Erreur fréquente à comprendre** Concaténer des entrées dans du JavaScript est toujours sûr.
>
> La construction de code avec des données non fiables crée un risque.

Atelier guidé : lire, expliquer et vérifier le code

```
const value = '<img src=x onerror=alert(1)>';
const label = document.createElement('div');
label.textContent = value;
```

La valeur ressemble à une balise avec un gestionnaire, mais textContent la conserve comme texte. Le navigateur ne doit donc pas créer l'image active de cette chaîne dans ce div. Inspectez le DOM puis comparez avec un laboratoire isolé de parsing HTML, sans utiliser de données réelles.

Le contrôle est local à cette affectation. Cette API ne parse pas la valeur comme du HTML. Ce serait une autre sémantique de sink HTML. La création de texte ne réalise pas cet appel.

Travail pratique : Affichez une note de laboratoire contenant des chevrons, guillemets et caractères Unicode dans le rendu initial puis dans une mise à jour DOM.

Critères de correction : Vérifiez le texte attendu et l'absence de nouveaux éléments ou gestionnaires d'événements interprétés. Inspectez chaque destination de données ; remplacez les insertions HTML inutiles et testez séparément les URL permises.

Libellé DOM : conserver le texte comme texte

Le besoin fonctionnel est d'afficher un libellé, donc aucune balise n'a besoin d'être interprétée. innerHTML donne pourtant au navigateur un contexte de parsing HTML. Une valeur reçue peut alors créer des éléments et attributs actifs au lieu de rester une simple chaîne visible.

Affectez la valeur à textContent sur l'élément prévu. Une chaîne contenant des chevrons ou des guillemets s'affiche alors comme texte, sans devenir le balisage décrit par ces caractères. Ne pré-encodez pas arbitrairement une valeur avant textContent, au risque d'afficher des entités littérales ou de mélanger les contrats.

Testez le DOM obtenu : texte exact et absence d'élément enfant provenant de la valeur. Si un autre produit exige du HTML enrichi, il doit disposer d'un parcours différent avec nettoyage maintenu et politique précise. CSP et HttpOnly restent utiles mais ne remplacent pas le bon choix de destination pour ce libellé.

Points de vigilance : textContent insère du texte, sans analyser des balises ni des attributs événementiels. Une liste noire partielle manque d'autres constructions et contextes dangereux. HttpOnly limite la lecture d'un cookie, pas les actions qu'un script injecté peut effectuer. Un encodage URL n'est pas le contrat d'insertion du texte dans le DOM.

Documentation de référence ↗

Exemple commenté

```
const title = document.querySelector("[data-lab-title]");
title.textContent = userSuppliedLabel;
```

userSuppliedLabel reste ici du texte dans cet élément. Cet exemple n’autorise pas à placer la même valeur librement dans une URL ou du JavaScript.

Exercice de mise en pratique

Affichez une note de laboratoire contenant des chevrons, guillemets et caractères Unicode dans le rendu initial puis dans une mise à jour DOM.

Voir une proposition de correction

Vérifiez le texte attendu et l’absence de nouveaux éléments ou gestionnaires d’événements interprétés. Inspectez chaque destination de données ; remplacez les insertions HTML inutiles et testez séparément les URL permises.

- Identifier la destination exacte.
- Préférer les API de texte.
- Ne pas désactiver l’échappement pour corriger la mise en page.

Contrôle de connaissances

Quel mécanisme convient à un libellé qui doit rester du texte dans un élément DOM ?

> Réponse expliquée : textContent — L’API conserve le rôle textuel de la valeur.

Cours associé : 20. XSS : données, code et contexte de sortie ↗

Documentation de référence ↗ https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

21. CSP : politique de chargement et déploiement progressif

Définir une politique compatible avec les besoins démontrés de la page.

Comprendre le mécanisme

Content Security Policy restreint des sources et comportements du document selon ses directives. Une politique de script peut réduire les conséquences de certaines injections, sans corriger tous les défauts de rendu ni supprimer l'autorisation serveur. default-src fournit une valeur de repli pour certaines directives de chargement, mais pas pour toutes les directives. Une politique doit donc être lue dans son ensemble plutôt qu'assimilée à un mot-clé magique.

Raisonner et vérifier

Commencez par l'inventaire des ressources et des scripts réellement nécessaires. Le mode Report-Only permet d'observer des violations sans promettre un blocage. Réduisez les exceptions, vérifiez les parcours dynamiques puis activez une politique imposée. Les rapports peuvent révéler des URL ou des fragments de contexte : définissez collecte, minimisation et accès au lieu d'envoyer aveuglément ces données vers un tiers.

Observer avant bloquer

Report-Only observe une politique sans la bloquer. L'observation permet un déploiement progressif mais ne constitue pas le blocage. Report-Only produit une observation des violations selon ce mode sans imposer le blocage correspondant. Utilisez-le pour préparer une politique, puis vérifiez séparément la politique d'application effectivement envoyée.

> **Erreur fréquente à comprendre** Report-Only garantit le blocage de chaque violation.
>
> L'observation permet un déploiement progressif mais ne constitue pas le blocage.

Ressources nécessaires

Une CSP doit correspondre aux ressources nécessaires. Les dépendances et contextes du site doivent être inventoriés. Inventoriez scripts, styles, images et connexions réellement utilisés. Une politique trop large perd son intérêt ; une politique trop stricte non testée peut casser connexion, paiement ou affichage des erreurs.

> **Erreur fréquente à comprendre** Une politique copiée garantit tous les parcours.
>
> Les dépendances et contextes du site doivent être inventoriés.

Génération du nonce

Le nonce doit provenir d'une génération maîtrisée. Le serveur doit garder la maîtrise de la valeur admise. Le nonce doit venir d'un hasard cryptographique et suivre la réponse concernée. Une valeur fixe ou issue d'un timestamp permettrait de prévoir la capacité d'exécution que la politique essaie de limiter.

> **Erreur fréquente à comprendre** Une entrée utilisateur peut choisir le nonce de confiance.
>
> Le serveur doit garder la maîtrise de la valeur admise.

Encodage conservé

Une CSP ne remplace pas l'échappement contextuel. Les points de sortie doivent rester contrôlés. La CSP ne transforme pas une concaténation HTML dangereuse en donnée sûre. Corrigez les points d'injection et le contexte de sortie même lorsqu'un script de démonstration semble bloqué par le navigateur.

> **Erreur fréquente à comprendre** La présence d'un champ CSP corrige toutes les injections.
>
> Les points de sortie doivent rester contrôlés.

Pages d'erreur

Les pages d'erreur peuvent nécessiter leur propre vérification. Les réponses et ressources peuvent différer selon le parcours. Une erreur peut passer par un autre template ou un autre composant. Vérifiez ses headers et ressources aussi, afin que le chemin dégradé ne perde pas la politique de la page nominale.

> **Erreur fréquente à comprendre** Tester l'accueil prouve toutes les CSP d'administration.
>
> Les réponses et ressources peuvent différer selon le parcours.

Rapports minimisés

Les rapports peuvent contenir des données à minimiser. Le diagnostic peut contenir des URL ou informations sensibles. Les rapports de violation peuvent exposer des URL ou fragments de contexte. Limitez collecte, accès et conservation ; le canal de diagnostic doit être traité comme un traitement de données propre.

> **Erreur fréquente à comprendre** Tout rapport CSP peut être publié sans examen.
>
> Le diagnostic peut contenir des URL ou informations sensibles.

Directives distinctes

Des directives ont des rôles différents. Les directives ciblent des catégories différentes. script-src, connect-src et frame-ancestors ne contrôlent pas les mêmes capacités. Expliquez chaque choix et ses règles de repli ; default-src n'est pas une réponse universelle à toutes les directives.

> **Erreur fréquente à comprendre** script-src contrôle à lui seul toutes les intégrations de frames.
>
> Les directives ciblent des catégories différentes.

Politique reçue

Une politique effective doit être observée dans la réponse. Le trajet final et ses intermédiaires comptent. Inspectez les headers après passage par les proxies et les éventuelles politiques multiples. Une ligne dans le code source de configuration ne prouve pas le résultat effectivement interprété par le client.

> **Erreur fréquente à comprendre** Un fichier de configuration suffit à prouver son déploiement.
>
> Le trajet final et ses intermédiaires comptent.

Inline

Les ressources inline demandent une politique adaptée. Sa position dans la page ne suffit pas à établir son origine. Les scripts inline demandent une admission précise, par exemple nonce ou hash selon le cas. Évitez d'ajouter unsafe-inline pour résoudre une panne sans comprendre la capacité plus large ainsi réintroduite.

> **Erreur fréquente à comprendre** Tout script inline est automatiquement de confiance.
>
> Sa position dans la page ne suffit pas à établir son origine.

Retrait d'exception

Le retrait d'une exception doit être testé. Élargir une liste de sources peut affaiblir la restriction. Après suppression d'une origine admise, testez la fonctionnalité légitime et le refus du chargement retiré. Une politique peut sembler renforcée tout en laissant un autre chemin équivalent autorisé.

> **Erreur fréquente à comprendre** Ajouter des jokers partout durcit nécessairement la CSP.
>
> Élargir une liste de sources peut affaiblir la restriction.

Atelier guidé : lire, expliquer et vérifier le code

```
Content-Security-Policy-Report-Only: script-src 'self'

Le navigateur observe un script qui violerait cette politique.
```

La réponse contient seulement une politique Report-Only et le navigateur observe une violation potentielle. Ne concluez pas que le script a été bloqué par cette politique. Comparez observation et application dans un laboratoire, puis vérifiez que le parcours nominal continue de fonctionner sous la politique appliquée.

Le mode d'observation ne fournit pas le blocage de la politique. C'est la distinction de ce mode. Le nom complet et son mode comptent. Un diagnostic ne modifie pas les points d'injection.

Travail pratique : Inventoriez les ressources d'une page de laboratoire, déployez une politique d'observation puis une politique bloquante minimale compatible.

Critères de correction : Testez chargement normal, module JavaScript, téléchargement et ressource refusée. Distinguez rapport reçu et blocage effectif. Retirez les permissions non justifiées et vérifiez que les rapports ne contiennent pas de secrets métier.

Exemple commenté

```
Content-Security-Policy: default-src 'self'; object-src 'none'; base-uri 'none'; frame-ancestors
'self'
```

Ce fragment illustre plusieurs restrictions. Sa pertinence dépend des ressources de la page ; il n'est pas une configuration universelle à copier en production.

Exercice de mise en pratique

Inventoriez les ressources d'une page de laboratoire, déployez une politique d'observation puis une politique bloquante minimale compatible.

Voir une proposition de correction

Testez chargement normal, module JavaScript, téléchargement et ressource refusée. Distinguez rapport reçu et blocage effectif. Retirez les permissions non justifiées et vérifiez que les rapports ne contiennent pas de secrets métier.

- Lire les directives individuellement.

- Distinguer observation et application.
- Limiter les exceptions.

Contrôle de connaissances

Une politique Report-Only bloque-t-elle nécessairement les violations observées ?

> Réponse expliquée : Non, son rôle est l'observation selon ce mode — Ce mode ne doit pas être présenté comme une politique imposée.

Cours associé : 21. CSP : politique de chargement et déploiement progressif ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy

22. Nonces CSP : imprévisibilité et cohérence

Lier l'autorisation d'un script au document prévu sans réutilisation aveugle.

Comprendre le mécanisme

Un nonce CSP est une valeur imprévisible associée à des scripts explicitement autorisés pour un document. La valeur annoncée dans la politique doit correspondre à l'attribut des éléments concernés. Réutiliser une constante publique enlève cette propriété d'imprévisibilité. Ajouter un nonce à tout balisage reçu du client accorderait précisément une permission à du contenu non fiable ; le serveur doit maîtriser la création des éléments autorisés.

Raisonner et vérifier

Prévoyez la génération et le rendu dans le même parcours, ainsi que leurs interactions avec le cache HTML. Une politique actualisée avec un corps ancien peut casser le chargement ; une stratégie de partage mal conçue peut réutiliser un contexte non voulu. Le nonce n'est ni un token de session ni une signature JWT. Testez sa rotation, l'absence de valeur connue à l'avance et le refus d'un script qui n'appartient pas au rendu de confiance.

Imprévisibilité

Un nonce doit être imprévisible et renouvelé selon le contrat de réponse. La valeur fixe ne sépare pas les réponses et devient réutilisable. Générez un nonce cryptographique neuf pour la réponse selon le contrat. Une constante publiée dans le code ou un compteur rendrait prévisible la valeur que l'injection devrait ne pas pouvoir anticiper.

> **Erreur fréquente à comprendre** Une constante publique suffit comme nonce de sécurité.
>
> La valeur fixe ne sépare pas les réponses et devient réutilisable.

Correspondance

Le nonce doit correspondre entre politique et éléments autorisés. Le navigateur vérifie la correspondance prévue par la politique. La politique nonce- et l'attribut des scripts admis doivent porter la même valeur. Un mismatch bloque le script légitime ; un nonce copié trop largement peut au contraire admettre du contenu hostile.

> **Erreur fréquente à comprendre** Des valeurs différentes autorisent le script par similarité.
>
> Le navigateur vérifie la correspondance prévue par la politique.

Source cryptographique

random_bytes convient à une génération cryptographique. La précision temporelle ne fournit pas l'imprévisibilité requise. random_bytes fournit des octets adaptés au secret temporaire recherché. Évitez dates et générateurs non cryptographiques ; traitez une erreur de génération comme un échec contrôlé de la réponse.

> **Erreur fréquente à comprendre** Un timestamp seul constitue une source d'entropie adaptée.
>
> La précision temporelle ne fournit pas l'imprévisibilité requise.

Encodage base64

Base64 représente l'aléatoire sans l'augmenter. L'encodage ne crée pas de bits secrets supplémentaires. L'encodage rend les octets transportables sans ajouter d'entropie. Dix-huit octets forment six groupes de trois, donc 24 caractères base64 dans cet exemple, sans caractère de bourrage supplémentaire dans cette sortie.

> **Erreur fréquente à comprendre** Encoder deux fois double l'entropie du nonce.
>
> L'encodage ne crée pas de bits secrets supplémentaires.

Cache cohérent

Le cache doit conserver la cohérence du nonce et du document. La politique et les éléments du document doivent rester associés. Si HTML et header sont mis en cache séparément, ils peuvent porter des nonces différents. Définissez une stratégie cohérente pour la réponse complète et vérifiez aussi le risque de réutilisation de valeur.

> **Erreur fréquente à comprendre** Changer seulement l'en-tête d'une page cachée est toujours cohérent.
>
> La politique et les éléments du document doivent rester associés.

Admission ciblée

Un script hostile ne doit pas recevoir automatiquement le nonce. Le contrôle doit préserver la séparation de provenance. N'ajoutez pas le nonce automatiquement à tout élément script trouvé dans une entrée HTML. Seuls les scripts de confiance définis par le rendu doivent recevoir cette capacité d'exécution.

> **Erreur fréquente à comprendre** Tout élément DOM créé reçoit légitimement le nonce de confiance.
>
> Le contrôle doit préserver la séparation de provenance.

Pas un droit métier

Le nonce n'est pas un jeton d'autorisation métier. La politique de scripts ne décide pas des permissions. Le nonce CSP concerne l'admission de certains contenus par le navigateur. Il ne donne aucune permission sur une facture et ne remplace pas un jeton CSRF ou une décision serveur.

> **Erreur fréquente à comprendre** Connaître le nonce donne les droits administrateur.
>
> La politique de scripts ne décide pas des permissions.

Échec maîtrisé

Les erreurs de génération doivent être traitées proprement. Le mode de dégradation ne doit pas annuler la propriété attendue. Si la génération échoue, ne basculez pas vers un nonce fixe ou une politique ouverte. Préservez une réponse d'erreur contrôlée et des diagnostics privés sans exposer les détails de la génération.

> **Erreur fréquente à comprendre** Une panne aléatoire justifie un nonce prévisible permanent.
>
> Le mode de dégradation ne doit pas annuler la propriété attendue.

Injection corrigée

Une XSS doit toujours être corrigée à sa source. La CSP complète les contrôles d'entrée et de sortie. Même avec des scripts bloqués, une injection peut changer contenu ou formulaires. Corrigez la source de données/code et utilisez CSP comme défense complémentaire dont les limites sont explicites.

> **Erreur fréquente à comprendre** Le nonce rend sûre toute concaténation HTML.
>
> La CSP complète les contrôles d'entrée et de sortie.

Deux chemins

Un test doit couvrir les scripts autorisés et refusés. Le cas négatif est nécessaire pour vérifier la restriction. Testez script légitime avec nonce correct, script sans nonce et script avec valeur incorrecte. Vérifiez exécution effective dans le navigateur, pas seulement présence textuelle des attributs HTML.

> **Erreur fréquente à comprendre** Un script nominal exécuté prouve que tout autre est bloqué.
>
> Le cas négatif est nécessaire pour vérifier la restriction.

Atelier guidé : lire, expliquer et vérifier le code

```
<?php
declare(strict_types=1);
$nonce = base64_encode(random_bytes(18));
echo strlen($nonce);
```

Calculez la longueur encodée à partir des dix-huit octets générés, puis vérifiez-la à l'exécution. Ce code illustre la génération, pas l'intégration complète avec les scripts et le cache. Ajoutez un rendu de laboratoire et contrôlez correspondance des valeurs et renouvellement entre réponses.

C'est la taille avant encodage. 144 est le nombre de bits d'entrée. Le contenu varie, mais cette longueur est fixe. 18 octets forment six groupes de trois, soit 24 caractères base64.

Travail pratique : Générez deux réponses de laboratoire et comparez leurs politiques, corps et valeurs de nonce. Ajoutez un élément non autorisé sans nonce.

Critères de correction : Vérifiez correspondance à l'intérieur de chaque réponse et renouvellement selon la stratégie retenue. Testez ensuite cache, fragment rendu et erreur de template pour détecter une valeur fixe ou une autorisation ajoutée trop largement.

Exemple commenté

```
<?php
$nonce = base64_encode(random_bytes(18));
// Utiliser cette valeur dans la politique et dans les scripts maîtrisés du même document.
// Ne pas appliquer un nonce à du HTML libre fourni par le client.
```

La génération utilise une source d'aléa adaptée. La cohérence politique-document et la provenance du script restent à vérifier dans le rendu complet.

Exercice de mise en pratique

Générez deux réponses de laboratoire et comparez leurs politiques, corps et valeurs de nonce. Ajoutez un élément non autorisé sans nonce.

Voir une proposition de correction

Vérifiez correspondance à l'intérieur de chaque réponse et renouvellement selon la stratégie retenue. Testez ensuite cache, fragment rendu et erreur de template pour détecter une valeur fixe ou une autorisation ajoutée trop largement.

- Nonce imprévisible.
- Même contexte politique-document.
- Pas d'autorisation automatique du contenu client.

Contrôle de connaissances

Un nonce CSP constant et public conserve-t-il le but d'imprévisibilité ?

> Réponse expliquée : Non — Une valeur connue à l'avance n'apporte pas cette propriété.

Cours associé : 22. Nonces CSP : imprévisibilité et cohérence ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Global_attributes/nonce

23. Encadrement et clickjacking

Décider qui peut intégrer la page, indépendamment des cadres qu'elle charge.

Comprendre le mécanisme

Une interface intégrée dans un cadre peut être présentée de façon trompeuse et amener un utilisateur à agir dans un contexte inattendu. frame-ancestors indique quels ancêtres peuvent intégrer le document. Cette directive ne répond pas à la même question que frame-src, qui concerne des cadres chargés par la page. Elle ne doit pas être supposée couverte automatiquement par default-src.

Raisonner et vérifier

Définissez une politique de cadrage par type de page : interface personnelle, formulaire sensible ou contenu destiné à l'intégration. Vérifiez tous les ancêtres pertinents et les besoins de partenaires connus. Le contrôle doit être servi dans le contexte pris en charge pour la directive ; une balise meta n'est pas un substitut universel aux en-têtes. Une protection de cadrage ne remplace pas le contrôle d'autorisation de l'action.

Parents autorisés

frame-ancestors limite les parents pouvant intégrer la page. La directive porte sur les ancêtres de navigation. frame-ancestors décide quels parents peuvent intégrer le document. Une page sensible peut refuser tout encadrement ou admettre une liste précise selon ses parcours légitimes ; servez la politique dans le canal prévu.

> **Erreur fréquente à comprendre** frame-ancestors choisit les images autorisées.
>
> La directive porte sur les ancêtres de navigation.

Sens des frames

frame-src et frame-ancestors ont des sens différents. La direction de l'intégration est différente. frame-src limite les frames que cette page charge, tandis que frame-ancestors concerne les pages qui l'intègrent. Une restriction de l'un ne prouve pas automatiquement la propriété de l'autre.

> **Erreur fréquente à comprendre** frame-src définit tous les sites pouvant encadrer la page.
>
> La direction de l'intégration est différente.

Pas de repli universel

frame-ancestors ne dépend pas d'un repli default-src. Cette directive demande sa propre définition. default-src ne fournit pas un repli à frame-ancestors. Si l'objectif est de limiter les parents, la politique doit traiter cette directive elle-même et être vérifiée dans le navigateur.

> **Erreur fréquente à comprendre** default-src interdit implicitement tous les parents par la même règle.
>
> Cette directive demande sa propre définition.

Clic trompé

Le clickjacking peut tromper un clic sur une interface légitime. La composition visuelle peut suffire à détourner l'intention. Le clickjacking peut superposer une interface qui masque une action réelle. L'utilisateur clique alors une page légitime sans comprendre la cible ; le risque ne nécessite pas toujours une injection dans cette page.

> **Erreur fréquente à comprendre** Le clickjacking exige une modification du code serveur.
>
> La composition visuelle peut suffire à détourner l'intention.

Usages embarqués

Un parcours embarqué légitime doit être inventorié. Certaines intégrations sont des besoins à décider explicitement. Inventoriez portails partenaires et widgets qui doivent intégrer la page. Une interdiction globale non testée peut casser ces parcours ; une admission trop large peut inclure des parents peu maîtrisés.

> **Erreur fréquente à comprendre** Interdire tout parent ne peut jamais gêner le produit.
>
> Certaines intégrations sont des besoins à décider explicitement.

Pages sensibles

La politique doit être servie sur les pages sensibles. Les réponses peuvent avoir des en-têtes différents. Appliquez la politique sur formulaires et chemins d'erreur concernés, pas seulement sur la page d'accueil. Vérifiez la réponse effective au travers du proxy pour chaque point d'entrée pertinent.

> **Erreur fréquente à comprendre** Protéger l'accueil suffit à protéger toute administration.
>
> Les réponses peuvent avoir des en-têtes différents.

Test navigateur

Un refus d'intégration doit être testé dans un navigateur. Navigation directe et encadrement sont des contextes différents. Créez un parent admis et un parent refusé en laboratoire, puis observez le chargement réel. Un test qui cherche seulement le mot frame-ancestors dans le code ne mesure pas cette frontière.

> **Erreur fréquente à comprendre** Un lien externe équivaut au test d'une iframe.
>
> Navigation directe et encadrement sont des contextes différents.

Droits conservés

L'autorisation métier reste nécessaire. La protection visuelle ne donne pas de permission. Un parent autorisé peut présenter une action que l'utilisateur n'a pas le droit d'exécuter. Le serveur doit conserver autorisation, validation et protection d'intention selon le contexte de la mutation.

> **Erreur fréquente à comprendre** Interdire les frames autorise tout utilisateur à payer.
>
> La protection visuelle ne donne pas de permission.

Parent de confiance

Un parent admis doit rester un contexte de confiance. Le canal sécurisé n'établit pas cette confiance métier. Admettre un domaine partenaire donne une capacité d'encadrement. Évaluez sa maîtrise et les sous-domaines inclus ; une règle trop large peut inclure du contenu que d'autres personnes contrôlent.

> **Erreur fréquente à comprendre** Toute origine avec HTTPS est un parent acceptable.
>
> Le canal sécurisé n'établit pas cette confiance métier.

Compatibilité

Les mécanismes de compatibilité doivent être cohérents. Les comportements doivent être vérifiés sur les clients visés. Si plusieurs mécanismes sont servis, gardez leur intention cohérente et connaissez le périmètre des clients. Des headers contradictoires compliquent le diagnostic et peuvent produire des résultats différents selon le navigateur.

> **Erreur fréquente à comprendre** Des politiques contradictoires sont toujours interprétées identiquement.
>
> Les comportements doivent être vérifiés sur les clients visés.

Atelier guidé : lire, expliquer et vérifier le code

```
Content-Security-Policy: default-src 'self'; frame-src 'self'

Besoin : refuser l'encadrement par des sites non admis.
```

La politique montrée contrôle les ressources et les frames chargées, mais ne fournit pas la restriction de parents demandée. Ajoutez une politique frame-ancestors conforme aux parcours légitimes puis testez une intégration autorisée et une interdite dans des contextes distincts.

Répéter la même directive ne change pas sa direction. Le type n'exprime pas une liste d'ancêtres. frame-src concerne les frames chargées, pas tous les parents admis. Le cookie ne définit pas cette restriction.

Travail pratique : Testez une page sensible dans un cadre same-origin puis dans un cadre d'une seconde origine de laboratoire.

Critères de correction : Vérifiez le refus ou l'autorisation attendus selon la politique. Testez également la navigation directe et les droits de mutation : le refus de cadrage ne doit pas être votre seule défense.

Exemple commenté

```
Content-Security-Policy: frame-ancestors 'none'
```

Ce choix interdit l'intégration dans le contexte de cette directive. Il peut être inadapté à un contenu dont le contrat prévoit explicitement une intégration partenaire.

Exercice de mise en pratique

Testez une page sensible dans un cadre same-origin puis dans un cadre d'une seconde origine de laboratoire.

Voir une proposition de correction

Vérifiez le refus ou l'autorisation attendus selon la politique. Testez également la navigation directe et les droits de mutation : le refus de cadrage ne doit pas être votre seule défense.

- Distinguer frame-src et frame-ancestors.
- Choisir la politique par usage.
- Tester le contexte d'en-tête réel.

Contrôle de connaissances

Quelle directive décide principalement qui peut intégrer le document ?

> Réponse expliquée : frame-ancestors — Elle porte sur les ancêtres autorisés.

Cours associé : 23. Encadrement et clickjacking ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy/frame-ancestors

24. HSTS : persistance d'une exigence HTTPS

Préparer le périmètre et la durée avant d'imposer la politique.

Comprendre le mécanisme

Strict-Transport-Security permet au navigateur de mémoriser une exigence de transport HTTPS pour un hôte dans les conditions du mécanisme. Ce choix n'installe pas un certificat valide et ne répare pas un service indisponible en TLS. Les options de sous-domaines étendent le périmètre et demandent un inventaire préalable. Le préchargement ajoute encore un processus et des conséquences distinctes d'un simple en-tête reçu.

Raisonner et vérifier

Déployez progressivement, après vérification des sous-domaines et du renouvellement des certificats. Une modification de configuration n'efface pas instantanément toutes les politiques déjà mémorisées par les clients. Testez les redirections, les liens générés et les contextes derrière proxy. Une courte durée de laboratoire facilite l'observation sans transformer une erreur de périmètre en engagement durable.

Politique mémorisée

HSTS mémorise une politique de transport côté client. Le contrôle ne provisionne pas les certificats. HSTS informe le client d'une exigence HTTPS pour le périmètre concerné. Il ne génère aucun certificat et ne répare pas une chaîne invalide ; ces responsabilités doivent être traitées séparément.

> **Erreur fréquente à comprendre** HSTS crée automatiquement un certificat serveur.
>
> Le contrôle ne provisionne pas les certificats.

Sous-domaines

La portée includeSubDomains demande un inventaire. La politique peut étendre ses exigences aux sous-domaines. includeSubDomains étend le périmètre au-dessous du domaine concerné. Inventoriez services anciens, noms délégués et dépendances avant activation, car ils devront eux aussi satisfaire la politique de transport.

> **Erreur fréquente à comprendre** includeSubDomains ne peut affecter aucun sous-domaine.
>
> La politique peut étendre ses exigences aux sous-domaines.

Durée

Un max-age élevé prolonge la politique mémorisée. Les clients peuvent conserver l'état reçu précédemment. Un grand max-age garde l'exigence longtemps dans les clients qui l'ont apprise. Une modification serveur ne peut pas effacer instantanément tout cet état déjà distribué dans les navigateurs.

> **Erreur fréquente à comprendre** Un changement serveur efface instantanément tout état client.
>
> Les clients peuvent conserver l'état reçu précédemment.

Apprentissage sûr

HSTS s'apprend via le canal sécurisé prévu. La politique ne doit pas être établie par une source non authentifiée. Le client apprend HSTS via le canal sécurisé prévu. Un header reçu sur un trajet non conforme ne doit pas être présenté comme une garantie équivalente pour établir cette confiance.

> **Erreur fréquente à comprendre** Un champ HSTS sur toute réponse HTTP non sécurisée suffit.
>
> La politique ne doit pas être établie par une source non authentifiée.

Déploiement progressif

Une montée progressive limite les surprises. La portée et le renouvellement doivent être maîtrisés. Commencez avec un périmètre et une durée contrôlés, vérifiez les erreurs et augmentez selon le plan. Cette progression doit rester compatible avec vos exigences de transport et les clients concernés.

> **Erreur fréquente à comprendre** Activer la durée maximale sans test constitue toujours la meilleure reprise.
>
> La portée et le renouvellement doivent être maîtrisés.

Certificat

Un certificat invalide doit être corrigé. Le mécanisme de transport ne corrige pas l'identité du pair. Le service doit présenter un certificat valide et renouvelé. HSTS peut rendre certaines erreurs plus bloquantes pour le visiteur ; sa présence augmente l'importance d'une exploitation TLS correcte.

> **Erreur fréquente à comprendre** HSTS répare un mauvais nom de certificat.
>
> Le mécanisme de transport ne corrige pas l'identité du pair.

Premier contact

Le premier contact dépend du contexte et des politiques préchargées. L'apprentissage et le préchargement ont des modèles distincts. La protection du premier accès dépend de l'état client et d'éventuelles politiques préchargées. Évitez de promettre une connaissance préalable universelle simplement parce qu'un header est désormais configuré.

> **Erreur fréquente à comprendre** HSTS annoncé protège rétroactivement toutes les premières visites.
>
> L'apprentissage et le préchargement ont des modèles distincts.

Retrait

Le retrait doit tenir compte de l'état déjà mémorisé. L'absence de nouveau champ ne supprime pas nécessairement l'ancien état. Un rollback doit tenir compte des clients qui ont mémorisé la politique. Maintenez HTTPS pendant la transition prévue et vérifiez comment la nouvelle politique est reçue par les clients déjà concernés.

> **Erreur fréquente à comprendre** Supprimer l'en-tête efface forcément la politique antérieure.
>
> L'absence de nouveau champ ne supprime pas nécessairement l'ancien état.

Métier distinct

Le transport sécurisé ne vérifie pas les droits métier. Les permissions restent dans l'application. Une connexion HTTPS exigée peut transporter une demande interdite. Vérifiez toujours l'utilisateur, la ressource précise et l'intention ; la persistance de transport n'établit aucun droit de modification.

> **Erreur fréquente à comprendre** HSTS valide automatiquement tout paiement.
>
> Les permissions restent dans l'application.

Vérification de portée

Les tests doivent inclure les sous-domaines concernés. Les services inclus peuvent avoir des configurations différentes. Testez domaine principal et sous-domaines inclus avec certificats et routages réels. Le succès de la page principale ne démontre pas le maintien du transport sur un service secondaire oublié.

> **Erreur fréquente à comprendre** Tester un seul hôte démontre toute la portée annoncée.
>
> Les services inclus peuvent avoir des configurations différentes.

Atelier guidé : lire, expliquer et vérifier le code

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Le header engage une durée d'un an et inclut les sous-domaines. Expliquez ces deux dimensions avant de le recopier. Vérifiez l'inventaire et le renouvellement des services concernés, puis distinguez cette exigence de transport de la validité du certificat et des permissions applicatives.

includeSubDomains élargit précisément la portée. La portée et le maintien de TLS doivent correspondre aux services inclus. HSTS ne provisionne pas les certificats. Les clients peuvent conserver la politique.

Travail pratique : Inventoriez les hôtes et dépendances avant d'envisager une politique couvrant les sous-domaines. Simulez un renouvellement de certificat.

Critères de correction : Vérifiez chaque service HTTPS et le comportement d'un navigateur neuf puis déjà exposé à la politique. Documentez durée, périmètre et procédure d'incident. N'activez pas une extension de périmètre sur la seule base du domaine principal.

Exemple commenté

```
Strict-Transport-Security: max-age=300
```

Cette courte durée est illustrative pour un laboratoire HTTPS contrôlé. Elle ne constitue pas une recommandation de durée finale ni une demande de préchargement.

Exercice de mise en pratique

Inventoriez les hôtes et dépendances avant d'envisager une politique couvrant les sous-domaines. Simulez un renouvellement de certificat.

Voir une proposition de correction

Vérifiez chaque service HTTPS et le comportement d'un navigateur neuf puis déjà exposé à la politique. Documentez durée, périmètre et procédure d'incident. N'activez pas une extension de périmètre sur la seule base du domaine principal.

- Inventorier les sous-domaines.
- Assurer le renouvellement TLS.

- Prévoir la persistance côté client.

Contrôle de connaissances

HSTS crée-t-il un certificat serveur valide ?

> Réponse expliquée : Non, il impose une politique de transport au navigateur — Le certificat est un autre élément de la chaîne TLS.

Cours associé : 24. HSTS : persistance d'une exigence HTTPS ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Strict-Transport-Security

25. Types de contenu et refus d’interprétation imprévue

Aligner le type annoncé avec le contenu et son usage.

Comprendre le mécanisme

Content-Type décrit la représentation ; une extension de fichier ne suffit pas à garantir que le contenu lui correspond. Certains comportements de navigateur peuvent essayer de déduire un type lorsque le contexte est ambigu. X-Content-Type-Options: nosniff limite des comportements de ce genre selon les contextes pris en charge. Il ne valide pas magiquement un upload et n’accorde pas une confiance générale au fichier.

Raisonner et vérifier

Servez les ressources maîtrisées avec des types explicites et séparez les fichiers utilisateurs des documents applicatifs sensibles. Selon l’usage, un téléchargement forcé ou une origine dédiée peut compléter la politique. Détectez le contenu, imposez les formats admis et vérifiez les réponses réellement envoyées par les différentes couches. Testez le navigateur final : un validateur de suffixe dans un formulaire n’est pas une preuve d’interprétation sûre.

Type déclaré

Content-Type décrit le type du contenu concerné. Le nom ne constitue pas à lui seul le contrat du message. Content-Type doit correspondre à la représentation réellement servie. Ne réutilisez pas aveuglément le type fourni pendant l’upload, car le client pouvait annoncer une valeur sans rapport avec les octets reçus.

> **Erreur fréquente à comprendre** L’extension du fichier remplace toujours Content-Type.
>
> Le nom ne constitue pas à lui seul le contrat du message.

Interprétation

nosniff limite certains comportements de détection du navigateur. Il ne transforme pas le contenu reçu. nosniff limite des comportements de détection dans les contextes navigateur concernés. Il ne lit pas le fichier pour certifier son innocuité ni ne remplace la validation métier d’un document importé.

> **Erreur fréquente à comprendre** nosniff désinfecte automatiquement tous les fichiers.
>
> Il ne transforme pas le contenu reçu.

Validation du fichier

La validation d’upload doit examiner le fichier selon son usage. Une déclaration client ne prouve pas le contenu. Examinez format, structure et limites selon l’usage prévu. Une extension PDF et un MIME client cohérent ne garantissent pas qu’un parseur acceptera un document sûr et compatible avec vos besoins.

> **Erreur fréquente à comprendre** Déclarer image/png prouve que le fichier est une image acceptable.
>
> Une déclaration client ne prouve pas le contenu.

Import et restitution

Servir un fichier est une décision distincte de son import. Origine, type et présentation influencent les risques. Un fichier accepté peut être servi dans un autre contexte et sous une origine sensible. Définissez stockage, disposition et type de réponse séparément de la décision d'import initiale.

> **Erreur fréquente à comprendre** Un fichier accepté peut être servi dans tout contexte sans analyse.
>
> Origine, type et présentation influencent les risques.

Chargement script

Une ressource script doit avoir un type approprié. Les règles de chargement et de type doivent être respectées. Un script servi avec un type incompatible peut être refusé selon les règles de chargement. Vérifiez le résultat navigateur et corrigez le type plutôt que retirer la protection pour masquer l'erreur.

> **Erreur fréquente à comprendre** Le navigateur doit exécuter tout texte téléchargé comme script.
>
> Les règles de chargement et de type doivent être respectées.

Accès au téléchargement

Le téléchargement ne dispense pas de contrôle d'accès. La présentation ne donne pas la permission. La disposition attachment ne rend pas un document public par nature. Vérifiez l'autorisation avant d'ouvrir et transmettre les octets, y compris pour une URL devinable ou un identifiant valide.

> **Erreur fréquente à comprendre** Un type application/pdf autorise tout lecteur.
>
> La présentation ne donne pas la permission.

Erreurs

Les erreurs doivent aussi avoir un type cohérent. Les consommateurs ont besoin d'un contrat correct. Une réponse d'erreur renvoyée à la place d'une ressource doit garder un type et un corps cohérents. Les chemins de panne ne doivent pas devenir un moyen de servir du contenu interprété autrement.

> **Erreur fréquente à comprendre** Un corps HTML peut toujours être traité comme JSON sans risque.
>
> Les consommateurs ont besoin d'un contrat correct.

HTML

Un type sûr ne remplace pas l'échappement HTML. Le contexte de sortie reste à protéger. Déclarer text/html ne protège pas une valeur injectée dans ce HTML. L'encodage contextuel ou la sanitisation reste nécessaire ; le type annonce une interprétation et ne rend pas ses données sûres.

> **Erreur fréquente à comprendre** Content-Type rend toute interpolation HTML inoffensive.
>
> Le contexte de sortie reste à protéger.

Réponse réelle

Les tests doivent observer les réponses réellement servies. Il faut vérifier le message final. Observez headers après proxy et CDN, puis le comportement du client. Une configuration de stockage correcte peut être contournée par une autre route de fichiers qui expose le même contenu différemment.

> **Erreur fréquente à comprendre** Un suffixe renommé prouve l'en-tête déployé.
>
> Il faut vérifier le message final.

Origine de distribution

Les contenus actifs demandent un périmètre de distribution adapté. Les formats et leurs interprétations diffèrent. Les contenus actifs non fiables peuvent nécessiter une origine séparée avec capacités réduites. Une séparation doit être conçue avec cookies, CORS et navigation, pas seulement avec un nouveau chemin URL.

> **Erreur fréquente à comprendre** Toutes les images et tous les documents ont le même comportement actif.
>
> Les formats et leurs interprétations diffèrent.

Atelier guidé : lire, expliquer et vérifier le code

```
Content-Type: text/plain
X-Content-Type-Options: nosniff

console.log('lab');
```

Le contenu ressemble à du JavaScript mais est servi text/plain avec nosniff. Dans un contexte de chargement script concerné, le client doit appliquer les règles de type et ne pas interpréter librement le texte comme code. Vérifiez ce comportement dans un navigateur de laboratoire.

Le client doit appliquer les règles de type pertinentes au chargement. Cette protection limite certaines détections, elle ne requalifie pas le contenu. Un mot du corps n'est pas une autorisation d'exécution. Les règles de type font partie du contrat navigateur.

Travail pratique : Dans le laboratoire, servez une ressource maîtrisée puis une copie avec un type incorrect. Comparez les chargements dans les contextes pertinents.

Critères de correction : Vérifiez types, disposition de téléchargement et absence d'exécution indésirable. Pour les fichiers clients, ajoutez un contrôle de contenu et une zone de stockage sans capacité d'exécution applicative.

Exemple commenté

```
Content-Type: application/json
X-Content-Type-Options: nosniff
```

L'exemple annonce un document JSON. Le serveur doit encore produire ce format et éviter de refléter librement des données dans un contexte interprétable.

Exercice de mise en pratique

Dans le laboratoire, servez une ressource maîtrisée puis une copie avec un type incorrect. Comparez les chargements dans les contextes pertinents.

Voir une proposition de correction

Vérifiez types, disposition de téléchargement et absence d'exécution indésirable. Pour les fichiers clients, ajoutez un contrôle de contenu et une zone de stockage sans capacité d'exécution applicative.

- Type explicite.
- Validation de contenu indépendante.
- Tests dans le contexte de chargement réel.

Contrôle de connaissances

nosniff valide-t-il le contenu métier d’un fichier reçu ?

> Réponse expliquée : Non, c’est une politique d’interprétation côté navigateur — La validation serveur du fichier reste une autre responsabilité.

Cours associé : 25. Types de contenu et refus d’interprétation imprévue ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Content-Type-Options

26. Referrer-Policy et données dans les URL

Limiter la diffusion de contexte sans supposer qu'un en-tête efface toutes les traces.

Comprendre le mécanisme

Le champ Referer peut transmettre un contexte de navigation selon la politique applicable. Referrer-Policy permet de limiter cette information pour les demandes concernées. Le nom historique Referer et celui de la politique sont différents. Les URL peuvent également apparaître dans l'historique, les journaux et les outils de supervision ; les secrets placés dans une chaîne de requête peuvent donc fuir par plusieurs canaux.

Raisonner et vérifier

Définissez une politique adaptée aux parcours sensibles puis vérifiez sa présence dans les réponses finales. Ne traitez pas cette directive comme une garantie d'absence de toute métadonnée : Origin, cookies et autres champs suivent leurs contrats propres. Réduisez d'abord la quantité de données sensibles dans les URL et dans les événements collectés. Les tests doivent couvrir navigation, sous-ressource et redirection selon le besoin.

Informations envoyées

Referrer-Policy contrôle des informations de référence transmises. Son périmètre ne couvre pas tous les stockages. Referrer-Policy règle les informations de référence transmises dans les parcours concernés. Elle agit sur des envois futurs selon le contexte et ne constitue pas un mécanisme général d'effacement de données.

> **Erreur fréquente à comprendre** Referrer-Policy efface tout historique local.
>
> Son périmètre ne couvre pas tous les stockages.

No-referrer

no-referrer peut supprimer le champ Referer du parcours concerné. La politique d'envoi n'efface pas les traces antérieures. Cette politique peut supprimer Referer sur une navigation concernée. Vérifiez au destinataire ce qui est réellement reçu ; l'absence du champ n'empêche pas d'autres canaux de conserver l'URL visitée.

> **Erreur fréquente à comprendre** no-referrer supprime tous les journaux serveur déjà écrits.
>
> La politique d'envoi n'efface pas les traces antérieures.

Secrets en URL

Les secrets ne doivent pas être placés dans les URL. Historique, logs et autres canaux peuvent conserver l'URL. Un token durable peut apparaître dans historique, captures, journaux ou outils de supervision. Retirez-le du design lorsque possible et utilisez les mécanismes de transport prévus pour les credentials.

> **Erreur fréquente à comprendre** Une politique de referrer rend toute URL secrète partout.
>
> Historique, logs et autres canaux peuvent conserver l'URL.

Granularité

L'origine et l'URL complète sont des granularités différentes. Le détail transmis dépend de la politique et du contexte. L'origine ne contient pas chemin et query, contrairement à une URL complète. Choisissez la politique selon les données que ces composants peuvent révéler et les parcours sortants réellement utilisés.

> **Erreur fréquente à comprendre** Envoyer l'origine transmet nécessairement tout le chemin.
>
> Le détail transmis dépend de la politique et du contexte.

Parcours sortants

Les liens sortants doivent être testés selon les parcours. La chaîne de navigation peut changer le résultat. Testez lien, redirection et ressource chargée dans les contextes pertinents. Une observation sur une navigation unique ne décrit pas automatiquement toutes les transmissions de référence du produit.

> **Erreur fréquente à comprendre** Un test sur la page d'accueil couvre chaque redirection.
>
> La chaîne de navigation peut changer le résultat.

Fuite existante

Une donnée déjà diffusée demande un traitement spécifique. La modification n'est pas une révocation de données copiées. Une valeur déjà présente dans des logs reste une copie à traiter. Selon sa capacité, retirez ou remplacez le secret et appliquez la procédure de conservation ou de nettoyage appropriée.

> **Erreur fréquente à comprendre** Changer la politique retire toutes les copies distantes.
>
> La modification n'est pas une révocation de données copiées.

Journaux

La politique de confidentialité doit inclure les logs. D'autres traces peuvent révéler le parcours. La confidentialité doit inclure logs proxy, application et APM. Réduire Referer n'empêche pas le serveur de journaliser directement une query sensible reçue dans la demande initiale.

> **Erreur fréquente à comprendre** Le referrer est la seule source de données de navigation.
>
> D'autres traces peuvent révéler le parcours.

Pas une identité

Une référence reçue n'authentifie pas l'utilisateur. Le contexte de navigation n'est pas une identité vérifiée. Referer peut manquer et ne doit pas être utilisé comme credential. Il peut contribuer à certains diagnostics ou contrôles contextuels, sans remplacer une authentification et une autorisation serveur.

> **Erreur fréquente à comprendre** Referer suffit comme preuve de propriété d'un compte.
>
> Le contexte de navigation n'est pas une identité vérifiée.

Observation destinataire

Les tests doivent contrôler la valeur observée au destinataire. Le navigateur applique la politique lors des requêtes. Utilisez un endpoint factice qui enregistre seulement les champs utiles au test. Comparez la valeur reçue à la politique attendue plutôt que déduire l'effet du seul header de départ.

> **Erreur fréquente à comprendre** Lire seulement le HTML prouve tout envoi réseau.
>
> Le navigateur applique la politique lors des requêtes.

Canal

Une protection du canal reste nécessaire. Le chiffrement relève du transport sécurisé. Une politique de référence ne chiffre pas les échanges. Maintenez HTTPS et ses validations pour protéger le transport, tout en minimisant les données que les extrémités reçoivent et conservent.

> **Erreur fréquente à comprendre** Referrer-Policy chiffre tous les échanges HTTP.
>
> Le chiffrement relève du transport sécurisé.

Atelier guidé : lire, expliquer et vérifier le code

```
Referrer-Policy: no-referrer

Journal antérieur : GET /lab/reset?token=public-fixture-only
```

Le header no-referrer n'efface pas la ligne de journal antérieure. La valeur illustrative y a déjà été copiée. Distinguez prévention des transmissions futures et traitement d'une diffusion existante, puis vérifiez l'ensemble des traces où une vraie capacité pourrait avoir été conservée.

Le champ n'exécute pas une purge. Ce n'est pas son périmètre. Une révocation demande un mécanisme applicatif. Il faut gérer la collecte et les copies séparément.

Travail pratique : Inspectez un parcours de laboratoire contenant un paramètre fictif sensible, puis définissez une représentation qui n'expose pas ce contexte inutilement.

Critères de correction : Relevez les champs reçus par une destination locale et les événements de votre propre journal. Appliquez une politique explicite puis vérifiez les canaux restants ; remplacez le paramètre sensible par un identifiant opaque ou un mécanisme adapté.

Exemple commenté

```
Referrer-Policy: no-referrer
```

Ce choix réduit l'envoi du champ de référence dans le périmètre prévu. Il ne supprime pas rétroactivement des URL déjà enregistrées ailleurs.

Exercice de mise en pratique

Inspectez un parcours de laboratoire contenant un paramètre fictif sensible, puis définissez une représentation qui n'expose pas ce contexte inutilement.

Voir une proposition de correction

Relevez les champs reçus par une destination locale et les événements de votre propre journal. Appliquez une politique explicite puis vérifiez les canaux restants ; remplacez le paramètre sensible par un identifiant opaque ou un mécanisme adapté.

- Ne pas placer des secrets inutilement dans les URL.

- Tester les champs réellement transmis.
- Minimiser aussi les journaux.

Contrôle de connaissances

Referrer-Policy supprime-t-elle automatiquement les secrets déjà présents dans les logs ?

> Réponse expliquée : Non — La politique de navigateur n'est pas un effacement rétroactif.

Cours associé : 26. Referrer-Policy et données dans les URL ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Referrer-Policy

27. Caches : fraîcheur, stockage et confidentialité

Nommer le cache et sa clé avant de discuter de durée.

Comprendre le mécanisme

Un navigateur, un proxy partagé et un cache applicatif sont des mécanismes distincts. Un objet peut être frais pour une certaine durée sans être autorisé pour tout utilisateur. Une clé de cache qui omet une dimension pertinente peut produire une réponse incorrecte ou divulguer un contenu privé. L'invalidation, la variation des représentations et les droits d'accès doivent être conçus ensemble.

Raisonner et vérifier

Testez les caches avec au moins deux identités et plusieurs variantes de représentation. Une seconde réponse rapide ne prouve pas qu'elle est correcte pour le nouvel acteur. Inventoriez les clés, les règles d'admission, les durées et le traitement des erreurs. Contrôlez aussi les caches internes à l'application : un en-tête HTTP prudent ne répare pas un résultat déjà mélangé par un cache serveur mal indexé.

Partage

Un cache partagé peut répondre à plusieurs utilisateurs. La portée du cache détermine les risques de partage. Un cache public peut répondre à plusieurs utilisateurs sans repasser par l'origine. Une donnée personnelle doit donc avoir une politique de stockage et de partage cohérente avant d'entrer dans ce cache.

> **Erreur fréquente à comprendre** Tout cache HTTP est privé à un seul compte.
>
> La portée du cache détermine les risques de partage.

Clé de représentation

La clé doit distinguer les représentations pertinentes. Les dimensions de personnalisation peuvent dépasser l'URL. La clé doit prendre en compte les dimensions réellement variables, telles que tenant ou langue. Une URL identique ne signifie pas que le contenu peut être partagé entre toutes les identités.

> **Erreur fréquente à comprendre** L'URL seule suffit pour toutes les données personnalisées.
>
> Les dimensions de personnalisation peuvent dépasser l'URL.

Deux acteurs

Tester deux utilisateurs peut révéler une fuite. La comparaison d'identités teste une frontière supplémentaire. Demandez la même route avec A puis B dans un laboratoire. Vérifiez contenu, headers et origine de la réponse ; recevoir les données de A pour B révèle une frontière de confidentialité rompue.

> **Erreur fréquente à comprendre** Un test avec un seul compte prouve l'isolation inter-comptes.
>
> La comparaison d'identités teste une frontière supplémentaire.

Origine évitée

Une réponse en cache peut éviter le passage à l'origine. Le cache peut servir sans ce traitement. Une réponse fraîche peut être servie sans exécuter le contrôleur. Ne supposez pas qu'un contrôle ajouté à l'origine protège une copie déjà publiée dans un cache intermédiaire.

> **Erreur fréquente à comprendre** Toute lecture de cache réexécute le contrôleur de droits.
>
> Le cache peut servir sans ce traitement.

Deux décisions

La politique de cache et l'autorisation sont distinctes. Le stockage n'accorde pas les droits. Autoriser une lecture et autoriser le stockage partagé sont des décisions distinctes. Une réponse légitime pour Alice peut devenir une fuite si elle est conservée puis réutilisée pour Bob.

> **Erreur fréquente à comprendre** Mettre en cache donne une permission métier à tous.
>
> Le stockage n'accorde pas les droits.

Erreurs et redirections

Les erreurs et redirections méritent une revue de cache. D'autres réponses participent au comportement des clients. Une redirection ou une erreur peut contenir une destination ou un détail personnalisé. Révisez leur stratégie de cache, pas seulement celle des réponses 200 contenant les données nominales.

> **Erreur fréquente à comprendre** Seuls les documents 200 peuvent affecter les parcours cachés.
>
> D'autres réponses participent au comportement des clients.

Langue

Les variantes linguistiques peuvent exiger une dimension de clé. Le contenu peut dépendre de l'en-tête de langue. Si Accept-Language influence le contenu, les variantes doivent être sélectionnées correctement. Testez deux valeurs au même chemin afin d'éviter une interface ou un document servi dans la mauvaise langue.

> **Erreur fréquente à comprendre** Une langue différente impose toujours la même représentation.
>
> Le contenu peut dépendre de l'en-tête de langue.

Cache applicatif

Un cache applicatif et un cache HTTP ont des règles distinctes. Les mécanismes doivent être conçus séparément. Une clé Redis et un cache HTTP ont des contrats différents. Vérifiez portée et invalidation de chacun ; corriger Cache-Control ne modifie pas automatiquement une entrée métier déjà partagée en mémoire.

> **Erreur fréquente à comprendre** Cache-Control contrôle automatiquement tout cache métier interne.
>
> Les mécanismes doivent être conçus séparément.

Fraîcheur

La durée de fraîcheur n'établit pas la confidentialité. Même une fenêtre courte peut exposer des données. Un contenu de quelques secondes peut être très confidentiel. max-age mesure une durée d'usage sans revalidation selon le contrat, pas un droit universel de lire les données pendant cette durée.

> **Erreur fréquente à comprendre** Un TTL court rend toute fuite acceptable et impossible.
>
> Même une fenêtre courte peut exposer des données.

Retrait

Le retrait d'un objet peut demander une stratégie d'invalidation. La propagation du retrait doit être définie. Supprimer un document à l'origine peut laisser des copies servies jusqu'à expiration ou purge. Définissez délai de retrait et invalidation selon la sensibilité, puis mesurez le résultat à chaque cache concerné.

> **Erreur fréquente à comprendre** Modifier la base invalide automatiquement tout CDN.
>
> La propagation du retrait doit être définie.

Atelier guidé : lire, expliquer et vérifier le code

```
A → GET /lab/account → body.user=A
B → GET /lab/account → body.user=A, Age: 120
Contrat : chaque utilisateur ne doit recevoir que son compte.
```

La demande de B reçoit body.user=A avec un Age positif. Le résultat viole le contrat personnel annoncé et suggère une réutilisation à examiner. Corrigez portée et clé, puis traitez les copies déjà présentes et rejouez le scénario à deux utilisateurs avec et sans cache chaud.

La durée ne supprime pas la fuite. CORS n'accorde pas ces droits métier. La réponse de A est exposée dans le parcours de B. Le résultat reçu ne prouve pas une identité légitime commune.

Travail pratique : Chargez le même chemin avec deux utilisateurs de laboratoire puis avec deux langues. Identifiez tous les caches qui peuvent intervenir.

Critères de correction : Observez l'origine du résultat et comparez les champs privés. Vérifiez que la clé et la politique distinguent les contextes incompatibles, puis testez modification de contenu, déconnexion et erreur de dépendance.

Exemple commenté

```
Cache navigateur ≠ cache partagé ≠ cache métier
Clé métier illustrative : tenant + langue + version du catalogue
Réponse personnelle : politique de partage spécifique
```

Le schéma décrit des dimensions possibles, à choisir selon les résultats réels. Il ne justifie pas de mettre une page personnelle dans un cache public.

Exercice de mise en pratique

Chargez le même chemin avec deux utilisateurs de laboratoire puis avec deux langues. Identifiez tous les caches qui peuvent intervenir.

Voir une proposition de correction

Observez l'origine du résultat et comparez les champs privés. Vérifiez que la clé et la politique distinguent les contextes incompatibles, puis testez modification de contenu, déconnexion et erreur de dépendance.

- Identifier tous les caches.

- Séparer fraîcheur et permission.
- Tester deux identités au minimum.

Contrôle de connaissances

Une réponse fraîche est-elle nécessairement autorisée pour tout utilisateur ?

> Réponse expliquée : Non, fraîcheur et confidentialité sont distinctes — La validité temporelle ne définit pas le périmètre de partage.

Cours associé : 27. Caches : fraîcheur, stockage et confidentialité ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Caching

28. Cache-Control : no-cache, no-store et private

Lire les directives sans leur attribuer un sens intuitif erroné.

Comprendre le mécanisme

no-cache ne signifie pas une interdiction générale de stockage : la réutilisation demande une validation selon les règles applicables. no-store demande de ne pas stocker le message dans le périmètre prévu. private indique qu'une réponse n'est pas destinée à un cache partagé. Ces directives répondent à des questions différentes ; les combiner doit traduire un besoin, pas une liste de mots supposés sûrs.

Raisonner et vérifier

Une directive ne garantit pas l'effacement d'une copie antérieure ni le comportement de tout système externe qui aurait déjà enregistré les données. Réduisez les données sensibles et testez le navigateur ainsi que les intermédiaires sous votre contrôle. Pour une page personnelle, choisissez une politique explicite et vérifiez l'absence de cache applicatif transversal. Documentez les besoins de navigation arrière et de travail hors ligne.

No-cache

no-cache demande une validation avant réutilisation selon le contrat. Stockage et réutilisation sont des décisions distinctes. Le stockage peut exister, mais une validation est requise avant réutilisation selon la sémantique applicable. Ne traduisez pas cette directive par aucune copie conservée : ce serait la propriété d'une autre instruction.

> **Erreur fréquente à comprendre** no-cache interdit tout stockage.
>
> Stockage et réutilisation sont des décisions distinctes.

No-store

no-store demande de ne pas stocker le message concerné. Cette directive ne promet pas un effacement universel rétroactif. Cette directive demande de ne pas stocker le message concerné. Elle ne constitue pas un effacement rétroactif universel de toutes les copies, historiques ou captures déjà créées ailleurs.

> **Erreur fréquente à comprendre** no-store efface toutes les copies antérieures et historiques.
>
> Cette directive ne promet pas un effacement universel rétroactif.

Private

private interdit le stockage par un cache partagé selon sa sémantique. La restriction de stockage n'est pas une preuve d'identité. Un cache partagé ne doit pas stocker selon cette sémantique, tandis qu'un cache privé peut encore être concerné par les autres règles. Vérifiez la différence entre navigateur et CDN.

> **Erreur fréquente à comprendre** private authentifie automatiquement le lecteur.
>
> La restriction de stockage n'est pas une preuve d'identité.

Public

public peut permettre un cache partagé selon les autres règles. L'auteur doit choisir une politique adaptée aux données. Une réponse marquée public peut être admissible à un cache partagé selon les autres conditions. Révisez immédiatement ce choix s'il porte des données personnelles et recherchez les copies déjà exposées.

> **Erreur fréquente à comprendre** public signifie que le contenu ne contient nécessairement aucun secret.
>
> L'auteur doit choisir une politique adaptée aux données.

Max-age

max-age définit une durée de fraîcheur. Le cache et l'identité ont des cycles distincts. Cette durée règle la fraîcheur, pas la sensibilité de la donnée. Une réponse personnelle ne devient pas partageable simplement parce que son max-age est court et que son contenu paraît récent.

> **Erreur fréquente à comprendre** max-age définit la durée de validité du compte utilisateur.
>
> Le cache et l'identité ont des cycles distincts.

S-maxage

s-maxage vise les caches partagés selon son contrat. La directive n'a pas ce rôle d'authentification. Cette directive cible les caches partagés dans son contrat. Comparez son effet avec max-age et les autres directives avant de conclure à une durée identique côté navigateur et côté intermédiaire.

> **Erreur fréquente à comprendre** s-maxage définit seulement le mot de passe du navigateur.
>
> La directive n'a pas ce rôle d'authentification.

Personnalisation

Une politique doit être cohérente avec les données personnalisées. La durée ne supprime pas la frontière d'identité. Identifiez les données liées à une identité avant de sélectionner la politique. Ne laissez pas un middleware global ajouter public sur un compte utilisateur simplement parce que le reste du site est public.

> **Erreur fréquente à comprendre** Une réponse privée devient partageable sans risque avec un TTL court.
>
> La durée ne supprime pas la frontière d'identité.

Combinaison

Des directives doivent être interprétées ensemble. Les autres paramètres peuvent modifier les conditions. Interprétez les directives ensemble et selon le type de cache. Une vérification qui cherche seulement un mot dans le header peut manquer une combinaison contradictoire ou une directive plus spécifique.

> **Erreur fréquente à comprendre** Lire un seul mot suffit à connaître toute politique de cache.
>
> Les autres paramètres peuvent modifier les conditions.

Valeur déployée

Les champs réellement servis doivent être vérifiés. Le cache utilise le message reçu. Inspectez la réponse reçue après proxy et CDN. Une route peut être transformée par une règle d'hébergement ; le fichier de configuration de l'application ne suffit pas à prouver la politique servie.

> **Erreur fréquente à comprendre** Une intention dans un commentaire impose la politique HTTP.
>
> Le cache utilise le message reçu.

Autorisation

L'autorisation doit rester correcte indépendamment du cache. Le protocole de cache ne décide pas de la propriété métier. Même sans cache partagé, le serveur doit refuser un utilisateur tiers. La politique de stockage ne remplace pas la vérification du droit sur chaque ressource au moment de la demande.

> **Erreur fréquente à comprendre** Cache-Control remplace les permissions sur l'objet.
>
> Le protocole de cache ne décide pas de la propriété métier.

Atelier guidé : lire, expliquer et vérifier le code

```
Cache-Control: no-cache
ETag: "lab-v1"
```

La réponse associe no-cache et un ETag. Elle peut donc permettre un stockage suivi d'une validation avant réutilisation selon le contrat. Expliquez la différence avec no-store et private, puis observez un parcours de revalidation réel au lieu d'interpréter le nom de directive littéralement.

Un validateur de contenu n'est pas une identité. no-cache ne signifie pas no-store. Cette interdiction correspond à une autre directive. La politique ne promet pas une purge universelle rétroactive.

Travail pratique : Comparez un document de laboratoire servi avec no-cache puis no-store. Examinez validation réseau et présence éventuelle de stockage.

Critères de correction : Utilisez des données fictives et un cache contrôlé. Testez rechargement, retour de navigation et changement d'identité. Décrivez les observations sans promettre une suppression universelle de toutes les copies.

Exemple commenté

```
Cache-Control: private, no-store
```

Cette politique convient à certains contenus personnels, mais son adéquation dépend du contrat de la page et des mécanismes qui la servent.

Exercice de mise en pratique

Comparez un document de laboratoire servi avec no-cache puis no-store. Examinez validation réseau et présence éventuelle de stockage.

Voir une proposition de correction

Utilisez des données fictives et un cache contrôlé. Testez rechargement, retour de navigation et changement d'identité. Décrivez les observations sans promettre une suppression universelle de toutes les copies.

- no-cache n'est pas no-store.
- private n'est pas une authentification.
- Pas de promesse d'effacement rétroactif.

Contrôle de connaissances

Quelle lecture de no-cache est correcte ?

> Réponse expliquée : La réutilisation exige une validation selon le contrat — La directive ne se résume pas à interdire le stockage.

Cours associé : 28. Cache-Control : no-cache, no-store et private ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cache-Control

29. Vary et variantes de représentation

Faire correspondre clé HTTP et dimensions réellement utilisées.

Comprendre le mécanisme

Vary indique des champs de requête ayant contribué au choix de la représentation. Un cache qui réutilise une réponse doit tenir compte des variantes annoncées dans le cadre de son contrat. La langue, certains encodages ou une politique CORS peuvent modifier une réponse. Un champ absent de Vary alors qu'il change le résultat peut conduire à une variante incorrecte. Une liste trop large peut au contraire réduire fortement l'efficacité du cache.

Raisonner et vérifier

N'ajoutez pas mécaniquement tous les champs. Partez du code qui choisit le résultat et des règles du cache intermédiaire. Vary ne remplace pas une politique de confidentialité : annoncer une variation sur Cookie ne suffit pas, à lui seul, à rendre sûre une réponse personnelle publique. Testez variantes, valeur absente, ordre des demandes et invalidation après changement de politique.

Sélection

Vary indique des champs de requête influençant la sélection. La directive participe à la sélection des variantes. Vary nomme des champs de requête qui influencent le choix de représentation. Ce n'est pas une liste de droits ; il aide le cache à distinguer les demandes que le serveur traite différemment.

> **Erreur fréquente à comprendre** Vary décrit uniquement la taille du corps de réponse.
>
> La directive participe à la sélection des variantes.

Langue

Une réponse variant par langue peut annoncer cette dimension. La séparation de variantes n'est pas un chiffrement. Une réponse calculée selon Accept-Language peut annoncer cette dimension. Vérifiez deux langues sur la même URL avec cache chaud pour démontrer que le contenu correspondant est choisi.

> **Erreur fréquente à comprendre** Vary Accept-Language chiffre la langue de l'utilisateur.
>
> La séparation de variantes n'est pas un chiffrement.

Origin

Une politique différente selon Origin peut demander Vary Origin. Le cache doit tenir compte de la variation réelle. Si Access-Control-Allow-Origin varie selon l'origine reçue, le cache doit respecter cette variation pertinente. Sans séparation appropriée, une réponse préparée pour un portail peut être réutilisée dans un autre contexte.

> **Erreur fréquente à comprendre** Réfléchir Origin ne nécessite jamais une revue de cache.
>
> Le cache doit tenir compte de la variation réelle.

Pas une permission

Vary ne constitue pas seul une autorisation. La sélection de représentation ne vérifie pas la permission. Vary: Cookie ne dit pas qu'un contenu personnel est admissible dans tout cache partagé. Il faut une politique de confidentialité et de stockage complète, avec dimensions et accès effectivement maîtrisés.

> **Erreur fréquente à comprendre** Vary Cookie valide les droits métier de chaque lecteur.
>
> La sélection de représentation ne vérifie pas la permission.

Dimension oubliée

Une dimension oubliée peut mélanger des réponses. Les dimensions pertinentes doivent être exactes. Un tenant absent de la clé peut recevoir le contenu d'un autre. Testez précisément une seule différence entre les demandes afin d'identifier la dimension que la sélection doit préserver.

> **Erreur fréquente à comprendre** La présence d'un Vary quelconque garantit toutes les variantes.
>
> Les dimensions pertinentes doivent être exactes.

Cardinalité

Des dimensions inutiles peuvent réduire l'efficacité du cache. Une clé trop fragmentée réduit la réutilisation. Ajouter inutilement des champs très variables peut multiplier les entrées et réduire les hits. Choisissez les dimensions requises par la représentation sans supprimer celles qui assurent l'isolation nécessaire.

> **Erreur fréquente à comprendre** Ajouter tous les champs améliore toujours le taux de hit.
>
> Une clé trop fragmentée réduit la réutilisation.

Comparaison contrôlée

Les tests doivent comparer des requêtes différant sur une dimension. Il faut provoquer la variation à vérifier. Gardez méthode et chemin identiques puis changez un seul header pertinent. Vérifiez contenu et métadonnées afin de distinguer une mauvaise sélection de variante d'un autre changement de comportement.

> **Erreur fréquente à comprendre** Deux requêtes identiques prouvent la séparation des langues.
>
> Il faut provoquer la variation à vérifier.

Intermédiaires

Les intermédiaires doivent respecter le contrat déployé. Les caches métier peuvent utiliser d'autres mécanismes. Les caches réellement déployés doivent comprendre et respecter la stratégie retenue. Une configuration d'application correcte ne suffit pas si le CDN utilise une clé personnalisée incompatible.

> **Erreur fréquente à comprendre** Tous les caches internes suivent automatiquement Vary.
>
> Les caches métier peuvent utiliser d'autres mécanismes.

Confidentialité complète

La confidentialité demande une politique complète. Le cache privé ou non stockable peut rester nécessaire. Définissez d'abord si la réponse peut être stockée et partagée, puis comment ses variantes sont distinguées. Une clé très détaillée ne justifie pas la conservation de données qui doivent rester non stockées.

> **Erreur fréquente à comprendre** Vary seul rend toute donnée personnelle partageable.
>
> Le cache privé ou non stockable peut rester nécessaire.

Normalisation

Une normalisation doit préserver la signification de la dimension. Une normalisation excessive peut fusionner des représentations distinctes. Normaliser une langue ou une origine peut améliorer la sélection seulement si la signification reste préservée. Une simplification qui fusionne des contextes différents peut recréer une fuite ou servir une mauvaise politique.

> **Erreur fréquente à comprendre** On peut supprimer toute valeur de clé sans effet.
>
> Une normalisation excessive peut fusionner des représentations distinctes.

Atelier guidé : lire, expliquer et vérifier le code

```
Access-Control-Allow-Origin: https://portal-a.example.test
Vary: Accept-Language

Cache partagé ; même URL utilisée par plusieurs origines admises.
```

La réponse varie par origine mais Vary ne montre que la langue dans ce cache partagé. Examinez la stratégie complète et ajoutez la dimension manquante lorsque ce contrat l'exige. Rejouez deux origines admises sur la même URL et vérifiez leur header de partage respectif.

Le cache doit distinguer la variation effectivement annoncée par origine. Un secret ne doit pas être exposé pour construire la politique. Le statut seul ne sépare pas les variantes. Le visuel ne définit pas la clé HTTP.

Travail pratique : Servez deux variantes de langue sur un endpoint de laboratoire cachable puis inversez l'ordre des demandes.

Critères de correction : Vérifiez que chaque appelant reçoit la représentation attendue. Comparez clé effective et champs annoncés, puis testez une origine absente ou refusée. Retirez les variations sans effet plutôt que masquer une conception de cache incohérente.

Exemple commenté

```
Vary: Accept-Language, Origin
```

Ce fragment suppose que ces dimensions influencent effectivement la réponse servie et qu'une politique de cache adaptée existe déjà.

Exercice de mise en pratique

Servez deux variantes de langue sur un endpoint de laboratoire cachable puis inversez l'ordre des demandes.

Voir une proposition de correction

Vérifiez que chaque appelant reçoit la représentation attendue. Comparez clé effective et champs annoncés, puis testez une origine absente ou refusée. Retirez les variations sans effet plutôt que masquer une conception de cache incohérente.

- Lister les dimensions réelles.
- Tester l’ordre des demandes.
- Ne pas confondre Vary et contrôle d’accès.

Contrôle de connaissances

Que décrit Vary ?

> Réponse expliquée : Des champs qui influencent le choix de représentation — Le mécanisme concerne les variantes HTTP.

Cours associé : 29. Vary et variantes de représentation ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Vary

30. Revalidation et mises à jour conditionnelles

Distinguer économie de lecture et prévention des mises à jour perdues.

Comprendre le mécanisme

Une requête conditionnelle utilise un validateur comme un ETag ou une date selon le mécanisme prévu. If-None-Match intervient notamment dans la revalidation d'une représentation ; une réponse 304 permet alors d'éviter de renvoyer un corps complet dans le cas applicable. If-Match exprime un prérequis utile pour certaines mutations, afin de ne pas modifier une version différente de celle que le client croit traiter.

Raisonner et vérifier

Un ETag n'est pas une preuve d'autorisation. Pour prévenir une mise à jour perdue, le serveur doit relier le prérequis à l'écriture de façon cohérente : une comparaison suivie d'un UPDATE non protégé peut encore courir avec un autre traitement. Définissez aussi l'usage de validateurs forts ou faibles selon le contrat de l'opération. Testez le cas où deux clients partent de la même version.

Validateur

Un ETag peut servir de validateur de représentation. La version de contenu n'est pas une identité. Un ETag identifie une version de représentation selon sa construction. Il ne donne pas de droit d'accès ; le serveur doit le relier à la bonne ressource et au bon contexte utilisateur.

> **Erreur fréquente à comprendre** Un ETag authentifie automatiquement l'utilisateur.
>
> La version de contenu n'est pas une identité.

Réutilisation 304

Une réponse 304 réutilise une représentation selon le contexte. La réponse de non-modification n'envoie pas ce contenu complet. Une réponse 304 pertinente permet au client de réutiliser une représentation qu'il possède selon le contrat de revalidation. Elle ne transporte pas normalement un nouveau corps complet de cette représentation.

> **Erreur fréquente à comprendre** 304 contient toujours un nouveau document complet.
>
> La réponse de non-modification n'envoie pas ce contenu complet.

Lecture conditionnelle

If-None-Match peut conditionner une lecture. Le champ exprime une condition sur un validateur. If-None-Match permet de demander si une version possédée reste actuelle. Vérifiez le cas correspondant et le cas modifié, avec les headers nécessaires à la mise à jour du cache.

> **Erreur fréquente à comprendre** If-None-Match est un mot de passe du serveur.
>
> Le champ exprime une condition sur un validateur.

Écriture conditionnelle

If-Match peut protéger une écriture concurrente. Une modification peut survenir entre les deux étapes. If-Match peut exprimer que la mutation n'est admise que sur la version connue. Cela aide à éviter qu'un client écrase une modification concurrente qu'il n'a pas encore observée.

> **Erreur fréquente à comprendre** Lire puis écrire sans condition est toujours équivalent.
>
> Une modification peut survenir entre les deux étapes.

Lien atomique

La comparaison et l'écriture doivent être cohérentes. Le contrôle doit porter sur l'état au moment de la transition. Comparer la version puis écrire plus tard sans protection laisse une course. Utilisez une mise à jour conditionnelle ou une transaction adaptée et contrôlez le nombre de lignes effectivement modifiées.

> **Erreur fréquente à comprendre** Une comparaison ancienne suffit même après un changement concurrent.
>
> Le contrôle doit porter sur l'état au moment de la transition.

Précondition échouée

Un échec de précondition peut empêcher une mise à jour perdue. Le refus doit précéder l'effet non admis. Si la version a changé, refusez selon le contrat plutôt qu'appliquer silencieusement l'écriture. Le client peut alors relire, fusionner ou demander confirmation avec un état à jour.

> **Erreur fréquente à comprendre** Un échec autorise l'écriture avant la réponse de refus.
>
> Le refus doit précéder l'effet non admis.

Droit de lecture

Le droit de lecture doit rester vérifié. Le validateur ne donne pas la permission. Un validateur connu ne suffit pas à obtenir une donnée. Vérifiez l'autorisation avant de répondre, y compris lorsqu'un 304 pourrait révéler l'existence ou l'état d'une ressource confidentielle.

> **Erreur fréquente à comprendre** Connaître un ETag donne accès à toute ressource.
>
> Le validateur ne donne pas la permission.

Bonne représentation

Un validateur doit correspondre à la représentation considérée. Il doit refléter le contrat de comparaison choisi. Une version calculée pour une vue publique n'est pas automatiquement celle d'une vue personnalisée. Définissez quelle représentation l'ETag désigne et gardez cette portée dans la sélection et les tests.

> **Erreur fréquente à comprendre** Le même ETag fixe convient à tout contenu évolutif.
>
> Il doit refléter le contrat de comparaison choisi.

Faible ou fort

Les validateurs faibles et forts ne sont pas interchangeables partout. Les règles de comparaison dépendent de l'usage. Les validateurs faibles et forts ne sont pas acceptés de façon interchangeable dans toutes les préconditions. Appliquez les règles du protocole pour l'opération choisie au lieu de retirer simplement le préfixe faible.

> **Erreur fréquente à comprendre** Le préfixe faible n'a aucune conséquence sémantique.
>
> Les règles de comparaison dépendent de l'usage.

Deux écrivains

Le test de concurrence doit inclure deux écrivains. La course doit être observée ou simulée. Faites lire la version sept à deux clients puis lancez leurs écritures. Une seule doit gagner si la politique exige une version exacte ; vérifiez le contenu final et le refus de l'autre.

> **Erreur fréquente à comprendre** Un test séquentiel unique prouve l'absence de mise à jour perdue.
>
> La course doit être observée ou simulée.

Atelier guidé : lire, expliquer et vérifier le code

```
UPDATE lab_documents
SET content = 'new', version = version + 1
WHERE id = 42 AND version = 7;
```

Le prédicat id=42 AND version=7 lie la précondition à la modification SQL. Si une première écriture a déjà produit version huit, cette requête ne modifie aucune ligne. L'application doit vérifier ce résultat et le traduire en échec de précondition, sans annoncer un succès fictif.

Le SET ne s'applique pas à une ligne hors du prédicat. Le prédicat conditionne l'écriture. La permission est un contrôle distinct de cette concurrence. Le prédicat refuse la ligne dont la version vaut déjà 8.

Travail pratique : Avec deux clients de laboratoire, modifiez une même ressource depuis la version initiale puis tentez d'appliquer la seconde modification.

Critères de correction : Utilisez une version persistante et une écriture conditionnelle. Vérifiez que le second client reçoit le refus prévu et recharge l'état au lieu d'écraser silencieusement la première modification. Testez aussi le contrôle d'accès.

Exemple commenté

```
PATCH /lab/reservations/42
If-Match: "version-7"
Content-Type: application/json

{"note":"mise à jour de laboratoire"}
```

Le serveur doit vérifier le prérequis dans la même stratégie de cohérence que la mutation. Cet en-tête ne crée pas, seul, un verrou SQL.

Exercice de mise en pratique

Avec deux clients de laboratoire, modifiez une même ressource depuis la version initiale puis tentez d'appliquer la seconde modification.

Voir une proposition de correction

Utilisez une version persistante et une écriture conditionnelle. Vérifiez que le second client reçoit le refus prévu et recharge l'état au lieu d'écraser silencieusement la première modification. Testez aussi le contrôle d'accès.

- Choisir le validateur adapté.
- Lier prérequis et écriture.
- Autorisation indépendante de l'ETag.

Contrôle de connaissances

Une réponse 304 pertinente remplace-t-elle normalement un corps complet lors d'une revalidation ?

> Réponse expliquée : Oui, dans le contrat de revalidation applicable — Le client peut réutiliser sa représentation selon les règles.

Cours associé : 30. Revalidation et mises à jour conditionnelles ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Conditional_requests

31. Redirections, méthodes et destinations

Préserver le contrat de la demande sans transmettre le contexte à un tiers inattendu.

Comprendre le mécanisme

Une redirection indique une autre destination au client. Selon le statut utilisé, le traitement de la méthode et du corps peut différer. Un 307 conserve la méthode et le corps dans le contrat prévu ; il ne doit donc pas être choisi sans réfléchir à la destination d'un message sensible. Une redirection n'accorde pas non plus une permission sur la ressource finale. Les bibliothèques peuvent suivre automatiquement plusieurs sauts.

Raisonner et vérifier

Définissez les destinations permises à partir d'identifiants internes ou d'URL parsées et validées. Évitez les retours libres fournis par le client qui produiraient une redirection ouverte. Pour les clients serveur, vérifiez la politique sur les changements d'hôte, de schéma et les champs sensibles ; ne supposez pas une protection identique de toutes les bibliothèques. Testez chaque saut, pas seulement le premier statut.

Méthode conservée

307 conserve la méthode et le corps selon sa sémantique. Ce statut préserve le contrat de renvoi de la requête. Un suivi de 307 conserve méthode et corps selon sa sémantique. Une redirection de POST sensible peut donc transmettre à nouveau son contenu ; examinez la destination avant de suivre automatiquement.

> **Erreur fréquente à comprendre** 307 transforme toujours POST en GET.
>
> Ce statut préserve le contrat de renvoi de la requête.

Destination admise

Les destinations doivent suivre une politique contrôlée. Une redirection libre peut détourner le parcours. Une URL de retour reçue du client doit être limitée au contrat prévu. Préférez une référence interne ou un chemin relatif maîtrisé lorsque le produit n'a pas besoin de destinations externes.

> **Erreur fréquente à comprendre** Toute URL fournie par le client est un retour sûr.
>
> Une redirection libre peut détourner le parcours.

Credentials

Un client doit revoir les credentials lors d'un changement de destination. Les credentials ne doivent pas être diffusés arbitrairement. Lors d'un changement d'hôte ou de contexte, réévaluez transmission d'Authorization, cookies et corps. Une bibliothèque peut avoir ses protections propres ; vérifiez-les sans supposer que toute redirection conserve ou retire tous les secrets.

> **Erreur fréquente à comprendre** Tout token doit suivre chaque redirection vers tout hôte.
>
> Les credentials ne doivent pas être diffusés arbitrairement.

Chaîne bornée

Une chaîne de redirections demande des bornes. Boucles et chaînes consomment temps et ressources. Limitez nombre de sauts et durée totale pour éviter boucles et consommation excessive. Conservez la trace des destinations factices dans les tests afin de comprendre pourquoi une reprise échoue.

> **Erreur fréquente à comprendre** Les redirections peuvent être suivies sans limite de nombre.
>
> Boucles et chaînes consomment temps et ressources.

SSRF à chaque saut

Le contrôle SSRF doit couvrir chaque saut admis. Une redirection peut changer la destination effective. Une URL initiale admise peut répondre vers une IP interne. Appliquez la politique de sortie à chaque destination et à sa résolution, pas seulement à la première chaîne reçue.

> **Erreur fréquente à comprendre** Valider seulement la première URL couvre toute destination finale.
>
> Une redirection peut changer la destination effective.

URL structurée

Une URL doit être comparée après parsing approprié. Le texte trompeur peut apparaître dans d'autres composants. Comparez schéma, hôte et port après parsing approprié. Une recherche textuelle peut admettre un domaine hostile contenant le nom attendu dans son userinfo, son chemin ou une partie trompeuse.

> **Erreur fréquente à comprendre** Une sous-chaîne visuelle suffit à valider tout retour.
>
> Le texte trompeur peut apparaître dans d'autres composants.

Droit indépendant

Un statut de redirection ne prouve pas l'autorisation métier. La navigation ne crée pas de permission. Un statut 302 ou 307 ne prouve pas que l'utilisateur pouvait accéder à la ressource initiale ou finale. Vérifiez les permissions là où l'effet ou la lecture sensible a lieu.

> **Erreur fréquente à comprendre** Une réponse 302 donne des droits administrateur.
>
> La navigation ne crée pas de permission.

Retour relatif

Un retour relatif maîtrisé peut limiter les destinations. Des syntaxes particulières peuvent avoir un autre sens. Une route interne connue ou un chemin relatif strictement validé peut réduire la surface d'open redirect. Vérifiez aussi les formes qui changent d'autorité selon le parseur, au lieu d'accepter toute chaîne commençant par slash.

> **Erreur fréquente à comprendre** Toute chaîne commençant par une barre est nécessairement sûre sans parsing.
>
> Des syntaxes particulières peuvent avoir un autre sens.

Caches

Les redirections peuvent interagir avec les caches. Le comportement dépend du statut et de la politique. Une redirection peut être mémorisée selon statut et directives. Révisez les destinations personnalisées et les changements permanents afin qu'un cache ne diffuse pas un retour sensible à un autre utilisateur.

> **Erreur fréquente à comprendre** Une redirection ne peut jamais être mémorisée.
>
> Le comportement dépend du statut et de la politique.

Destination finale

Les tests doivent observer la destination finale. Des sauts ultérieurs peuvent modifier le résultat. En laboratoire, observez chaque saut et la destination réellement appelée. Une réponse initiale conforme ne démontre pas que les étapes suivantes respectent la politique de sortie ou de confidentialité.

> **Erreur fréquente à comprendre** Le premier statut suffit à prouver le parcours complet.
>
> Des sauts ultérieurs peuvent modifier le résultat.

Atelier guidé : lire, expliquer et vérifier le code

```
HTTP/1.1 307 Temporary Redirect
Location: https://other.example.test/lab/receive
```

Le 307 pointe vers un autre hôte et peut faire suivre un corps sensible. Avant le suivi, examinez destination, credentials et sémantique de la reprise. Ajoutez un serveur factice récepteur dans le laboratoire pour vérifier ce que le client transmet effectivement à chaque saut.

La portée des secrets doit rester contrôlée. Une destination reçue ne prouve pas sa confiance. 307 préserve la méthode et peut préserver un corps sensible. Ce n'est pas la sémantique de 307.

Travail pratique : Comparez une redirection d'après-formulaire et un déplacement temporaire d'endpoint dans un laboratoire local. Relevez méthodes et corps reçus à destination.

Critères de correction : Désactivez d'abord le suivi automatique pour inspecter la réponse, puis activez-le avec une politique bornée. Testez destination non admise et changement de schéma. Vérifiez qu'aucun secret n'atteint un récepteur non prévu.

Exemple commenté

```
HTTP/1.1 307 Temporary Redirect
Location: https://api.example.test/lab/new-endpoint
```

Le choix conserve le contrat de méthode lors du suivi. Il convient seulement si cette destination et le transfert du message sont intentionnels.

Exercice de mise en pratique

Comparez une redirection d'après-formulaire et un déplacement temporaire d'endpoint dans un laboratoire local. Relevez méthodes et corps reçus à destination.

Voir une proposition de correction

Désactivez d'abord le suivi automatique pour inspecter la réponse, puis activez-le avec une politique bornée. Testez destination non admise et changement de schéma. Vérifiez qu'aucun secret n'atteint un récepteur non prévu.

- Choisir le statut consciemment.
- Valider les destinations.

- Borner le suivi automatique.

Contrôle de connaissances

Quel effet caractérise 307 dans son contrat de redirection ?

> Réponse expliquée : La conservation de la méthode et du corps lors du suivi prévu — Le client doit tenir compte de cette conservation.

Cours associé : 31. Redirections, méthodes et destinations ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/307

32. Host, Forwarded et confiance des mandataires

Définir qui est autorisé à décrire la requête d'origine.

Comprendre le mécanisme

Un reverse proxy peut transmettre l'hôte, le schéma ou une adresse d'origine au serveur applicatif. Des champs comme Forwarded ou des variantes historiques X-Forwarded-* ne doivent pas être crus simplement parce qu'ils sont présents. Si le client externe peut fournir la même information sans être neutralisé, l'application risque de générer de mauvais liens, de journaliser une fausse origine ou d'appliquer une décision fondée sur une entrée hostile.

Raisonner et vérifier

Configurez explicitement les mandataires de confiance et les champs qu'ils reconstruisent. Distinguez connexion observée par PHP et connexion externe terminée en TLS au frontal. Utilisez une origine publique de confiance pour les liens sensibles lorsque le besoin s'y prête. Testez les accès directs à l'application et le passage normal par le proxy ; une configuration correcte suppose aussi que les chemins non prévus sont bornés.

Provenance

Forwarded est une donnée à interpréter selon sa provenance. Un client peut envoyer un champ de ce nom. Un client direct peut envoyer Forwarded avec proto=https ou une fausse IP. La valeur n'est fiable que dans une chaîne où son producteur et le nettoyage des entrées externes sont maîtrisés.

> **Erreur fréquente à comprendre** Le nom Forwarded rend toute valeur fiable.
>
> Un client peut envoyer un champ de ce nom.

Proxies admis

Les adresses de proxies admis doivent être maîtrisées. L'affirmation reçue ne prouve pas l'émetteur. Configurez des adresses ou réseaux de confiance correspondant à l'infrastructure réelle. Une confiance globale peut permettre à tout appelant d'influencer des décisions fondées sur des métadonnées supposées internes.

> **Erreur fréquente à comprendre** Tout émetteur d'un X-Forwarded-For est un proxy de confiance.
>
> L'affirmation reçue ne prouve pas l'émetteur.

Nettoyage

Le proxy peut supprimer et reconstruire les champs attendus. Une chaîne fiable demande une politique de frontière. Le proxy d'entrée peut supprimer les headers clients concernés puis reconstruire les valeurs fiables. Vérifiez cette règle et les accès directs éventuels au backend, qui pourraient contourner cette frontière.

> **Erreur fréquente à comprendre** Conserver tous les champs entrants garantit leur authenticité.
>
> Une chaîne fiable demande une politique de frontière.

Schéma perçu

Le schéma perçu influence les URL générées. Les composants peuvent utiliser cette information pour générer des URL. Le schéma reconstruit influence cookies Secure, redirections et URL absolues. Derrière une terminaison TLS, une mauvaise configuration peut produire des liens HTTP ou des boucles sans que le navigateur utilise ce schéma initialement.

> **Erreur fréquente à comprendre** Le schéma annoncé ne peut affecter aucun lien absolu.
>
> Les composants peuvent utiliser cette information pour générer des URL.

Host

Le Host doit suivre les contraintes du service. Une entrée hostile peut influencer des URL et décisions. Un lien de réinitialisation construit depuis un Host libre peut viser un domaine contrôlé par l'appelant. Limitez les hôtes du service et utilisez un contexte maîtrisé pour générer les liens sensibles.

> **Erreur fréquente à comprendre** Tout Host reçu est une destination publique légitime.
>
> Une entrée hostile peut influencer des URL et décisions.

Adresse et quotas

La confiance trop large peut affecter la limitation par IP. Un émetteur peut alors influencer l'identité réseau perçue. Un attaquant qui choisit une fausse IP annoncée peut contourner un quota naïf. Définissez quelle adresse représente le client selon la chaîne réelle et testez accès direct et accès via proxy.

> **Erreur fréquente à comprendre** Faire confiance à toutes les adresses améliore toujours les quotas.
>
> Un émetteur peut alors influencer l'identité réseau perçue.

Topologie testée

Le trajet réel doit être reproduit dans les tests. Le chemin peut transformer les métadonnées. Reproduisez les sauts, terminaisons et headers dans l'environnement de vérification. Un test direct sur localhost ne démontre pas que la reconstruction d'identité réseau fonctionne derrière le proxy hébergé.

> **Erreur fréquente à comprendre** Un appel direct local prouve la chaîne des proxies publics.
>
> Le chemin peut transformer les métadonnées.

Réseaux évolutifs

Les réseaux de confiance peuvent évoluer. La configuration doit suivre l'infrastructure réelle. Des adresses d'intermédiaires peuvent changer lors d'une migration. Prévoyez la maintenance et la validation régulière de la liste admise ; une ancienne plage trop large peut conserver une confiance inutile.

> **Erreur fréquente à comprendre** Une liste non relue demeure correcte pour toute topologie future.
>
> La configuration doit suivre l'infrastructure réelle.

Trajet interne

Le transport interne demande sa propre vérification. La terminaison TLS marque une frontière de protection. TLS au frontal ne démontre pas le chiffrement entre proxy et application. Vérifiez réseau, authentification et transport de ce segment selon les données et les capacités qu'il transporte.

> **Erreur fréquente à comprendre** HTTPS public chiffre automatiquement le saut interne.
>
> La terminaison TLS marque une frontière de protection.

Corrélation

La corrélation ne doit pas divulguer les secrets. Des métadonnées minimisées peuvent suffire au diagnostic. Transmettez des identifiants de trace non sensibles et validés. Ne faites pas voyager cookies ou bearer tokens dans un identifiant de diagnostic pour faciliter le rapprochement des journaux.

> **Erreur fréquente à comprendre** Déboguer les proxies exige les cookies complets des clients.
>
> Des métadonnées minimisées peuvent suffire au diagnostic.

Atelier guidé : lire, expliquer et vérifier le code

```
Forwarded: for=192.0.2.42;proto=https;host=portal.example.test
```

La ligne Forwarded contient une adresse et un schéma, mais aucune preuve de son producteur. Décrivez les conditions nécessaires pour les accepter : proxy admis, nettoyage des valeurs externes et impossibilité de contourner ce trajet. Vérifiez ensuite les liens générés et la limitation par IP.

Le nom ne protège pas sa provenance. Le serveur doit lier ces métadonnées à un producteur admis. La déclaration client n'observe pas tout le trajet. Le champ ne vérifie pas une identité.

Travail pratique : Simulez un champ d'origine falsifié depuis le client, puis vérifiez les liens générés et les attributs de cookies derrière le proxy de laboratoire.

Critères de correction : Comparez requête directe et requête relayée. Vérifiez les champs effectivement reconstruits, les proxys déclarés et la génération d'URL. Confirmez qu'un appelant non autorisé ne peut pas choisir l'origine d'un lien sensible.

Exemple commenté

```
Client externe → proxy de confiance 192.0.2.10 → application
Le proxy neutralise les assertions entrantes et produit le contexte autorisé.
```

L'adresse est réservée aux exemples de documentation. Elle doit être remplacée dans une vraie architecture par la liste précise des équipements maîtrisés.

Exercice de mise en pratique

Simulez un champ d'origine falsifié depuis le client, puis vérifiez les liens générés et les attributs de cookies derrière le proxy de laboratoire.

Voir une proposition de correction

Comparez requête directe et requête relayée. Vérifiez les champs effectivement reconstruits, les proxys déclarés et la génération d'URL. Confirmez qu'un appelant non autorisé ne peut pas choisir l'origine d'un lien sensible.

- Mandataires explicites.
- Neutralisation des champs client concurrents.

- Origine publique maîtrisée.

Contrôle de connaissances

La présence de Forwarded prouve-t-elle que son contenu vient d’un proxy de confiance ?

> Réponse expliquée : Non — La provenance doit être vérifiée par l’architecture.

Cours associé : 32. Host, Forwarded et confiance des mandataires ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Forwarded

33. SSRF et sorties HTTP du serveur

Réduire les destinations et la puissance d'un client HTTP piloté par des données.

Comprendre le mécanisme

Une SSRF utilise la capacité du serveur à effectuer des requêtes vers des destinations choisies ou influencées par un appelant. Le serveur peut disposer d'accès réseau que le navigateur n'a pas. Une simple vérification de préfixe de chaîne ne couvre pas les formes d'URL, résolutions DNS, adresses ou redirections possibles. La politique doit décrire les protocoles, destinations, ports et chemins réellement nécessaires.

Raisonner et vérifier

Préférez des identifiants de fournisseurs connus à une URL libre. Contrôlez chaque saut de redirection et les adresses effectivement utilisées avec les protections réseau appropriées. Bornez temps, taille de réponse et nombre de sauts. Les restrictions de sortie doivent rester effectives même si la validation applicative comporte un défaut. Les tests utilisent des récepteurs de laboratoire, sans explorer des réseaux dont vous n'êtes pas responsable.

Politique de sortie

Une URL serveur doit suivre une politique de destination. TLS vers une destination hostile ne la rend pas admise. Définissez schémas, hôtes, ports et opérations distantes admis. Une URL syntaxiquement valide peut viser un service interne ou une capacité privilégiée ; sa validité ne suffit pas à l'autoriser.

> **Erreur fréquente à comprendre** Toute URL HTTPS client est une sortie fiable.
>
> TLS vers une destination hostile ne la rend pas admise.

Identifiant fournisseur

Un identifiant de fournisseur peut être résolu côté serveur. Une résolution maîtrisée réduit les destinations disponibles. Le client peut choisir partner-42 tandis que le serveur résout une configuration contrôlée. Ce modèle réduit la liberté de destination, mais doit encore vérifier que l'acteur a droit à ce fournisseur et cette opération.

> **Erreur fréquente à comprendre** L'appelant doit pouvoir choisir tout hôte interne.
>
> Une résolution maîtrisée réduit les destinations disponibles.

Redirections

Les redirections demandent une validation à chaque étape. Une redirection peut viser un autre réseau. Un partenaire admis peut renvoyer vers localhost ou un réseau privé. Validez chaque saut ou refusez le suivi selon le contrat ; la validation initiale ne couvre pas la chaîne complète.

> **Erreur fréquente à comprendre** La première URL admise valide tous les sauts suivants.
>
> Une redirection peut viser un autre réseau.

DNS

La résolution DNS doit être considérée dans le modèle. La destination réseau effective peut différer des attentes. La résolution peut changer entre validation et connexion. Utilisez un client et une architecture qui maintiennent la politique sur la destination réellement jointe, avec contrôle réseau complémentaire et tests de résolution maîtrisés.

> **Erreur fréquente à comprendre** Comparer seulement le nom visuel suffit dans tous les cas.
>
> La destination réseau effective peut différer des attentes.

Réseaux sensibles

Les réseaux privés et métadonnées sensibles doivent être considérés. Les sorties peuvent exposer des capacités internes. Incluez loopback, réseaux internes et services de métadonnées dans le modèle. Ne réduisez pas l'analyse à une seule chaîne 127.0.0.1 : adresses, familles et représentations doivent être traitées par des composants adaptés.

> **Erreur fréquente à comprendre** Le serveur peut contacter librement toute adresse depuis un formulaire.
>
> Les sorties peuvent exposer des capacités internes.

Budgets

Des limites de taille et de durée restent nécessaires. Un service autorisé peut être lent ou volumineux. Même une destination autorisée peut répondre lentement ou sans limite. Bornez connexion, durée globale et volume lu, y compris après décompression si le client l'effectue automatiquement.

> **Erreur fréquente à comprendre** Une destination admise garantit un coût borné de réponse.
>
> Un service autorisé peut être lent ou volumineux.

Egress

Une politique de sortie réseau complète le contrôle applicatif. Les couches apportent des garanties différentes. Une politique réseau de sortie peut bloquer des destinations que l'application n'aurait jamais dû joindre. Cette défense complète le contrôle applicatif et limite l'impact d'une erreur de validation ou d'une nouvelle route vulnérable.

> **Erreur fréquente à comprendre** Un filtre de chaîne remplace universellement tout contrôle réseau.
>
> Les couches apportent des garanties différentes.

Schémas

Les schémas d'URL doivent être bornés. Les capacités des schémas diffèrent et doivent être limitées. N'admettez que les schémas nécessaires, généralement HTTPS dans ce contrat d'import. Un parseur acceptant plusieurs protocoles ne signifie pas que votre produit doit exposer toutes leurs capacités.

> **Erreur fréquente à comprendre** Tout schéma parsable convient à un client HTTP métier.
>
> Les capacités des schémas diffèrent et doivent être limitées.

Laboratoire

Un test SSRF doit rester dans un laboratoire autorisé. Le périmètre d'essai doit protéger les tiers et l'infrastructure. Utilisez services factices et destinations autorisées pour tester refus et redirections. Les tests ne doivent pas sonder des infrastructures tierces ou des métadonnées réelles pour démontrer la propriété pédagogique.

> **Erreur fréquente à comprendre** Un exercice peut scanner arbitrairement les services internes réels.
>
> Le périmètre d'essai doit protéger les tiers et l'infrastructure.

Opération précise

L'autorisation doit porter sur l'opération distante admise. Le rôle ne rend pas toute destination légitime. Une destination partenaire autorisée peut exposer plusieurs chemins aux capacités différentes. Limitez méthode, route et données transmises selon le besoin métier, pas seulement le nom du serveur distant.

> **Erreur fréquente à comprendre** Un rôle utilisateur suffit à autoriser toute sortie réseau.
>
> Le rôle ne rend pas toute destination légitime.

Atelier guidé : lire, expliquer et vérifier le code

```
{
  "allowedHost": "partner.example.test",
  "initialUrl": "https://partner.example.test/lab/export",
  "redirectLocation": "http://127.0.0.1/internal"
}
```

L'URL initiale correspond au partenaire, mais Location vise une interface locale interne. Le suivi doit être refusé par la politique montrée. Ajoutez un test qui observe l'absence de connexion vers la cible interdite, puis un saut conforme pour vérifier le fonctionnement nominal du client.

Les redirections doivent rester dans le contrat de sortie. La confiance ne se propage pas arbitrairement. HTTP ne fournit pas ce chiffrement. La destination initiale ne prouve pas les autres pairs.

Travail pratique : Construisez deux récepteurs locaux, l'un admis et l'autre refusé. Simulez une redirection du premier vers le second.

Critères de correction : Vérifiez le refus au bon saut et la borne de temps. Ajoutez les cas de port non prévu, schéma non admis et résolution incompatible avec la politique. Contrôlez également les règles de sortie réseau du service.

SSRF : appliquer la politique à la connexion réelle

Le produit n'a besoin que de partenaires identifiés. Remplacer une URL libre par un identifiant résolu côté serveur réduit fortement la surface, mais la configuration partenaire et son parcours réseau demandent encore une vérification.

Une résolution DNS peut changer et une réponse initialement admise peut rediriger vers une cible interdite. La politique doit porter sur la destination réellement jointe, revalider chaque redirection et être complétée par un filtrage réseau de sortie. Limitez schémas, ports, méthodes et chemins au besoin, avec délais et taille maximale de réponse.

Ne tentez pas de résoudre le problème par une simple recherche textuelle de localhost ou par une seule résolution avant un second appel indépendant. Utilisez des composants adaptés et une topologie qui préserve le lien entre décision et connexion. Les tests se font avec DNS et serveurs factices autorisés, en vérifiant surtout l'absence d'appel aux destinations interdites et un chemin nominal partenaire qui continue de fonctionner.

Points de vigilance : Un domaine peut figurer dans le chemin, les identifiants ou un sous-domaine trompeur. Une politique de destination appliquée à la connexion, aux redirections et au réseau réduit les contournements. Une

résolution ultérieure ou une redirection peut viser une autre destination. TLS authentifie un nom ; il n'autorise pas l'application à joindre n'importe quelle ressource.

Documentation de référence ↗

Exemple commenté

```
Entrée : providerId = "catalogue-lab"
Configuration serveur : destination HTTPS connue
Client HTTP : délais et taille bornés, redirections contrôlées
```

L'identifiant sélectionne une destination configurée plutôt que donner au client le contrôle direct d'une URL arbitraire.

Exercice de mise en pratique

Construisez deux récepteurs locaux, l'un admis et l'autre refusé. Simulez une redirection du premier vers le second.

Voir une proposition de correction

Vérifiez le refus au bon saut et la borne de temps. Ajoutez les cas de port non prévu, schéma non admis et résolution incompatible avec la politique. Contrôlez également les règles de sortie réseau du service.

- Destinations explicites.
- Chaque saut contrôlé.
- Budgets de temps et de volume.

Contrôle de connaissances

Quel changement réduit fortement la surface d'un import distant ?

> Réponse expliquée : Choisir un fournisseur configuré par identifiant plutôt qu'accepter toute URL — La destination n'est plus entièrement pilotée par l'entrée.

Cours associé : 33. SSRF et sorties HTTP du serveur ↗

Documentation de référence ↗ https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

34. Uploads, ZIP et budgets par couche

Valider le contenu reçu sans confondre découpage technique et import autonome.

Comprendre le mécanisme

Un upload traverse plusieurs plafonds : proxy, moteur PHP, formulaire et logique de traitement. Le type MIME déclaré et le nom client sont des assertions non fiables. Une archive ajoute un risque de chemins dangereux, de fichiers inattendus et d'expansion excessive. La taille compressée, la somme décompressée, le nombre d'entrées et la taille de chaque image sont des dimensions distinctes à borner.

Raisonner et vérifier

Un découpage binaire d'un ZIP ne produit pas nécessairement plusieurs archives indépendantes lisibles par un importeur. Pour fractionner des réalisations, chaque partie doit contenir son manifeste JSON et les images qu'il référence. Si une seule image dépasse le plafond réseau, répartir les lignes ne suffit pas : il faut adapter une copie de l'image ou choisir un transfert séparé compatible avec le contrat d'import.

Budget multipart

Le multipart peut dépasser la taille du ZIP seul. L'enveloppe et les autres champs ajoutent des octets. Prévoyez fichier, métadonnées et champs de formulaire dans la limite de requête. Un ZIP sous un plafond exprimé pour le corps complet peut encore être refusé lorsque l'enveloppe le fait dépasser.

> **Erreur fréquente à comprendre** Un ZIP sous le seuil implique toute requête sous ce seuil.
>
> L'enveloppe et les autres champs ajoutent des octets.

Découpe binaire

Un fractionnement binaire ne crée pas des ZIP indépendants. L'importeur a besoin d'une structure d'archive complète. split produit des fragments d'un flux, pas plusieurs archives ZIP indépendantes. Un importeur qui attend chaque fois une archive complète ne peut pas utiliser directement ces fragments sans mécanisme de réassemblage.

> **Erreur fréquente à comprendre** Chaque segment binaire est automatiquement importable comme archive.
>
> L'importeur a besoin d'une structure d'archive complète.

Parties autonomes

Des parties importables doivent inclure leurs données référencées. Le contrat de l'import exige les références nécessaires. Pour reconstruire plusieurs imports, chaque archive doit contenir un manifeste valide et toutes les images référencées par ses éléments. Si une seule image dépasse le plafond, diviser uniquement les lignes ne résout pas ce cas.

> **Erreur fréquente à comprendre** Une partie JSON peut toujours omettre toutes ses images.
>
> Le contrat de l'import exige les références nécessaires.

Expansion

La taille décompressée doit être bornée. Le volume extrait peut être bien supérieur. Limitez taille totale extraite et taille par fichier pendant la lecture. Un petit fichier compressé peut demander beaucoup de stockage et de mémoire ; son poids réseau ne borne pas ces ressources.

> **Erreur fréquente à comprendre** Une archive de petite taille ne peut épuiser le disque.
>
> Le volume extrait peut être bien supérieur.

Chemins

Les chemins des entrées doivent rester dans le périmètre d'extraction. Des chemins hostiles peuvent sortir du répertoire prévu. Refusez les chemins absolus et les sorties du répertoire prévu après résolution contrôlée. Examinez aussi liens et formes spécifiques à la plateforme pour éviter qu'une entrée écrive hors du périmètre d'extraction.

> **Erreur fréquente à comprendre** Tout chemin fourni par le ZIP est une destination autorisée.
>
> Des chemins hostiles peuvent sortir du répertoire prévu.

Nombre de fichiers

Le nombre d'entrées doit être limité. De nombreuses entrées peuvent consommer des ressources. Bornez entrées, profondeur et opérations, même lorsque chaque fichier est petit. Une archive peut épuiser temps, inodes ou descripteurs sans dépasser une simple limite de taille totale.

> **Erreur fréquente à comprendre** Seule la taille du premier fichier détermine le coût total.
>
> De nombreuses entrées peuvent consommer des ressources.

Doublons

L'import doit définir le traitement des doublons. La reprise exige une politique explicite. Définissez si un identifiant existant est refusé, fusionné ou mis à jour. Cette politique doit rester stable entre parties et reprises ; sinon un fractionnement peut créer des doublons ou écraser des réalisations.

> **Erreur fréquente à comprendre** Réimporter une partie ne peut jamais répéter un effet.
>
> La reprise exige une politique explicite.

Original conservé

Un original doit être conservé avant transformation. Une copie permet de vérifier et reprendre sans perte. Travaillez sur une copie et conservez l'archive source avant transformation. Vérifiez manifestes et références après reconstruction afin de pouvoir revenir à l'original si un regroupement omet une image.

> **Erreur fréquente à comprendre** Découper exige de détruire l'archive source.
>
> Une copie permet de vérifier et reprendre sans perte.

Type réel

Le type déclaré par le client doit être vérifié. La déclaration n'est pas une analyse du contenu. Le nom .zip et le MIME client sont des affirmations. Le parseur et les règles de contenu doivent vérifier format et structure admise avant extraction ou publication des fichiers.

> **Erreur fréquente à comprendre** Content-Type application/zip prouve une archive valide.
>
> La déclaration n'est pas une analyse du contenu.

Import privilégié

L'autorisation et le CSRF restent nécessaires sur l'import admin. Le format ne prouve pas les droits de l'acteur. Un formulaire d'administration reste soumis à authentification, droit d'import et protection CSRF adaptée. Une limite de taille ou un ZIP valide n'établit pas l'autorisation de modifier le portfolio.

> **Erreur fréquente à comprendre** Une archive valide donne accès à l'administration.
>
> Le format ne prouve pas les droits de l'acteur.

Atelier guidé : lire, expliquer et vérifier le code

```
# Illustration de découpe binaire uniquement :
split -b 500000 portfolio.zip lab-part-
# L'importeur attend pour chaque envoi une archive ZIP complète.
```

La commande split découpe seulement le flux binaire de portfolio.zip. L'importeur exige pourtant un ZIP complet à chaque envoi. Reconstituez des archives autonomes à partir des données et dépendances, puis vérifiez leur validité et leur taille multipart estimée avant de les proposer à l'import.

Les octets sont coupés sans reconstruction logique. Le suffixe ne répare pas la structure. Le seuil ne recrée pas les références ou contenus. Il faut reconstruire des archives conformes avec leurs données nécessaires.

Travail pratique : Fractionnez un petit jeu fictif en archives autonomes, puis testez taille réseau, intégrité, chemins et doublons à la réimportation.

Critères de correction : Vérifiez chaque archive avec le lecteur réel, importez toutes les parties dans une base de test isolée et comparez champs et images. Rejouez l'import pour contrôler l'idempotence. Gardez les sources intactes pour comparer les résultats.

PDF : admission, stockage et restitution sont trois décisions

Un nom .pdf et un MIME client application/pdf restent des déclarations. Vérifiez le format admis et sa structure avec des composants adaptés, en bornant taille, temps et ressources du traitement. Aucun scanner unique ne garantit l'absence de tout danger dans un document.

Générez un nom interne indépendant du nom original et stockez hors d'un périmètre exécutable ou publiquement listable. Conservez une relation métier vers le propriétaire. Lors de la restitution, autorisez l'action avant lecture puis servez un type explicite, une disposition adaptée et une politique de cache cohérente avec la confidentialité.

Testez fichier malformé, dépassement, nom de chemin hostile et demande par un autre compte. Vérifiez aussi les routes alternatives de fichiers et les copies temporaires. Un upload correctement refusé ne doit laisser aucun document consultable, tandis qu'un import accepté ne doit pas contourner l'autorisation simplement parce que son URL est opaque.

Points de vigilance : Les métadonnées client ne sont pas fiables et le stockage exécutable augmente le risque. Une extension ne prouve pas le type ni l'innocuité du contenu. La limite de taille ne contrôle ni la nature du fichier ni son

exécution ou son accès. Ces contrôles se complètent ; une analyse de contenu adaptée reste utile et aucun scanner ne garantit l'absence de danger.

Documentation de référence ↗

Exemple commenté

```
partie-01.zip
  realisations.json  → sous-ensemble de réalisations
  images/a.png       → uniquement les visuels référencés
partie-02.zip
  realisations.json  → autre sous-ensemble
  images/b.png
```

Chaque partie possède une structure d'archive complète. Les références du JSON doivent correspondre exactement aux chemins internes attendus par l'importeur.

Exercice de mise en pratique

Fractionnez un petit jeu fictif en archives autonomes, puis testez taille réseau, intégrité, chemins et doublons à la réimportation.

Voir une proposition de correction

Vérifiez chaque archive avec le lecteur réel, importez toutes les parties dans une base de test isolée et comparez champs et images. Rejouez l'import pour contrôler l'idempotence. Gardez les sources intactes pour comparer les résultats.

- Bornes compressées et décompressées.
- Manifeste autonome par partie.
- Validation avec l'importeur réel.

Contrôle de connaissances

Une archive sous la limite réseau est-elle nécessairement sûre à décompresser ?

> Réponse expliquée : Non, expansion, chemins et contenu restent à contrôler — La taille reçue ne couvre pas tous les risques d'archive.

Cours associé : 34. Uploads, ZIP et budgets par couche ↗

Documentation de référence ↗ https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

35. Téléchargements et Content-Disposition

Séparer autorisation du fichier et présentation au navigateur.

Comprendre le mécanisme

Content-Disposition peut indiquer qu'une représentation doit être proposée en pièce jointe et fournir un nom de téléchargement. Ce nom est une métadonnée de présentation, pas une autorisation ni un chemin serveur sûr. Les variantes de paramètres permettent de traiter des noms selon un encodage défini ; une construction manuelle à partir d'une entrée libre peut introduire une syntaxe incorrecte. Utilisez l'API du framework prévue pour cet en-tête.

Raisonner et vérifier

Le serveur doit résoudre un identifiant de document vers une ressource autorisée, sans laisser l'appelant choisir arbitrairement un chemin du système de fichiers. Contrôlez les permissions à chaque téléchargement sensible et définissez cache et durée de disponibilité. Un lien signé est un contrat à part entière : sa portée, son expiration et sa diffusion doivent être explicites, sans confondre signature du lien et identité de toute personne qui le possède.

Présentation

Content-Disposition décrit la présentation du fichier. La permission doit être vérifiée séparément. Content-Disposition: attachment suggère un téléchargement au client. Cette instruction ne vérifie ni propriétaire ni confidentialité ; un document interdit reste une fuite même si le navigateur propose de l'enregistrer.

> **Erreur fréquente à comprendre** attachment accorde un droit de lecture au client.
>
> La permission doit être vérifiée séparément.

Résolution maîtrisée

Un identifiant métier peut être résolu en chemin contrôlé. Le serveur doit limiter les ressources distribuées. Recevez un identifiant métier puis résolvez côté serveur un chemin autorisé. N'exposez pas un paramètre de chemin libre qui donnerait au client la possibilité de choisir un fichier système.

> **Erreur fréquente à comprendre** Le client doit choisir librement tout chemin absolu.
>
> Le serveur doit limiter les ressources distribuées.

Nom de fichier

Les noms de fichiers doivent passer par une API adaptée. La construction doit empêcher les caractères et syntaxes indésirables. Utilisez une API de construction de disposition pour gérer caractères et encodage. Un nom provenant du client doit être contrôlé et ne doit pas être concaténé dans une ligne HTTP brute.

> **Erreur fréquente à comprendre** Concaténer tout nom reçu dans un en-tête est sûr.
>
> La construction doit empêcher les caractères et syntaxes indésirables.

Avant lecture

Le propriétaire de l'objet doit être vérifié avant lecture. Une ressource existante peut être privée. Vérifiez l'acteur et le sujet avant d'ouvrir ou streamer le document. Une erreur tardive peut arriver après émission partielle de données, même si le statut final semble refuser l'accès.

> **Erreur fréquente à comprendre** L'existence du fichier suffit à l'autoriser.
>
> Une ressource existante peut être privée.

URL opaque

Une URL opaque reste sensible selon son contrat. L'opacité ne prouve pas la permission. Une URL non devinable peut être une capacité au porteur selon son contrat. Protégez durée, diffusion et logs ; l'opacité ne remplace pas automatiquement une autorisation quand celle-ci est exigée.

> **Erreur fréquente à comprendre** Un nom difficile à deviner remplace toujours l'autorisation.
>
> L'opacité ne prouve pas la permission.

Type explicite

Le type servi doit correspondre au contrat. Le serveur doit connaître la représentation distribuée. Servez le format prévu avec un Content-Type correct et les protections d'interprétation adaptées. Une extension de téléchargement ne doit pas être l'unique source de vérité sur la représentation reçue.

> **Erreur fréquente à comprendre** L'extension client définit toujours le bon Content-Type.
>
> Le serveur doit connaître la représentation distribuée.

Cache

Les réponses privées demandent une politique de cache. La présentation ne définit pas seule le stockage. Un document personnel exige une stratégie de stockage et de partage cohérente. Vérifiez navigateur, CDN et routes alternatives ; attachment n'empêche pas à lui seul la conservation d'une copie.

> **Erreur fréquente à comprendre** attachment interdit automatiquement tous les caches partagés.
>
> La présentation ne définit pas seule le stockage.

Erreurs minimisées

Les erreurs doivent éviter de révéler des chemins internes. Le diagnostic public doit rester minimisé. Un fichier absent ne doit pas exposer son chemin interne complet ou les répertoires du serveur. Retournez une catégorie publique stable et gardez le diagnostic privé nécessaire à l'exploitation.

> **Erreur fréquente à comprendre** Un échec de lecture doit afficher tout le système de fichiers.
>
> Le diagnostic public doit rester minimisé.

Budgets

Le téléchargement doit respecter des budgets de ressources. Les limites de service restent nécessaires. Bornez taille, durée et concurrence de téléchargement selon la capacité. Un streaming peut réduire la mémoire sans rendre gratuits bande passante, descripteurs et occupation de connexion.

> **Erreur fréquente à comprendre** Un droit valide autorise une réponse infinie.
>
> Les limites de service restent nécessaires.

Document tiers

Un test doit tenter la lecture d'un document tiers. La frontière d'autorisation exige un cas négatif. Créez deux comptes et demandez avec B le document de A. Vérifiez refus et absence d'octets confidentiels, puis un cas propriétaire qui reçoit exactement le document prévu.

> **Erreur fréquente à comprendre** Le téléchargement nominal prouve l'isolation entre comptes.
>
> La frontière d'autorisation exige un cas négatif.

Atelier guidé : lire, expliquer et vérifier le code

```
HTTP/1.1 200 OK
Content-Type: application/pdf
Content-Disposition: attachment; filename="lab-report.pdf"

Contexte : document d'un autre compte, aucune autorisation vérifiée.
```

Le document appartient à un autre compte et aucun contrôle n'est exécuté. Les headers PDF et attachment ne corrigent pas cette absence. Placez l'autorisation avant la lecture des octets puis vérifiez le refus avec un identifiant valide d'un tiers et une réponse sans contenu partiel.

Répéter le mode de présentation ne change pas les droits. Le renommage ne crée pas d'autorisation. Les champs de présentation ne valident pas l'accès à l'objet. Le nom ne donne pas la permission.

Travail pratique : Créez deux documents fictifs appartenant à deux acteurs et testez leur téléchargement par identifiant. Ajoutez un nom de fichier Unicode.

Critères de correction : Vérifiez les permissions avant ouverture du fichier, générez l'en-tête avec l'API prévue et testez le nom présenté. Refusez les identifiants inconnus ou interdits sans exposer de chemin interne ni de trace.

Exemple commenté

```
Content-Type: application/pdf
Content-Disposition: attachment; filename="rapport-lab.pdf"
Cache-Control: private, no-store
```

Les champs organisent le rendu et le stockage de la réponse. Ils ne démontrent pas que l'utilisateur a le droit d'obtenir le document.

Exercice de mise en pratique

Créez deux documents fictifs appartenant à deux acteurs et testez leur téléchargement par identifiant. Ajoutez un nom de fichier Unicode.

Voir une proposition de correction

Vérifiez les permissions avant ouverture du fichier, générez l'en-tête avec l'API prévue et testez le nom présenté. Refusez les identifiants inconnus ou interdits sans exposer de chemin interne ni de trace.

- Identifiant métier vers chemin maîtrisé.
- Autorisation avant lecture.
- En-têtes construits par API.

Contrôle de connaissances

Content-Disposition: attachment accorde-t-il un droit d’accès au fichier ?

> Réponse expliquée : Non, il décrit sa présentation — L’autorisation doit être appliquée avant la distribution.

Cours associé : 35. Téléchargements et Content-Disposition ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Disposition

36. Limitation de débit et budgets d’abus

Choisir l’unité protégée plutôt qu’un compteur arbitraire.

Comprendre le mécanisme

429 indique une limitation de demandes selon une politique serveur. Une politique utile précise la ressource protégée, les dimensions de comptage, la fenêtre et le comportement de reprise. Une adresse réseau partagée peut représenter plusieurs utilisateurs ; une adresse annoncée dans un champ non fiable peut être falsifiée. Selon le risque, acteur, opération, coût et ressource métier peuvent être des dimensions plus pertinentes qu’un seul nombre de connexions.

Raisonner et vérifier

Un limiteur ne remplace pas l’autorisation et doit rester cohérent entre instances si son contrat est global. Le client peut recevoir une indication Retry-After, sans obtenir un droit illimité de réessayer. Testez les bordures de fenêtre, les actions coûteuses et la récupération après expiration. Une tentative refusée ne doit pas avoir déjà produit l’effet métier qu’on voulait borner.

Signal 429

429 indique une limitation selon une politique. La cause protocolaire n’est pas cette preuve d’identité. Le statut annonce une limitation selon la politique du service. Le client doit pouvoir distinguer ce refus d’une panne interne et connaître les conditions de reprise lorsque le contrat les fournit.

> **Erreur fréquente à comprendre** 429 prouve une erreur de mot de passe.
>
> La cause protocolaire n’est pas cette preuve d’identité.

Dimensions

Le quota doit préciser ressource et fenêtre. L’interprétation demande dimensions et durée. Précisez ressource, acteur, tenant, fenêtre et coût compté. Un nombre sans unité ni portée ne décrit pas une règle testable ; dix demandes par seconde diffèrent de dix exports simultanés.

> **Erreur fréquente à comprendre** Un nombre sans unité suffit à définir la politique.
>
> L’interprétation demande dimensions et durée.

IP partagée

Une IP partagée peut représenter plusieurs acteurs. NAT et réseaux partagés compliquent cette équivalence. Un NAT peut regrouper plusieurs personnes derrière une adresse. Une limite par IP seule peut pénaliser des utilisateurs légitimes ; combinez les dimensions adaptées sans faire confiance à une identité réseau fournie librement.

> **Erreur fréquente à comprendre** Une IP correspond toujours à un utilisateur unique.
>
> NAT et réseaux partagés compliquent cette équivalence.

Proxies

Une adresse annoncée doit suivre la confiance proxy. Le champ reçu peut être contrôlé par un client. L'adresse cliente doit venir de la chaîne de confiance configurée. Si le client peut choisir X-Forwarded-For utilisé par le quota, il peut changer de compteur sans changer d'acteur réel.

> **Erreur fréquente à comprendre** Tout X-Forwarded-For donne une identité fiable au limiteur.
>
> Le champ reçu peut être contrôlé par un client.

Plusieurs instances

Un quota global demande une coordination adaptée. Les états locaux peuvent être indépendants. Des compteurs locaux indépendants multiplient la capacité globale apparente. Utilisez une coordination adaptée au contrat et testez deux instances lorsque vous annoncez un plafond partagé par tout le service.

> **Erreur fréquente à comprendre** Un compteur local est toujours global sur dix instances.
>
> Les états locaux peuvent être indépendants.

Retry-After

Retry-After peut guider la reprise. Une indication de délai n'est pas une autorisation illimitée. Une valeur de 30 dans la forme numérique indique ici des secondes. Le client doit respecter le contrat et ajouter une stratégie de reprise bornée, sans relancer immédiatement une rafale identique.

> **Erreur fréquente à comprendre** Retry-After accorde un nombre illimité de retries.
>
> Une indication de délai n'est pas une autorisation illimitée.

Coût variable

Le coût métier peut être une dimension utile. Les opérations consomment des ressources différentes. Un export lourd peut coûter davantage qu'une lecture simple. Une politique pondérée ou une limite d'opérations actives peut mieux protéger la capacité qu'un simple compteur uniforme de requêtes.

> **Erreur fréquente à comprendre** Toute requête a nécessairement le même coût.
>
> Les opérations consomment des ressources différentes.

Refus précoce

Un refus pertinent doit précéder l'effet coûteux. Le travail a déjà été effectué si le contrôle arrive après. Décidez avant l'effet coûteux lorsque la règle l'exige. Retourner 429 après avoir appelé le prestataire ou généré un gros fichier ne protège pas la ressource que le quota devait borner.

> **Erreur fréquente à comprendre** Envoyer l'email puis limiter protège cet envoi.
>
> Le travail a déjà été effectué si le contrôle arrive après.

Bornes de fenêtre

Les bordures de fenêtre doivent être testées. Le comportement aux seuils demande des essais spécifiques. Testez début et fin de fenêtre et concurrence proche du plafond. Les algorithmes de fenêtre peuvent autoriser des rafales différentes ; décrivez le comportement mesuré au lieu de promettre une fluidité absolue.

> **Erreur fréquente à comprendre** Un seul appel nominal prouve le traitement des bursts.
>
> Le comportement aux seuils demande des essais spécifiques.

Autorisation conservée

La limitation complète l'autorisation. Le budget n'accorde pas de permission métier. Un acteur sous son quota ne gagne aucun droit sur les données d'un tiers. Limitation et contrôle d'accès doivent rester deux décisions explicites, avec leurs propres cas de refus.

> **Erreur fréquente à comprendre** Un quota disponible donne accès à tout objet.
>
> Le budget n'accorde pas de permission métier.

Atelier guidé : lire, expliquer et vérifier le code

```
HTTP/1.1 429 Too Many Requests
Retry-After: 30
Cache-Control: no-store
```

La réponse demande une attente de trente secondes dans cette forme de Retry-After. Définissez aussi le nombre maximal de nouvelles tentatives et le budget global. Côté serveur, vérifiez que le refus intervient avant le travail coûteux et que les compteurs correspondent au périmètre annoncé.

L'annonce ne supprime pas le quota. Cette forme de Retry-After exprime ici des secondes. Ce champ ne décrit pas ce cycle de compte. La reprise peut encore être refusée selon le contrat.

Travail pratique : Définissez une politique de laboratoire pour lecture de catalogue et envoi d'un email coûteux. Testez deux utilisateurs derrière une même adresse.

Critères de correction : Vérifiez que les dimensions choisies évitent de mélanger aveuglément tous les acteurs et bornent réellement le travail coûteux. Testez plusieurs instances, concurrence et reprise, puis documentez les faux refus attendus.

Exemple commenté

```
HTTP/1.1 429 Too Many Requests
Retry-After: 30
Cache-Control: no-store
```

Le délai illustratif est exprimé ici en secondes. La stratégie réelle doit définir quota, coût et traitement des demandes concurrentes.

Exercice de mise en pratique

Définissez une politique de laboratoire pour lecture de catalogue et envoi d'un email coûteux. Testez deux utilisateurs derrière une même adresse.

Voir une proposition de correction

Vérifiez que les dimensions choisies évitent de mélanger aveuglément tous les acteurs et bornent réellement le travail coûteux. Testez plusieurs instances, concurrence et reprise, puis documentez les faux refus attendus.

- Compter la bonne ressource.
- Confiance des adresses explicite.
- Pas d’effet avant un refus pertinent.

Contrôle de connaissances

Que doit préciser une politique de limitation utile ?

> Réponse expliquée : Ressource, dimensions, fenêtre et reprise — Le compteur doit correspondre au risque et au coût.

Cours associé : 36. Limitation de débit et budgets d’abus ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/429

37. Authentification HTTP et autorisation métier

Interpréter les challenges sans réduire tous les refus au mot de passe.

Comprendre le mécanisme

WWW-Authenticate décrit un challenge selon un schéma d'authentification. Le client et le serveur doivent partager le contrat du schéma et le contexte auquel l'identité s'applique. Une réponse signalant l'absence d'authentification ne se confond pas avec un refus d'action pour un acteur déjà identifié. Des règles de non-divulgation peuvent aussi influencer la réponse publique. Dans tous les cas, les événements internes restent corrélables sans exposer les secrets.

Raisonner et vérifier

Un encodage de credentials n'est pas un chiffrement ; le canal sécurisé et la validation restent nécessaires. Un bearer token accorde une capacité à son détenteur selon le profil prévu : le transport, le stockage, la durée et la validation du jeton font partie du modèle. Un client authentifié doit encore prouver la permission sur l'objet et l'opération demandés. Testez identité absente, jeton invalide et permission insuffisante séparément.

Challenge

WWW-Authenticate porte un challenge selon un schéma. Le champ décrit un mécanisme d'authentification. WWW-Authenticate annonce un challenge selon le schéma utilisé. Le client et le serveur doivent partager ce contrat ; un texte arbitraire ne met pas en œuvre un protocole d'authentification complet.

> **Erreur fréquente à comprendre** WWW-Authenticate chiffre automatiquement le mot de passe.
>
> Le champ décrit un mécanisme d'authentification.

Bearer validé

Un bearer token doit être validé. Le nom du schéma ne vérifie pas la valeur. Recevoir Authorization: Bearer indique seulement une valeur présentée. Vérifiez preuve ou introspection, profil, durée et retrait selon le mécanisme avant de construire une identité utilisable par le métier.

> **Erreur fréquente à comprendre** La présence du mot Bearer prouve la validité.
>
> Le nom du schéma ne vérifie pas la valeur.

Permission

Une identité valide ne donne pas tous les droits. L'autorisation porte sur l'objet et l'opération. Un utilisateur reconnu peut rester interdit sur une réservation. Contrôlez l'action sur cet objet après authentification, avec test propriétaire et tiers ; la connexion ne suffit pas à prouver cette relation.

> **Erreur fréquente à comprendre** Un login réussi autorise toute réservation tierce.
>
> L'autorisation porte sur l'objet et l'opération.

Basic et base64

Base64 est un encodage. L'encodage est réversible sans secret. La représentation base64 du schéma Basic est réversible sans secret. Elle ne chiffre pas lab:public ; le canal doit protéger les credentials et leurs copies doivent être exclues des traces ordinaires.

> **Erreur fréquente à comprendre** Base64 rend les credentials confidentiels contre le lecteur.
>
> L'encodage est réversible sans secret.

Transport

Le canal doit protéger les credentials sensibles. La signature ne protège pas contre toute interception et réutilisation. Utilisez un canal authentifié et chiffré selon le contrat. Une politique de header correcte ne répare pas une transmission vers une mauvaise destination ou une validation TLS désactivée.

> **Erreur fréquente à comprendre** Un token signé dispense de HTTPS.
>
> La signature ne protège pas contre toute interception et réutilisation.

Catégories de refus

Une politique peut distinguer identité absente et droit insuffisant. Le contrat public doit aussi tenir compte de la non-divulgation. Distinguez dans la politique identité absente ou invalide et permission insuffisante. Le choix de réponse publique doit rester cohérent sans divulguer inutilement l'existence de ressources confidentielles.

> **Erreur fréquente à comprendre** Tous les refus doivent révéler les détails internes du compte.
>
> Le contrat public doit aussi tenir compte de la non-divulgation.

Profil partagé

Le schéma et son profil doivent être partagés. Les règles de validation doivent être explicites. Bearer ne dit pas si le token est JWT ou opaque ni quelle audience est attendue. Documentez la validation convenue avec l'émetteur et refusez les profils qui ressemblent seulement au bon format.

> **Erreur fréquente à comprendre** Une chaîne longue suffit à définir un protocole sûr.
>
> Les règles de validation doivent être explicites.

Logs

Les journaux doivent éviter les credentials bruts. Le diagnostic peut utiliser des catégories et corrélations. Retirez Authorization et cookies des événements ordinaires. Une trace de diagnostic qui conserve la valeur entière peut fournir à son lecteur la même capacité que le porteur légitime.

> **Erreur fréquente à comprendre** Déboguer un challenge exige de publier les tokens.
>
> Le diagnostic peut utiliser des catégories et corrélations.

Redirections

Un client doit éviter de transmettre ses secrets à toute redirection. La destination et le périmètre doivent être contrôlés. Réévaluez credentials lors d'un changement de destination du client HTTP. Vérifiez les règles de la bibliothèque et les sauts admis ; un secret ne doit pas suivre librement toute URL retournée.

> **Erreur fréquente à comprendre** Les credentials suivent automatiquement toute destination admise visuellement.
>
> La destination et le périmètre doivent être contrôlés.

Tests distincts

Les tests doivent distinguer token invalide et permission insuffisante. Authentification et autorisation sont deux décisions. Utilisez un token invalide pour l'authentification et un token valide d'un tiers pour l'autorisation. Ces cas isolent deux propriétés différentes et évitent qu'un rejet précoce masque une règle d'objet absente.

> **Erreur fréquente à comprendre** Un seul test connecté couvre les deux frontières.
>
> Authentification et autorisation sont deux décisions.

Atelier guidé : lire, expliquer et vérifier le code

```
Authorization: Basic bGFiOnB1YmxpYw==

Fixture publique : lab:public ; aucune requête réelle n'est envoyée.
```

La valeur Basic encode les credentials publics de laboratoire, sans les chiffrer. Décodez-la localement puis expliquez pourquoi HTTPS et minimisation des traces restent nécessaires. Aucun appel réel n'est requis pour comprendre la différence entre représentation du header, protection du canal et décision de permission.

Le schéma et l'encodage ne protègent pas seuls le secret sur le canal. Il n'y a pas cette clé de chiffrement. Le canal reste une protection nécessaire. Le serveur doit valider identité et permissions.

Travail pratique : Écrivez une matrice de réponses pour API publique, API authentifiée et action réservée au propriétaire.

Critères de correction : Testez le challenge et les messages publics sans enregistrer le token. Vérifiez que le refus de droit ne relance pas inutilement une authentification et qu'aucune mutation ne précède la décision d'autorisation.

Exemple commenté

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="lab-api"
Cache-Control: no-store
```

Le schéma annonce le contexte d'authentification de ce laboratoire. La présence d'un champ Bearer côté client ne valide pas son contenu.

Exercice de mise en pratique

Écrivez une matrice de réponses pour API publique, API authentifiée et action réservée au propriétaire.

Voir une proposition de correction

Testez le challenge et les messages publics sans enregistrer le token. Vérifiez que le refus de droit ne relance pas inutilement une authentification et qu'aucune mutation ne précède la décision d'autorisation.

- Schéma et profil explicites.
- Transport sécurisé.
- Permission vérifiée sur l'objet.

Contrôle de connaissances

La réception d'un bearer token suffit-elle à identifier un utilisateur autorisé ?

> Réponse expliquée : Non, validation du jeton puis autorisation restent nécessaires — Le format reçu n'est pas une preuve valide.

Cours associé : 37. Authentification HTTP et autorisation métier ↗

Documentation de référence ↗ https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/WWW-Authenticate

38. Clients HTTP serveur : délais, tailles et reprises

Borner la consommation d'une dépendance avant d'ajouter des retries.

Comprendre le mécanisme

Un client HTTP serveur possède plusieurs étapes : résolution, connexion, négociation, attente et lecture du corps. Une option nommée timeout peut viser une période d'inactivité plutôt qu'une durée totale ; lisez le contrat exact de la bibliothèque. Une réponse qui arrive lentement par petits morceaux peut donc consommer plus de temps que prévu si seul un délai partiel est configuré. Le volume du corps doit également être borné.

Raisonner et vérifier

Fixez un budget de bout en bout compatible avec celui de l'appelant. Les reprises ajoutent du travail et peuvent aggraver une saturation, surtout lorsqu'un effet distant est incertain. Réservez-les aux cas compatibles avec le contrat d'idempotence, bornez leur nombre et leur durée, et appliquez une attente adaptée. Vérifiez aussi l'annulation et le traitement des erreurs : une tâche client abandonnée ne prouve pas que le service distant a cessé son effet.

Contrat du timeout

Un timeout peut viser l'inactivité plutôt que toute la durée. La sémantique dépend de la bibliothèque. Une option peut mesurer inactivité ou une étape de connexion plutôt que durée totale. Consultez le client employé et testez le comportement ; le nom timeout ne garantit pas un budget global uniforme.

> **Erreur fréquente à comprendre** Toute option timeout définit le même budget total.
>
> La sémantique dépend de la bibliothèque.

Volume

Le volume de réponse doit être borné. Temps et taille sont des limites indépendantes. Limitez les octets lus et, si pertinent, décompressés. Une réponse rapide mais énorme peut épuiser la mémoire disponible ou l'espace disque malgré une échéance temporelle correctement configurée.

> **Erreur fréquente à comprendre** Un délai court garantit une faible mémoire consommée.
>
> Temps et taille sont des limites indépendantes.

Retries

Les retries consomment un budget supplémentaire. Il crée un travail et un trafic supplémentaires. Chaque tentative ajoute du travail au service distant et consomme votre budget. Une dépendance saturée peut devenir encore moins disponible si tous ses clients reprennent immédiatement sans limite ni attente.

> **Erreur fréquente à comprendre** Un retry est toujours gratuit pour la dépendance.
>
> Il crée un travail et un trafic supplémentaires.

Mutation reprise

La reprise d'une mutation demande un contrat d'idempotence. L'effet distant peut avoir réussi avant la perte de réponse. Un timeout ne prouve pas l'absence d'effet. Pour une création ou un paiement, conservez une identité stable et utilisez le mécanisme d'idempotence ou de réconciliation du fournisseur avant toute nouvelle intention.

> **Erreur fréquente à comprendre** Tout timeout autorise une nouvelle mutation sans risque.
>
> L'effet distant peut avoir réussi avant la perte de réponse.

Flux lent

Une lecture lente peut rester active par petits morceaux. Il faut distinguer inactivité et durée de bout en bout. Un serveur peut envoyer quelques octets régulièrement et éviter un timeout d'inactivité. Une durée totale maximale reste nécessaire pour borner l'occupation du worker dans ce scénario.

> **Erreur fréquente à comprendre** Recevoir un octet annule nécessairement tout budget total.
>
> Il faut distinguer inactivité et durée de bout en bout.

Annulation

L'annulation locale ne prouve pas l'arrêt distant. Les effets peuvent continuer côté service. Arrêter l'attente locale ne démontre pas que le traitement distant s'est arrêté. Conservez un état incertain et recherchez le résultat si l'opération peut avoir produit un effet durable.

> **Erreur fréquente à comprendre** Fermer le client annule forcément un commit distant.
>
> Les effets peuvent continuer côté service.

Budget de l'appelant

Un budget doit être compatible avec celui de l'appelant. Les durées s'additionnent et consomment des ressources. Le temps accordé à une dépendance doit laisser place au traitement et à la réponse du service appelant. Empiler des délais plus longs que la requête entrante peut créer un travail devenu inutile.

> **Erreur fréquente à comprendre** Une dépendance peut attendre indéfiniment sans affecter le parcours.
>
> Les durées s'additionnent et consomment des ressources.

Tentatives bornées

Le nombre de tentatives doit être borné. La reprise peut amplifier la panne. Fixez nombre maximal, délais et budget global, avec jitter lorsque pertinent. Une règle d'arrêt doit rester applicable même si chaque tentative individuelle semble terminer dans son propre délai.

> **Erreur fréquente à comprendre** Une dépendance saturée doit recevoir des retries infinis.
>
> La reprise peut amplifier la panne.

Erreurs de phases

Les erreurs simulées doivent couvrir connexion et lecture. Les phases possèdent des échecs différents. Testez connexion impossible, lecture interrompue et flux lent avec services factices. Un seul refus immédiat ne vérifie pas les budgets et états incertains des autres phases.

> **Erreur fréquente à comprendre** Un succès rapide couvre toutes les pannes de transport.
>
> Les phases possèdent des échecs différents.

Diagnostics minimisés

Le diagnostic doit minimiser les secrets des appels. La surveillance ne doit pas exposer les capacités utilisées. Conservez fournisseur, phase, durée et catégorie d'échec sans URL contenant un secret ni header d'authentification. Un message d'exception de client doit être revu avant d'être exposé ou journalisé tel quel.

> **Erreur fréquente à comprendre** Les logs clients doivent recopier chaque Authorization.
>
> La surveillance ne doit pas exposer les capacités utilisées.

Atelier guidé : lire, expliquer et vérifier le code

```
# Options d'un appel HttpClient :
timeout: 1.0
max_duration: 3.0
```

Dans ce contrat HttpClient, timeout et max_duration couvrent deux dimensions temporelles. Le premier peut viser l'inactivité, le second une durée totale maximale. Testez un flux qui envoie régulièrement de petits morceaux pour démontrer pourquoi ces deux limites ne sont pas interchangeables.

L'idempotence demande un contrat séparé. Une durée n'est pas une limite de volume. Le budget ne vérifie pas l'identité TLS. Les options répondent à deux dimensions temporelles différentes.

Travail pratique : Simulez une dépendance silencieuse, une dépendance envoyant lentement et une réponse trop volumineuse.

Critères de correction : Mesurez le temps total et la mémoire, puis vérifiez le refus borné et l'absence de reprise incontrôlée. Ajoutez un cas où la réponse d'une mutation est perdue ; démontrez la politique d'idempotence avant d'activer un retry.

Exemple commenté

```
Budget du parcours : 3 s
Connexion + traitement + lecture + reprises éventuelles ≤ budget choisi
Corps maximal : borne applicative explicite
```

Les nombres sont des paramètres de laboratoire, pas des valeurs universelles. La mesure doit couvrir le temps et le volume réellement consommés.

Exercice de mise en pratique

Simulez une dépendance silencieuse, une dépendance envoyant lentement et une réponse trop volumineuse.

Voir une proposition de correction

Mesurez le temps total et la mémoire, puis vérifiez le refus borné et l'absence de reprise incontrôlée. Ajoutez un cas où la réponse d'une mutation est perdue ; démontrez la politique d'idempotence avant d'activer un retry.

- Lire le sens des options.

- Budget total et volume bornés.
- Reprises compatibles avec l’effet.

Contrôle de connaissances

Une option timeout garantit-elle toujours une durée totale de requête ?

> Réponse expliquée : Non, son contrat peut viser une autre étape ou l’inactivité — Il faut vérifier la sémantique de l’option.

Cours associé : 38. Clients HTTP serveur : délais, tailles et reprises ↗

Documentation de référence ↗ https://symfony.com/doc/8.0/http_client.html

39. Journaux HTTP sans fuite et sans faux contexte

Conserver des preuves utiles tout en minimisant les données collectées.

Comprendre le mécanisme

Les logs de trafic et d'authentification doivent permettre de relier un incident à une version, une opération et un résultat. Ils ne doivent pas devenir un inventaire de cookies, Authorization, mots de passe ou corps personnels. Les entrées contrôlées par un client peuvent aussi déformer un journal si elles sont assemblées sans encodage adapté. Un identifiant de corrélation facilite l'enquête, sans prouver à lui seul une identité.

Raisonner et vérifier

Définissez champs, niveaux, accès et conservation selon un besoin précis. Une erreur complète peut rester dans un stockage privé restreint tandis que la réponse publique porte un code stable. Testez les caractères de contrôle, les volumes et les pannes du système de logs. Conservez la capacité de diagnostiquer une demande refusée tôt au proxy, sans augmenter arbitrairement la collecte sensible dans toute la chaîne.

Credentials

Les logs doivent éviter les bearer tokens bruts. Une fuite de log peut alors permettre un rejeu. Un bearer token brut dans les logs devient une copie potentiellement rejouable. Supprimez-le des champs collectés et vérifiez également proxy, APM et exceptions, qui peuvent journaliser la requête automatiquement.

> **Erreur fréquente à comprendre** Un journal ordinaire doit conserver chaque token complet.
>
> Une fuite de log peut alors permettre un rejeu.

Entrées hostiles

Une structure adaptée aide à encoder les entrées non fiables. Des entrées peuvent déformer la lecture des événements. Utilisez une journalisation structurée avec encodage adapté. Un champ client contenant retour de ligne ou délimiteur ne doit pas pouvoir fabriquer une fausse entrée ou modifier la lecture du contexte.

> **Erreur fréquente à comprendre** Concaténer les retours à la ligne clients est toujours sûr.
>
> Des entrées peuvent déformer la lecture des événements.

Corrélation

Un requestId sert à corréler. L'unicité apparente ne prouve pas l'identité. requestId rapproche les événements mais n'authentifie pas l'acteur. Validez ou générez cette référence selon sa provenance et gardez séparée l'identité réellement établie par le mécanisme de sécurité du serveur.

> **Erreur fréquente à comprendre** Un requestId client authentifie l'utilisateur.
>
> L'unicité apparente ne prouve pas l'identité.

Version

La version déployée aide à situer un incident. La version peut aider sans collecter cette donnée sensible. Associez l'événement à la release et au composant concernés. Cette information aide à relier un incident à une configuration réelle plutôt qu'à une trace ancienne ou à un autre nœud.

> **Erreur fréquente à comprendre** Le hash du mot de passe est indispensable à chaque événement.
>
> La version peut aider sans collecter cette donnée sensible.

Accès

Les droits d'accès aux logs doivent être définis. Les diagnostics peuvent contenir des informations sensibles. Les journaux peuvent révéler comportements et données personnelles même sans mot de passe. Définissez qui peut les lire, exporter et partager, avec une traçabilité adaptée aux besoins d'exploitation.

> **Erreur fréquente à comprendre** Tous les journaux doivent être publics par défaut.
>
> Les diagnostics peuvent contenir des informations sensibles.

Conservation

Une durée de conservation doit être justifiée. Les besoins et risques de collecte doivent être maîtrisés. Choisissez durée et finalité au lieu de conserver indéfiniment par défaut. Les exports et sauvegardes participent aussi à cette conservation ; une suppression du store actif ne retire pas toutes les copies.

> **Erreur fréquente à comprendre** Conserver indéfiniment toute trace est toujours nécessaire.
>
> Les besoins et risques de collecte doivent être maîtrisés.

Panne de collecte

Les pannes de journalisation doivent être prévues. Disque, débit et services distants peuvent échouer. Prévoyez saturation, indisponibilité et erreurs du pipeline de logs. Le service doit avoir une politique explicite : la journalisation ne doit pas bloquer sans borne ni masquer silencieusement des événements indispensables.

> **Erreur fréquente à comprendre** Le système de logs ne peut jamais manquer de ressources.
>
> Disque, débit et services distants peuvent échouer.

Avant l'application

Le proxy peut journaliser un refus avant l'application. Un refus peut se situer en amont. Un 413 peut être journalisé uniquement au proxy. Corrélez les couches et n'interprétez pas l'absence de trace PHP comme une preuve que le navigateur n'a émis aucune demande.

> **Erreur fréquente à comprendre** L'absence de log PHP prouve l'absence de toute requête réseau.
>
> Un refus peut se situer en amont.

Public et privé

Le diagnostic public peut rester plus limité que le privé. Les besoins d’accès et de confidentialité diffèrent. Le client peut recevoir une catégorie stable et un identifiant, tandis que le support consulte un diagnostic protégé. Ne publiez pas la trace interne pour faciliter une investigation ponctuelle.

> **Erreur fréquente à comprendre** La trace privée doit toujours être affichée au client.
>
> Les besoins d’accès et de confidentialité diffèrent.

Partage expurgé

Les preuves doivent être expurgées avant partage. Les fixtures et résultats peuvent être sans données réelles. Avant de joindre une preuve à un ticket, retirez cookies, tokens et données clientes inutiles. Vérifiez le fichier final, y compris captures et champs masqués seulement visuellement dans l’outil d’origine.

> **Erreur fréquente à comprendre** Un dossier d’exercice exige des secrets de production.
>
> Les fixtures et résultats peuvent être sans données réelles.

Atelier guidé : lire, expliquer et vérifier le code

```
{
  "requestId": "lab-42",
  "operation": "orders.read",
  "Authorization": "Bearer public-fixture-only",
  "result": "denied"
}
```

Le champ Authorization contient la valeur complète, même si la fixture est publique. Dans une demande réelle, ce modèle de journalisation créerait une copie de credential. Remplacez-le par une catégorie de mécanisme et un résultat minimisé, puis vérifiez que la valeur ne réapparaît pas dans les exceptions ou traces automatiques.

L’opération générique peut être utile selon la politique. Le format JSON ne constitue pas la donnée sensible montrée. Un bearer token peut devenir une capacité rejouable si le log fuite. Le résultat aide au diagnostic sans imposer ce secret.

Travail pratique : Produisez une matrice de journalisation pour succès, refus de droit, 413 et timeout. Ajoutez une entrée client contenant un retour à la ligne fictif.

Critères de correction : Vérifiez la structure de l’événement et l’absence de secret. Comparez journaux proxy et application avec une corrélation bornée. Testez la politique de conservation et la visibilité des diagnostics selon le rôle d’exploitation.

Événements d’authentification utiles sans secrets

Pour comprendre des échecs de connexion, conservez résultat, catégorie de mécanisme, horodatage, composant et corrélation non sensible. Un token ou un mot de passe brut n’est pas nécessaire pour compter les refus et retrouver le chemin d’un incident.

Utilisez des événements structurés et encodez les valeurs non fiables afin qu’un nom contenant un retour de ligne ne fabrique pas une autre entrée. Protégez accès, intégrité et durée de conservation ; les journaux peuvent rester sensibles même après suppression des credentials.

Vérifiez toute la collecte : proxy, application, exceptions et APM peuvent enregistrer la même demande selon des règles différentes. Une liste noire dans un seul logger ne garantit pas l’absence de copies ailleurs. Avec une fixture identifiable, provoquez un refus et inspectez les artefacts finaux. Avant partage dans un ticket, expurgez de nouveau les preuves et ne considérez pas un requestId fourni par le client comme une identité authentifiée.

Points de vigilance : Les tokens, cookies et mots de passe peuvent devenir des moyens d'usurpation dans les logs. L'absence de traces empêche de détecter et d'investiguer les attaques. La minimisation préserve les signaux utiles ; les données non fiables doivent aussi être encodées pour éviter l'injection de logs. D'autres données et secrets restent exposés ; la confidentialité exige un contrôle global des accès.

Documentation de référence ↗

Exemple commenté

```
{"requestId":"lab-42","operation":"reservation.cancel","result":"denied","release":"abc123"}
```

L'événement décrit une décision avec une corrélation et une version. Il n'embarque pas le cookie ou le token ayant accompagné la demande.

Exercice de mise en pratique

Produisez une matrice de journalisation pour succès, refus de droit, 413 et timeout. Ajoutez une entrée client contenant un retour à la ligne fictif.

Voir une proposition de correction

Vérifiez la structure de l'événement et l'absence de secret. Comparez journaux proxy et application avec une corrélation bornée. Testez la politique de conservation et la visibilité des diagnostics selon le rôle d'exploitation.

- Pas de credentials bruts.
- Données non fiables encodées.
- Accès et conservation maîtrisés.

Contrôle de connaissances

Quel contenu éviter dans un événement HTTP ordinaire ?

> Réponse expliquée : Le bearer token brut — Il peut devenir une capacité d'usurpation.

Cours associé : 39. Journaux HTTP sans fuite et sans faux contexte ↗

Documentation de référence ↗ https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

Module 40 / 40 · 90 min

40. Revue avancée et dossier de preuves HTTP

Transformer une liste de contrôles en démonstration limitée et reproductible.

Comprendre le mécanisme

Une revue crédible relie un actif, une menace, une exigence et un test observable. Une liste d'en-têtes présente sur une page ne démontre pas la sécurité de tout un portail. Les parcours authentifiés, les interfaces d'administration, les erreurs, les téléchargements et les intermédiaires peuvent appliquer des politiques différentes. Le périmètre, la version et les conditions de test doivent être explicites pour que le résultat ait un sens.

Raisonner et vérifier

Utilisez les standards et guides publics comme repères, puis documentez les écarts et les contrôles compensatoires effectivement vérifiés. Un examen d'entraînement mesure une sélection de connaissances ; il ne délivre pas une certification de produit ni une assurance exhaustive. Conservez des preuves expurgées, les tests négatifs et la procédure de non-régression. Attribuez à chaque risque résiduel un traitement et un propriétaire plutôt que le masquer dans un score global.

Périmètre

Une preuve doit préciser le périmètre testé. Les routes, identités et états peuvent différer. Indiquez composants, versions, identités et parcours réellement vérifiés. Une capture de page d'accueil ne démontre pas les contrôles d'import, d'API ou de document personnel qui suivent d'autres chemins.

> **Erreur fréquente à comprendre** Une capture d'accueil certifie tout le portail.
>
> Les routes, identités et états peuvent différer.

Exigence observable

Une exigence doit être reliée à un contrôle observable. La présence d'un composant n'est pas une preuve de propriété. Formulez une propriété comme refus sans mutation pour une identité invalide. Reliez-la à un contrôle puis à des assertions de réponse et d'état ; cela rend le résultat interprétable par un réviseur.

> **Erreur fréquente à comprendre** Le nom d'une technologie démontre sa bonne configuration.
>
> La présence d'un composant n'est pas une preuve de propriété.

Tests de frontière

Les tests négatifs vérifient les frontières de refus. Les refus sans effet font partie des garanties. Essayez mauvaise méthode, acteur tiers, entrée malformée et limite dépassée. Ces scénarios vérifient des refus que le parcours nominal ne démontre pas, même lorsque celui-ci est entièrement fonctionnel.

> **Erreur fréquente à comprendre** Seuls les succès métier sont utiles à une revue.
>
> Les refus sans effet font partie des garanties.

Risque résiduel

Un risque résiduel doit rester visible et attribué. Le score ne remplace pas le traitement des limites. Une frontière non couverte doit rester déclarée avec propriétaire et traitement prévu. Un dossier honnête permet une décision ; un score global sans lacunes visibles peut masquer les risques les plus sensibles.

> **Erreur fréquente à comprendre** Un score global permet d'effacer tout risque non testé.
>
> Le score ne remplace pas le traitement des limites.

Portée du QCM

Un QCM mesure des connaissances sélectionnées. L'implémentation et son environnement demandent une vérification. Le QCM évalue des connaissances sélectionnées dans un contexte pédagogique. Il ne certifie pas le portail déployé ni toutes ses dépendances ; les preuves d'implémentation et d'exploitation restent nécessaires.

> **Erreur fréquente à comprendre** Un QCM réussi certifie la sécurité du service réel.
>
> L'implémentation et son environnement demandent une vérification.

Versions

Les versions doivent accompagner les résultats. Une modification peut changer le comportement. Conservez version d'application, runtime, proxy et configuration avec les résultats. Après une modification de trajet ou de bibliothèque, identifiez quelles preuves doivent être rejouées au lieu de réutiliser tout le dossier sans revue.

> **Erreur fréquente à comprendre** Un résultat reste valable pour toute version future.
>
> Une modification peut changer le comportement.

Reproductibilité

Les preuves doivent être reproductibles et expurgées. Le secret n'est pas nécessaire à la compréhension du résultat. Fournissez étapes, données factices et résultat attendu sans credentials réels. Un autre ingénieur doit pouvoir reproduire le contrôle sur un environnement autorisé et comprendre les limites de l'expérience.

> **Erreur fréquente à comprendre** Publier les credentials rend le test plus professionnel.
>
> Le secret n'est pas nécessaire à la compréhension du résultat.

Non-régression

Les contrôles stables peuvent devenir des tests de non-régression. Les propriétés doivent être maintenues dans le temps. Transformez les propriétés stables en tests adaptés, notamment rejet avant effet et isolation entre acteurs. Évitez les assertions qui copient seulement l'implémentation sans vérifier le contrat métier.

> **Erreur fréquente à comprendre** Une revue manuelle dispense de vérifier toute évolution.
>
> Les propriétés doivent être maintenues dans le temps.

Dépendances

Les dépendances font partie du périmètre décrit. Le parcours dépend de plusieurs couches et services. Incluez les composants qui prennent réellement les décisions : CDN, proxy, store de session et fournisseur externe. Une propriété du système peut échouer dans un composant absent du rapport applicatif.

> **Erreur fréquente à comprendre** L'application n'est affectée par aucun composant externe.
>
> Le parcours dépend de plusieurs couches et services.

Acceptation

Le responsable doit connaître les conditions d'acceptation. Une assurance doit annoncer ce qu'elle couvre réellement. Le responsable doit connaître critères, preuves manquantes et délai de traitement des risques. Un résultat technique n'a de sens opérationnel que si la propriété et les conditions de son acceptation sont explicites.

> **Erreur fréquente à comprendre** Une limite non documentée équivaut à une garantie.
>
> Une assurance doit annoncer ce qu'elle couvre réellement.

Atelier guidé : lire, expliquer et vérifier le code

```
Test : token invalide
Réponse : 401
Observation base : une commande a pourtant été créée
Exigence : refus sans mutation
```

La réponse 401 est accompagnée d'une commande créée en base. Le test échoue donc à l'exigence de refus sans mutation, malgré le statut attendu. Corrigez l'ordre des contrôles puis vérifiez à nouveau base, messages et appels aval avant de considérer la frontière validée.

Une connaissance évaluée ne change pas l'état du système. Les preuves doivent couvrir le résultat métier, pas seulement la réponse. L'effet observé viole l'exigence. L'écart doit rester visible et être corrigé.

Travail pratique : Réalisez la revue du portail de synthèse en réutilisant les tests des trente-neuf pages précédentes. Choisissez aussi deux limites non couvertes.

Critères de correction : Présentez les preuves sur les parcours retenus, les refus sans effet et les critères de reprise. Documentez les limites, versions et dépendances. Faites relire les hypothèses puis transformez les contrôles stables en tests de non-régression.

Carte à capture manuelle : autorisation, capture et expiration

L'autorisation et la capture sont deux étapes distinctes dans le flux manuel posé. Une réponse d'autorisation peut réserver des fonds sans qu'un ordre de capture ait encore été exécuté. Le backend doit conserver cet état intermédiaire et ne pas annoncer une capture finale.

Modélisez au moins la référence prestataire, le montant autorisé, les transitions permises et l'échéance applicable. La fenêtre dépend du moyen et du contrat ; évitez une constante universelle déduite d'un exemple. À l'expiration ou au refus ultérieur, le système doit réconcilier son état plutôt que conserver indéfiniment un succès supposé.

Testez autorisation réussie sans capture, capture confirmée, expiration et notification répétée avec le simulateur du fournisseur. Les effets métier, comme livraison ou facture définitive, doivent suivre l'état exigé par votre contrat. Une réponse HTTP reçue et une authentification client réussie ne remplacent pas la preuve de la transition de paiement correspondante.

Points de vigilance : Une autorisation ne constitue pas un règlement définitif au marchand. Le cycle de vie ne s'arrête pas à l'autorisation ; les délais et capacités varient selon le moyen de paiement et le PSP. Un remboursement restitue

des fonds ; il ne remplace pas une autorisation. Une autorisation peut expirer et une capture peut échouer.

Documentation de référence ↗

Exemple commenté

```
Actif → menace → exigence → contrôle → test → résultat → risque résiduel
Version et périmètre du laboratoire joints au dossier.
```

Cette chaîne rend une décision vérifiable et révisable. Elle évite d'assimiler un score de QCM à une validation complète du système réel.

Exercice de mise en pratique

Réalisez la revue du portail de synthèse en réutilisant les tests des trente-neuf pages précédentes. Choisissez aussi deux limites non couvertes.

Voir une proposition de correction

Présentez les preuves sur les parcours retenus, les refus sans effet et les critères de reprise. Documentez les limites, versions et dépendances. Faites relire les hypothèses puis transformez les contrôles stables en tests de non-régression.

- Périmètre explicite.
- Preuves positives et négatives.
- Risques résiduels visibles.

Contrôle de connaissances

Un score élevé au QCM certifie-t-il la sécurité du portail déployé ?

> Réponse expliquée : Non, une vérification du système et de son périmètre reste nécessaire — L'évaluation de connaissances n'est pas un audit exhaustif.

Cours associé : 40. Revue avancée et dossier de preuves HTTP ↗

Documentation de référence ↗ https://owasp.org/www-project-web-security-testing-guide/

Projet de synthèse — Revue HTTP d’un portail de réservation

Sur un portail de laboratoire, documentez le trajet navigateur, proxy, application et dépendances. Produisez une politique HTTP cohérente et les tests qui démontrent ses limites.

- Matrice méthodes, ressources, acteurs et statuts attendus.
- Cookies, CORS, CSRF, CSP et caches validés sur des parcours authentifiés et anonymes.
- Confiance proxy explicite, limites d’upload et budgets de temps mesurés.
- Journaux minimisés, scénarios de reprise et preuves de non-régression.
- Chaque décision renvoie à la menace, au contrôle, au test et au propriétaire.

Voir une proposition de correction

Commencez par un inventaire des actifs et une capture de parcours nominal. Séparez politiques navigateur et contrôles serveur ; vérifiez ensuite les chemins de refus. Testez le cache avec deux identités, les limites à chaque couche et les dépendances simulées. Conservez configuration, version testée, résultats expurgés et critères d’acceptation ; corrigez les écarts avant d’annoncer une couverture.

Examen d’entraînement indépendant. Le nombre de questions, la durée et le seuil sont des choix pédagogiques DIGITAL AVENIR, sans équivalence avec un barème officiel. Les réponses sont présentes dans la page pour l’autoformation ; aucun titre de certification n’est délivré.

Accéder à l’examen blanc