Projet Cybersécurité

Enseignants : Alexandre BEZE, Quentin CHASSEL

Repo GitHub (fork) : https://github.com/gabrielle-manga/cracks.git

RAPPORT DE PENTEST

MANGA ABANDA Ange Gabrielle

AKOUTEY Armelle Deo-Gratias

FISA 4A CYBERSECURITE

POLYTECH DIJON

TABLE DES MATIERES

SOMMAIRE

6.6 VULN-06 – Divulgation de token via rst.php
6.7 VULN-07 – Politique de mot de passe faible et absence de lockout

7. Correctifs et gestion des patchs

8. RED TEAM VS BLUE TEAM

(Cette section décrit uniquement les correctifs réellement implémentés et déployés)

7.1 Synthèse des patchs appliqués

- **Tableau : ID vulnérabilité → branche Git → commit → statut**

7.2 Détail des correctifs par vulnérabilité

7.2.1 Correctif VULN-01 – Sécurisation des identifiants admin
7.2.2 Correctif VULN-02 – Protection contre l’usurpation et le XSS
7.2.3 Correctif VULN-03 – Suppression de l’exposition des tokens
7.2.4 Correctif VULN-07 – Politique de mot de passe et limitation des tentatives
7.2.5 Correctif VULN-05 – Protection de la base de données SQLite

(Objectif, patch appliqué, fichiers modifiés, tests et résultats)

Résumé exécutif

Ce rapport présente l'audit de sécurité mené sur l'application **web cracks**, un outil fictif destiné au partage et au vote de « blagues foireuses ». Ce travail a été réalisé dans le cadre du module **Projet Cybersécurité** à Polytech Dijon, par le binôme MANGA ABANDA Ange Gabrielle et AKOUTEY Armelle Déo-Gratias.

L'objectif est de conduire un **test d'intrusion ciblé** afin d'identifier les vulnérabilités majeures de l'application, d'en évaluer l'impact potentiel et de proposer des corrections adaptées. Pour ce faire, nous avons adopté une méthodologie inspirée de **PTES** et de l'**OWASP Testing Guide**, articulée autour des étapes suivantes : cadrage et mise en place de l'environnement de test, reconnaissance et

cartographie des surfaces d'attaque, exploitation des vulnérabilités, application de correctifs, puis échanges inter-équipes dans le cadre du **Red vs Blue**.

L'audit a permis de mettre en évidence plusieurs failles de sécurité, principalement liées à l'authentification, au contrôle des accès et à la validation des entrées utilisateurs. Chaque vulnérabilité a été documentée avec ses conditions de reproduction, ses preuves et son impact. Les correctifs correspondants ont ensuite été proposés, implémentés et validés par des tests unitaires et manuels.

1. Introduction

Le projet consiste à auditer l'application web cracks, déployée dans un environnement de test dédié. L'audit repose sur une approche combinant la méthodologie PTES et le guide OWASP Testing, afin de cartographier les surfaces d'attaque, identifier les vulnérabilités, démontrer leur exploitation et valider les correctifs appliqués. Le rapport s'adresse à la fois à un public technique, qui trouvera le détail des failles et preuves, et à un public non technique, qui pourra s'appuyer sur les synthèses et recommandations de sécurité.

1.1 Objet du rapport & objectifs

L'objet du rapport est de documenter l'ensemble du processus d'audit et de correction appliqué à l'application cracks, en suivant une démarche rigoureuse inspirée des standards de tests d'intrusion.
Les objectifs poursuivis sont :

- Identifier les vulnérabilités présentes dans l'application (authentification, gestion des sessions, validation des entrées, contrôle des accès, exposition de données sensibles).
- Évaluer leur impact sur les trois piliers de la sécurité (confidentialité, intégrité, disponibilité).
- Reproduire et prouver les exploitations à l'aide de scénarios concrets et reproductibles.
- Proposer et implémenter des correctifs adaptés et conformes aux bonnes pratiques.
- Valider l'efficacité des patchs par des tests techniques (unitaires et manuels).
- Partager les résultats dans une logique collaborative, notamment à travers la phase Red vs Blue avec les autres équipes.

1.2 Acteurs et binôme

L'audit est réalisé par le binôme MANGA ABANDA Ange Gabrielle et AKOUTEY Armelle Déo-Gratias, étudiantes à Polytech Dijon dans le cadre du module Projet Cybersécurité.

1.3 Commit / branche / version testée

L'audit de sécurité s'appuie sur un fork du dépôt GitHub d'origine cracks, mis à disposition par les enseignants du module.
Le fork de référence utilisé pour ce projet est accessible publiquement à l'adresse suivante : https://github.com/esirem-chassel/cracks
Le projet est installé et exécuté localement conformément à la procédure fournie qui décrit notamment la configuration d'un environnement Apache2, PHP et SQLite sur l'URL locale http://cracks.local/

L'analyse se concentre sur la branche principale (main), à partir du commit initial du projet correspondant à l'état d'installation de base. Les correctifs appliqués au cours de l'audit sont ensuite enregistrés et suivis à travers des pull requests et des commits ultérieurs, chacun étant associé à l'ID du finding concerné.
Ce choix garantit une traçabilité complète entre les vulnérabilités identifiées, leur reproduction et les patchs proposés, en s'appuyant directement sur l'historique du dépôt GitHub.

2. Contexte & périmètre

L'audit de sécurité se déroule dans le cadre académique du module Projet Cybersécurité. L'application cracks est un projet fictif conçu spécifiquement pour l'apprentissage pratique du pentesting et de la sécurisation d'applications web. Le contexte de l'audit est donc contrôlé : l'application est déployée sur un environnement de test dédié et ne contient pas de données réelles.

L'objectif est de simuler les conditions d'un test d'intrusion en environnement réel, tout en restant dans un cadre sécurisé et maîtrisé. Pour ce faire, les auditeurs disposent du code source complet de l'application (approche boîte blanche) et d'un environnement reproductible (serveur Apache2, base SQLite, configuration locale). Le périmètre couvre les fonctionnalités principales accessibles via l'interface web, sans extension à d'autres environnements système ou réseau.

2.1 Présentation succincte de l'application cracks

L'application cracks est une plateforme web de type communautaire. Ses principales fonctionnalités sont :

- Inscription et authentification des utilisateurs
- Publication de contenus (ajout de « cracks »)
- Interaction via un système de votes
- Consultation publique du contenu publié

Le modèle repose sur une gestion simple des utilisateurs et d'une base de données SQLite. Les routes principales comprennent l'inscription (/register), la connexion (/login), la soumission de contenu (/submit), la consultation publique (/view), ainsi que la gestion des votes.

2.2 Environnements de test (OS, versions, DB, etc.)

L'audit est réalisé dans un environnement local configuré spécifiquement pour ce projet. Les principales caractéristiques techniques sont les suivantes :

- Système d'exploitation : Raspberry Pi OS / Ubuntu (selon poste de TP)
- Serveur web : Apache2
- Langage serveur : PHP (avec support SQLite3)
- Base de données : SQLite3 initialisée via init.sql
- Nom de domaine local : cracks.local déclaré dans /etc/hosts
- Dépôt de référence : https://github.com/gabrielle-manga/cracks/issues

La configuration est mise en place conformément à la procédure d'installation fournie.

2.3 Portée : fonctionnalités et endpoints testés

L'audit se concentre exclusivement sur l'application cracks dans son périmètre web. Les tests incluent :

- Authentification : robustesse des mécanismes de connexion et de gestion des sessions.
- Gestion des utilisateurs : inscription, stockage des identifiants, droits d'accès.
- Validation des entrées : publication de cracks, soumission de formulaires, votes.
- Affichage public : consultation des cracks et gestion des accès non authentifiés.
- Interactions avec la base de données : requêtes et traitement des données utilisateurs.

Ne sont pas inclus dans le périmètre :

- L'infrastructure réseau sous-jacente (autres services du serveur hôte).
- Les attaques de type DoS/DDoS.
- Les environnements de production externes (l'application est isolée en local).

Le périmètre vise donc à couvrir les fonctionnalités principales accessibles via le navigateur, en reproduisant le point de vue d'un attaquant ciblant directement l'application web.

3. Méthodologie

3.1 Cadre de travail (PTES / OWASP adapté)

Le projet s'appuie sur un cadre méthodologique hybride, combinant les standards **PTES** (Penetration Testing Execution Standard) et **OWASP** Testing Guide, adaptés au contexte pédagogique et au périmètre restreint de l'application cracks.

La méthodologie **PTES** fournit une trame complète pour organiser un test d'intrusion en **sept phases** : pré-engagement, collecte d'informations, modélisation des menaces, analyse des vulnérabilités, exploitation, post-exploitation et reporting. Elle sert ici de structure générale pour planifier et documenter l'audit.

La méthodologie **OWASP** est utilisée en complément, car elle cible spécifiquement les applications web et propose un ensemble de tests systématiques (authentification, autorisation, gestion des sessions, validation des entrées, injections, logique métier, etc.). Elle permet de détailler concrètement les vérifications à mener sur les endpoints et les fonctionnalités de l'application.
Dans le cadre de ce projet :

- PTES définit l'ossature du rapport et la progression logique de l'audit.
- OWASP guide les tests techniques sur l'application web, notamment l'exploration des failles connues (ex. SQLi, XSS, mauvaise gestion des sessions, contrôle d'accès défaillant).

3.2 Outils et scripts autorisés / interdits

L'audit de sécurité est réalisé dans un cadre pédagogique strict, avec des règles précises concernant l'usage des outils et scripts. Ces règles visent à garantir l'équité entre les binômes, à éviter une automatisation excessive et à privilégier l'apprentissage des démarches manuelles de test.

Outils autorisés

- Outils systèmes de base : curl, wget, telnet, navigateurs web avec consoles développeur intégrées.
- Analyseur de requêtes HTTP : utilisation d'un proxy comme Burp Suite (édition communautaire) ou OWASP ZAP, limités aux fonctions de proxy et d'interception pour analyser et manipuler les requêtes.
- Scripts simples faits maison : scripts en Python, Bash ou PHP écrits par les étudiants eux-mêmes, dans le but d'automatiser ponctuellement des tests ou de reproduire une vulnérabilité spécifique.

- SQLite3 en ligne de commande : pour interagir avec la base de données locale et comprendre les effets d'une injection ou d'une mauvaise validation des entrées.

Outils interdits

- Suites complètes de pentesting telles que Metasploit, Nessus, OpenVAS ou SQLMap, car elles automatisent une partie trop importante du travail.
- Scanners massifs de vulnérabilités ou d'énumération, qui ne correspondent pas au périmètre restreint du projet.
- Scripts externes non justifiés ou copiés depuis des ressources en ligne, afin de garantir que l'ensemble des reproductions et exploitations soit compris et maîtrisé par le binôme.

Justification pédagogique

Le cadre impose de privilégier une approche manuelle et compréhensive des tests, afin de renforcer la capacité à analyser le code source, à identifier les vulnérabilités et à développer des correctifs pertinents. L'usage d'outils automatisés lourds est volontairement exclu pour obliger à raisonner sur la logique applicative et non uniquement sur la détection brute de failles.

4. Phase de reconnaissance & cartographie

4.1 Découverte des endpoints & surfaces d'attaque

Le but de cette première phrase de pénétration est d'inventorier par inspection manuelle les endpoints accessibles et les fichiers potentiellement exposés:

- */*
- */?inc=login (formulaire de connexion)*
- */rst.php (page / formulaire d'administration / reset)*
- */?inc=rst (page de réinitialisation avec id et code)*
- */admin*
- *Fichiers potentiellement accessibles: /init.sql, /config.php, fichiers .db ou .sqlite, /inc/**

Fichiers clés trouvés dans le code:

- Init.sql: compte admin exposé
- rst.php / inc/rst.php: génération & affichage du lien de réinitialisation de compte
- Auth.php: gestion des sessions et récupération pwd
- Inc/login.php: point d'entrée login

4.2 Analyse statique rapide du code (fichiers sensibles)

4.3 Inventaire des paramètres et inputs utilisateurs

Nous avons recensé les paramètres visibles par endpoint, ainsi que les différents risques observés:

Endpoint	Méthode	Paramètres observés	Risque observé
/rst.php	POST / HTML form	mdp, login, valid	Divulgation de token / lien de reset renvoyé
/?inc=rst	GET	id, code	Acceptation du token pour réinitialisation
/?inc=login	POST	username, password	Absence de lockout
/comments ou endpoint de contenu (si présent)	POST	content	XSS possible
init.sql	/	login, pwd	Credentials par défaut stockés en dur dans le code

5. Résultats de l'audit — synthèse

5.1 Méthodologie de classification et niveaux de criticité

L'évaluation du niveau de criticité de chaque faille repose sur la méthodologie CVSS v3.1 (Common Vulnerability Scoring System) et sur le Risk Rating OWASP.

Chaque vulnérabilité est donc classée en fonction de la probabilité d'exploitation ainsi que de l'impact.
Pour mieux nous y référer voici un tableau récapitulatif de la classification des failles, qui nous servira pour la suite :

Niveau de criticité	Score CVSS (approx.)	Description	Priorité de traitement
Critique (Critical)	9.0 – 10.0	Exploitable à distance, sans authentification, peut entrainer une compromission totale	P0 — à corriger immédiatement
Élevé (High)	7.0 – 8.9	Permet d'obtenir un accès privilégié, d'altérer des données sensibles ou de contourner la sécurité	P1 — à corriger dans les plus brefs délais, sous 48 h
Moyen (Medium)	4.0 – 6.9	Exploitable avec l'interaction utilisateur ou avec des conditions spécifiques	P2 — à corriger lors d'une maintenance ultérieure
Faible (Low)	0.1 – 3.9	Faible impact	P3 — à corriger lors d'une mise à jour
Informationnel (Info)	0.0	Observation ou bonne pratique à tenir sans risque direct	/

5.2 Tableau récapitulatif des findings

ID	Issue (Réf)	Titre	Gravité	Résumé	Impact
VULN-01	*#1*	Identifiants administrateur par défaut dans *init.sql* (admin:admin)	Elevé - P1	Présence d'identifiants par défaut	accès administrateur immédiat et possibilité de compromission totale
VULN-02	*#2*	XXS + Contournement PHPSESSID	Elevé - P1	XSS + possible publication/modification en tant qu'admin sans session valide	usurpation d'identité et actions malveillantes
VULN-03	*#3*	Exposition des tokens de réinitialisation via rst.php	Elevé - P1	Tokens de réinitialisation exposés	forte possibilité pour un attaquant de réinitialiser le mot de passe d'un utilisateur ciblé sans inscription, sans possession d'email, ni preuve d'identité > prise de contrôle des comptes
VULN-04	*#4*	XSS — Injection — Session Hijack	Elevé - P1	Reflet / persistant XSS permettant vol de cookies/session	usurpation de compte administrateur et compromission de l'application entière

VULN-05	*#5*	SQLite Exposure – Admin Hash Crack	Elevé - P1	Base SQLite accessible et exportable contenant les hash admin	craquage du mot de passe (online & offline) admin si le hash faible. Risque d'accès administrateur.
VULN-06	*#6*	Token disclosure (PoC : curl)	Elevé - P1	`rst.php` renvoie en clair un lien de réinitialisation (token) lorsqu'on poste des paramètres attendus	Réinitialisation du mot de passe d'un compte (ex. `admin`) et prise de contrôle du compte sans authentification. compromission de la confidentialité et de l'intégrité des comptes
VULN-07	*#7*	Politique de mot de passe faible et absence de verrouillage sur /?inc=login	Moyen - P2	Pas de verrouillage/anti-brute force et règles de password faibles	attaques facilitées par force brute / credential stuffing + accès aux données sensibles

6. Détails techniques — findings (template par faille)

Pour les détails techniques, voir Fork Ange Gabrielle MANGA x Armelle AKOUTEY > Issues

7. Corrections et gestion des patchs

7.1 Synthèse des corrections appliquées

ID Vulnérabilité	Branche Git / Commit	Statut
VULN-01	fix/vuln-01: Changement admin:admin & Hash pwds	Déployé
VULN-02	fix/vuln-02: correction propriété du owner + empecher usurpation du champ owner et du XSS stocké	Déployé
VULN-03	fix/vuln-03: suppression du $systemMdp et de l'affichage du lien de réinitialisation	Déployé
VULN-07	Fix/vuln-07: politique de mot de passe et limitation de tentatives de connexion	Déployé
VULN-05	fix-vuln-05: fixe exposure db	Déployé

7.2 Détail par correction

7.2.1 [VULN-01]

- **Réference**
  Branche : fix/vuln-01
  Commit : fix/vul-01: Changement admin:admin & Hash pwds
  Date de correction : 04/12/2025

- **Objectif du patch**
  - Supprimer les identifiants par défaut admin:admin présents dans init.sql
  - Remplacer le hachage MD5 qui est faible par un mécanisme plus sécurisé (bcrypt)
  - Dans Auth.php Mettre à jour l'algorithme d'authentification pour le mot de passe via un algorithme > *password_verify()*
  - Mise à jour d'un mot de passe fort généré, autre que "admin" dans la base de données

- **Patch appliqué**
  - Modif MAJ Remplacement du hash bcrypt du mot de passe admin dans init.sql

```
┌──(kali㉿kali)-[/var/www/html/cracks]
└─$ php gen_hash.php

$2y$12$./f/liDL8uL.Fxo9rrnTb.H6Wn9bTcY.PDeU63JPdP51kQMXjMxbW
```

```
insert into users (login, pwd, isadmin)
values('admin', '$2y$12$FJUCljgyY9fgU76SMSGNjusk1O.hNB/nYImERukyScCd09qeUmRsO', 1);
```

  - Auth.php: Nous avons supprimé l'utilisation de la fonction md5() pour la génération des mots de passe, car aujourd'hui elle est obsolète et vulnérable aux attaques par bruteforce. On a donc switché ce système par password_hash() pour la création et le stockage des mots de passe, et password_verify() pour leur vérification. Nous avons aussi modifié les fonctions subscribe(), tryLog() et resetPwd() du fichier avec des méthodes plus sécurisés via bcrypt.

- **Fichiers modifiés**

Init.sql
Auth.php
Base de données sqlite pour la modification du mot de passe admin

- **Résultats**
  - Il n'y a plus de mot de passe affiché en clair dans le code
  - Authentification réussie et verifiée avec password_verify

7.2.2 [VULN-02]

- **Réference**
  Branche : fix/vuln-02
  Commit : Correction propriété du owner + empecher usurpation du champ owner et du XSS stocké
  Date de correction : 04/12/2025

- **Objectif du patch**
  - Empecher un attaquant de faire de l'usurpation d'un utilisateur, et à l'occurence de l'admin en envoyant un owner=1 via POST inc/add.php
  - Préserver le fonctionnement normal du site : les utilisateurs sans comptes peuvent également poster des cracks
  - Supprimer la possibilité de faire une injection js en postant un crack
  - Sécuriser le site web de telle façon à ce que le owner ne soit jamais défini par l'utilisateur mais par le serveur; et aussi le contenu doit être affiché via un **htmlspecialchars()**

- **Patch appliqué**
  - Suppression du champ "owner" côté client dans inc/add.php

```
4           $content = $_REQUEST['content'] ?? '';
5           $owner = !empty($_SESSION['userid']) ? $_SESSION['userid'] : 0;
6
7           $q = 'insert into cracks (content, owner, datesend) '
8                   . ' values("'.nl2br($content).'", "'.$owner.'", '.time().')';
9           $db->query($q);
```

- Suppression du champ “hidden” côté html dans inc/add.php

```
23                          required="required"></textarea>
24                  <input type="submit" name="val" value="Ajouter ce crack" />
25              </p>
```

- Suppression de la possibilité de faire du XSS dans config.php

```
27          echo '<p>'.Markdown::_(htmlspecialchars($crack['content'], ENT_QUOTES, 'UTF-8')).'</p>';
```

- **Fichiers modifiés**

  Inc/add.php

  config.php

- **Tests effectués & résultats**
  - Post sans session : le owner intercepté par l’attaquant n’est plus visible et est ignoré, l’usurpation devient donc impossible

Request

Pretty Raw Hex

```
1  POST /?inc=add HTTP/1.1
2  Host: cracks.local
3  Content-Length: 46
4  Cache-Control: max-age=0
5  Accept-Language: fr-FR,fr;q=0.9
6  Origin: http://cracks.local
7  Content-Type: application/x-www-form-urlencoded
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://cracks.local/?inc=add
12 Accept-Encoding: gzip, deflate, br
13 Cookie: PHPSESSID=00e7e9ee2fcd1160f23cc757279d00aa; cid=43
14 Connection: keep-alive
15
16 content=salutanotestburp2&val=Ajouter+ce+crack
```

burpsuite

Accueil Cracks Nouv

Les dix derniers cracks !

salutanotestburp2	salutanotestburp	salut	salut	salut 	hello ano	<script>alert(1)</script> *Envoyé par admin le 2025-12-04 13:23*	salut *Envoyé pa*

- Injection XSS: le texte s'affiche à présent sous sa forme littérale grâce à **htmlspecialchars()**

- Un utilisateur anonyme (qui n'a pas de compte) ne peut poster qu'avec la valeur du owner=0 et jamais défini côté client

7.2.3 [VULN-03]

- **Réference**
  Branche : fix/vuln-03
  Commit : suppression du $systemMdp et de l'affichage du lien de réinitialisation
  Date de correction : 04/12/2025

- **Objectif du patch**
  - Supprimer le mot de passe codé en dur ($systemMdp='pwd1234'), qui permettait à n'importe qui ayant la vue sur le code, de générer un lien de réinitialisation
  - Affiche de la page crack/rst.php
  - Respecter les recommandations OWASP > ne pas afficher les tokens, ni les informations sensibles aux vus et sus de tous

- **Patch appliqué**
  - Suppression complète de $systemMdp et suppression de l'exposition du token

```php
<?php

require_once 'config.php';
session_start();

if(empty($_SESSION['userid'])) {
    echo 'Accès interdit !';
    exit;
}
```

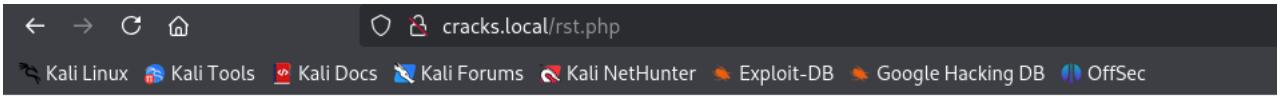
```php
<?php if(!empty($_REQUEST['valid'])) {
    $found = Auth::getInstance()->getCodeFromLogin($_REQUEST['login']);

    echo '<p>Si ce compte existe, un lien de réinitialisation a été généré.</p>';
}
?>
</body>
```

- **Fichiers modifiés**

  rst.php (à la racine)

- **Tests effectués & Résultats**
  - Génération d'un token via crack/rst.php en précisant le login



Création code reset

login | Obtenir le lien

Envoyer ce code à l'utilisateur qui a oublié son mdp

Si ce compte existe, un lien de réinitialisation a été généré.

  - Vérification que le token ait bien été généré

```
sqlite> SELECT id, login, pwd FROM users WHERE login = 'ange';
2|ange|76aed74b74e08ab93257c9fe41da1511
```

  - Réinitialisation du token avec l'url basée comme ça:

http://cracks.local/?inc=rst&id=”préciser l’id”&code=”TOKEN”

7.2.4 [VULN-07]

- **Réference**
  Branche : fix/vuln-07
  Commit : fix/vuln07: politique de mot de passe et limitation de tentatives de connexion
  Date de correction : 11/12/2025

- **Objectif du patch**
  - Renforcer la politique de mot de passe : au moins 8 caractères, au moins un chiffre, lettres MAJ/MIN, caractères spéciaux imposés
  - Introduire une première couche de protection contre les attaques par force brute en limitant les tentatives de connexions excéssives (ne doivent pas dépasser 3 tentatives)
  - Garantir un comportement cohérent avec les bonnes pratiques OWASP (longueur minimale, complexité, lockout après plusieurs échecs).

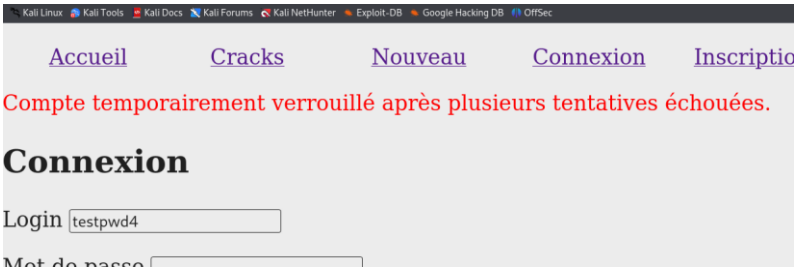
- **Patch appliqué**
  - Politique de mot de passe dans inc/sub.php

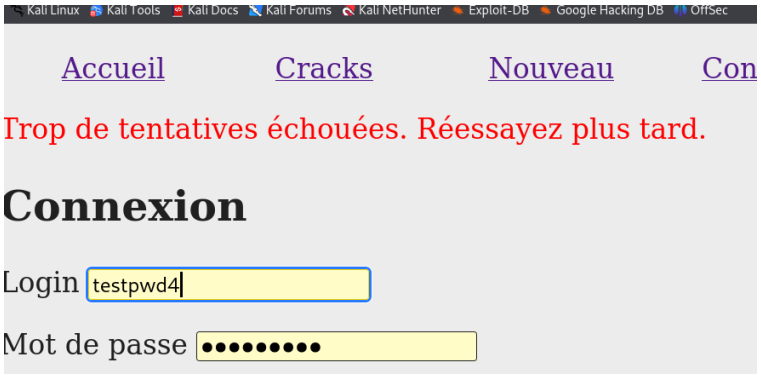
    Validations côté serveur pour refuser les mots de passe ne respectant pas :

    - longueur ≥ 8 caractères
    - présence d’au moins une MAJ/MIN
    - présence d’un chiffre/caractères spécial
    - Les inscriptions ne respectant pas ces critères sont désormais refusées avec un message d’erreur explicite

  - Mise en place d’un système de “lockout” dans inc/login.php

- $_SESSION['login_attempts'] comptabilise les tentatives échouées par login
- Après plus de 3 tentatives échouées (> 3), le compte est temporairement verrouillé pendant 3minutes.
- Cette contrainte de verrouillage réduit le bruteforce et empêche de continuer à tester des mots de passe sans interruption





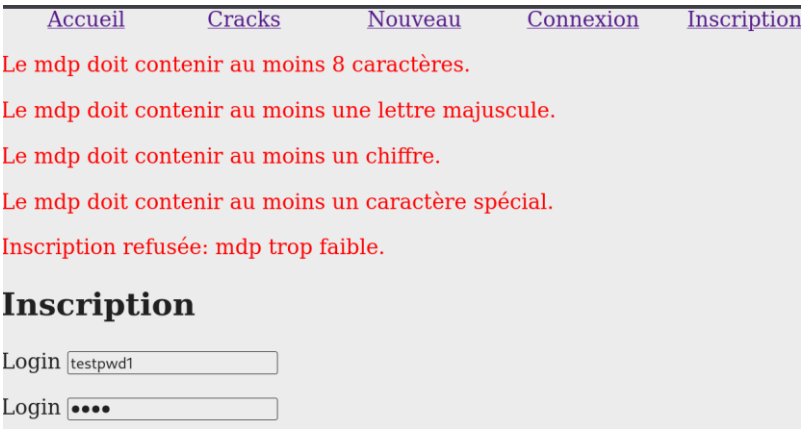
- **Fichiers modifiés**

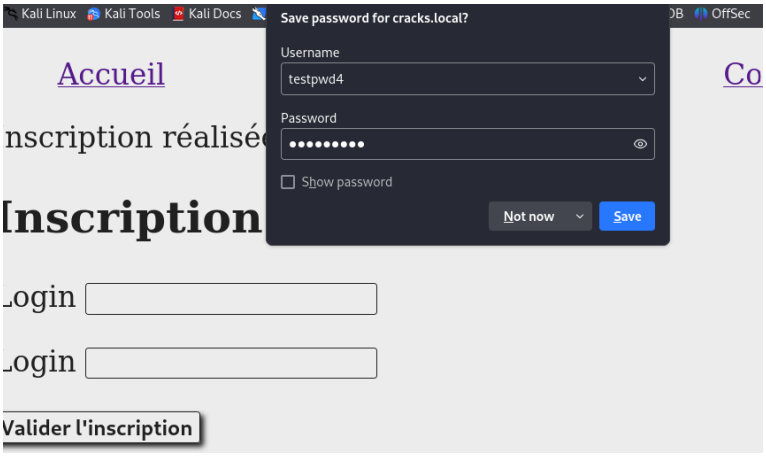
  Inc/sub.php

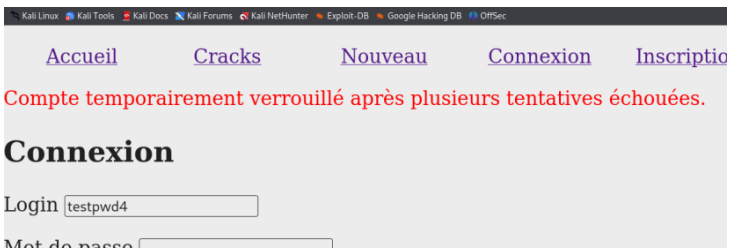
  Inc/login.php

- **Tests effectués & résultats**
  - Test1 : politique de mot de passe

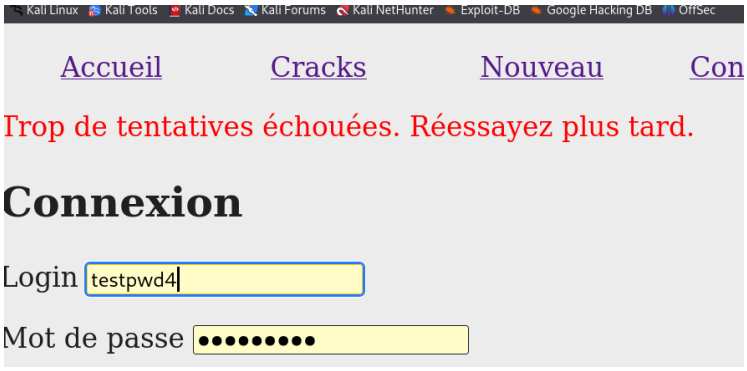
Login	Mdp	Résultat
testpwd1	test	échec
Testpwd2	testpwd2	échec
testpwd3	Testpwd2*	échec
testpwd4	Testpwd4*	Succès





- Test 2 : verification lockout
  -
  -
  - 

  -



7.2.4 [VULN-07]

- **Réference**
  Branche : fix/vuln-05-database-exposure
  Commit : fix/vuln-05-database-exposure
  Date de correction : 11/12/2025

- **Objectif du patch**
  - Empêcher l'accès public au répertoire /database/, qui permettait le téléchargement direct du fichier cracks.db
  - Supprimer toute possibilité d'exfiltration de la base de données
  - Mettre en conformité l'architecture avec les bonnes pratiques OWASP :
    - ne jamais exposer une base de données dans le web
    - limiter l'accès aux fichiers critiques

- **Patch appliqué**
  - Déplacement de la base de données hors du webroot

    La base de données SQLite `cracks.db était` initialement située dans : */var/www/html/cracks/database/cracks.db* et on l'a déplacé vers un répertoire non accessible depuis la base de données */var/www/cracks-data/cracks.db*

  - Modification du DSN SQLite

On a MAJ le chemin dans dans le fichier db.json
Ainsi c'est passé de *"dsn":"sqlite:/var/www/cracks-data/cracks.db"* à *"dsn":"sqlite:/var/www/html/cracks/database/cracks.db"*
L'application continue donc d'accéder à la base de données sans être exposée.

- Blocage de l'accès au répertoire /database/
  Options -Indexes
  Deny from all
  Cela empeche l'accès direct aux ressources.
- Nous avons également attribuer de la propriété du dossier et du fichier à l'utilisateur Apache (www-data).

- **Fichiers modifiés**
  Db.json

- **Tests effectués & résultats**

8. RED TEAMS VS BLUE TEAMS

(voir dans le git > ISSUES)