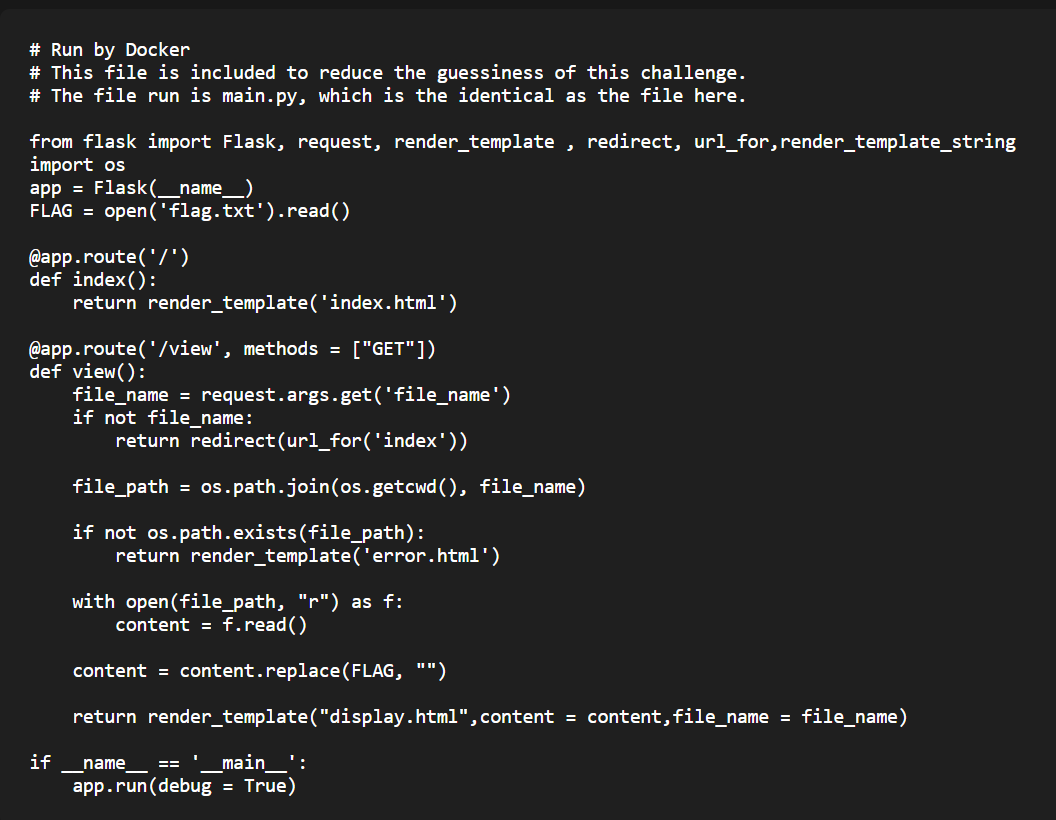
View Source Revenge Write Up

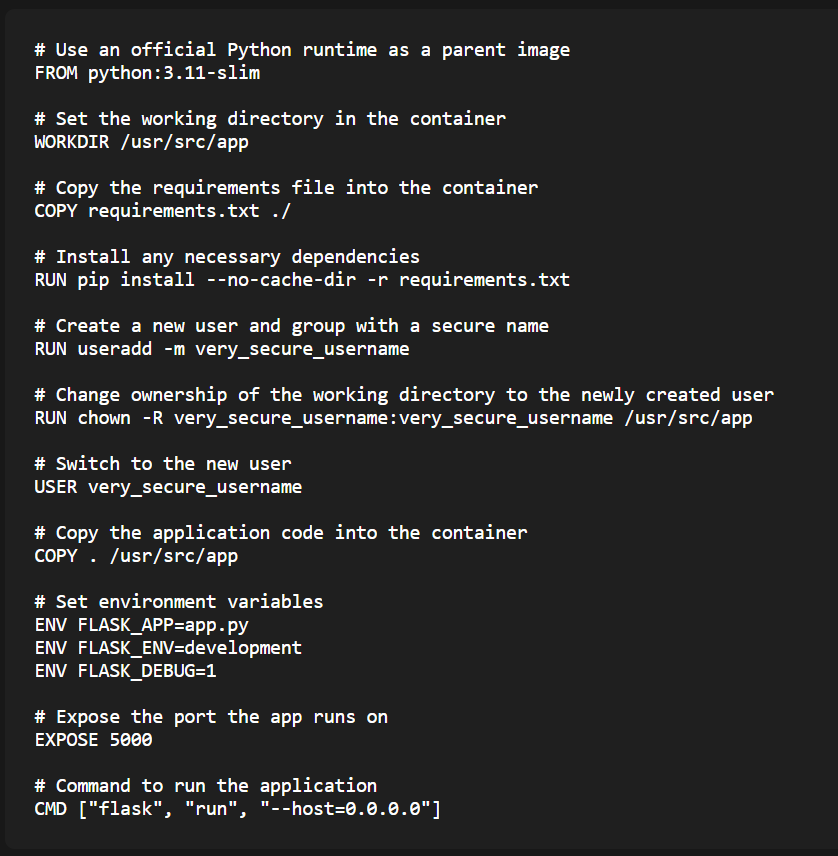
# Path Traversal

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



Dockerfile



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,



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