**Web Exploitation:**

caas:

Started off Web Exploitation looking into caas, which was a cowsay problem. After looking a bit into the cowsay problem, I was trying to find a way to properly exploit the application to reveal the message. At this point, I have really started relying on YouTube quite often, therefore back to the Video Streaming platform it was.

With the help of reading the Wikipedia article and a youtube video on what cowsay was, I realized that I could put in bash commands in the cowsay to see what it would do.

After putting in the input and trying to see what is present in the ls and then cat the files, I found the flag.

Answer: picoCTF{moooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo0o}

References:

<https://en.wikipedia.org/wiki/Cowsay>

<https://www.youtube.com/watch?v=XvbBt8HKWJo>

Forbidden Paths:

On reading the description of the problem, I have to find the flag.txt file in a very specific directory and I have no authorized access to it.

My first instinct was to change the inspect element of the code if possible, to extract the flag. Well, since the code of the HTML is changing every time, an input is set, there is no way for me to inspect and mess with the HTML code. Time to learn what nginx is…

After a bit of digging and realizing that the file path was meant to be read in reverse instead of reading it the opposite way, I realized that to get to the parent directory, I was going to need to specify the entire file path, knowing how the Reader was reading the text now.

After input the file path and not getting a response again, I had to refer another document. After realizing that the ‘..’ meant to go back one directory, then I quickly wrote the text required and then boom, I got the answer.

Answer: picoCTF{7h3\_p47h\_70\_5ucc355\_e5a6fcbc}

References:

<https://gist.github.com/leommoore/2701379>

<https://infosecwriteups.com/hacking-nginx-best-ways-7c576cc17ccc>

Local Authority:

Another challenge to complete. I saw this as a password trial and error problem and realized that it would be useless to keep trying to brute force it. Inspecting and checking the HTML code of the site was again useless, so it was back to the drawing board to figure out how to solve this question.

After trying some attempts, I looked into the JS code for the Log In failed site to see if I was able to crack it via that. Upon realizing that one of the methods was not present in the immediate JS code, I had to go back and see what other trial and error I could do to solve this question.

After messing around a bit, I realized that the code was actually directing me first to another location and then to the main one. After a little bit of digging into the source code, I found the main file, which was the secure.js file. Upon inspecting the username and password required to crack the flag, I quickly input it to get the answer and continue with more challenges.

Answer: picoCTF{j5\_15\_7r4n5p4r3n7\_05df90c8}

References:

<https://stackoverflow.com/questions/2940423/how-to-find-hidden-properties-methods-in-javascript-objects#:~:text=You%20can't%20easily%20do,down%20to%20list%20all%20methods>.

<https://itnext.io/controlling-access-to-data-in-javascript-903e80213168>

<https://www.youtube.com/watch?v=PgYT8FX4Jm8>

where are the robots:

Time to solve another one. And this was way to simple to crack for some reason. I had already dealt with another file which had this file named robots.txt. On just simply trying it out, I got access to another HTML file. Upon inputting that HTML file, I got the flag, which seemed to be way too simple for these types of challenges.

Answer: picoCTF{ca1cu1at1ng\_Mach1n3s\_1bb4c}

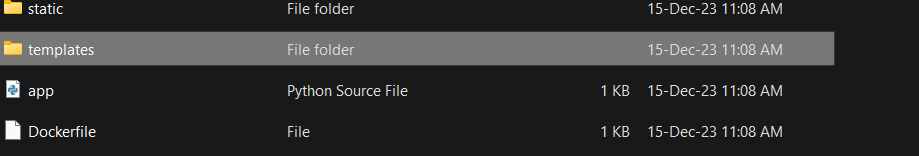
Inspect HTML:



Are you kidding me??

Answer: picoCTF{1n5p3t0r\_0f\_h7ml\_8113f7e2}

notepad:

After downloading the tar file, I was greeted with these input data. On checking the app.py file, I didn’t notice any immediate things that could work with.

After spending some time looking into it, I couldn’t figure out a direction to take this, therefore I resorted to using the internet to solve my issue.

After looking into the Dockerfile for a while, I realized that there was a file that I needed to find that was called flag.txt (which should hopefully contain the file that I require).

Quickly realizing that this was way too hard for me to complete at this point, I left it for the time being, and started working towards another challenge.