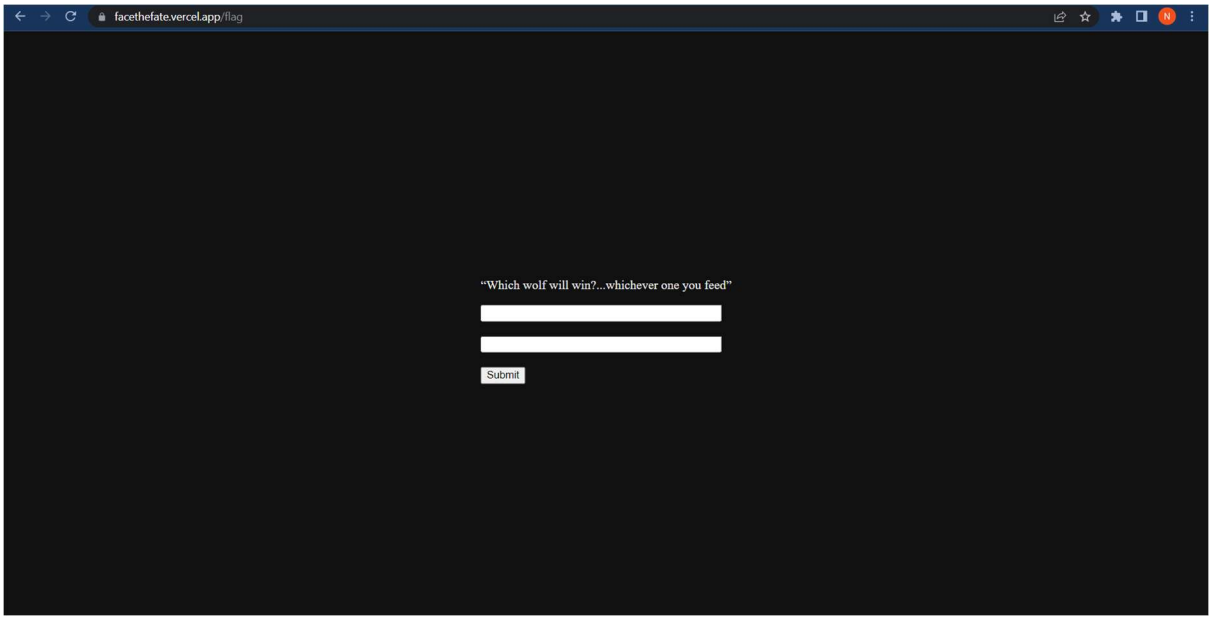
NOVA CTF 2023 WRITE UP

CHALLENGE: FACE THE FATE

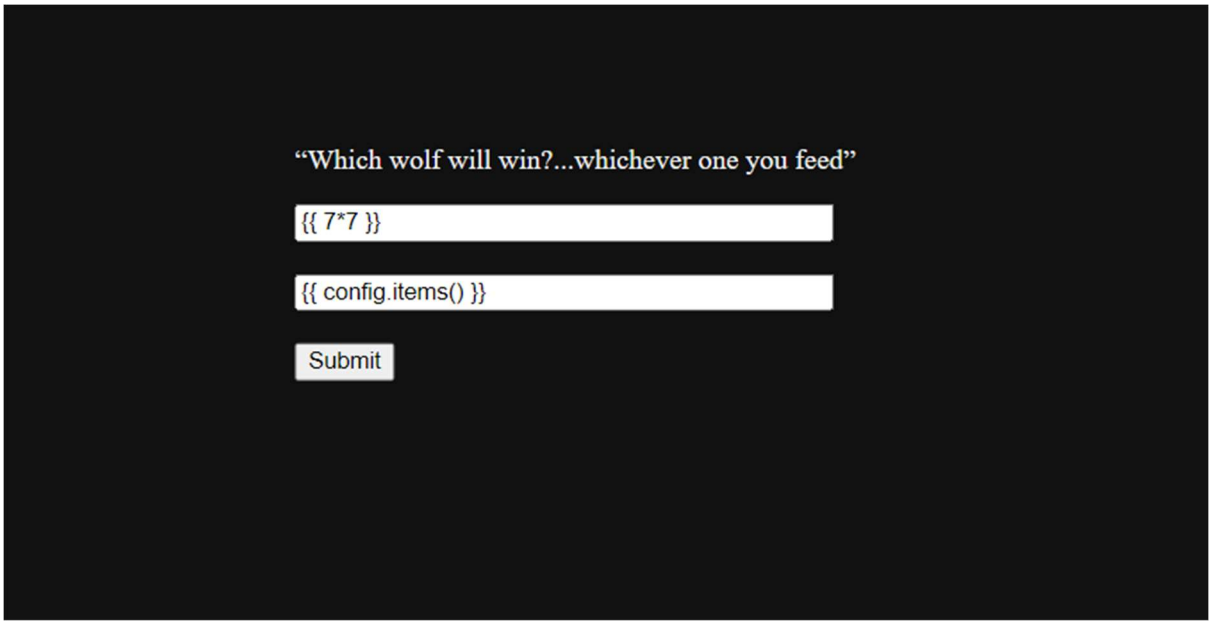
CATEGORY : WEB

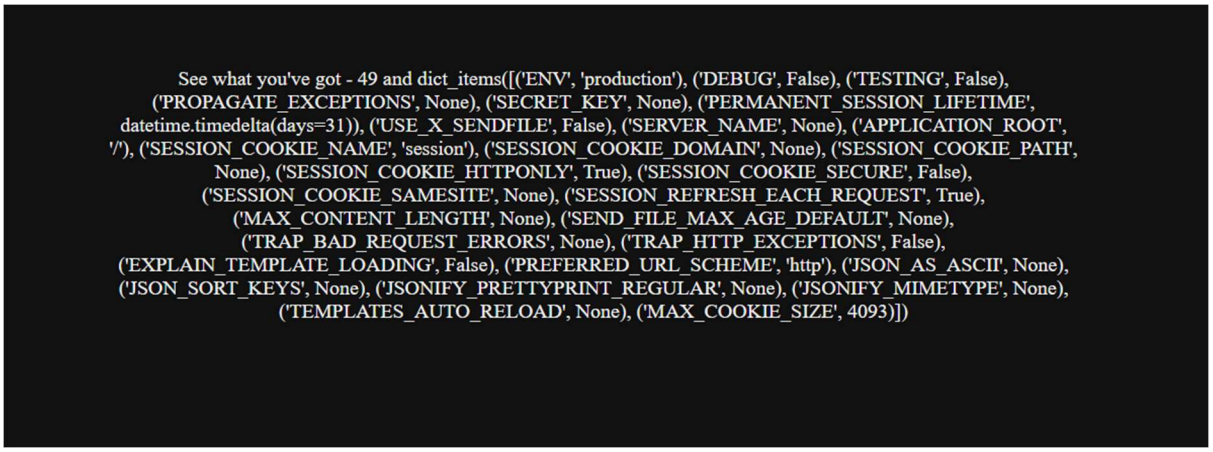
SOLUTION:

When we visit the website, the landing page contains just a background image and greeting message. On brute forcing the URLs of the web application, we can see that **/flag** URL takes to a page it contains two input boxes and a submit button.



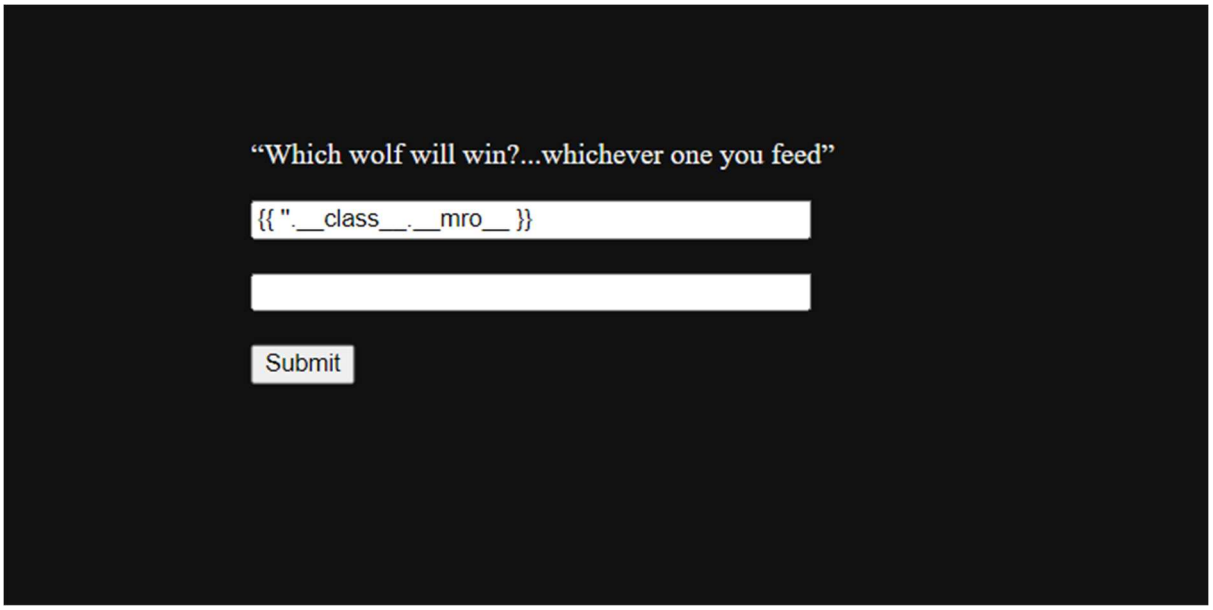
There is no clue in the inspection window. Hmmm….bewildering! On trying out some inputs or by using tools like wappelyzer, we can find that python and jinja are used.

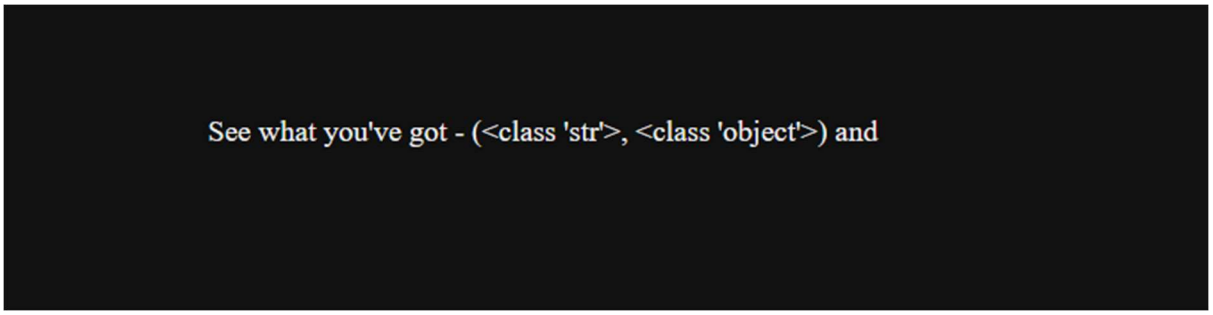




Config object is a dictionary-like object that contains all of the configuration values for the application. In most cases, this includes sensitive values such as database connection strings, credentials to third party services, the SECRET_KEY, etc. Viewing these configuration items is as easy as injecting a payload of *{{ config.items() }}*.

Our initial step is to choose a new-style object for accessing the object base class. To achieve this, we can utilize an empty string ('') object of type str. Subsequently, we can employ the mro attribute to obtain the inherited classes of the object. To exploit the SSTI vulnerability, we can insert the payload *{{ ''.__class__.__mro__ }}*.





To access the root object class, we can use an index of 1 to select the class type object. Once we're at the root object, we can use the subclasses attribute to extract all the classes used in the application. To exploit the SSTI vulnerability, we can insert the payload *{{ ''.__class__.__mro__[1].__subclasses__() }}*

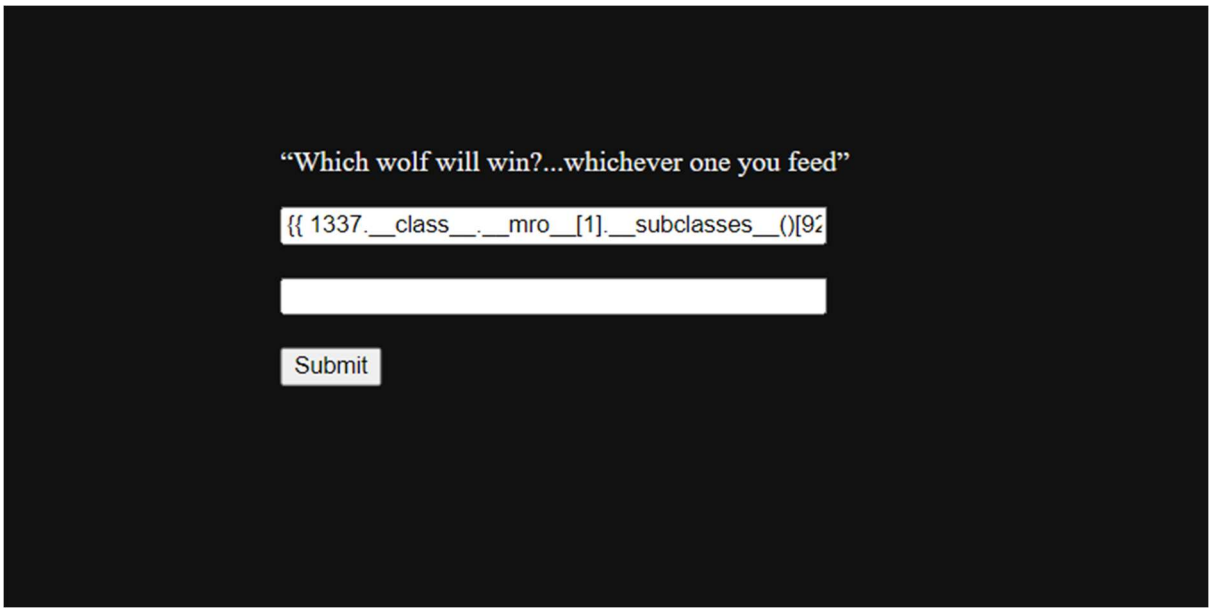
There are distinct sets of classes in various applications meaning that there are no universal methods for exploitation. We must tailor our exploits to the specific classes available, allowing us to craft scenario-specific exploits.

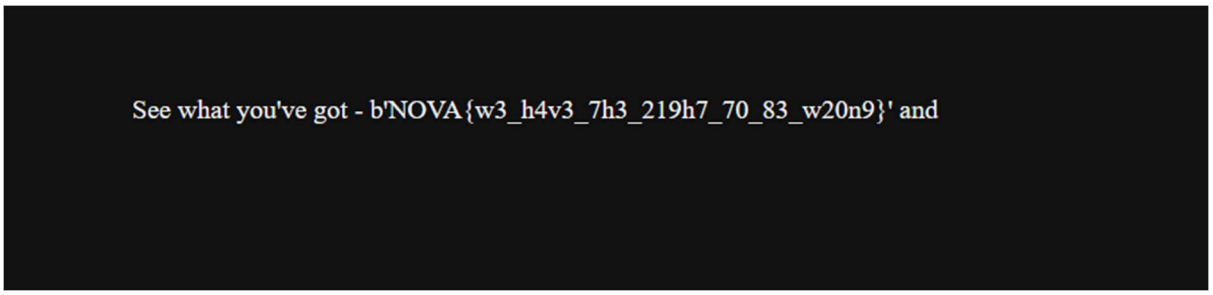
In order to access files on the server, we must first determine the index of the _io._IOBase class. In our case, the class is located at index 92, although this value may differ in other scenarios.

After identifying the _io.IOBase class, we can use the subclasses() method once again to list available classes until we find the index of the _io._RawIOBase class. Finally, we can use this method again to arrive at the _io.FileIO class, which permits us to read files on the server.

Having identified the necessary class, we can now use the _io.FileIO class to read files from the server by creating a file object with the appropriate payload.

{{ 1337.__class__.__mro__[1].__subclasses__()[92].__subclasses__()[0].__subclasses__()[0]('flag/flag.txt').read() }}





"Mission Accomplished Ethan!"