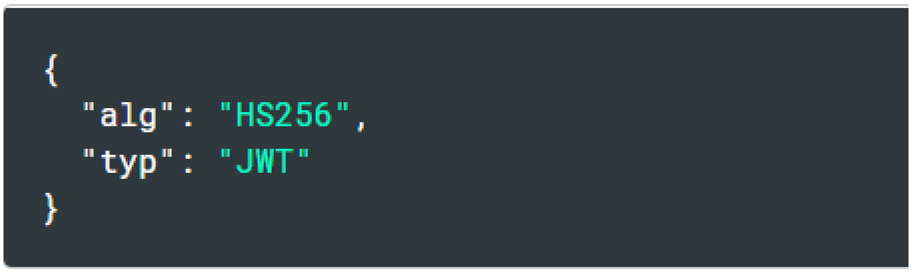
JWT Attack

1. What is JWT ?

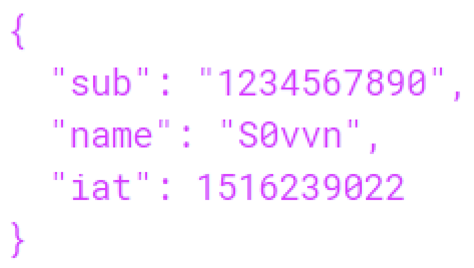
JWT stand for Json Web Token. It is a string that contain a signed data structure (header, payload và signature):

- Header : Contain some information about token , look like this:



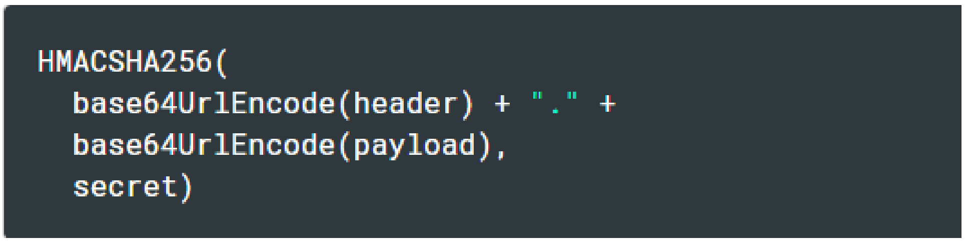
```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Payload : Contain about an entity (typically, the user) and additional data.



```
{
  "sub": "1234567890",
  "name": "S0vvn",
  "iat": 1516239022
}
```

- Signature : Signature is created by combine the header and the payload after encode. For example use HMAC algorithm to encrypt as following:



```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Every information are contained within the jWT token

Stateless server

II. What is JWT attack ?

JWT attack occur when JWT authentication mechanisms fail, enabling malicious actors to craft tokens and impersonate the user of a web application.

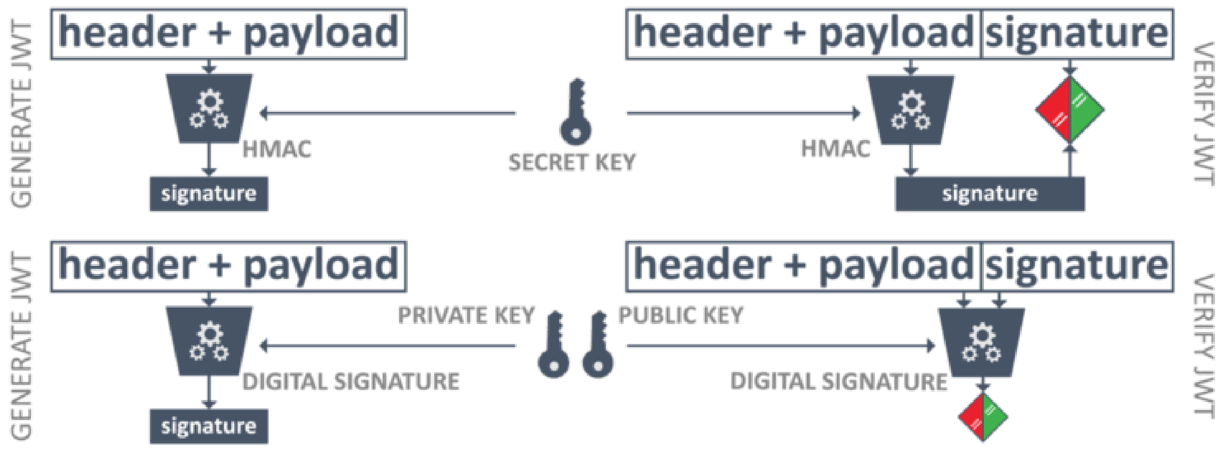
III. Several common attack vector and how to perform it?

1. Sensitive data in the JWT

- Decode the Header and Payload to get the information

2. Change the signing algorithm

- ***None Algorithm Attack***: Change alg into None and delete the signature to check
- ***HMAC Algorithm Attack***: If it use RSA, change the alg into HMAC and use publickey to decrypt

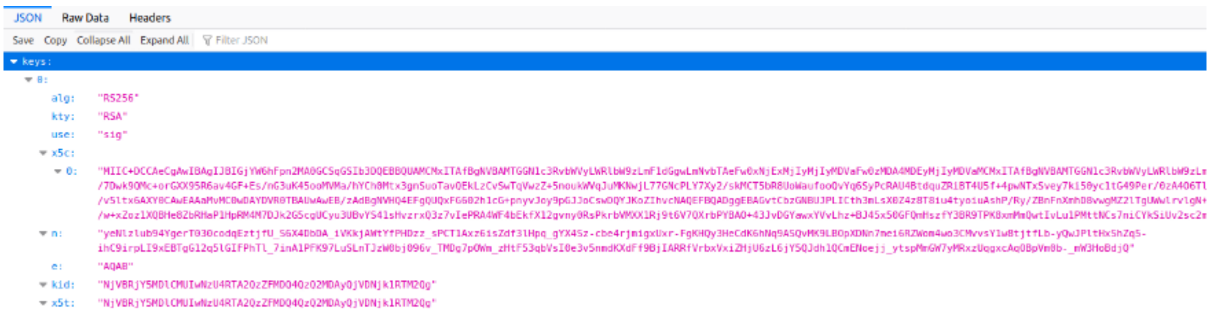


3. Crack the key

- Several tools that can brute force the HS256 signature on a JWT if the secret key is not enough strong:
  - jwtbrute, a .NET implementation.
  - This Python script I wrote that uses PyJWT to do the decoding.
  - John the Ripper

4. Forged Header Parameter

The JSON Web Key Set (JWKS) is a set of public keys which are used for token verification. Here is an example:



```
keys:
  0:
    alg:    "RS256"
    kty:    "RSA"
    use:    "sig"
    x5c:
      0:    "MIIC+DCCAeCgAwIBAgIJBIGjYW6hFpn2MA0GCSqGSIb3DQEBBQUAMCMxITAfBgNVBAMTGGN1c3RvbWVyLWRlbW9zLmF1dGgwLmNvbTAeFw0xNjExMjIyMjIyMDVaFw0zMDA4MDEyMjIyMDVaMCMxITAfBgNVBAMTGGN1c3RvbWVyLWRlbW9zLm
            /7Dwk9QMc+orGXX95R6av4GF+Es/nG3uK45ooMVMa/hYCh8Mtx3gn5uoTavQEkLzCvSwTqVwzZ+5nouKWVqJuMKNwjL77GNcPLY7Xy2/skMCT5bR8UoWaufooQvYq6SyPcRAU4BtdquZR1BT4U5f+4pwNTx5vey7ki50yc1tG49Per/0zA4O6Tl
            /v5ltx6AXY8CAwEAAaMvMC0wDAYDVR0TBAUwAwEB/zAdBgNVHQ4EFgQUQxFG602h1cG+pnyvJoy9pGJJoCswDQYJKoZIhvcNAQEFBQADggEBAGvtCbzGNBUJPLICth3mLsX0Z4z8T8iu4tyoiuAshP/Ry/ZBnFnXmhD8vwgMZ2lTgUWwlrvlgN+
            /w+xZoz1XQBHe8ZbRHaP1HpRM4M7DJk2G5cgUCyu3UBvYS41sHvzrxQ3z7vIePRA4WF4bEkfX12gvny0RsPkrbWMXX1Rj9t6V7QXrbPYBAO+43JvDGYawxYVvLhz+BJ45x50GFQmHszfY3BR9TPK8xmMmQwtIvLu1PMttNCs7niCYkSiUv2sc2m
    n:      "yeNlzlub94YgerT030codqEztjfU_S6X4DbDA_1VKkjAWtYfPHDzz_sPCT1Axz6isZdf31Hpq_gYX4Sz-cbe4rjmigxUxr-FgKHQy3HeCdK6hNq9ASQvMK9LBOpXDNn7mei6RZWom4wo3CMvvsY1w8tjtfLb-yQwJPltHxShZq5-
            ihC9irpLI9xEBTgG12q5lGIFPhTl_7inA1PFK97LuSLnTJzW0bj096v_TMDg7pOWm_zHtF53qbVsI0e3v5nmdKXdFf9BjIARRfVrbxVxiZHjU6zL6jY5QJdh1QCmENoejj_ytspMmGW7yMRxzUqgxcAq0BpVm0b-_mW3HoBdjQ"
    e:      "AQAB"
    kid:    "NjVBRjY5MDlCMUIwNzU4RTA2QzZFMDQ4QzQ2MDAyQjVDNjk1RTM2Qg"
    x5t:    "NjVBRjY5MDlCMUIwNzU4RTA2QzZFMDQ4QzQ2MDAyQjVDNjk1RTM2Qg"
```

Public keys in JWKS

This file is stored in a Trusted server and the Application can point to this file via **“jku”** and **“x5u”** Header parameters, but if we being able to manipulate the URL via tricks like Open redirect, adding @ symbol after the hostname etc.

Then we can redirect the Application to our malicious server instead of the Trusted server and this way we can generate new tokens because both public and private keys are controlled by us.

- **JSON Set URL (jku):**

This parameter points to a set of public keys in JSON-encoded format (attributes **n** and **e** in JWKS) and “jwt_tool” auto generate the JWKS file named *“jwttool_custom_jwks.json”* for this attack at the first run of tool after installation.

Command:

python3 jwt_tool.py <JWT> -X s -ju "https://attacker.com/jwttool_custom_jwks.json"

- **X.509 URL (x5u):**

This parameter points to X.509 public key certificate or chain of certificates (attribute **x5c** in JWKS) and you can generate this certificate with the corresponding private key like this:

```
openssl req -newkey rsa:2048 -nodes -keyout private.pem -x509 -days 365 -out
attacker.crt -subj "/C=AU/L=Brisbane/O=CompanyName/CN=pentester"
```

Here using OpenSSL, certificate got created in “attacker.crt” which now can be embedded in a JWKS file with “x5c” attribute and the exploitation can be done like this:

Command:

```
python3 jwt_tool.py <JWT> -S rs256 -pr private.pem -I -hc x5u -hv
"https://attacker.com/custom_x5u.json"
```

- **Embedded Public Keys:**

If server embed public keys directly in the token via **“jwk”** (JSON Web Key) or **“x5c”** (X.509 Certificate Chain) parameters then try to replace them with your own public keys and sign the token with the corresponding Private key.

5. HTTP Response Header Injection

Suppose if an Application is restricting any manipulated URL in “jku” or “x5c” parameters, then we can use Response Header injection vulnerability to add inline JWKS in the HTTP response and force the Application to use this for Signature verification.

To demonstrate this attack, I have written a Python script which can be found here.

IV. How to use JWT token safely ?

In order to make the JWT use safe and secure, it is recommended to:

- Use secure connection when transferring tokens;
- Never transfer users’ sensitive data in the tokens;
- Limit JWT lifespan and use “refresh tokens” mechanism;
- Use long key phrases;
- Keep a white list of authorised signature algorithms on the application side;

- Work, ideally, with one signature algorithm only;
- Choose well-known and reliable libraries for JWT operation;
- Always validate and sanitise the data received from users.

Reference:

Blog: https://www.pingidentity.com/en/company/blog/posts/2019/jwt-security-nobody-talks-about.html

Cyberpolygon: https://cyberpolygon.com/materials/security-of-json-web-tokens-jwt/

Stackexchange: https://security.stackexchange.com/questions/64541/can-i-prevent-a-replay-attack-of-my-signed-jwts

Blog: https://medium.com/101-writeups/hacking-json-web-token-jwt-233fe6c862e6

Securityflag: https://knowledge-base.secureflag.com/vulnerabilities/broken_authentication/broken_json_web_token_vulnerability.html

Sjoerdlangkemper: https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication/