JWT attacks

What are JWTs?

JSON web tokens (JWTs) are a standardized format for sending cryptographically signed JSON data between systems.

They can theoretically contain any kind of data, but are most commonly used to send information ("claims") about users as part of authentication, session handling, and access control mechanisms.

Everything is client-side stored.

JWT format

A JWT consists of 3 parts:

- a header
- a payload
- a signature
  They are separated by a dot (.)

```
eyJraWQiOiI5MTM2ZGRiMy1jYjBhLTRhMTktYTA3ZS1lYWRmNWE0NGM4YjUiLCJhbGciOiJSUzI1NiJ9.

eyJpc3MiOiJwb3J0c3dpZ2dlciIsImV4cCI6MTY0ODAzNzE2NCwibmFtZSI6IkNhcmxvcyBNb250b3lhIiwic3ViIjoiY2FybG9zIiwic
m9sZSI6ImJsb2dfYXV0aG9yIiwiZW1haWwiOiJjYXJsb3NAY2FybG9zLW1vbnRveWEubmV0IiwiaWF0IjoxNTE2MjM5MDIyfQ.

SYZBPIBg2CRjXAJ8vCER0LA_ENjII1JakvNQoP-Hw6GG1zfl4JyngsZReIfqRvIAEi5L4HV0q7_9qGhQZvy9ZdxEJbwTxRs_6Lb-
fZTDpW6lKYNdMyjw45_alSCZ1fypsMWz_2mTpQzil0lOtps5Ei_z7mM7M8gCwe_AGpI53JxduQOaB5HkT5gVrv9cKu9CsW5MS6ZbqYXpG
yOG5ehoxqm8DL5tFYaW3lB50ELxi0KsuTKEbD0t5BCl0aCR2MBJWAbN-
xeLwEenaqBiwPVvKixYleeDQiBEIylFdNNIMviKRgXiYuAvMziVPbwSgkZVHeEdF5MQP1Oe2Spac-6IfA
```

The header and the payload are just encoded with base64.

For example we can easily decode the payload because is in base 64:

```
{
        "iss": "portswigger",
         "exp": 1648037164,
         "name": "Carlos Montoya",
         "sub": "carlos",
         "role": "blog_author",
         "email": "carlos@carlos-montoya.net",
         "iat": 1516239022
 }
```

JWT signature

The signature is the most important thing because is used by the server to understand if the JWT is legit.

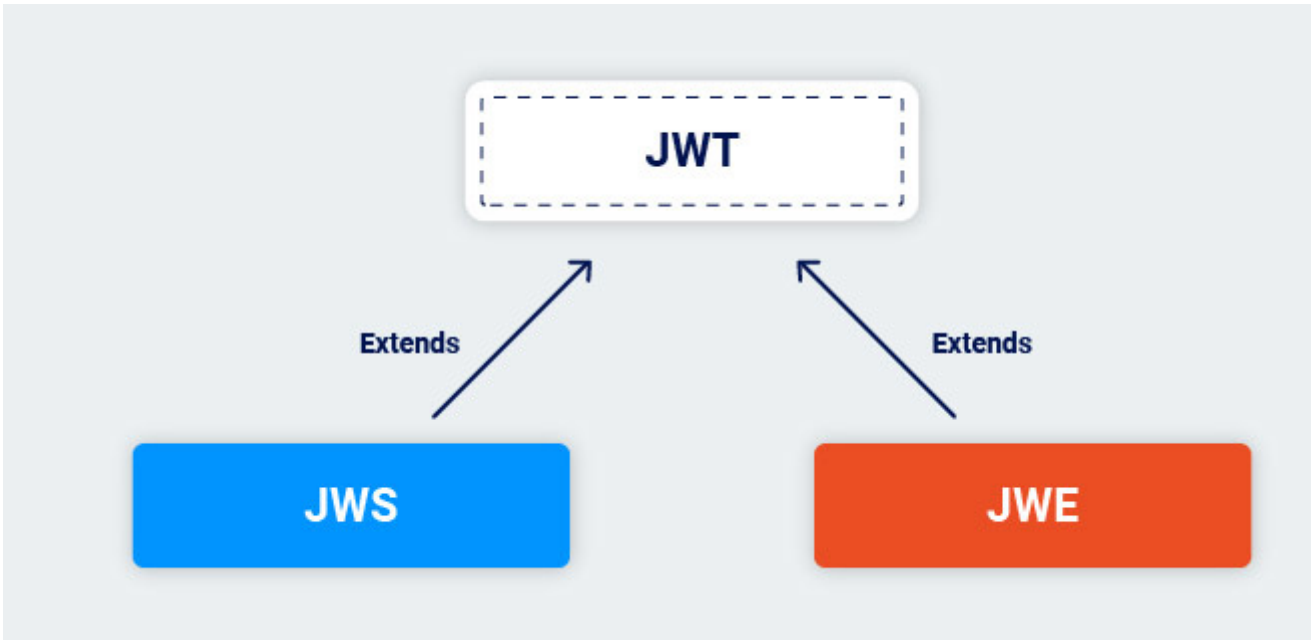
In general the server uses as signature the hash value of the header and the payload.
In some cases this hash value is also encrypted.

So if an attacker changes the payload then the signature will not match anymore.
The attacker can modify the payload and use the malicious JWT only if he is able to use the same hash function and the same encryption key.

JWT vs JWS vs JWE

JWT only defines that the information must be transferred using a JSON.

The JWT spec is extended by both the JSON Web Signature (JWS) and JSON Web Encryption (JWE) specifications, which define concrete ways of actually implementing JWTs.



JWS are JWT with a signature

JWE are JWT that use the siganture and the encryption of the signature.

What is the impact of JWT attacks?

This type of attack is very severe because if an attacker can change the JWT token then he is able to escalate privileges or to impersonate other users.

How do vulnerabilities to JWT attacks arise?

In general they depend on implementation flaws and this happens when the signature of the JWT is not verified properly.

In addition if the server key is leaked in some way, or can be guessed or brute-forced, an attacker can generate a valid signature for any arbitrary token, compromising the entire mechanism.

Exploiting flawed JWT signature verification

In general this happens when the siganture is not properly verified.

Accepting arbitrary signatures

JWT libraries typically provide one method for verifying tokens and another that just decodes them.

- For example, the Node.js library `jsonwebtoken` has `verify()` and `decode()`.
- Occasionally, developers confuse these two methods and only pass incoming tokens to the `decode()` method. This effectively means that the application doesn't verify the signature at all.

Accepting tokens with no signature

Among other things, the JWT header contains an `alg` parameter.
This tells the server which algorithm was used to sign the token and, therefore, which algorithm it needs to use when verifying the signature.

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

By default, the `alg` parameter can be set to `none`, which indicates a so-called "unsecured JWT".

```
{
        "alg": "none",
        "typ": "JWT"
}
```

- this forces the server to accept the token if `none` is not disabled, because we have no signature algorithm to use

Due to the obvious dangers of this, servers usually reject tokens with no signature.

We can sometimes bypass these filters using classic obfuscation techniques, such as mixed capitalization and unexpected encodings.

Brute-forcing secret keys

When implementing JWT applications, developers sometimes make mistakes like forgetting to change default or placeholder secrets.

So basically it could be possible to bruteforce secret keys using wordlists.

Brute-forcing secret keys using hashcat

The idea is to use hashcat.

First of all we can see directly if a string is a JWT using this command

```
hashcat -a 0
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM
5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c" path_wordlist
```

- remember to escape bash chars like $ using $

From this we can find the type of the encoded string in output:

```
16500 | JWT (JSON Web Token) | Network Protocol
```

So then we can run:

```
hashcat -a 0 -m 16500
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM
5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c" path_wordlist
```

JWT header parameter injections

The following ones are of particular interest to attackers.

- jwk (JSON Web Key) - Provides an embedded JSON object representing the key.
- jku (JSON Web Key Set URL) - Provides a URL from which servers can fetch a set of keys containing the correct key.
- kid (Key ID) - Provides an ID that servers can use to identify the correct key in cases where there are multiple keys to choose from. Depending on the format of the key, this may have a matching kid parameter.

Injecting self-signed JWTs via the jwk parameter

JWK

A JWK (JSON Web Key) is a standardized format for representing keys as a JSON object.

```
{
        "kid": "ed2Nf8sb-sD6ng0-scs5390g-fFD8sfxG",
        "typ": "JWT",
        "alg": "RS256",
        "jwk": {"kty": "RSA", "e": "AQAB", "kid": "ed2Nf8sb-sD6ng0-scs5390g-fFD8sfxG", "n":
"yy1wpYmffgXBxhAUJzHHocCuJolwDqql75ZWuCQ_cb33K2vh9m" }
}
```

You can exploit this behavior by signing a modified JWT using your own RSA private key, then embedding the matching public key in the `jwk` header.

Injecting self-signed JWTs via the jku parameter

Instead of embedding public keys directly using the `jwk` header parameter, some servers let you use the `jku` (JWK Set URL) header parameter to reference a JWK Set containing the key. When verifying the signature, the server fetches the relevant key from this URL.

```
{ "keys": [ { "kty": "RSA", "e": "AQAB", "kid": "75d0ef47-af89-47a9-9061-7c02a610d5ab", "n": "o-
yy1wpYmffgXBxhAUJzHHocCuJolwDqql75ZWuCQ_cb33K2vh9mk6GPM9gNN4Y_qTVX67WhsN3JvaFYw-fhvsWQ" }, { "kty":
"RSA", "e": "AQAB", "kid": "d8fDFo-fS9-faS14a9-ASf99sa-7c1Ad5abA", "n": "fc3f-
yy1wpYmffgXBxhAUJzHql79gNNQ_cb33HocCuJolwDqmk6GPM4Y_qTVX67WhsN3JvaFYw-dfg6DH-asAScw" } ] }
```

Injecting self-signed JWTs via the kid parameter

`kid` (Key ID) parameter, which helps the server identify which key to use when verifying the signature.

Verification keys are often stored as a JWK Set.
In this case, the server may simply look for the JWK with the same `kid` as the token.

However, the JWS specification doesn't define a concrete structure for this ID - it's just an arbitrary string of the developer's choosing.

If this parameter is also vulnerable to directory traversal, an attacker could potentially force the server to use an arbitrary file from its filesystem as the verification key.

```
{
    "kid": "../../path/to/file",
    "typ": "JWT",
    "alg": "HS256",
    "k": "asGsADas3421-dfh9DGN-AFDFDbasfd8-anfjkvc"
}
```

One of the simplest methods is to use `/dev/null`, which is present on most Linux systems.

As this is an empty file, reading it returns an empty string.

Therefore, signing the token with a empty string will result in a valid signature.

```
{
    "kid": "../../dev/null",
    "typ": "JWT",
    "alg": "HS256",
    "k": "asGsADas3421-dfh9DGN-AFDFDbasfd8-anfjkvc"
}
```

How to prevent JWT attacks

- Use an up-to-date library for handling JWTs
- Make sure that you perform robust signature verification on any JWTs that you receive
- Enforce a strict whitelist of permitted hosts for the `jku` header
- Make sure that you're not vulnerable to path traversal or SQL injection via the `kid` header parameter

Other important things:

- Always set an expiration date for any tokens that you issue.
- Avoid sending tokens in URL parameters where possible.
- Enable the issuing server to revoke tokens (on logout, for example).