

JSON web tokens

JSON web tokens are a type of get right of entry to tokens that are widely used in industrial applications. They are based totally on the JSON layout and consists of a token signature to ensure the integrity of the token.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c|

How do JSON Web Tokens work?

A JSON internet token consists of three components: a header, a payload, and a signature.

The header

The header part of a JSON web token identifies the algorithm used to generate the signature. It is a base64url encoded string of a JSON blob like this:

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

base64url encoded string: eyBhbGciOiJIUzI1NiwgdHlwIDogSldUIH0K

(Base64url encoding is a modified version of base64 for the URL format. It is comparable to base64, but uses exclusive non-alphanumeric characters and omits padding.)

The most common algorithms used are **HMAC** and **RSA algorithms**.

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

The payload

The payload area includes the records that is truly used for get right of entry to control. This section, too, is base64url encoded earlier than being used in the token.

```
{  
  "user_name" : "admin",  
}
```

base64url encoded string: eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg

The signature

The signature is the part that is used to validate that the token has now not been tampered with. It is calculated by means of concatenating the header with the payload, then signing with the algorithm certain in the header.

```
signature = HMAC-SHA256(base64urlEncode(header) + '.' + base64urlEncode(payload),  
secret_key)
```

```
// Let's simply say the fee of secret_key is "key".
```

```
-> signature characteristic returns 4Hb/6ibbViPOzq9SJflsNGPWsk6B8F6EqVrkNjpXh7M
```

For this particular token, the string

“eyJhbGcgOiB1Zl1NiWgdHlwiDogSldUIH0K.eyJB1c2VyX25hbWUgOiBhZG1pbiB9Cg.4Hb/6ibbViPOzq9SJflsNGPWsk6B8F6EqVrkNjpXh7M” is signed with the HS256 algorithm with the secret key “key”. The ensuing string is 4Hb/6ibbViPOzq9SJflsNGPWsk6B8F6EqVrkNjpXh7M.

The entire token

You get the whole token with the aid of concatenating every part (header, payload, and signature) with a “.” in between each section.

```
eyJhbGcgOiB1Zl1NiWgdHlwiDogSldUIH0K.eyJB1c2VyX25hbWUgOiBhZG1pbiB9Cg.4Hb/6ibbViPOzq9SJflsNGPWsk6B8F6EqVrkNjpXh7M
```

Ways to bypass JSON Web Token controls

When implemented correctly, JSON net tokens provide a secure way to become aware of the user due to the fact the records contained in the payload section can't be tampered with. (Since the consumer does not have get entry to to the secret key, she cannot signal the token herself.)

But if carried out incorrectly, there are methods that an attacker can bypass the protection mechanism and forge arbitrary tokens.

Change up the algorithm type

One of the approaches that attackers can forge their very own tokens is by tampering with the alg field of the header. If the application does not avoid the algorithm type used in the

JWT, an attacker can specify which algorithm to use, which could compromise the security of the token.

None algorithm

JWT supports a “none” algorithm. If the alg field is set to “none”, any token would be considered legitimate if their signature section is set to empty. For example, the following token would be regarded valid:

eyJiYWxnliA6lCJOb25lliwglR5cClgOiAiSldUliB9Cg.eyJB1c2VyX25hbWUgOiBhZG1pbiB9Cg.

It is simply the base64url encoded versions of these two blobs, and no signature is present.

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

This characteristic was once originally used for debugging purposes. But if no longer became off in a manufacturing environment, it would permit attackers to forge any token they desire through setting the alg subject to “none”. They can then impersonate all people on the web page with the aid of the use of the forged tokens.

2. HMAC algorithm

The two most common kinds of algorithms used for JWTs are HMAC and RSA. With HMAC, the token would be signed with a key, then later validated with the equal key. As for RSA, the token would first be created with a non-public key, then established with the corresponding public key.

HMAC -> signed with a key, established with the equal key

RSA -> signed with a personal key, tested with the corresponding public key

It is imperative that the secret key for HMAC tokens and the personal key for RSA tokens are kept a secret since they are used to signal the tokens.

Now let's say that there is an utility that used to be at first designed to use RSA tokens. The tokens are signed with a non-public key A, which is saved a secret from the public. Then the tokens are tested with public key B, which is handy to anyone. This is k as lengthy as the tokens are constantly treated as RSA tokens.

Token signed with key A -> Token validated with key B (RSA scenario)

Now if the attacker modifications the alg to HMAC, she would possibly be able to create legitimate tokens via signing the solid tokens with the RSA public key B.

This is because initially when the token is signed with RSA, the program verifies it with the RSA public key B. When the signing algorithm is switched to HMAC, the token is nevertheless demonstrated with the RSA public key B, but this time, the token can be signed with the equal public key B (since it's the use of HMAC).

Token signed with key B -> Token demonstrated with key B (HMAC scenario)

Provide a non-valid signature

It is also feasible that the signature of the token is by no means confirmed after it arrives at the application. This way an attacker may want to definitely bypass the safety mechanism with the aid of offering an invalid signature.

Brute force the secret key

It ought to also be feasible to brute pressure the key used to signal a JWT.

The attacker has a lot of data to start with: she knows the algorithm used to sign the token, the payload that used to be signed and the resulting signature. If the key used to signal the token is no longer complex enough, she might be in a position to brute force it easily.

Leak the secret key

If an attacker is not in a position to brute pressure the key, she might attempt leaking the secret key instead. If some other vulnerability (like a directory traversal, XXE, SSRF) exists that permits the attacker to read the file where the key fee is stored, the attacker can steal the key and signal arbitrary tokens.

KID manipulation

KID stands for "Key ID". It is an elective header subject in JWTs, and it permits developers to specify the key to be used for verifying the token. The suitable usage of a KID parameter appears like this:

```
{  
  "alg" : "HS256",  
  "typ" : "JWT",  
  "kid" : "1"    // use key quantity 1 to affirm the token  
}
```

Since this subject is managed with the aid of the user, it can be manipulated by way of attackers and lead to risky consequences.

Directory traversal

Since the KID is regularly used to retrieve a key file from the file system, if it is not sanitized before use, it can lead to a listing traversal attack. When this is the case, the attacker would be able to specify any file in the file machine as the key to be used to verify the token.

```
"kid": "../../../public/css/main.css"  
// use the publicly reachable file main.css to affirm the token
```

For example, the attacker can pressure the software into the usage of a publicly on hand file as the key, and sign an HMAC token the use of that file.

2. SQL injection

The KID may want to also be used to retrieve the key from a database. In this case, it might be viable to make use of SQL injection to bypass JWT signing.

If SQL injection is viable on the KID parameter, the attacker can use this injection to return any fee she wants.

```
"kid": "aaaaaaa' UNION SELECT 'key';--"  
// use the string "key" to verify the token
```

For example, this above injection will cause the software to return the string “key” (since the key named “aaaaaaa” doesn’t exist in the database). The token will then be demonstrated with the string “key” as the secret key.

Header parameter manipulation

In addition to a key ID, JSON internet token requirements also provide developers with the potential to specify keys by way of a URL.

JKU header parameter

JKU stands for “JWK Set URL”. It is an optionally available header area used to specify a URL that points to a set of keys that are used to verify the token. If this field is allowed and no longer exact restricted, an attacker could host their very own key file and specify that the application makes use of it to verify tokens.

```
jku URL -&gt; file containing JWK set -&gt; JWK used to affirm the token
```

2. JWK header parameter

The optionally available JWK (JSON Web Key) header parameter allows attackers to embed the key used to confirm the token directly in the token.

3. X5U, X5C URL manipulation

Similar to the jku and jwk headers, the x5u and x5c header parameters allow attackers to specify the public key certificate or certificate chain used to verify the token. x5u specifies the records in the URI structure while x5c lets in the certificates values to be embedded in the token.

Other JWT protection issues

There are additionally other JWT problems that arise when they are not effectively implemented. These are now not very common, however without a doubt really worth looking out for

Information leak

Since JSON internet tokens are used for get admission to control, they often comprise information about the user.

If the token is not encrypted, everyone can base64 decode the token and study the token's payload. So if the token includes touchy information, it might grow to be a supply of facts leaks. A correct implemented signature section of the JSON web token offers statistics integrity, now not confidentiality.

Command injection

Sometimes when the KID parameter is passed immediately into an insecure file read operation, it is possible to inject instructions into the code flow.

One of the features that should enable this type of attack is the Ruby open() function. This feature lets in attackers to execute machine instructions by surely tacking the command onto the input after the KID file name:

```
"key_file" | whoami;
```

This is just one example. Theoretically, vulnerabilities like this can appear on every occasion an application passes any of the header parameters unsensitized into any function such as system(), exec(), etc.

Ultimately, JSON web tokens are simply every other structure of consumer input. They should usually be treated with scepticism and sanitized rigorously.

Resources

<https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens>

<https://systemweakness.com/hacking-jwt-d29f39e202d5>

