

1.1.0.0 Check bearer token

Bearer token

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted

 [RFC 7519: JSON Web Token \(JWT\)](#)

Bearer Token Structure

The JWT is compounded by 3 parts, that are separated by a dot. Each part is coded in base64

HEADER: ALGORITHM & TOKEN TYPE

```
1 eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IkpQSW1JbnRyYW5ldF9SUzI1NiJ9
```


If we decoded in base 64¹, we get:

```
1 {
2   "alg": "RS256",
3   "typ": "JWT",
4   "kid": "APIIntranet_RS256"
5 }
```

This specification further specifies the use of the following Header Parameters in both the cases where the JWT is a JWS and where it is a JWE.

The fields that can appear in this section are defined here:  [JWSHeader - nimbus-jose-jwt 2.21 javadoc](#)

In our world we have:

- **"alg"** (Algorithm): algorithm that was used for signing the token. This parameter has the same meaning, syntax, and processing rules as the "alg" Header Parameter defined in Section 4.1.1 of [JWS], except that the Header Parameter identifies the cryptographic algorithm used to encrypt or determine the value of the CEK. The encrypted content is not usable if the "alg" value does not represent a supported algorithm, or if the recipient does not have a key that can be used with that algorithm. A list of defined "alg" values for this use can be found in  [RFC 7518: JSON Web Algorithms \(JWA\)](#)
- **"typ"** (Type): token type. The "typ" (type) Header Parameter defined by [JWS] and [JWE] is used by JWT applications to declare the media type [IANA.MediaTypes] of this complete JWT. This is intended for use by the JWT application when values that are not JWTs could also be present in an application data structure that can contain a JWT object; the application can use this value to disambiguate among the different kinds of objects that might be present. It will typically not be used by applications when it is already known that the object is a JWT. This parameter is ignored by JWT implementations; any processing of this parameter is performed by the JWT application. If present, it is RECOMMENDED that its value be "JWT" to indicate that this object is a JWT. While media type names are not case sensitive, it is RECOMMENDED that "JWT" always be spelled using uppercase characters for compatibility with legacy implementations. Use of this Header Parameter is OPTIONAL.
- **"kid"** (Key ID): Key ID. This field identifies which entity signed the token. So, the one that wants to validate the token, will need to retrieve the public key associated to this key ID (probably using the [keymanager](#)). This parameter has the same meaning, syntax, and processing rules as the "kid" Header Parameter defined in Section 4.1.4 of [JWS], except that the key hint references the public key to which the JWE was encrypted; this can be used to determine the private key needed to decrypt the JWE. This parameter allows originators to explicitly signal a change of key to JWE recipients.

- **"cty"** (Content Type): The "cty" (content type) Header Parameter defined by [JWS] and [JWE] is used by this specification to convey structural information about the JWT. In the normal case in which nested signing or encryption operations are not employed, the use of this Header Parameter is NOT RECOMMENDED. In the case that nested signing or encryption is employed, this Header Parameter MUST be present; in this case, the value MUST be "JWT", to indicate that a Nested JWT is carried in this JWT. While media type names are not case sensitive, it is RECOMMENDED that "JWT" always be spelled using uppercase characters for compatibility with legacy implementations.

PAYLOAD: DATA

```
1 eyJpc3MiOiJBUEltSW50cmFuZlZlcjZzdWIiOiJCMDAxMDkiLCJhdwQiOiJTVjBU0VSRU5JVfkiLCJuYmYiOiJlMDA2MzI3ODUuMjc1LCJleHAiOiJ
```

If we decoded in base 64¹, we get:

```
1 {
2   "iss": "APIIntranet",
3   "sub": "B00109",
4   "aud": "SARASERENITY",
5   "nbf": 1500632785.275,
6   "exp": 1500891985.275,
7   "iat": 1500632785.275,
8   "jti": "4aecdc9b-4920-4b59-b25d-7ecd4ba26c49"
9 }
```

The fields are defined here: [JWTClaimsSet - nimbus-jose-jwt 4.0 javadoc](#)

- **"iss"** (Issuer): The "iss" (issuer) claim identifies the principal that issued the JWT. The processing of this claim is generally application specific. The "iss" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.
- **"sub"** (Subject): The "sub" (subject) claim identifies the principal that is the subject of the JWT. The claims in a JWT are normally statements about the subject. The subject value MUST either be scoped to be locally unique in the context of the issuer or be globally unique. The processing of this claim is generally application specific. The "sub" value is a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.
- **"aud"** (Audience): The "aud" (audience) claim identifies the recipients that the JWT is intended for. Each principal intended to process the JWT MUST identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the "aud" claim when this claim is present, then the JWT MUST be rejected. In the general case, the "aud" value is an array of case-sensitive strings, each containing a StringOrURI value. In the special case when the JWT has one audience, the "aud" value MAY be a single case-sensitive string containing a StringOrURI value. The interpretation of audience values is generally application specific. Use of this claim is OPTIONAL.
- **"nbf"** (Not before): The "nbf" (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing. The processing of the "nbf" claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the "nbf" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.
- **"exp"** (expiration time): The "exp" (expiration time) claim identifies the expiration time on or after which the JWT MUST NOT be accepted for processing. The processing of the "exp" claim requires that the current date/time MUST be before the expiration date/time listed in the "exp" claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.
- **"iat"** (issued at): The "iat" (issued at) claim identifies the time at which the JWT was issued. This claim can be used to determine the age of the JWT. Its value MUST be a number containing a NumericDate value. Use of this claim is OPTIONAL.
- **"jti"** (JWT ID): The "jti" (JWT ID) claim provides a unique identifier for the JWT. The identifier value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to a different data object; if the application uses multiple issuers, collisions MUST be prevented among values produced by different issuers as well. The "jti" claim can be used to prevent the JWT from being replayed. The "jti" value is a case-sensitive string. Use of this claim is OPTIONAL.

SIGNATURE

Computing the MAC of the encoded JOSE Header and encoded JWS Payload with the HMAC SHA-256 algorithm and base64url encoding the HMAC value in the manner specified in [JWS] yields this encoded JWS Signature

Example bearer token

eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJqb2UiLA0KICJleHAiOiJleMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlIHNvbS9pc19yb290Ijp0cnVlfQ.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk

Content token

How it looks like

```
1 eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IkpQSW1JbnRyYW5ldF9SUzI1NiJ9.eyJpc3MiOiJBUeIltSW50cmFuZlZlZdWlI0iJCMi
```

Header: *eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IkpQSW1JbnRyYW5ldF9SUzI1NiJ9*

```
1 {
2   "alg": "RS256",
3   "typ": "JWT",
4   "kid": "APIIntranet_RS256"
5 }
```

Payload:

eyJpc3MiOiJBUeIltSW50cmFuZlZlZdWlI0iJCMiDAxMDkiLCJhdWQiOiJTVVJBU0VSRU5JVfkiLCJuYmYiOiJlMDA2MzI3ODUuMjc1LCJleHAiOiJlMDA0OTE5ODUuMjc1LCJpYXQiOiJlMDA2MzI3ODUuMjc1LCJqdGkiOiI0YWVjZGM5Yi00OTlwLTRiNTktYjI1ZC03ZW5kNGJhMjZjNDkifQ

```
1 {
2   "iss": "APIIntranet",
3   "sub": "B00109",
4   "aud": "SARASERENITY",
5   "nbf": 1500632785.275,
6   "exp": 1500891985.275,
7   "iat": 1500632785.275,
8   "jti": "4aecdc9b-4920-4b59-b25d-7ecd4ba26c49"
9 }
```

Signature:

s9drzQBAljrD_3dqhPbD9u6z7WdcjeStyPZhyRAvOf4p0TWZf08nogMEwwJKOO3ZM8hatMqAnpwR8a54TcQKv3ERSREWkz16Ldwa
gBToasmXf_PV4i_hPNhUXBMZlnCEmB32PA8HZdUSkolzBFTHJv2I-XyV_5H7mb9g6dygEqMBv8NfKtt6xY54QQkCYoAV8Feas-
pjCpHYKpc2Bk_TvLOhZkp4DLQFU045viOTSEfolF9xlaG7LTxyLxzaxL9NhNoV__kEKD-
6ZBoAlFlu2fgDScZVAPf4wF4UEd8DdU0nwlbjnQM0d0rBd4PBuKW1ynamGUqAEIcWz_UE3CLWKQ

Tool for generating a JWT token

General info

The security team has provided us ([SECSERVSUP-168](#)) a tool for generating a JWT token in a easy way

The tool is just this HTML page: [generateJWT.html](#) → this generator will create audience as a String.

Updated version: [generateJWTv2.html](#) → this generator will create audience as a Object. It is needed for the latest checkToken customNode.

Example of use

Just fill the data that you want the token to have (in header and payload):

(Step1) Set Claim.

Set claim value of JWT token.

Key ID(kid)

Issuer(iss)

Subject(sub)

Audience(aud)

Not before Time(nbf)

Expiration Time(exp)

Issue At Time(iat)

JWT ID(jti)

Session Id(sid)

Security Context(secCtx)

Choose the algorithm and enter the proper value. For instance, what we use in most of the cases is RS256, so we have to enter a private key. We can extract it from a .pem file, for example, and in that case it is the needed to insert here this text:

```
1 -----BEGIN ENCRYPTED PRIVATE KEY-----  
2 MIIFDjBAbg...  
3 -----END ENCRYPTED PRIVATE KEY-----
```

And the pass phrase/passcode associated to that key (in order to be able to decrypt it)

(Step2) Choose issuer key and JWS signing algorithm.

CHOOSE 'Used for STS testing' to use keys of STS tests. Choose RS256 (SHA256withRSA RSA2048bit:z4) to use a private key generated for testing purposes

passcode for private key:

Finally, just click on *Sign it!* and, if everything goes fine, we'll have a JWT token

(Step3) Press "Sign it!"

To generate signed JWT just pass 'Sign it!'.

```
Xqg2ehAT9eBp0b1E1Wtk7JITHoZdxXCboa1F949W0q3DrRBqKBXhr7M619pHut
RG6s6z2vqI5_aLnJ_TevP8AbJ90FFq0_FKb0UsfxZKagVupDzrmbCX0jCSnsQX
1eBrhbAh6FdOUgX_BLDiw
```

In this case, the generated token is:

```
1 eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IkpQSW1JbnRlcm5ldEdlcm1hbnlfU1MyNTYifQ.eyJpc3MiOiJBUEltSW50ZXJuZXRhZXJ
```

That is compounded by:

- Header:

```
1 {
2   "alg": "RS256",
3   "typ": "JWT",
4   "kid": "APIInternetGermany_RS256"
5 }
```

- Payload:

```
1 {
2   "iss": "APIInternetGermany",
3   "sub": "uid:HBHGIMTX",
4   "nbf": 1501082825,
5   "exp": 1501086425,
6   "iat": 1501082825,
7   "jti": "id123456",
8   "sid": "B4657888EAE1F9027E0FA938"
9 }
```

How to generate a PEM file

Private key

```
1 ssh-keygen -t rsa -b 2048 -f private_key.pem
```

Public key

```
1 ssh-keygen -y -f private_key.pem > key.pub
```

Convert public key to PEM file

```
1 ssh-keygen -f key.pub -e > public_key.pem
```

Validate token

1. Obtain the JWT of the request
2. Split the token into its parts (header, payload, signature)
3. Decode the header and the payload in JSON format.
4. Verify that the **"typ"** in the header is *JWT*

5. Verify that the header algorithm matches the expected one and that the requester is a valid requester, using the fields: "alg" and "iss"
6. Obtain the public key using the Token Manager and the "kid" key.
7. Verify the token signature using the public key and the encryption algorithm. (signature = header + "." + payload)
8. Verify that the token has not expired.
9. Verify that the "aud" matches the expected one.
10. Return that the token is valid or if any verification failed, return KO.

Serenity library for validating it

<https://nexus.alm.gsnetcloud.corp/#browse/search/maven=attributes.maven2.artifactId%3Dsecurity-credentials>

More detail:  [RFC 7516: JSON Web Encryption \(JWE\)](#)