api$jwt Documentation$

Greger Teigre Wedel

Apr 05, 2021

Contents:

This Python library is a wrapper around pyjwt to support JWT creation and validation as well as payload handling (scopes, auth level, etc).

This documentation can be found at https://api-jwt.readthedocs.io/en/latest/.

Source code repo is at https://github.com/gregertw/api_jwt.

CHAPTER 1

License

CHAPTER 2

README and Getting Started

This Python library is a wrapper around pyjwt to support JWT creation and validation as well as payload handling (scopes, auth level, etc).

2.1 Use case

The typical use is a micro services architecture where a single auth service is responsible for issuing JWT tokens for clients you may or may not trust. In order to scale without a central point of failure, the JWT token should contain all necessary information for each micro service to trust the identity of the requestor, as well what the requestor can access. This is done through embedding meta information in the JWT token and signing it with a private key only known to the authn/authz service.

With only knowledge of the public key, any service can verify the signature of the JWT and thus prove it's authenticity.

NOTE!!! Do not store secrets in the payload, it is not encrypted and can easily be decoded and read.

2.2 Typical use

Make your own wrapper through sub-classing the APIJwt class. In the initialisation, load the private key (for the authn/authz service) and public key(s) for the all other services. In the authn/authz service that encodes JWTs, set all the extra payload keys that should be allowed (i.e. info you want to convey to the other services receiving the JWT), and then set the allowed values for each of the keys. The decode does not validate the payload, just the signature, so these configurations are thus not needed.

You can now use the encode() and decode() functions inherited from APIJwt to encode (and sign) the JWT, as well as decode.

2.3 Example use

Example wrapper where you add set your own configuration parameters

```
from api_jwt import APIJwt
class HudyaJWT(APIJwt):
    def __init__(self, *args, **kwargs):
        if settings.JWT_KEY_PRIVATE:  # This could be loaded from os.env(), it should␣
↪be base64 encoded
                                      # It should be in pem format and not be␣
↪encrypted
            privkey = base64.b64decode(settings.JWT_KEY_PRIVATE).decode('utf-8')
        else:
            privkey = None
        if settings.JWT_KEY_PUBLIC:   # This could be loaded from os.env(), it should␣
↪be base64 encoded in pem format
            pubkey = base64.b64decode(settings.JWT_KEY_PUBLIC).decode('utf-8')
        else:
            pubkey = None
        super().__init__(
            public_keys=pubkey,
            private_key=privkey,
            ttl=int(settings.JWT_TOKEN_TTL),  # This is in seconds
            *args, **kwargs)

        # The below is only required for encoding
        self.set_allowed('level', [
            0.0,  # Level 0, no authentication
            1.0,  # External auth
            2.0,  # Password/single-factor
            3.0,  # Multi-factor
            3.1,  # Yubikey
            3.5,  # External multi-factor
            4.0   # Certificate-level
        ])
        self.set_allowed('keys', {
            'user': 'auth_user',
            'support': 'auth_support',
            'admin': 'auth_admin'
        })
        self.set_allowed('scopes', {
            'PER_KEY': {  # Use single key with 'PER_KEY' to set allowed values based␣
↪on key
                'user': ['user:all', 'NO', 'SE', 'DK'],
                'support': [
                    'support:all',
                    'support:insurance',
                    'support:power',
                    'support:mobile'
                ],
                'admin': ['admin:all', 'user:all']
            }
        })
```

With this class, you can encode a JWT:

```
jwt_obj = HudyaJWT()
token = jwt_obj.encode(
    subject='user@domain.com',
    level='3.1',
    factor='yubikey',
```

(continues on next page)

(continued from previous page)

```
    target='user@domain.com',  # Used if the target of the scopes is different from␣
↪subject
    key='support,
    exp=3600,
    scopes=['support:mobile'],
    dnt=0,  # Normal user, full tracking
)
if token is None:
    raise
```

Decoding is super-simple:

```
jwt_data = HudyaJWT()
payload = jwt_data.decode(token)
if not jwt_data.is_valid:
    raise ValidationError("JWT is not valid")
if 'support:mobile' in payload['scope']:
    print("Access granted!")
```

2.4 How to release

Remember to update version in setup.py.

Running `docker-compose up -d` without env var COMMAND set will start up the container and run run.sh, which will result in the tests being run and the container be kept running for subsequent docker exec commands.

To do a test build and release, run docker-compose with `COMMAND="build"` as an env variable (which will be passed into run.sh as a parameter). In this case, the .pypirc file in the root dir of the docker build will be copied in. A pypitest server entry is expected.

To release, make sure you have the pypi server entry in .pypirc for release and run with COMMAND set to "release", e.g.:

```
export COMMAND="release";docker-compose up -d
```

NOTE!! Due to the volume set up in docker-compose.yml, the sources will be in sync inside and outside of the container, so there is no need to rebuild the container.

Do not forget to change CHANGELOG.md.

CHAPTER 3

CHANGELOG

3.1 Apr 5, 2021

- Bump version to 1.2.2
- Upgrade all libs and dependencies to python 3.7.10 (cryptography supports only 3.7.*)
- Moved from alpine image to regular python docker image to reduce cryptography issues in alpine
- Removed unsupported pytest path param from ini

3.2 Dec 22, 2019

- Bump version to 1.2.1
- Upgrade to latest cryptography library and its dependencies
- Fix rendering of readme at pypi.org

3.3 Feb 23, 2019

- Open-sourced library at 1.2.0, reflecting multiple internal releases

3.4 Feb 18, 2019

- First version ready for open-sourcing

3.5 Pre-Feb 18, 2019

See repo commit history

CHAPTER 4

api_jwt source docs

class api_jwt.**APIJwt**(*public_keys=None*, *private_key=None*, *ttl=None*, ***kwargs*)

Bases: object

Main class to represent a JWT used for API purposes. The internal structure of the JWT is set when the class is instantiated. This structure will be used for encode/decode operations. If no parameters are supplied, the defaults are used. If no certificate or key is supplied, a dummy test pair is used, causing a log warning.

For the extras parameter (kwargs): An extras key that is set to None will be pruned from the JWT. With no extras set, these default extras will be included. Default extras:

```
extras = {
    'level': 0.0,  # Authentication level, allowed['levels'] specifies the valid␣
↪levels
    'factor': '',  # Authentication factor used, e.g. password, otp, etc
    'target': '',  # Target id for scopes
    'dnt': 0,      # Do Not Track
    'scopes': [],  # Scopes this token gives access to
}
```

For the allowed parameter (kwargs): The allowed dict defines the valid values for each of the extras. If a new extra is to be used, a new key with valid paramters should also be included. The 'keys' and 'scopes' keys are special:

```
allowed = {
    'keys': {
        'user': 'auth_user',
        'admin': 'auth_admin'
    },
    'level': [
        0.0,  # Level 0, no authentication
        1.0,  # Externally authenticated
        2.0,  # Password/single-factor
        3.0,  # Multi-factor
        4.0   # Certificate-level
```

(continues on next page)

(continued from previous page)

```
    ],
    'dnt': [
        0,  # or not set - normal user
        1,  # reservation - don't track this user
        2,  # not anonymous - don't track and don't store data anonymously
        3   # Test user - this is a test user, don't skew metrics
    ],
    'scopes': {
        'PER_KEY': {  # Use single key with 'PER_KEY' to set allowed values based␣
↪on key
            'user': ['user:all'],
            'admin': ['admin:all']
        }
    }
}
```

Parameters

- **public_keys** – Single string or list of strings with public pem keys for signing verification
- **private_key** – Single string with pem key for
- **ttl** – Default time to live in seconds for the encoded JWT (can be overridden in encode())
- **extras** – A dict with non-mandatory JWT parameters that will be included in the encoded JWT.
- **allowed** – A dict with allowed values for keys and scopes and for the extras

add_public_keys(*keys=None*)

Add more public keys to the list of keys used to validate JWT signature

Parameters **keys** – Single or list of keys (no json allowed)

decode(*token=None*)

Decode a token, properties is_valid, is_bad, an is_expired will be set. All tokens are verified against the list of public keys

Parameters **token** – JWT token to decode

Returns Either None if doken was not successfully decoded or a dict with the payload

encode(*subject=None*, *key='user'*, *exp=None*, ***kwargs*)

Encode the token with subject as the identity this token concerns (sub), using key to identify the key type (used by some gateways like kong in the iss parameter) and with exp (or default) expiry in seconds. Additional key/value pairs will be validated against extras and the values of each against allowed. If an extra does not have a key in allowed, any value except None will be included in the token payload.

Use kwargs, extras={} or allowed={}, as short-cuts to calling set_allowed() and set_extras() before encode(), i.e. do everything at once in the encode() call.

Parameters

- **subject** – id of the subject of this token
- **key** – a valid key from _allowed['keys'], typically used in iss to match key on external system
- **exp** – expiry time in seconds

- **kwargs** – key/value pairs to include in the payload.

Returns None if not successfully encoded JWT token

expiry
Expiry date of the token

Returns Expiry timestamp for token

Return type datetime

is_bad
Set if decoded a token and token contained errors

Returns Bad token or not

Return type bool

is_expired
Set if decoded a token and token was expired :return: Expired token or not :rtype: bool

is_valid
Set if decoded a token and contains True if token is valid

Returns whether token is valid or not

Return type bool

jwt
Contains the jwt token

Returns JWT token

Return type string

override_keys(*public_keys=None*, *private_key=None*)
Override the public and private keys 'on the fly'. Preferred method is at instantiation

Parameters

- **public_keys** – list or json list of keys, or single public key for decoding
- **private_key** – private key for encoding

reset_keys()
Reset back the keys to the dummy keys (both public and private)

set_allowed(*key*, *values*)
Set allowed values for the extras in the payload

Parameters

- **key** – Payload name
- **values** – Dict for keys and scopes, list for other extras

set_extras(*key*, *default*)
Set extra allowed payload parameters for encoded tokens

Parameters

- **key** – Payload key name
- **default** – Default value, None if it should be pruned from jwt if not set

user
Contains the entire payload of a decoded token

Returns payload

Return type dict

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index