asymmetric-jwt-auth Documentation

Release 1.0.0.post9

Craig Weber

Jun 02, 2023

CONTENTS

CHAPTER

ONE

WHAT?

This is an library designed to handle authentication in *server-to-server* API requests. It accomplishes this using RSA public / private key pairs.

CHAPTER

TWO

WHY?

The standard pattern of using username and password works well for user-to-server requests, but is lacking for server-to-server applications. In these scenarios, since the password doesn't need to be memorable by a user, we can use something far more secure: asymmetric key cryptography. This has the advantage that a password is never actually sent to the server.

CHAPTER

THREE

HOW?

A public / private key pair is generated by the client machine. The server machine is then supplied with the public key, which it can store in any method it likes. When this library is used with Django, it provides a model for storing public keys associated with built-in User objects. When a request is made, the client creates a JWT including several claims and signs it using it's private key. Upon receipt, the server verifies the claim to using the public key to ensure the issuer is legitimately who they claim to be.

The claim (issued by the client) includes components: the username of the user who is attempting authentication, the current unix timestamp, and a randomly generated nonce. For example:

```
{
    "username": "guido",
    "time": 1439216312,
    "nonce": "1"
}
```

The timestamp must be within ± 20 seconds of the server time and the nonce must be unique within the given timestamp and user. In other words, if more than one request from a user is made within the same second, the nonce must change. Due to these two factors no token is usable more than once, thereby preventing replay attacks.

To make an authenticated request, the client must generate a JWT following the above format and include it as the HTTP Authorization header in the following format:

```
Authorization: JWT <my_token>
```

Important note: the claim is *not* encrypted, only signed. Additionally, the signature only prevents the claim from being tampered with or re-used. Every other part of the request is still vulnerable to tamper. Therefore, this is not a replacement for using SSL in the transport layer.

Full Documentation: https://asymmetric-jwt-auth.readthedocs.io

CHAPTER

FOUR

CONTENTS

4.1 Installation

4.1.1 Dependencies

We don't re-implement JWT or RSA in this library. Instead we rely on the widely used PyJWT and cryptography libraries as building blocks.. This library serves as a simple drop-in wrapper around those components.

4.1.2 Django Server

Install the library using pip.

```
pip install asymmetric_jwt_auth
```

Add `asymmetric_jwt_auth` to the list of `INSTALLED_APPS` in `settings.py`

```
INSTALLED_APPS = (
    ...
    'asymmetric_jwt_auth',
    ...
)
```

Add `asymmetric_jwt_auth.middleware.JWTAuthMiddleware` to the list of `MIDDLEWARE_CLASSES` in `settings.py`

```
MIDDLEWARE_CLASSES = (
    ...
    'asymmetric_jwt_auth.middleware.JWTAuthMiddleware',
)
```

Create the new models in your DB.

```
python manage.py migrate
```

This creates a new relationship on the `django.contrib.auth.models.User` model. `User` now contains a one-to-many relationship to `asymmetric_jwt_auth.models.PublicKey`. Any number of public key's can be added to a user using the Django Admin site.

The middleware activated above will watch for incoming requests with a JWT authorization header and will attempt to authenticate it using saved public keys.

4.2 Usage

4.2.1 Unencrypted Private Key File

Here's an example of making a request to a server using a JWT authentication header and the requests HTTP client library.

```
from asymmetric_jwt_auth.keys import PrivateKey
from asymmetric_jwt_auth.tokens import Token
import requests

# Load an RSA private key from file
privkey = PrivateKey.load_pem_from_file('~/.ssh/id_rsa')
# This is the user to authenticate as on the server
auth = Token(username='crgwbr').create_auth_header(privkey)

r = requests.get('http://example.com/api/endpoint/', headers={
    'Authorization': auth,
})
```

4.2.2 Encrypted Private Key File

This method also supports using an encrypted private key.

```
from asymmetric_jwt_auth.keys import PrivateKey
from asymmetric_jwt_auth.tokens import Token
import requests

# Load an RSA private key from file
privkey = PrivateKey.load_pem_from_file('~/.ssh/id_rsa',
    password='somepassphrase')
# This is the user to authenticate as on the server
auth = Token(username='crgwbr').create_auth_header(privkey)

r = requests.get('http://example.com/api/endpoint/', headers={
    'Authorization': auth
})
```

4.2.3 Private Key File String

If already you have the public key as a string, you can work directly with that instead of using a key file.

```
from asymmetric_jwt_auth.keys import PrivateKey
from asymmetric_jwt_auth.tokens import Token
import requests

MY_KEY = """-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCh3FtGHks62gHd
KF/oreZGfswTsOijlCbmHvYhO34TpTSXqpcZ1UFOPReFBU2caOdlMbNTshpwjDVr
/TepUcl9xzQqLuKDthI8wyXRZKnSbTzRiWwJn72D5YboOuCOkZBTvJoGE2wq1HkM
/bRubzjXVL1UupXYYQ7MEqkHXT+XCFFm6/9CPuhvBKp1ULMw1vu3kseobQzE4XsF
5gQtcipMQoV9aRnK1cICeYL2GT1G3NRn+WvIPVSAIdnXqA+2Y90VXt+43wUE2ttp
AKV3PpXodUOOw9XE+ZVizBXyoicbyQlSmbjyz08BZ+CLgcIaYmCf4itt53a2VF/v
```

(continues on next page)

(continued from previous page)

```
ePHIKBfRAgMBAAECggEBAIUeIGbzhTWalEvZ578KPkeeAqLzLPFTaAZ8UjqUniT0
CuPtZaXWUIZTEiPRb7oCQMRl8rET2lDTzx/IOl3jqM3r5ggHVT2zoR4d9N1YZ55r
Psipt5PWr1tpiuE1gvdd2hA0HYx/rscuxXucsCbfDCV0SN4FMjWp5SyK8D7hPuor
ms6EJ+JgNWGJvVKbnBXrtfZtBaTW4BuIu8f2WxuHG3ngQl4jRR8Jnh5JniMROxy8
MMx3/NmiU3hfhnhU2l1tQTn1t9cvciOF+DrZjdv30h1NPbexL+UczXFWb2aAYMtC
89iNadfqPdMIZF86Xg1dgLaYGOUa7K1xSCuspvUI2lECgYEA1tV9fwSgNcWqBwS5
TisaqErVohBGqWB+74NOq6SfV9zM226QtrrU8yNlAhxQfwjDtqnAon3NtvZENula
dsev99JLjtJFfV7jsqgz/ybEJ3tkEM/EiQU+eGfp58Dq3WpZb7a2PA/hDnRXsJDp
w7dq/fTzkAmlG02CxpVDCc9R2m0CgYEAwOBPD6+zYQCguXxk/3COQBVpjtFzouqZ
v5Oy3WVxSw/KCRO7/hMVCAAWI9JCTd3a44m8F8e03UoXs4u1eR49H5OufLilT+lf
ImdbAvQMHb5cLPr4oh884ANfJih71xTmJnAJ8stX+HSGkKxs9yxVYoZWTGi/mw6z
FttOYzAx1HUCgYBR9GWIlBIuETbYsJOkX0svEkVHKuBZ8wbZhgT387gZw5Ce0SIB
o2pjSohY8sY+f/BxeXaURlu4xV+mdwTctTbK2n2agVqjBhTk7cfQOVCxIyA8TZZT
Ex4Ovs17bJvsVYrC1DfW19PqOLXPFKko0YrOUKittRA4RyxxZzWIw38dTQKBgCEu
tgth0/+NRxmCQDH+IEsAJA/xEu7lY5wlAfG7ARnD1qNnJMGacNTWhviUtNmGoKDi
0lxY/FHR7G/0Sj1TKXrkQnGspqwv3zEhDPReHjODy4Hlj578ttFnYxhCgMPJEatt
PRjrSPAyw+/h6kE//FSd/fzZTJWVmtQE2OCRqxD9AoGASiN9htvqvXldVDMoR2F2
F+KRA2lXYg78Rg+dpDYLJBk6t8c9e7/xLJATgZy3tLC5YQcpCkrfoCcztdmOiiVt
Q55GCaDNUu1Ttwlu/6yocwYPPS4pP2/qUUDzzBoCEg+PfXSOAsLrGHQ3YLoqbw/H
DxwoXAVLIrFyhFJdklMTnZs=
-----END PRIVATE KEY-----
"""

privkey = PrivateKey.load_pem(MY_KEY.encode())
auth = Token(username='crgwbr').create_auth_header(privkey)

r = requests.get('http://example.com/api/endpoint/', headers={
    'Authorization': auth
})
```

4.3 API

4.3.1 Keys

class asymmetric_jwt_auth.keys.**PublicKey**(**args*, ***kwds*)

Represents a public key

property allowed_algorithms

Return a list of allowed JWT algorithms for this key, in order of most to least preferred.

property as_jwk

Return the public key in JWK format

property as_pem

Get the public key as a PEM-formatted byte string

property fingerprint

Get a sha256 fingerprint of the key.

classmethod load_openssh(*key: bytes*) → Union[*asymmetric_jwt_auth.keys.RSAPublicKey*, *asymmetric_jwt_auth.keys.Ed25519PublicKey*]

Load a openssh-format public key

classmethod load_pem(*pem: bytes*) → Union[*asymmetric_jwt_auth.keys.RSAPublicKey*, *asymmetric_jwt_auth.keys.Ed25519PublicKey*]

Load a PEM-format public key

classmethod load_serialized_public_key(*key: bytes*) → Tuple[Optional[Exception], Optional[Union[*asymmetric_jwt_auth.keys.RSAPublicKey*, *asymmetric_jwt_auth.keys.Ed25519PublicKey*]]]

Load a PEM or openssh format public key

class asymmetric_jwt_auth.keys.**RSAPublicKey**(*key: cryptography.hazmat.primitives.asymmetric.rsa.RSAPublicKey*)

Represents an RSA public key

property allowed_algorithms

Return a list of allowed JWT algorithms for this key, in order of most to least preferred.

property as_jwk

Return the public key in JWK format

class asymmetric_jwt_auth.keys.**Ed25519PublicKey**(*key: cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PublicKey*)

Represents an Ed25519 public key

property allowed_algorithms

Return a list of allowed JWT algorithms for this key, in order of most to least preferred.

class asymmetric_jwt_auth.keys.**PrivateKey**(**args*, ***kwds*)

Represents a private key

classmethod load_pem(*pem: bytes*, *password: Optional[bytes] = None*) → Union[*asymmetric_jwt_auth.keys.RSAPrivateKey*, *asymmetric_jwt_auth.keys.Ed25519PrivateKey*]

Load a PEM-format private key

classmethod load_pem_from_file(*filepath: os.PathLike*, *password: Optional[bytes] = None*) → Union[*asymmetric_jwt_auth.keys.RSAPrivateKey*, *asymmetric_jwt_auth.keys.Ed25519PrivateKey*]

Load a PEM-format private key from disk.

class asymmetric_jwt_auth.keys.**RSAPrivateKey**(*key: cryptography.hazmat.primitives.asymmetric.rsa.RSAPrivateKey*)

Represents an RSA private key

classmethod generate(*size: int = 2048*, *public_exponent: int = 65537*) → *asymmetric_jwt_auth.keys.RSAPrivateKey*

Generate an RSA private key.

pubkey_cls

alias of *asymmetric_jwt_auth.keys.RSAPublicKey*

class asymmetric_jwt_auth.keys.**Ed25519PrivateKey**(*key: cryptography.hazmat.primitives.asymmetric.ed25519.Ed25519PrivateK*

Represents an Ed25519 private key

classmethod generate() → *asymmetric_jwt_auth.keys.Ed25519PrivateKey*

Generate an Ed25519 private key.

pubkey_cls

alias of *asymmetric_jwt_auth.keys.Ed25519PublicKey*

4.3.2 Middleware

class asymmetric_jwt_auth.middleware.**JWTAuthMiddleware**(*get_response: Callable[[django.http.request.HttpRequest], django.http.response.HttpResponse]*)

Django middleware class for authenticating users using JWT Authentication headers

authorize_request(*request: django.http.request.HttpRequest*) → django.http.request.HttpRequest

Process a Django request and authenticate users.

If a JWT authentication header is detected and it is determined to be valid, the user is set as `request.user` and CSRF protection is disabled (`request._dont_enforce_csrf_checks = True`) on the request.

Parameters **request** – Django Request instance

4.3.3 Models

class asymmetric_jwt_auth.models.**PublicKey**(**args*, ***kwargs*)

Store a public key and associate it to a particular user.

Implements the same concept as the OpenSSH `~/.ssh/authorized_keys` file on a Unix system.

exception DoesNotExist

exception MultipleObjectsReturned

comment

Comment describing the key. Use this to note what system is authenticating with the key, when it was last rotated, etc.

key

Key text in either PEM or OpenSSH format.

last_used_on

Date and time that key was last used for authenticating a request.

save(**args*, ***kwargs*) → None

Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

user

Foreign key to the Django User model. Related name: `public_keys`.

class asymmetric_jwt_auth.models.**JWKSEndpointTrust**(**args*, ***kwargs*)

Associate a JSON Web Key Set (JWKS) URL with a Django User.

This accomplishes the same purpose of the PublicKey model, in a more automated fashion. Instead of manually assigning a public key to a user, the system will load a list of public keys from this URL.

exception DoesNotExist

exception MultipleObjectsReturned

jwks_url

URL of the JSON Web Key Set (JWKS)

user

Foreign key to the Django User model. Related name: `public_keys`.

4.3.4 Tokens

class asymmetric_jwt_auth.tokens.**Token**(*username: str*, *timestamp: Optional[int] = None*)
Represents a JWT that's either been constructed by our code or has been verified to be valid.

create_auth_header(*private_key:* asymmetric_jwt_auth.keys.PrivateKey) → str
Create an HTTP Authorization header

sign(*private_key:* asymmetric_jwt_auth.keys.PrivateKey) → str
Create and return signed authentication JWT

class asymmetric_jwt_auth.tokens.**UntrustedToken**(*token: str*)
Represents a JWT received from user input (and not yet trusted)

get_claimed_username() → Union[None, str]
Given a JWT, get the username that it is claiming to be *without verifying that the signature is valid*.

Parameters **token** – JWT claim

Returns Username

verify(*public_key:* asymmetric_jwt_auth.keys.PublicKey) → Union[None, *asymmetric_jwt_auth.tokens.Token*]
Verify the validity of the given JWT using the given public key.

4.3.5 Nonces

class asymmetric_jwt_auth.nonce.base.**BaseNonceBackend**

class asymmetric_jwt_auth.nonce.django.**DjangoCacheNonceBackend**
Nonce backend which uses DJango's cache system.

Simple, but not great. Prone to race conditions.

log_used_nonce(*username: str*, *timestamp: int*, *nonce: str*) → None
Log a nonce as being used, and therefore henceforth invalid.

validate_nonce(*username: str*, *timestamp: int*, *nonce: str*) → bool
Confirm that the given nonce hasn't already been used.

class asymmetric_jwt_auth.nonce.null.**NullNonceBackend**
Nonce backend which doesn't actually do anything

log_used_nonce(*username: str*, *timestamp: int*, *nonce: str*) → None
Log a nonce as being used, and therefore henceforth invalid.

validate_nonce(*username: str*, *timestamp: int*, *nonce: str*) → bool
Confirm that the given nonce hasn't already been used.

4.3.6 Model Repositories

class asymmetric_jwt_auth.repos.base.**BaseUserRepository**

class asymmetric_jwt_auth.repos.base.**BasePublicKeyRepository**

class asymmetric_jwt_auth.repos.django.**DjangoUserRepository**

get_user(*username: str*) → Union[None, django.contrib.auth.models.User]
Get a Django user by username

class asymmetric_jwt_auth.repos.django.**DjangoPublicKeyListRepository**

attempt_to_verify_token(*user: django.contrib.auth.models.User*, *untrusted_token:* asymmetric_jwt_auth.tokens.UntrustedToken) → Optional[*asymmetric_jwt_auth.tokens.Token*]

Attempt to verify a JWT for the given user using public keys from the PublicKey model.

class asymmetric_jwt_auth.repos.django.**DjangoJWKSRepository**

attempt_to_verify_token(*user: django.contrib.auth.models.User*, *untrusted_token:* asymmetric_jwt_auth.tokens.UntrustedToken) → Optional[*asymmetric_jwt_auth.tokens.Token*]

Attempt to verify a JWT for the given user using public keys the user's JWKS endpoint.

4.4 Change Log

4.4.1 1.0.0

- Updated cryptography dependency to `>=3.4.6`.
- Updated PyJWT dependency to `>=2.0.1`.
- Added support for EdDSA signing and verification.
- Added support for obtaining public keys via JWKS endpoints.
- Refactored many things into classes to be more extensible.

4.4.2 0.5.0

- Add new `PublicKey.last_used_on` field

4.4.3 0.4.3

- Fix exception thrown by middleware when processing a request with a malformed Authorization header.

4.4.4 0.4.2

- Fix performance of Django Admin view when adding/changing a public key on a site with many users.

4.4.5 0.4.1

- Fix middleware in Django 2.0.

4.4.6 0.4.0

- Add support for Django 2.0.
- Drop support for Django 1.8, 1.9, and 1.10.

4.4.7 0.3.1

- Made logging quieter by reducing severity of unimportant messages

4.4.8 0.3.0

- Improve documentation.
- Drop support for Python 3.3.
- Upgrade dependency versions.

4.4.9 0.2.4

- Use setuptools instead of distutils

4.4.10 0.2.3

- Support swappable user models instead of being hard-tied to `django.contrib.auth.models.User`.

4.4.11 0.2.2

- Fix README codec issue

4.4.12 0.2.1

- Allow PEM format keys through validation

4.4.13 0.2.0

- Validate a public keys before saving the model in the Django Admin interface.
- Add comment field for describing a key
- Make Public Keys separate from User in the Django Admin.
- Change key reference from User to settings.AUTH_USER_MODEL
- Adds test for get_claimed_username

4.4.14 0.1.7

- Fix bug in token.get_claimed_username

4.4.15 0.1.6

- Include migrations in build

4.4.16 0.1.5

- Add initial db migrations

4.4.17 0.1.4

- Fix Python3 bug in middleware
- Drop support for Python 2.6 and Python 3.2
- Add TravisCI builds

4.4.18 0.1.3

- Expand test coverage
- Fix PyPi README formatting
- Fix Python 3 compatibility
- Add GitlabCI builds

4.4.19 0.1.2

- Fix bug in setting the authenticated user in the Django session
- Fix bug in public key iteration

4.4.20 0.1.1

- Fix packaging bugs.

4.4.21 0.1.0

- Initial Release

4.5 License

PYTHON MODULE INDEX

a

INDEX