JWT

note by umluizlima

> JWTs is an easy and safe way to implement authorization

Content

JSON Web Token (JWT) is an open standard for transmitting information as JSON objects.

Its information can be verified and trusted because it is digitally **signed using a secret or a public/private key pair**.

Signed tokens can verify the integrity of the claims contained within. When signed using public/private key pairs it also certifies that the private key owner is the one who signed it.

Format

JWTs are strings in the following format:

`header.payload.signature`

Header typically consists of the token's type and its signing algorithm:

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

Payload contains the claims, of three types:

- Registered: predefined set of optional claims, such as `iss` (issuer), `exp` (expiration time), `sub` (subject), `aud` (audience), etc.;
- Public: can be defined at will or follow IANA JWT Registry;
- Private: custom claims;

```
{
  "sub": "123456",
  "name": "John Doe",
  "admin": true
}
```

Signature is created by combining the **base64Url** encoded header and payload strings and then encrypting it with the chosen algorithm and secret:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Usage

In **authentication**, a JWT is returned when the user successfully logs in. Tokens are credentials, so they should not be kept longer than required.

The user agent then sends the JWT along with requests, typically in the "Authorization" header using the "Bearer" schema:

```
Authorization: Bearer <token>
```

The server's protected routes will check for a valid JWT in the header and allow access to the protected resources if it's valid. If the data needed for verification is contained in the JWT a database query might not be necessary.

Example

JWT signed with a secret:

```
$ pip install pyjwt
$ python
>>> import jwt
>>> secret = "super-secret-string"
>>> payload = {"email": "user@email.com"}
>>> token = jwt.encode(payload=payload,
key=secret, algorithm="HS256").decode("utf-8")
>>> decoded_payload = jwt.decode(jwt=token,
key=secret, algorithms=["HS256"])
```

- Tampered tokens raise `InvalidSignatureError`;
- Expired tokens raise `ExpiredSignatureError` (when the `exp` claim is defined on the payload);

Signing JWTs with a secret is fine for monolithic systems where there's only one application signing and verifying the tokens.

For distributed systems, it is recommended to sign tokens using a public/private key pair where only one application will sign tokens with its private key, but other applications will be able to verify it by using the public key.

A public/private key pair can be generated by:

```
$ ssh-keygen -t rsa -b 4096
```

```
$ pip install cryptography
$ python
>>> import jwt
>>> payload = {"email": "user@email.com"}
>>> token = jwt.encode(payload=payload,
key=private_key,
algorithm="RS256").decode("utf-8")
>>> decoded_payload = jwt.decode(jwt=token,
key=public_key, algorithms=["RS256"])
```

References

- JSON Web Token Introduction;
- JSON Web Tokens with Public Key Signatures by Miguel Grinberg;