JSON Web Tokens

CMPU4023 - Enterprise Application Development

Pre-authentication

- JSON web tokens (JWTs) are an example of a pre-authentication and token scheme
- In pre-authentication schemes, the API caller performs the authentication steps in advance and then uses some issued token to make subsequent API requests
- The issued token, usually temporary in nature, acts as a cheaper-to-implement proxy for having to authenticate each request separately
- In addition to being less resource-intensive, JWT pre-authentication, being stateless, can also be easier to distribute, allowing for better performance and scalability characteristics of the API system

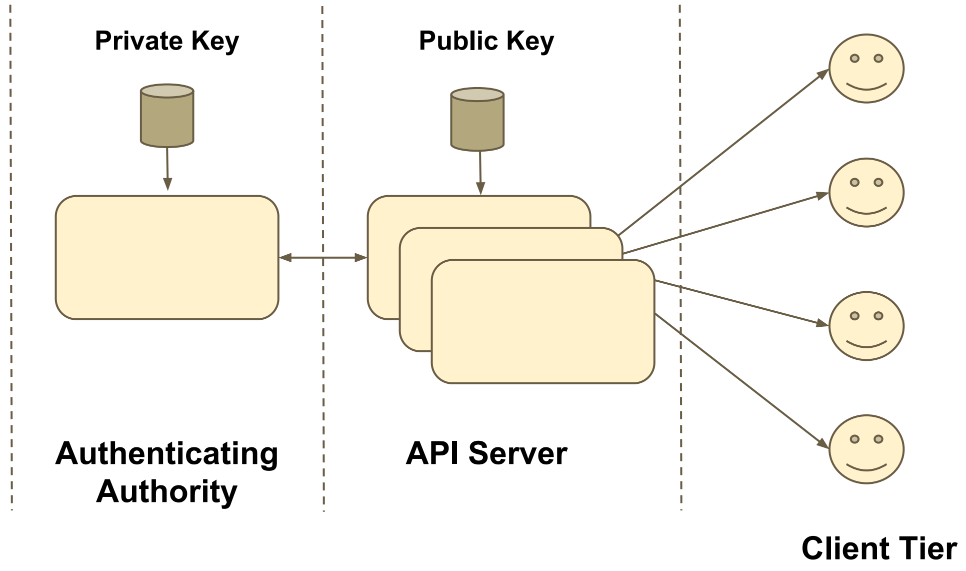
How JWTs Work

- JWT is a simple idea which is based on secure message signing

1. The client issues an authentication request which includes the requester credentials, called *claims*
2. The server verifies the claims for the requester with the authenticating authority
3. Assuming verified, the server then builds a BASE64-encoded token, which includes caller identity information and an expiry timestamp
4. The token is then signed using a secure MAC by the authenticating authority using its secret private key
5. The server responds to the client with the signed token to indicate success
6. In subsequent API requests, the client includes this token (called a bearer token) in the request header
7. The server verifies the token signature using the public key and un-bundles the token information to determine the user identity and token expiry timestamp
8. If successful, the server allows the request forward for further verification and processing

JWT Authentication

- The logical view of JWT authentication looks like something like this:



RFC 7519

- At the time of writing the JWT authentication scheme is covered by an RFC proposed standard which sets out the format of messages and the expected behaviour of participating parties
- The standard describes the format of a token as follows (dot-separated)

base64

Header	**Payload**	**Signature**

`{`
`  "typ": "JWT",`
`  "alg": "HS256"`
`}`
	`{`
	`  "sub": "user:12345",`
	`  "iat": "1300819380",`
	`  "exp": "1300829380"`
	`}`
		Hashed and signed over the previous parts:
		
		● Header
		● Payload

`eyJhbGciOiJIUzI1NiIsI`
`nR5cCI6IkpXVCJ9...`
	`eyJleHAiOjE0ODgxMzMzMjcsImlhdCI`
	`6MTQ4Nzg3NDEyNywic3ViIj...`
		`03f329983b86f7d9a9f5fef853`
		`05880101d5e302afaf...`

Signing Method

- The JWT specification allows for different crypto technologies to be used in the signing process
- The authenticating authority can include a field in the token specifying which hashing and encryption algorithm was used to create the token signature (e.g. HMAC-SHA256 or RSA-SHA256)
- If symmetric key signing is used, then both the issuer and the verifier need to possess the same shared secret
- If asymmetric key signing is used, then the issuer can hold the private key and the verifiers can use the public key

JWT Security Properties

- Operationally, it should be noted that the cryptography described in the RFC 7519 mechanism is used for signing tokens which establishes two important facts about a token

  1. The issuer's identity because of the secret key in its possession
  2. The token's integrity (i.e. that it has not been altered in any way)

- Importantly, RFC 7519 does not provide for token encryption so it is expected that services using JWTs would independently implement some token or transmission encryption system such as TLS
- The token <u>must not</u> be publicly divulged as an attacker could gain the same access as the compromised user

Javascript API

- Install crypto packages and tools with NPM

```
npm install jsonwebtoken
```

- Example usage in JS

```
var jwt = require('jsonwebtoken');

// sign with RSA SHA256 (i.e. using HMAC256 hashing)
var cert = fs.readFileSync('private.key');  // get private key from disk
var token = jwt.sign({
   sub: 'user:12345',
   iat: Math.floor(Date.now() / 1000)
}, cert, { algorithm: 'RS256'});
```

Bearer Token

- Once created, signed and distributed by the server, the token can then be used by the client to authenticate a particular identity (i.e. user)
- The client does this by including the token in each API request in a header field

```
Authorization: BEARER
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE0ODg0NjQ5NTAsImlhdCI6MTQ4ODIwNTc1MCw
ic3ViIjoiOWJhNGRlYWItMDdiMy00NTVjLTk1YTMtMDhkYmQ4MGFkMmUwIn0.NShG3qC2I_LBxZoKX-8UBz_
kFkWKSzJs6JrgDc27P9Lbd-OR9nIsV35Jk2uNvspJH2VyZ7bHS3RR-8CtTexRqcsozrkZsicBWbauX4ph3DU
LGST5ju3tVNXi-NsQoFHij-4BPGNMjjr4DftwnKmJeGA0dI4exZ0Q33AHJVjXNAEVA16x9FOBMkBfXXDQFKI
yJtg46GB3hd7IX8Di4WB8iV-99bsb911UmSb1FKrZQ32zhpFQ0ybms2RGxN1MeMfYeZLjB4c3BpkrV84ucl3
VoXd6qxWzuvWF9r6EyGa9kKxgtGIDOZB0kYCSLLKef9i2EDxyTCRmOK8HJvYwNdH-Vg
```

Expiry and Revocation

- It is common and a good practice to set some expiry time for server-issued tokens to force the API client to periodically reauthenticate
- This can be done including an expiry timestamp in the token which the client is free to read
- In other circumstances, the server may want to revoke a token altogether which can happen if:
    - The token is only intended to be user a fixed number of times (e.g. once)
    - The associated identity credentials have been changed since the token was issued
- To implement revocation, the server (authenticating authority) must keep track of issued tokens and their status - less flexible and less scalable in practice

Summary

- JWTs are an example of a pre-authentication scheme which is very suitable for securing service APIs
- The client and server has access to a shared key (such as a password) which is used to perform the initial authentication
- The issuing and verification of tokens can be separated and carried out by separate components if public key crypto is used
- It is a lightweight and scalable system to implement and can complement different kinds of APIs
- JWTs were developed for and mostly are used with HTTP-based protocols such as REST

JSON Web Tokens

CMPU4023 - Enterprise Application Development