Autenticazione e autorizzazione con Oauth2

Uso di JSON Web Token per gestire l'accesso alle risorse

Sommario

- Il protocollo OAUTH2
- JSON Web Token
- Un esempio di applicazione

Il protocollo OAUTH2

Oauth2 https://oauth.net/2/ è un protocollo standard per gestire l'autorizzazione ad accedere a determinate risorse da parte di applicazioni web. Il protocollo è stato sviluppato dal IETF Working Group Oauth (RFC 6749)

Esso definisce un meccanismo per permettere ad un utente di fornire ad un'applicazione web il permesso di accedere (in modo controllato) a proprie risorse che risiedono su un server in rete.

L'applicazione web ottiene l'autorizzazione per il tramite di un Authorization Server che gestisce l'autenticazione dell'utente e fornisce all'applicazione web la CAPABILITY di accedere alle risorse concesse dall'utente nella forma di un TOKEN (con scadenza).

Già visto … ?

Questo meccanismo è molto utilizzato e lo sperimentiamo (a nostra insaputa) tutte le volte che qualche servizio ci propone di registrarci attraverso le nostre credenziali Google, Facebook, ecc.

(per approfondire:https://developers.google.com/identity/protocols/oauth2)

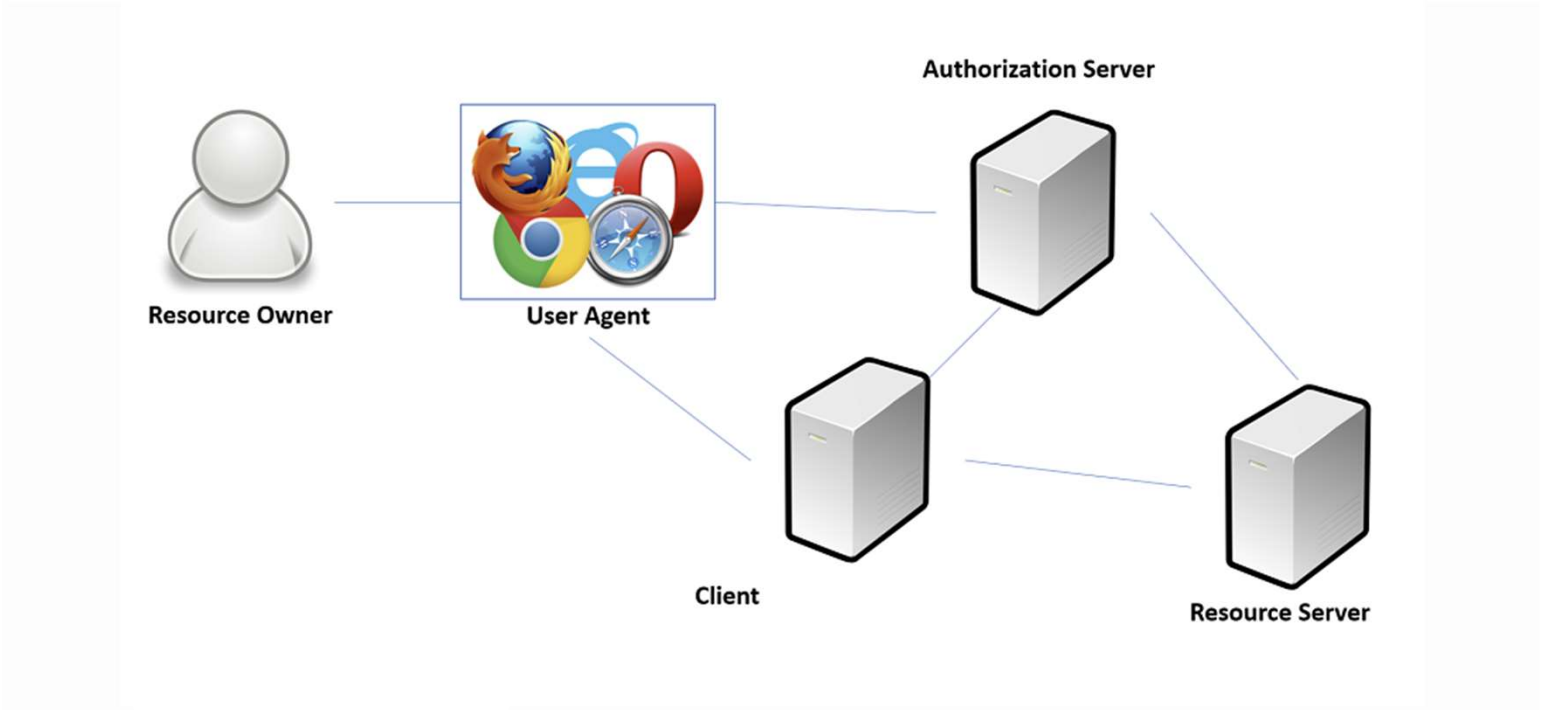
In questo caso l'interazione procede in questo modo: veniamo indirizzati sul servizio di autenticazione scelto (es. Google), qui inseriamo le credenziali di tale servizio (che la nuova applicazione NON potrà vedere), contestualmente ci viene chiesto se siamo disposti a condividere alcune informazioni con la nuova applicazione: per esempio l'indirizzo e-mail, e in qualche caso altre informazioni (per esempio l'età, i contatti, ecc.). Nel momento in cui accettiamo, l'applicazione acquisisce un TOKEN con il quale potrà accedere alle risorse che le abbiamo concesso (e solo a quelle) per un periodo di tempo limitato. Trascorso questo periodo avrà bisogno di aggiornare il Token per poter continuare ad accedere.

Terminologia

Per spiegare il funzionamento di Oauth2 è necessario introdurre alcuni termini, corrispondenti ai RUOLI coinvolti nel protocollo:

- Authorization server
- Resource server (gestore delle risorse dell'utente)
- User (resource owner) – che può utilizzare uno User Agent (browser)
- Client application (applicazione web che ha bisogno di poter accedere alle risorse dello User per offrirgli un certo servizio: per questo lo User dovrà concederle una capability per accedere – entro certi limiti – alle risorse necessarie)

Terminologia



Nel progetto per la distribuzione delle risorse idriche …

Per contestualizzare:

- l'utente (agricoltore o fornitore acqua) interagisce con un'applicazione web che corrisponde al ruolo di *Client Application*,
- il servizio che mantiene i dati corrisponde al ruolo del *Resource Server,*
- l'Authorization server è un servizio esterno (vedremo alcuni esempi: Keycloak, ma anche il server Fitbit) che si occupa di autenticare un utente, e successivamente fornisce alla *Client Application* un Token (con scadenza temporale definita) che questa potrà utilizzare per richiedere al *Resource Server* l'accesso (parziale) alle risorse dell'utente, in base a quanto indicato nel Token.

Tipi di Client Application: Confidential vs Public

Il protocollo prevede diverse forme di acquisizione dei permessi di accesso; ciò è motivato da un'esigenza di flessibilità: deve poter funzionare in diversi casi:

1. La client application è su un server protetto da possibili manipolazioni (e viene quindi classificata come CONFIDENTIAL)
2. La client application è una Single Page Application, costituita da codice javascript, che viene eseguito direttamente nel browser: quindi è potenzialmente attaccabile (in questo caso è classificata come PUBLIC)

Authorization Grant

Lo standard definisce diversi tipi di Authorization Grant. Noi vedremo in particolare quello denominato AUTHORIZATION CODE

Altri tipi, che non vedremo, sono:

- Proof Key of Code Exchange (PKCE)
- Client Credentials
- Device Code
- Implicit (ora sconsigliato e sostituito da PKCE)

Authorization code

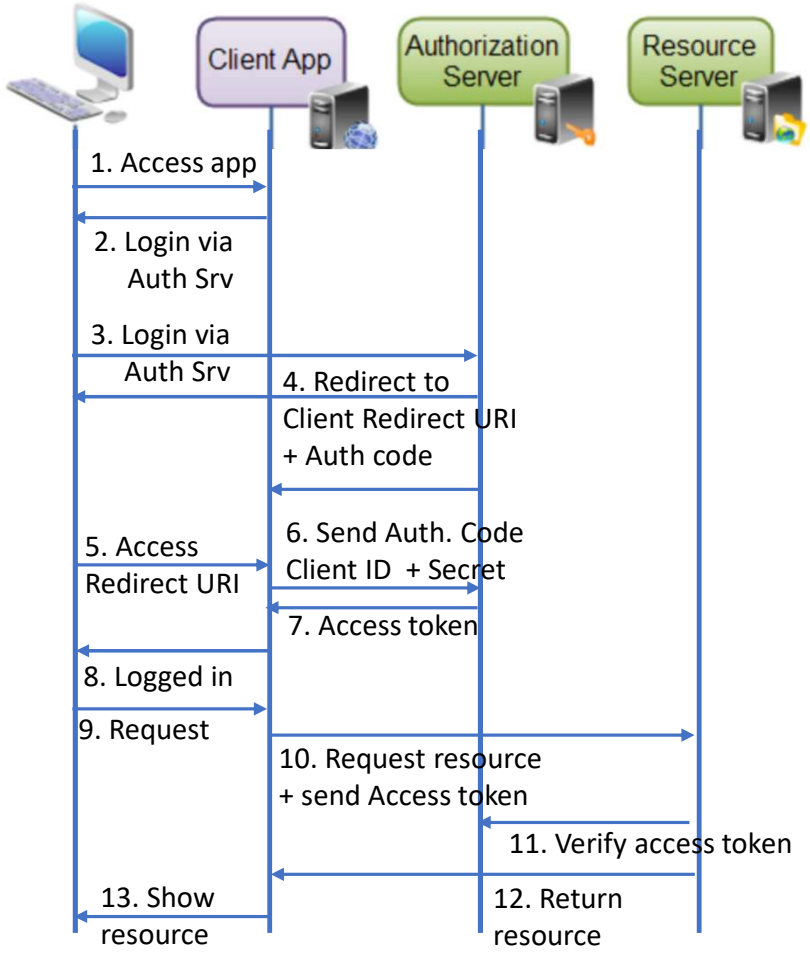
Prima di esaminare il funzionamento del protocollo per questo tipo di authorization grant occorre fare una premessa:

- la Client Application con cui l'utente interagisce per accedere al Resource Server è PREVENTIVAMENTE REGISTRATA presso l'Authorization Server. Nel caso tale applicazione sia considerata «CONFIDENTIAL» al momento della registrazione acquisisce un «SECRET» che si suppone possa proteggere da accessi indesiderati.

Nota: l'ultima assunzione (possesso del «SECRET») non è indispensabile, ma aggiunge un livello di sicurezza ulteriore.

Il protocollo

1.Il proprietario delle risorse (user) accede all'applicazione client.
2. L'applicazione client rinvia l'utente al login presso l'auth. server.
3. Per il login l'utente è ridiretto all'authorization server.
4. Quindi l'utente si autentica con l'authorization server. Se autenticato gli viene chiesto se intende concedere le risorse richieste all'applicazione. Se l'utente conferma è rinviato alla applicazione client (via Client Redirect URI) e con un Auth.Code
5. Il rinvio avviene mandando l'utente alla "redirect URI", specificato all'atto della registrazione. Insieme al rinvio, l'authorization server manda un codice che rappresenta l'autorizzazione (Auth. Code)
6. Quando si accede all'URI nell'applicazione client questa si connette direttamente all'authorization server. E invia il codice di autorizzazione, il client ID (ed eventualmente il segreto)
7. Se l'authorization server li ritiene validi ritorna un *token d'accesso*.
8. Lo user riceve conferma e può iniziare a fare le sue richieste alla client application (9)
10. L' applicazione client può ora usare il token per chiedere accesso alle risorse al resource server. Il token vale sia come autenticazione del client e del proprietario delle risorse (user), che come diritto di accesso alle risorse.
11. Il resource server può verificare la validità del token, in caso positivo risponde restituendo la risorsa (12, 13)



JSON Web Token (1/2)

- Il token usato per l'accesso può assumere diversi formati, uno dei più diffusi è il JWT – JSON Web Token

(per approfondimenti vedere jwt.io dove sono anche indicate varie librerie per la decodifica dei token, l'estrazione delle informazioni contenute nonché la verifica dell'autenticità del token)

- Il JWT comprende tre parti (codificate in BASE64 – base64url3):
  - Header
  - Payload (che potrà contenere varie informazioni, tra cui il tipo di permessi di accesso concessi con questo token)
  - Signature/Encryption data
- Il Resource Server può controllare l'autenticità del token verificando che il token sia firmato dall'Authorization Server (tramite la sua chiave pubblica).

JSON Web Token (2/2)

ESEMPIO:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

DECODIFICATI:

Header: {
"alg": "HS256",
"typ": "JWT"
}
	Payload: {
	"sub": "1234567890",
	"name": "John Doe",
	"iat": 1516239022
	}

Collegandosi a jwt.io e inserendo il JWT sopra, potete verificarne l'autenticità inserendo la parola *secret* nel campo Verify Signature.