GLI STANDARD SU CUI SI BASA KEYCLOAK

Gli standard su cui si basa Keycloak (come molti altri sistemi di autorizzazione e autenticazione) sono:

- Oauth2.0 (per autorizzare un'applicazione ad accedere alle risorse di un dato utente)
- OpenID Connect (Per l'autenticazione degli utenti)
- JSON Web Tokens (JWT) – per la rappresentazione della *capability* di accesso alle risorse

AUTORIZZARE L'ACCESSO CON OAuth 2.0

Oauth2.0 è un framework che facilita l'integrazione di servizi web. Lo scopo è permettere l'accesso a risorse di utenti ad applicazioni di terze parti senza condividere con queste ultime le credenziali dell'utente, e consente anche un controllo fine su quali dati sull'utente vengono condivisi.

Per comprendere OAuth 2.0 Occorre definire i ruoli coinvolti nel protocollo di autorizzazione.

- **Il proprietario delle risorse**: tipicamente è l'utente finale, proprietario delle risorse (tipicamente dei dati) alle quali l'applicazione deve accedere;
- **Il Server dove si trovano le risorse**: questo è il servizio che gestisce le risorse protette;
- **L'applicazione Client**: questa è l'applicazione che deve accedere (in modo protetto ed eventualmente limitato) alle risorse;
- **L'Authorization Server**: questo è il server che emette i permessi (le capabilities) che autorizzano l'applicazione Client ad accedere alle risorse del proprietario delle risorse. Questo è il ruolo di Keycloak.

Il protocollo Oauth2.0 prevede i seguenti passaggi: l'applicazione Client chiede all'Authorization Server di accedere ad una determinata risorsa per conto del proprietario della stessa risorsa. L'Authorization Server emette un access token che consente l'accesso alla risorsa per un tempo limitato. Dopo aver ricevuto l'access token l'applicazione Client fa accesso al server dove si trovano le risorse presentando l'access token come prova dell'autorizzazione acquisita.

Ci sono diverse modalità per ottenere l'access token, di seguito sono indicate le possibili sequenze (flow) per l'acquisizione del token:

- Client credentials flow (si usa quando il client chiede l'autorizzazione ad accedere a proprie risorse)
- Device flow (si usa quando l'autorizzazione viene richiesta per conto di un dispositivo, cosa interessante per i sistemi IoT, lato sensori/attuatori sul campo, dove è impossibile prevedere l'inserimento di credenziali)
- Authorization Code flow: quest'ultimo è quello che sperimenteremo ed è illustrato attraverso un Message Sequence Chart nella figura seguente (tratta dal libro [1]):

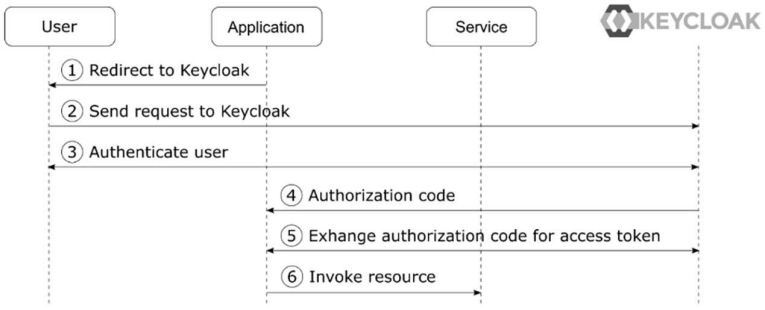

Figure 3.1 – OAuth 2.0 Authorization Code grant type simplified

Commentiamo brevemente i passi elencati nella figura precedente:

1. La Client application prepara una richiesta di autorizzazione (il bottone di login sulla pagina iniziale dell'applicazione) e quando l'utente chiede di accedere, ridirige la sua richiesta sulla pagina per l'autenticazione di keycloak.
2. Il browser dell'utente quindi si sposta su un endpoint del server Keycloak chiamato ***authorization endpoint***
3. Se l'utente non si era già autenticato, allora Keycloak richiede all'utente di inserire le credenziali per consentire alla client application di fare accesso alle risorse protette a nome dell'utente
4. A questo punto la Client application riceve un authorization code da Keycloak all'interno di una authorization response
5. Utilizzando l'authorization code l'applicazione richiede all'Authorization Server di fornirgli un token di accesso, inviando una richiesta al ***token endpoint*** di Keycloak.
6. L'Authorization Server a questo punto invia il token alla client application che potrà finalmente accedere alle risorse secondo le richieste dell'utente, presentando il token ad ogni richiesta che rivolgerà al server che gestisce le risorse.

Il token ha una vita limitata, può essere fornito un refresh token per rinnovarlo senza dover chiedere nuovamente all'utente di riautenticarsi.

Ci sono alcuni ulteriori dettagli per garantire che solo determinate client application possano utilizzare questo meccanismo: le client applications devono essere registrate presso l'authorization server ed essere accessibili attraverso una URL predefinita (l'authorization server non accetterà una redirezione verso una redirect URI diversa da quella registrata ed associata alla client application).

AUTENTICAZIONE DEGLI UTENTI CON OpenID Connect

La fase di autenticazione dell'utente avviene secondo lo standard Open ID Connect, un'estensione di Oauth2.0.

Questa funzionalità è quella che siamo abituati ad utilizzare quando usiamo per esempio le nostre credenziali Google o Facebook per accedere a servizi diversi e indipendenti da Google e Facebook. In questo modo gli utenti possono accedere a numerosi servizi senza dover gestire altrettante credenziali.

Anche OpenID Connect definisce alcuni ruoli:

- End User (l'equivalente del resource owner in Oauth2.0) – è la persona che deve autenticarsi;
- Relying party: si tratta dell'applicazione che vuole verificare l'identità dell'utente utilizzando un servizio di autenticazione esterno (quella che prima abbiamo chiamato client application)
- OpenID Provider: il provider che gestisce l'autenticazione dell'utente (e questo è il ruolo di Keycloak).

OpenID Connect usa il protocollo Authorization Code di OAuth2.0 ma quando avanza la richiesta specifica che ciò che viene richiesta è l'AUTENTICAZIONE e non l'AUTORIZZAZIONE ad accedere. Tuttavia è possibile integrare le due cose e dopo l'avvenuta autenticazione ottenere anche un access token.

La sequenza è quella già vista in precedenza, e la client application può ricevere un ID token (in cambio dell'authorization code) e in questo modo dispone dell'identità dell'utente e può iniziare una sessione con l'utente dove quest'ultimo è autenticato.

Lo standard OpenID Connect definisce anche un preciso formato per l'ID token che può quindi essere letto direttamente dall'applicazione.

I JSON WEB TOKENS

L'uso di JWT ha diversi vantaggi: essendo in formato JSON è facile parsificarli, inoltre sono stati definiti diversi standard che permettono di scambiare token firmati per poter verificarne l'integrità. I token prevedono due parti "header" e "payload" (che contiene un insieme di "claims"), ed eventualmente una firma digitale. Le tre sezioni sono codificate in base64url-encoded e sono concatenate (con un "." di separazione tra la prima e la seconda, e tra la seconda e la terza).

Nel caso di token firmati, si potrà verificarne l'integrità utilizzando la chiave pubblica di chi ha emesso il token.

Per le operazioni di verifica dei token vale la pena di fare riferimento alle numerose librerie disponibili (es. vedere jwt.io per un elenco di librerie affidabili).

PASSI DI INSTALLAZIONE DI KEYCLOAK E ALCUNE OPERAZIONI DI AMMINISTRAZIONE

Per installare keycloak scaricare lo zip contenente il server (consiglio di utilizzare la versione messa a disposizione sul DIR: non è la più recente, ma è quella per cui il codice di esempio su cui faremo sperimentazioni è testato), estrarre il contenuto del file scaricato in una cartella, aggiungere un utente amministratore con il seguente comando (KC_HOME è il pathname della directory dove è stato estratto il file .zip):

> To create an admin account on Linux or macOS, execute the following command in a terminal:

```
$ cd $KC_HOME
$ bin/add-user-keycloak.sh -u admin -p admin
```

> On Windows, execute the following command:

```
> cd %KC_HOME%
> bin\add-user-keycloak.bat -u admin -p admin
```

e poi avviare il server con il seguente script:

.\bin\standalone.bat si Windows oppure ./standalone.sh su Linux o Mac

In questo modo il server risponde sulla porta 8080. Se si vuole cambiare porta usare la seguente opzione:

./standalone.sh -Djboss.socket.binding.port-offset=363 (in questo caso la porta sarà 8080+364 = 8443)

Ora collegarsi a: localhost:8080/auth per accedere agli strumenti di amministrazione.

DUE ESEMPI DI APPLICAZIONI CHE UTILIZZANO KEYCLOAK

Il testo [1] propone alcuni esempi di applicazioni che mostrano come utilizzare i servizi di keycloak, ed inoltre sono molto utili per capire i passaggi del protocollo descritti nelle precedenti sezioni.

Queste applicazioni di esempio si possono scaricare liberamente da github dal seguente link:

https://github.com/PacktPublishing/Keycloak-Identity-and-Access-Management-for-Modern-Applications

LA PRIMA APPLICAZIONE

La prima applicazione si può utilizzare come test per verificare il corretto funzionamento di Keycloak dopo una prima installazione. Dopo aver scaricato il codice da github, la prima applicazione si può trovare nella sottocartella ch2 (riferita al capitolo 2 del libro).

Il codice è stato testato e funziona con versioni di keycloak non aggiornatissime: nel frattempo sono stati cambiati alcuni dettagli ed in particolare alcuni endpoint di keycloak, per questo sul DIR potete trovare uno .zip contenente la versione 12.0.4 di keycloak alla quale il testo (e il codice di esempio) si riferiscono.

L'applicazione di esempio si compone di due parti: una client application (frontend) e un resource server (backend). Per poterle provare occorre aver installato keycloak e creato un "realm" myrealm, registrato una client application "myclient", ed un utente "myuser" con uno specifico ruolo "myrole" (vedere le slide per alcune indicazioni su come effettuare queste operazioni: è anche possibile vedere le operazioni in alcuni video sul canale youtube indicato sul repository di github: https://www.youtube.com/playlist?list=PLeLcvrwLe187DykEKXg-9Urd1Z6MQT61d)

Il Frontend è una Single-Page Application implementata in JavaScript (usiamo Node.js) che prevede poche semplici azioni: login con Keycloak, mostra il nome utente estratto dal token, mostra l'ID token, mostra l'Access token, invoca un endpoint del backend che richiede il token per restituire una risorsa (fittizia). – risponde su localhost:8000 (e su keycloak l'applicazione sarà stata registrata con l'indicazione che la URI alla quale rinviare il controllo dopo l'autenticazione deve avere la forma http://localhost:8000/*)

Il backend è un server con interfaccia API REST, è anch'esso implementato in Node.js, e ha due soli endpoints: /public (non richiede autorizzazioni di accesso), /secured (richiede autorizzazioni di accesso e in particolare è accessibile solo ad utenti che abbiano ruolo "myrole" definito in "myrealm"). Per provare il suo funzionamento in modo diretto si può accedere a http://localhost:3000 e si può subito verificare che il link verso il "public endpoint" risponde con un messaggio "Public message!" mentre il "secured endpoint" risponde con un messaggio di errore (Access denied!) perché la richiesta non è corredata di un access token valido (quello che invece il frontend fornirà previa autenticazione dell'utente).

Il frontend ha la possibilità di richiedere un nuovo access token, utilizzando il refresh token, per evitare di chiedere all'utente di riautenticarsi quando scade l'access token stesso. Insieme al nuovo access token si ottiene anche un nuovo refresh token e la procedura si può ripetere più volte.

E' possibile osservare alcuni dei passaggi (in particolare lo scambio di authorization code e token tra client application e keycloak) osservando i messaggi scambiati tramite gli "strumenti per sviluppatori" del browser (sezione Rete).

Si può quindi provare ad esaminare il token estratto nel formato base64 su jwt.io per verificare la corrispondenza dei suoi contenuti con quelli mostrati dal frontend. E' anche possibile verificare la validità del token prelevando la chiave pubblica di keycloak (riferita al realm "myrealm") che si può ottenere tramite l'endpoint di keycloak: http://localhost:8080/auth/realms/myrealm

Per poter osservare da vicino tutti i passaggi è anche possibile sperimentare il programma che si può trovare nella sottocartella ch4 del software scaricato da github

BIBLIOGRAFIA:

[1] *Keycloak – Identity and Access Management for modern Applications*, Stian Thorgersen, Pedro Igor Silva, 2021 Packt Publishing

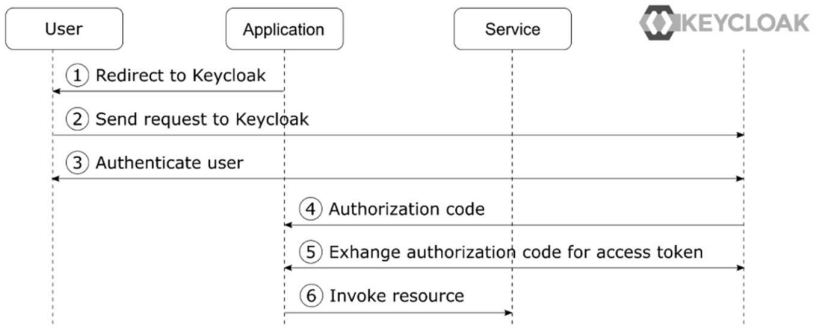

Figure 3.1 – OAuth 2.0 Authorization Code grant type simplified