Cloud Native Apps sichern mit OAuth2/OIDC

Veranstaltung
Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen
Michael Theis
SS 2018

Abgabe
01.06.2018

Christian Keller

Inhaltsverzeichnis

Einleitung 3

OAuth 2 3

Authorisation Code Grant Flow *4*

Implicit Grant Flow *6*

Resource Owner Credentials Grant Flow *7*

Client Credentials Grant Flow *8*

OpenID Connect 8

Vergleich 9

Gegenüberstellung OAuth 2.0 und OpenID Connect *9*

Tokenbasierte Verfahren vs. HTTP Basic Authentication *10*

Tokenbasierte Verfahren vs. Form Based Login *10*

Cloud Native App mit OAuth 2.0 sichern 11

Fazit 14

Quellen 14

Einleitung

Fast jede Person nutzt das Internet um Informationen zu erhalten oder zu kommunizieren. Dabei teilt sich die ganze Welt das Medium Internet. Was aber nicht geteilt werden soll, sind die privaten oder geheimen Inhalte der Informationen oder Unterhaltungen. Deswegen hat sich Verschlüsselung im World Wide Web stark verbreitet. Jeder gängige Browser zeigt dem Nutzer heutzutage an, ob die Kommunikation verschlüsselt ist und gibt dem Nutzer so ein Gefühl von Sicherheit. Wird auf personalisierte Daten oder Dienste zugegriffen, muss das System wissen wer zugreifen möchte und bietet zusätzlichen Schutz durch einen Login. Dieser beinhaltet eine Authentifizierung, die bestätigen soll, dass der zugreifende Nutzer der ist, für den er sich ausgibt und eine Autorisierung, die die Zugangsinformationen verwendet und überprüft, ob der Nutzer auch den richtigen Schlüssel für das Schloss vor den Daten hat. In der Softwareentwicklung ist es daher wichtig ein geeignetes Verfahren zu finden, das es ermöglicht die Informationen entsprechend zu Sichern. Für die Eignung müssen viele Eigenschaften des Systems in Betracht gezogen werden. Heute geht der Trend zu Cloud-Computing und somit beispielsweise horizontale Skalierung oder Verteilung wichtige Eigenschaften. Im Folgenden wird ein geeignetes Verfahren, das dem aktuellen Stand der Technik entspricht, beschrieben.

OAuth 2

Das Framework OAuth 2 beschreibt ein Verfahren zur Authentifizierung und Autorisierung. Dabei gibt es vier Rollen. Die erste Rolle ist der „Resource Owner“ oder auch Nutzer, der auf eine Ressource zugreifen möchte und dazu seine Informationen bei einem Anbieter freigeben muss. Als zweites spielt der Client eine wichtige Rolle. Er ist die Anwendung, die der Nutzer verwendet um auf die Ressourcen zuzugreifen. Der „Authorization Server“ ist eine API-Schnittstelle wie sie Github, Google, Facebook oder auch ein eigenes System bereitstellen kann. Über diese Schnittstelle können Anwendungen ihre Nutzer authentifizieren und auch autorisieren. Eine weitere Rolle ist der „Resource Server“. Er verwaltet die zugreifbaren Informationen oder Ressourcen und muss dabei die Authentifizierung und Autorisierung überprüfen, bevor er den Zugriff gewährt. Nachdem nun die einzelnen Rollen beschrieben sind, wird im Folgenden das

Protokoll näher beschrieben. Das Protokoll beschreibt verschiedene Abläufe um die Authentifizierung durchzuführen, die in den nächsten Abschnitten genauer beleuchtet werden.

Authorisation Code Grant Flow

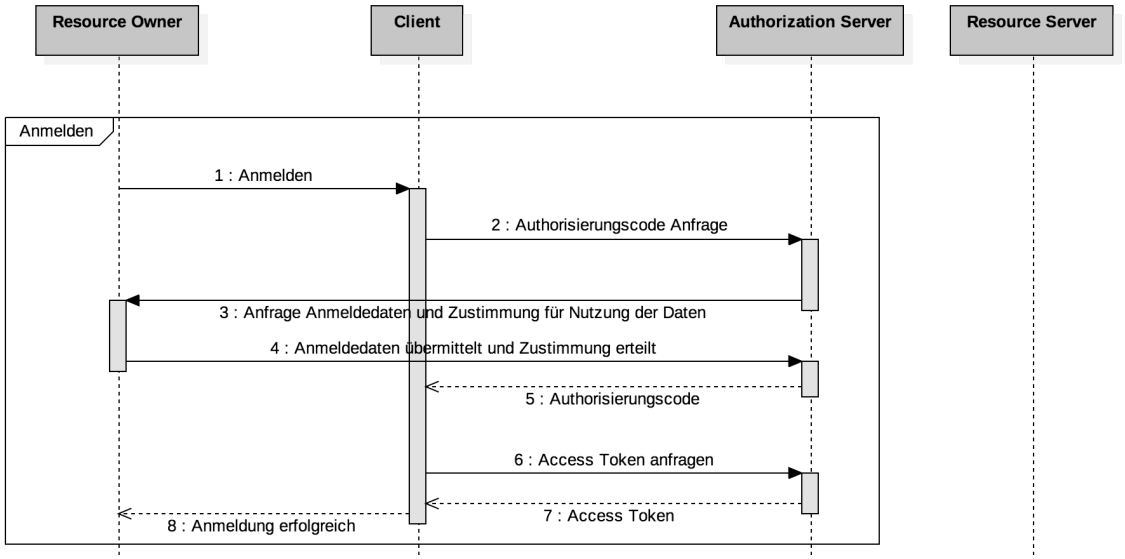

Abbildung 1 – OAuth 2.0 "Authorisation Code Grant" Flow (selbst erstellt)

Den Anfang stellt der Anmeldevorgang dar. Der Nutzer möchte SourceTree (ein Git-Client von Atlassian) verwenden, um auf ein privates Github-Repository zuzugreifen. Der Nutzer versucht sich bei dem Anbieter anzumelden und der Client fordert ihn auf, sich bei Github an dem Authentifizierungsendpunkt über eine Webseite anzumelden. Der Client ist hier die Anwendung SourceTree und der „Authorization Server" ist Github. Damit wären die Nachrichten 1 - 3 bereits abgeschlossen. Nachdem der Nutzer seine Anmeldedaten eingibt, wird er gefragt, ob SourceTree auf die Benutzerdaten zugreifen darf. Ist die Zustimmung erteilt, so sendet Github dem Client einen „Authorization Code". Dieser Code bestätigt dem Client, dass die Authentifizierung erfolgreich war und dass nun mit dem Code ein „Access Token" bei dem Anbieter an dem Tokenendpunkt angefragt werden kann. Außerdem ermöglicht es dieser Code, dass der Client keine Anmeldeinformationen direkt verarbeitet, da die Anmeldung direkt zwischen „Authorization Server" und dem Nutzer stattfindet. Außerdem wird der „Access Token" nicht an Dritte (inklusive „Resource Owner") weitergegeben. In der Spezifikation ist der

Token als Text beschrieben, der eine Vollmacht für den Zugriff auf geschützte Ressourcen darstellt. Damit ist der Anmeldevorgang abgeschlossen.

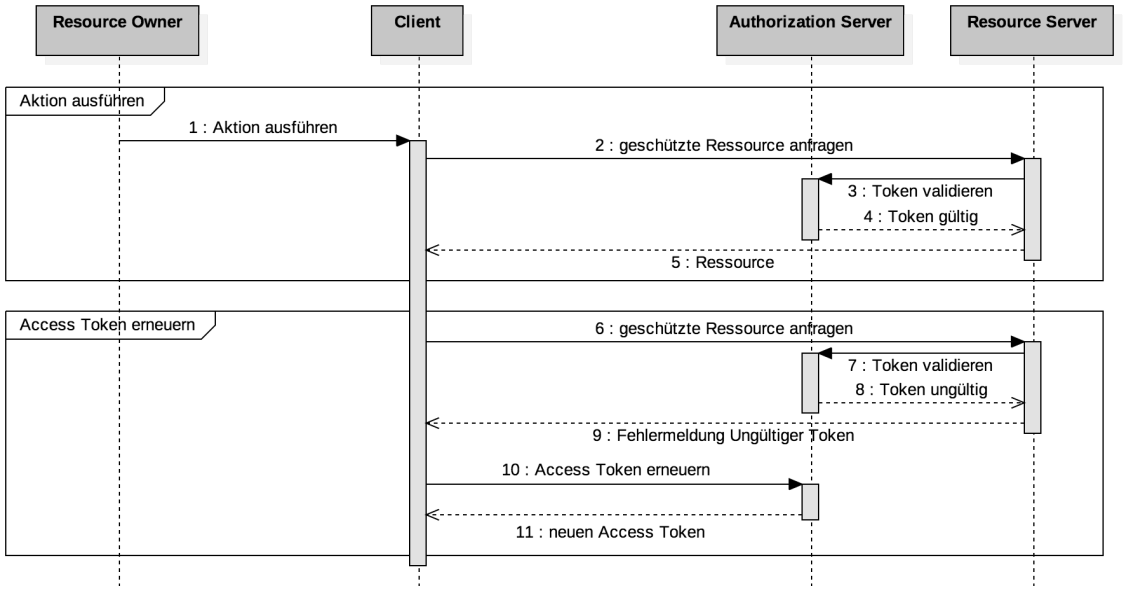

Abbildung 2 - OAuth 2.0 Resourcenzugriff (selbst erstellt)

Greift der Nutzer nun auf das Repository zu, so wird der Access Token mit der Anfrage an den „Resource Server" gesendet. In diesem Beispiel ist das ebenfalls Github, kann aber prinzipiell auch ein anderes System sein. Der Token wird nun an dem „Authorization Server" validiert. Ist er gültig und kann durch den „Resource Server" validiert werden, so erhält der Nutzer die angeforderten Daten.
Versucht der Nutzer mit einem ausgelaufenen Token auf die Daten zuzugreifen, so kann der Client über den optionalen Mechanismus eines „Refresh Tokens" eine weitere Vollmacht einholen und spart damit einen weiteren Anmeldevorgang. Ist die neue Legitimation abgeschlossen, so erhält der Nutzer wieder die benötigten Daten. Bietet der Anbieter oder das eigene System keinen „Refresh Token" muss sich der Nutzer erneut anmelden.

Implicit Grant Flow

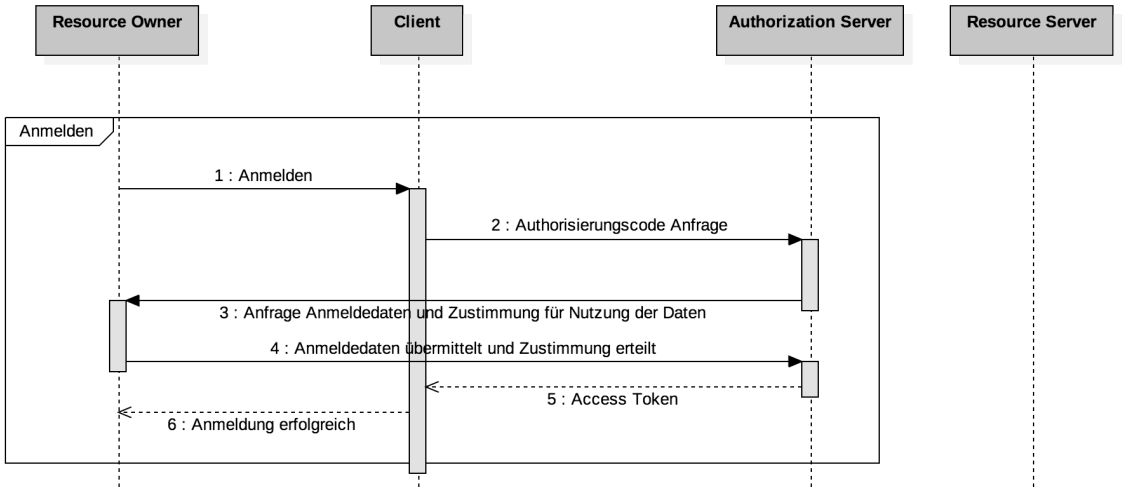

Abbildung 3 - OAuth2 "Implicit Grant" Flow (selbst erstellt)

Eine Alternative stellt der Ablauf „Implicit Grant“ dar. Es handelt sich um eine Abwandlung des „Authorisation Code“ Ablaufs, der für Webanwendungen optimiert ist. Dieser Ablauf ist nicht so sicher wie der „Authorisation Code“ Ablauf, da Dritte auf Browser-Daten jederzeit zugreifen können und der „Access Token“ potentiell nicht geheim gehalten werden kann. Aus Effizienzgründen wird auf einen „Authorisation Code“ und so auf die Kommunikation mit zwei verschiedenen Endpunkten verzichtet und der „Access Token“ wird als URL-Parameter an den Client übergeben. „Bei der impliziten Gewährung werden insbesondere aus Sicherheitsgründen keine Aktualisierungstoken ausgegeben. Ein Aktualisierungstoken hat einen breiteren Umfang als Zugriffstoken und gewährt mehr Rechte, weshalb der Schaden bei Verlust auch größer sein kann.“ [Microsoft, 2016]

Resource Owner Credentials Grant Flow

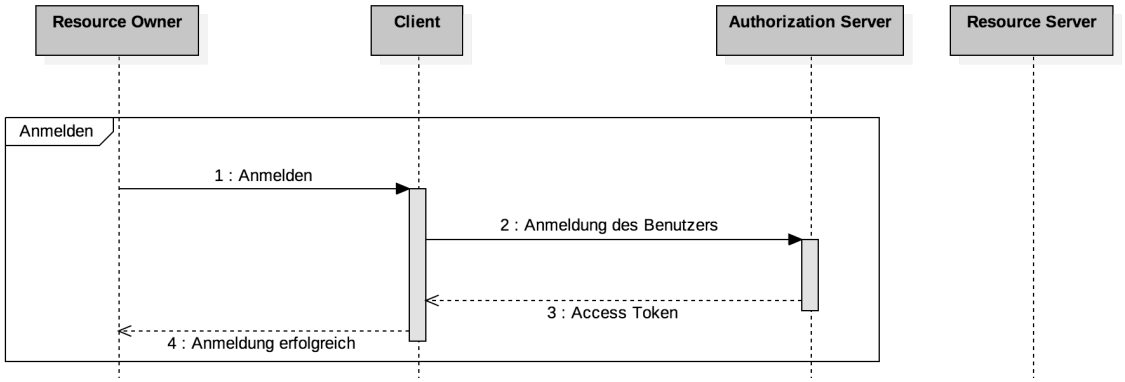

Abbildung 4 - OAuth 2.0 "Resource Owner Credentials Grant" Flow (selbst erstellt)

Der Ablauf des „Resource Owner Credentials Grant" ist eine analoge Umsetzung zu der grundsätzlichen HTTP-Authentifizierung (Basic Auth). Der Benutzer meldet sich mit Benutzername und Kennwort beim Client an, der wiederum die Daten an den „Authorization Server" an den Tokenendpunkt weiterleitet. Der „Authorization Server" validiert diese Daten und gewährt bei erfolgreicher Anmeldung einen „Access Token". Bei erfolgreicher Anmeldung ist der Client angehalten die Benutzerdaten zu verwerfen, da das Token ausreicht um Zugriff zu erlangen. Damit ist die Authentifizierung abgeschlossen und es kann wie üblich auf Ressourcen zugegriffen werden bzw. das Token erneuert werden, wenn es abgelaufen ist.

Client Credentials Grant Flow

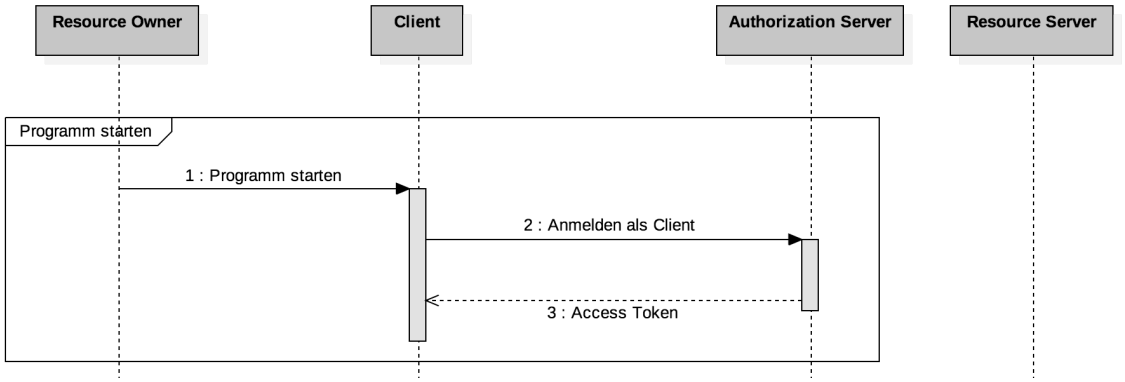

Abbildung 5 - OAuth 2.0 "Client Credentials Grant" Flow (selbst erstellt)

Die Spezifikation beschreibt noch eine weiter Variante um ein "Access Token" zu erhalten. Diese Art ist nur für vertrauenswürdige Anwendungen gedacht, denn es wird nicht der Benutzer authentifiziert, sondern der Client selbst. Dazu erhält der Client geheimes Wissen, mit dem er sich bei dem „Authorization Server" anmelden kann. Ist die Authentifizierung abgeschlossen und der Client besitzt das Zugriffstoken, wird auch hier wie in den anderen Abläufen auf Ressourcen zugegriffen.

OpenID Connect

OAuth 2.0 authentifiziert einen Nutzer und gibt einen Access Token aus, um mit diesem auf Ressourcen zuzugreifen. OpenID Connect erweitert dies und sendet im Authentifizierungsprozess neben dem Access Token auch ein ID Token mit. Dieser ID Token ist als JSON Web Token (JWT) standardisiert und enthält Zusatzinformationen. Genauer sieht ein ID Token wie folgt aus:

```
{
   "iss":"joe",
   "exp":1300819380,
   "http://example.com/is_root":true
}
```

Abbildung 6 - JSON Web Token [RFC 7519, 2015]

Ein ID Token entspricht einem Claim Set in JSON Notation. Ein Claim besteht aus einem Bezeichner und einem Wert. Viele Bezeichner sind dabei für OIDC standardisiert[1]. In dem

[1] http://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

Beispiel beschreibt das Claim „iss“ den „Issuer“ bzw. den Aussteller oder „exp“ den Ablaufzeitpunkt. Das ID Token wird als String in einer HTTP-Nachricht versendet und kann als zusätzliche Sicherheitsmaßnahme signiert und verschlüsselt werden. Um das ID Token beispielsweise zu signieren, kann JSON Web Signature (JWS) verwendet werden. Ein ID Token ist in drei Teile unterteilt, die durch einen Punkt getrennt werden und jeweils in Base64 codiert sind. Als erstes startet ein JOSE-Header, der Meta-Informationen des JWT als JSON Objekt enthält. Dann folgt der Inhalt also das JWT und zum Schluss enthält das ID Token noch die Signatur.

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

Abbildung 7 - ID Token in base64 Codierung [RFC 7519, 2015]

Des Weiteren verwendet OIDC für den Client in der Spezifikation den Begriff „Relaying Party“ und für den „Authorization Server“ – Identity Provider. Außerdem geht aus der Spezifikation hervor, dass OpenID Connect nur die Flows „Authorization Code Grant“ und „Implicit Grant“ beschreibt. Erweiternd gibt es einen hybriden Ablauf, der beide Abläufe kombiniert. OIDC spezifiziert einen zusätzlichen Endpunkt, der standardisierte Benutzerinformationen bereitstellt und durch OAuth 2.0 gesichert ist. Google stellt ein Werkzeug[2], das den Authentifizierungsprozess durchspielt und sich für eine genaue Analyse sehr gut eignet, bereit.

Vergleich

Gegenüberstellung OAuth 2.0 und OpenID Connect

OAuth 2.0 ist eine robuste und sichere Möglichkeit Token basierte Authentifizierung zu realisieren. Allerdings besitzt es einen Nachteil. Es spezifiziert keinen standardisierten Weg auf Benutzerdaten zuzugreifen. OpenID Connect versucht diesen Nachteil auszugleichen und erweitert OAuth 2.0 um Identitiätsinformationen und bietet Anwendungen an, mehr Details über ihre Nutzer zu erfahren sofern die Nutzer zustimmen. Meldet sich der Nutzer ab oder läuft ein Token aus, so muss der Authentifizierungsprozess in beiden Frameworks neu angestoßen werden. Läuft ein

[2] https://developers.google.com/oauthplayground

Token aus und ein Aktualisierungstoken wird unterstützt, kann ein neues Zugriffstoken ohne Interaktion des Nutzers angefragt werden.

	OAuth 2.0	OpenID Connect
Authorization Code Grant	Ja	Ja
Implicit Grant	Ja	Ja
Resource Owner Credentials Grant	Ja	Nein
Client Credentials Grant	Ja	Nein
Zusatzinformationen mittels JWT	Nein	Ja
UserInfo-Endpunkt	Nein	Ja

Abbildung 8 - Gegenüberstellung OAuth 2.0 mit OpenID Connect (selbst erstellt)

Tokenbasierte Verfahren vs. HTTP Basic Authentication

HTTP Basic Authentication nutzt den HTTP Header „Authorization" und gibt dabei immer den Benutzernamen und das Passwort als Base64 codierten Text an. Das hat im Vergleich zu tokenbasierten Verfahren den Nachteil, dass eine Anwendung die Benutzerdaten speichert. Tokenbasierte Verfahren speichern nur ein Token, das durch einen Authentifizierungsprozess mit Anmeldeinformationen erzeugt wird. OAuth 2.0 beispielsweise beschreibt im „Authorization Code Grant" Ablauf, dass der Client nie die Anmeldeinformationen des Nutzers verarbeitet. Eine Übertragungsverschlüsselung über SSL/TLS ist in beiden Varianten vorgesehen.

Tokenbasierte Verfahren vs. Form Based Login

Form-Based Login ist nicht genau spezifiziert und beschreibt das Vorgehen eines Logins über ein HTML-Formular. Die Benutzerdaten werden als HTTP-Content an einen Server gesendet. Meistens ist diese Art der Authentifizierung und Autorisierung mit Sessions verbunden und wird nur in Webanwendungen verwendet, da es für andere Clients nicht umsetzbar ist. Tokenbasierte Verfahren sind Client unabhängig und schränken nicht auf bestimmte Technologien ein.

Cloud Native App mit OAuth 2.0 sichern

Spring bietet eine einfache Umsetzung für die Absicherung von Spring Anwendungen mit OAuth 2.0. Genutzt werden kann ein externer Anbieter mit entsprechender API oder der Authorization Server von Spring selbst. Die einfachere und angenehmere Variante ist die Authentifizierung über einen externen Anbieter. Das hat den Vorteil, dass der Nutzer kein neuen Benutzeraccount anlegen muss. Außerdem spart es Kosten und Zeit, wenn die sicherheitsrelevanten Komponenten nicht selbst erstellt werden müssen. Nachteilig ist allerdings, dass als Unternemen eine weniger starke Bindung zum Kunden existiert und selbst keine Kundendaten erhoben werden. Die Sicherung einer Spring Anwendung kann ohne Wissen über die eigentliche Anwendung stattfinden, da es hauptsächlich Konfiguration ist.

Spring bietet eine Annotation `@EnableOAuth2Sso` im Produkt Security und aktiviert damit OAuth 2.0 für die Anwendung. Damit Spring weiß mit welchem Anbieter oder welchem Server der Authentifizierungsprozess abzuschließen ist, muss Konfiguration hinterlegt werden. Für Github kann beispielsweise folgende Konfiguration verwendet werden im YAML Format:

```
security:
  basic:
    enabled: false
  oauth2:
    client:
      client-id: 5fdfe26598c0cd2006ed
      client-secret: 9fc5e*********************************
      access-token-uri: https://github.com/login/oauth/access_token
      user-authorization-uri: https://github.com/login/oauth/authorize
      client-authentication-scheme: form
    resource:
      user-info-uri: https://api.github.com/user
```

Abbildung 9 – Spring Konfiguration (application.yml) (selbst erstellt)

Die Informationen `client-id` und `client-secret` werden vom Anbieter bei der Registrierung der Anwendung vergeben und sind geheim zu halten.

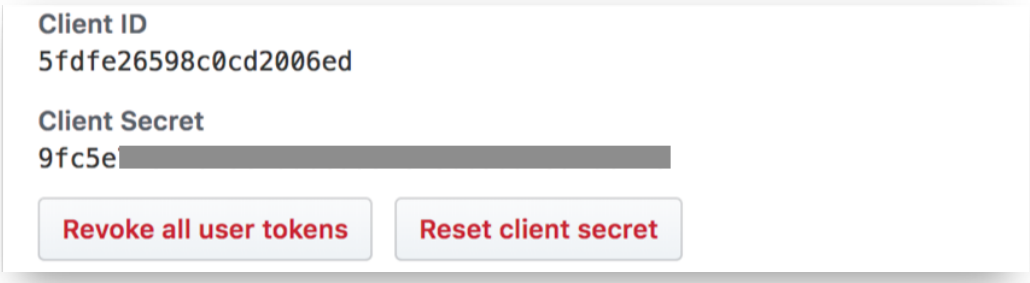

Abbildung 10 - geheimes Wissen der Github Anwendung

Des Weiteren ist bei der Registrierung auch eine Rücksprungadresse anzugeben. Diese Adresse dient dem Anbieter zur Identifizierung des Clients und leitet den Nutzer nach der Anmeldung auf diese Adresse.

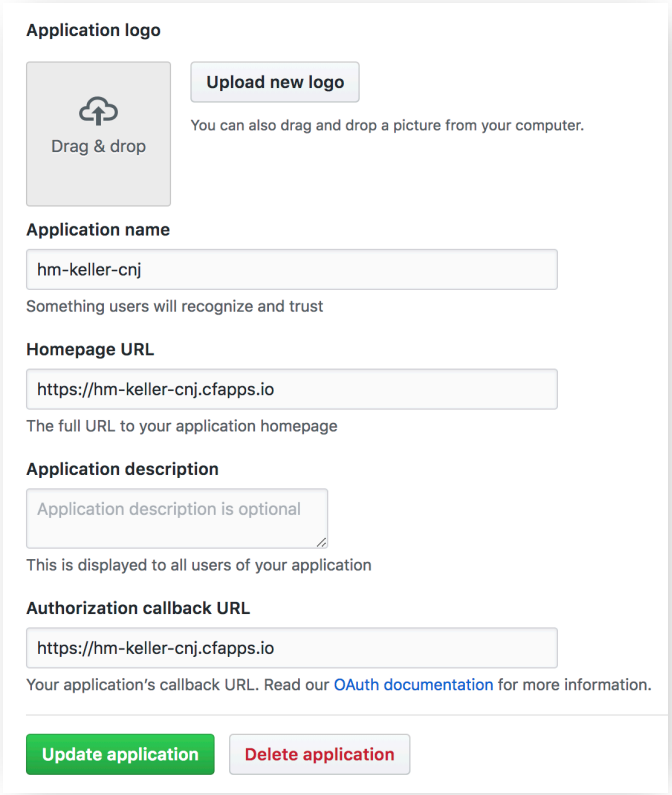

Abbildung 11 - Github Anwendungseinstellungen

Prinzipiell wäre damit der größte Teil erledigt. Startet der Nutzer nun die Anwendung, so wird er direkt zu Github weitergeleitet, muss sich dort authentifizieren und landet auf

der Rücksprungadresse. Ist der Nutzer bereits angemeldet, öffnet sich die Applikation und im Hintergrund erhält diese gegebenenfalls Benutzerinformationen.

Spring sichert automatisch alle Endpunkte durch OAuth 2.0. Nutzt eine Anwendung bestimmte Seiten wie eine Startseite und möchte diese auch ohne Authentifizierung zugänglich machen, so muss eine zusätzliche Konfiguration im Quellcode stattfinden. Im Beispiel ist nur das Anwendungs-Icon frei zugänglich.

```
@Configuration
public class WebConfiguration extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .antMatchers("/favicon.ico").permitAll()
                .anyRequest().authenticated()
                .and().csrf().disable();
    }
}
```

Abbildung 12 - Authentifizierungskonfiguration Endpunkte (selbst erstellt)

Damit ist die Konfiguration einer Spring Anwendung für die Absicherung mit OAuth 2.0 abgeschlossen. Soll in einem Endpunkt auf den Benutzernamen zugegriffen werden, kann der Endpunkt ein Objekt vom Typ Principal erhalten.

```
@GetMapping
public Info getInfo(Principal principal) {
        String username = principal.getName();
        …
}
```

Abbildung 13 - Endpunkt Info mit Principal (selbst erstellt)

Dieses Objekt hat ein Attribut `name`, das den Benutzernamen des angemeldeten Benutzers enthält. Außerdem können nähere Informationen zur Authentifizierung und Autorisierung über den Typen Authentication in einem Endpunkt erfragt werden.

```
@GetMapping
public Info getInfo(Authentication authentication) { … }
```

Abbildung 14 - Endpunkt Info mit Authentication (selbst erstellt)

Der Typ Authentication bietet Informationen zu den zugewiesenen Benutzergruppen, näheren Details wie IP-Adresse des Authentifizierungsservers und den Token.

Fazit

OAuth 2.0 bietet eine Reihe von Abläufen, wie sich Nutzer bei Anwendungen sicher authentifizieren und autorisieren können. Der Client sendet geheime Informationen wie beispielsweise Benutzername und Passwort an einen Server und erhält einen Token bei erfolgreicher Authentifizierung. Diesen sendet er bei jeder weiteren Anfrage mit und erhält so Zugriff auf Daten. OpenID Connect erweitert dies um zusätzliche Informationen in Form eines ID Tokens, das Informationen wie den Aussteller oder ein Ablaufdatum der Information enthält. Meiner Meinung nach ist es leicht eine Anwendung mithilfe von Spring Security durch OAuth 2.0 abzusichern, denn es muss hauptsächlich konfiguriert werden und erfordert keinen großen Entwicklungsaufwand. Außerdem verlagern sich die Benutzerinformationen zu externen Anbietern und der Fokus kann auf die Anwendungsentwicklung gerichtet werden.

Quellen

Microsoft, 2016:

Grundlegendes zum Ablauf der impliziten OAuth2-Gewährung in Azure Active Directory (AD)
https://docs.microsoft.com/de-de/azure/active-directory/develop/active-directory-dev-understanding-oauth2-implicit-grant (Abruf, 28.05.2018)

RFC 6746, 2012:

OAuth 2.0 Spezifikation
https://tools.ietf.org/html/rfc6749 (Abruf, 28.05.2018)

RFC 7519, 2015:

JSON Web Token (JWT)
https://tools.ietf.org/html/rfc7519#section-3.1 (Abruf, 28.05.2018)

OpenID Connect, 2015:

OpenID Connect – Core 1.0 Spezifikation
http://openid.net/specs/openid-connect-core-1_0.html (Abruf, 28.05.2018)

Abbildungsverzeichnis

Abbildung 1 – OAuth 2.0 “Authorisation Code Grant” Flow (selbst erstellt) 4
Abbildung 2 - OAuth 2.0 Resourcenzugriff (selbst erstellt) 5
Abbildung 3 - OAuth2 "Implicit Grant" Flow (selbst erstellt) 6
Abbildung 4 - OAuth 2.0 "Resource Owner Credentials Grant" Flow (selbst erstellt) 7
Abbildung 5 - OAuth 2.0 "Client Credentials Grant" Flow (selbst erstellt) 8
Abbildung 6 - JSON Web Token [RFC 7519, 2015] 8
Abbildung 7 - ID Token in base64 Codierung [RFC 7519, 2015] 9
Abbildung 8 - Gegenüberstellung OAuth 2.0 mit OpenID Connect (selbst erstellt) 10
Abbildung 9 – Spring Konfiguration (application.yml) (selbst erstellt) 11
Abbildung 10 - geheimes Wissen der Github Anwendung 12
Abbildung 11 - Github Anwendungseinstellungen 12
Abbildung 12 - Authentifizierungskonfiguration Endpunkte (selbst erstellt) 13
Abbildung 13 - Endpunkt Info mit Principal (selbst erstellt) 13
Abbildung 14 - Endpunkt Info mit Authentication (selbst erstellt) 13

Erklärung

Hiermit erkläre ich, dass ich die Studienarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

(Unterschrift)