inchorus OAuth2 Plugin Dokumentation

Thomas Gertler

Publication Date: 2023-01-25

Contents

1 Einführung

Das OAuth2 Plugin bietet Ihnen die Möglichkeit für Ihr Gadget eine weitere Authentifizierungsmethode basierend auf dem OAuth 2.0 Protokoll anzubieten, indem es das Gadget Core Plugin erweitert. Dadurch ist ein Login an einem zentralen Authentifizierungsserver (Identity Provider, IP) möglich, um z.B. ein Single Sign-On (SSO) zu realisieren.

Die Authentifizierungsmethode kann analog zur FORM basierten oder BASIC Authentifizierung verwendet werden und ist daher einfach austauschbar oder parallel konfigurierbar. Sie kann, mit Weiterleitung zum IP, als Login am Frontend verwendet werden oder bei z.B. bei API Anfragen direkt den Access Token zur Authentifizierung entgegen nehmen.

Folgende Funktionalitäten stehen zur Verfügung:

- Automatische Weiterleitung zur Login Seite des Identity Providers.
- Automatisches Einloggen, wenn der Nutzer am Identity Provider angemeldet ist.
- Automatischer Logout am Identity Provider konfigurierbar.
- Automatischer Logout und Weiterleitung zur Login Seite des IP bei Ablauf der Token Lifetime.
- Übermittlung des Benutzernamens und des vollen Namens, sofern der Nutzer ihn angegeben hat.
- Auslesen des Access-, Refresh- und ID-Tokens.
- Konfigurieren der Endpoint-URLs zum Identity Provider.
- Vom Login Prozess unabhängige Utility Methoden wie das Auslesen des Access Tokens aus dem Request, Prüfen des Access Tokens oder das Abfragen der User-Infos mit dem Request Token

Das Plugin basiert auf dem Nimbus OAuth 2.0 SDK with OpenID Connect extensions (https://connect2id.com/products/nimbus-oauth-openid-connect-sdk).

Als Identity Provider wird in dieser Dokumentation Keycloak (https://www.keycloak.org/) verwendet und vorgestellt.

2 Wie wird das Plugin verwendet?

Plugin einbinden

Öffnen Sie die Datei pom.xml Ihres inchorus Gadget Projektes und ergänzen Sie folgende Zeilen im Abschnitt "dependencies":

```
<dependency>
 <groupId>de.guh.plugin</groupId>
 <artifactId>inchorus-oauth2-plugin</artifactId>
 <version>1.x.x</version>
</dependency>
```

Initialisieren des OAUTH2 Plugins

Die OAuth Authentifizierung wird auf die gleiche Weise wie die FORM und BASIC Authentifizierung initialisiert und verwendet. Je nach Gadget Version muss dafür unterschiedlicher Code aufgerufen werden. Im folgenden wird die Methode für aktuelle Gadget Versionen beschrieben. Kann bei alten Gadgets ohne `Authorizationhandler.java` Klasse die Gadget Core Version nicht aktualisiert werden, kann die Authentifizierung auf die bisherige Methode verwendet werden, weitere Informationen dazu im Abschnitt Anmerkungen für ältere Gadget Versionen ().

In der `Authorizationhandler.java` Klasse wird für jede zu schützende Resource folgender Code hinzugefügt:

```
ProtectedResource pr = new ProtectedResource(
 ProtectedResource.type_PATH,
 "startswith",
 "/command.html"
);
pr.addRequiredAuthorization(Gadget.gdSecurity.authorization_AUTHENTICATED);
pr.addRequiredAuthentication(Gadget.gdSecurity.authentication_OAUTH2);
Gadget.gdSecurity.addProtectedResource(pr);
```

Zu den bisherigen Authentifizierungsmethoden unterscheidet sich hier nur der Teil `Gadget.gdSecurity.authentication_OAUTH2` mit dem die OAuth Authentifizierung ausgewählt wird. In diesem Beispiel wird für alle Anfragen, die über Command aufgerufen werden, eine OAuth Authentifikation benötigt.

In der `Authenticationhandler.java` Klasse des Gadgets wird daraufhin beim Aufruf der Resource und erfolgreicher Authentifizierung am Identity Provider die Anmeldung am Gadget ausgeführt. Da der eigentliche Authentifizierungsvorgang extern stattfindet, wird hier der authentifizierte Benutzername in `string1` und der volle Benutzername in `string2` übergeben, sofern

der Nutzer diese Informationen am Identity Provider bereitgestellt hat. Diese Daten können dann weiterverwendet und z.B. in der Session gespeichert werden, wie in der folgenden Beispielimplementierung:

```
@Override
public String authenticate(GadgetRequest gdRequest, GadgetSession gdSession,
 GadgetResponse gdResponse, String string1, String string2) {
  gdRequest.log.debug("...authenticate()...[running]");

  if (string1 == null || string1.isBlank())
    return GadgetStatuscodes.ERROR_MISSING_CREDENTIALS;

  // Store username in Session
  gdSession.saveParameterInSession("username", string1);

  return GadgetStatuscodes.SUCCESS_AUTHENTICATE_SESSION;
}
```

Dieser Abschnitt wird nur erreicht, wenn die Authentifizierung am Identity Provider bereits erfolgreich war, durch die Rückgabe von `GadgetStatuscodes.SUCCESS_AUTHENTICATE_SESSION` wird die Authentifizierung und der Login am Gadget bestätigt. Bei weiteren Aufrufen mit der selben Session wird dieser Teil nicht mehr durchlaufen. Beim ersten Anmelden können hier z.B. mit den statischen Utility Methoden der `OAuth2Commands` Klasse weitere Informationen abgefragt werden. Der Access Token kann z.B. mit `getAccessTokenFromRequest(GadgetRequest gdRequest)` aus dem Request ausgelesen werden und mit diesem Token können mittels `getUserInfo(BearerAccessToken accessToken, URI userInfoEndpointURL, GadgetLogger log)` weitere Infos über den Authentifizierten Benutzer abgefragt werden.

Konfigurieren des Gadgets

Um die Authentifizierung zu ermöglichen muss ein Identity Provider eingerichtet sein und das Gadget an ihm registriert werden. Für eine lokale Entwicklungsumgebung lässt sich testweise `Keycloak` verwenden. Eine geeignete Einrichtung von `Keycloak` ist im Abschnitt *Section 3, "Keycloak als Identity Provider"* beschrieben.

Der Identity Provider stellt die notwendigen URLs und IDs zur Verfügung, mit denen er gesteuert werden kann. Diese URLs müssen in die Konfiguration (`config.xml`) des Gadgets eingetragen werden:

```
<oauth2>
  <authendpointurl>
    <![CDATA[http://localhost:8080/realms/inchorus/protocol/openid-connect/auth]]>
  </authendpointurl>
  <tokenendpointurl>
```

```
    <![CDATA[http://localhost:8080/realms/inchorus/protocol/openid-connect/token]]>
  </tokenendpointurl>
  <logoutendpointurl>
    <![CDATA[http://localhost:8080/realms/inchorus/protocol/openid-connect/logout]]>
  </logoutendpointurl>
  <userinfoendpointurl>
    <![CDATA[http://localhost:8080/realms/inchorus/protocol/openid-connect/userinfo]]>
  </userinfoendpointurl>
  <introspectionendpointurl>
    <![CDATA[http://localhost:8080/realms/inchorus/protocol/openid-connect/token/
introspect]]>
  </introspectionendpointurl>
  <clientid>myapp</clientid>
  <clientsecret>09c87729-30a3-46fc-89f0-e8a10099e2a9</clientsecret>
  <synchronizelogout>true</synchronizelogout>
</oauth2>
```

Ist im Identity Provider kein Client Secret eingetragen und damit als "public client" konfiguriert, kann `<clientsecret>` weggelassen werden, ansonsten wird er als "confidential client" behandelt und `<clientsecret>` muss in die Konfiguration eingetragen werden. Mit `<synchronizelogout>` kann gesteuert werden, ob beim Aufrufen der Logout URL des Gadgets der Benutzer auch vom Identity Provider ausgeloggt werden soll. Zu beachten ist, dass der Benutzer im umgekehrten Fall nicht automatisch am Gadget ausgeloggt wird, wenn er sich am Identity Provider ausgeloggt hat. Damit der Benutzer in diesem Fall ausgeloggt wird, muss am Identity Provider eingestellt werden, dass die Logout URL des Gadgets aufgerufen wird, wenn sich der Nutzer am IP ausloggt.

Einloggen über OAuth2

Ist das Gadget konfiguriert und der Identity Provider eingerichtet wird der Benutzer automatisch auf die Login Seite des Identity Providers weitergeleitet, wenn er versucht eine mit dem OAUTH2 Plugin geschützte Seite aufzurufen. Ist der Benutzer bereits am Identity Provider angemeldet wird er automatisch am Gadget eingeloggt. Das bedeutet seine Session wird als authentifiziert festgelegt. Zu beachten ist hier, dass danach nicht bei jedem weiteren Request eine Überprüfung des Access Tokens durchgeführt wird, es wird lediglich das Ablaufdatum des Access Token geprüft. Andernfalls wird nur in der Session des Benutzers überprüft, ob er bereits authentifiziert wurde und die Session noch aktiv ist. Zur weiteren Überprüfung sind in der Session unter `'accesstoken'` der `BearerAccessToken` und unter `'idtoken'` der `JWT` ID Token gespeichert. Wird die Anfrage direkt mit einem vorher generierten Access Token ausgeführt - z.B. bei Anfragen einer API aus Postman oder Swagger heraus - enthält die Session kein zusätzliches ID Token. Das Ablaufdatum wird als UNIX Timestamp (auf Sekunden gerundet) unter `authenticationexpirationtime` in der Session gespeichert.

Weitere Überprüfung des Access Tokens

Um den Access Token zu überprüfen, kann die statische Methode `testAccessToken` aus der `OAuth2Commands` Klasse verwendet werden. Zu beachten ist hierbei, dass dabei eine Anfrage an den Identity Provider gestellt wird, die zusätzliche Überprüfung ist also zeitintensiv und sollte mit Bedacht angewendet werden. Mit `getTokenExpirationTime()`, `getTokenLifetime()` kann der Ablaufzeitpunkt bzw. die restliche Gültigkeitsdauer in Sekunden abgefragt werden. Ist in der Konfiguration die `introspectionendpointurl` gesetzt, wird auch hier eine Anfrage an den IP gestellt, andernfalls wird der Zeitpunkt direkt aus dem Token ausgelesen und ist damit weniger Zeitkritisch. Die Methode `isExpired()` prüft immer nur den Wert im Token und gibt direkt die Antwort als Boolean zurück. Die Methode `accessTokenIntrospection()` liefert die Informationen des Introspection Endpoints und `getUserInfo()` die Informationen über den Benutzer vom User Info Endpoint.

Ausloggen über OAuth2

Ist in der Konfiguration der Wert von `synchronizelogout` auf `true` gesetzt, wird der Benutzer auch am IP abgemeldet, wenn er sich am Gadget abmeldet und damit unter Umständen auch an anderen Anwendungen, an denen er mittels SSO angemeldet ist. Andernfalls bleibt der Benutzer am IP angemeldet und wird nur vom Gadget abgemeldet.

Damit der Benuter am Gadget abgemeldet wird, wenn er sich am IP abmeldet, muss im IP konfiguriert werden, dass die Logout URL des Gadgets aufgerufen wird. In Regel hat diese z.B. die folgende Form: `http://localhost:8080/myapp/logout/`.

Bei Ablauf der Token Lifetime wird bei API Anfragen eine Unauthorized Response zurückgegeben und der Token aus eventuellen Sessions gelöscht. Bei der Anmeldung über das Frontend, wenn ein Refresh Token in der Session gespeichert wurde, wird zuerst versucht den Access Token über den Refresh Token am IP zu verlängern. Ist dies nicht möglich weil der Refresh Token abgelaufen oder der Benutzer am IP abgemeldet ist, wird er wieder zur Login Seite des IP geleitet.

Anmerkungen zur Token Lifetime: Die Token Lifetime ist unabhängig von der Session Lifetime die im Gadget konfiguriert wird, läuft die Session ab, muss auch wieder ein neuer Login vorgenommen werden. Umgekehrt kann der Token ablaufen bevor die Session abläuft und muss dann verlängert werden. Ist der Refresh Token noch gültig wird der Access Token im Hintergrund erneuert und der Benutzer bleibt, ohne Weiterleitung zum IP, am Gadget angemeldet, es findet aber eine erneute Überprüfung des Access Tokens statt. Demnach kann die Lifetime des Access Tokens am IP relativ kurz gewählt werden, während die Lifetime des Refresh Tokens dem Session Timeout des Gadgets bzw. des Identity Providers entsprechen kann.

Anmerkungen für ältere Gadget Versionen

Bei älteren Gadgets ohne Authorizationhandler Klasse muss beim Initialisieren des Gadgets folgender Code aufgerufen werden, um für die entsprechenden Pfade die OAuth2 Authentifikation zu aktivieren:

```
Authentication acOAUTH2 = new Authentication("OAUTH2");
acOAUTH2.addPathProtection("startswith", "/index.html");
acOAUTH2.addPathProtection("startswith", "/command.html");
gdSecurity.addAuthentication(acOAUTH2);
```

Standardmäßig befindet sich die Methode `addAuthentication()` in der Hauptklasse des Gadgets und kann um obigen Code ergänzt werden.

Die Anpassung des `StandardAuthenticationhandlers` erfolgt ebenso wie bei aktuellen Gadgets wie oben beschrieben.

3 Keycloak als Identity Provider

Als erstes muss Keycloak heruntergeladen und auf eine beliebige Methode, die unter Getting Started (https://www.keycloak.org/guides#getting-started) beschrieben ist, mit einem Admin-User eingerichtet und gestartet werden. Danach kann er unter http://localhost:8080/admin konfiguriert werden:

1. Einen "realm" erstellen mit einem beliebigen Namen, der Teil der endpoint URLs wird, bspw. "inchorus":
   - Dropdown List → Create realm → Realm name: "inchorus" → Create
2. Einen Benutzer anlegen, der sich am Gadget anmelden kann:
   - Users → Add user → Username: "testuser", First Name: "Max", Last Name: "Mustermann" → Create
   - Ein Passwort für diesen Nutzer anlegen: Credentials → Set Password (Temporary: Off) → Save password
   - Unter http://localhost:8080/realms/inchorus/account/ lässt sich der Account verwalten und der Benutzer kann über `Sign in` ein- und später wieder ausgeloggt werden (wobei "inchorus" in der URL dem unter Punkt 1 gewähltem realm Namen entspricht)
3. Das Gadget bei Keycloak registrieren und konfigurieren, bspw. für das Gadget "myapp":
   - Clients → Create client → Client Protocol: "openid-connect", Client ID: "myapp" → Next

    - Für einen "confidential client" `Client authentication` auf `On` stellen, `Authorization` kann auf `Off` bleiben.
    - Als `Authentication Flow` wird `Standard flow` ausgewählt, um die authorization code basierte Anmeldung zu aktivieren, die vom OAuth2 Plugin verwendet wird, alle anderen Checkboxen können auf `Off` bleiben. Um z.B. in Postman ohne Weiterleitung über den Browser das Access Token abzurufen, kann zusätzlich `Direct access grants` ausgewählt werden.
    - Save
- Die URL zum Gadget hinzufügen: Clients → myapp → Settings → Root URL: "http://localhost:55080/myapp/" → Save. Zusätzlich die URL mit `*` Wildcard unter `Valid redirect URIs` hinzufügen
- Damit beim Logout am Identity Provider der Benutzer auch am Gadget ausgeloggt wird, muss unter `Logout Settings` die Option `Front channel logout` aktiviert werden und unter `Front-channel logout URL` die Logout URL des Gadgets angegeben werden z.B. `http://localhost:55080/myapp/logout/`
    - für API Anfragen über z.B. Postman kann stattdessen die `Backchannel logout URL` gesetzt werden um eine Browser Weiterleitung zu vermeiden.
- Unter Clients → myapp → Credentials → Client secret den Client Secret für die `config.xml` kopieren

Die Enpoint-URLs entsprechen nun den unter Konfigurieren des Gadgets () in der `config.xml` eingestellten Werten. Eine Übersicht dieser URLs und weiterer Einstellungen findet sich unter `http://localhost:8080/realms/inchorus/.well-known/openid-configuration`.

4 Changelog

Version 1.0.9

- Update auf Gadget Core Version 2.1.24
- Hinzufügen von `accessTokenIntrospection()` um Informationen über den Access Token am Introspection Endpoint abzufragen und optionale Abfrage der Token Lifetime über den Introspection Endpoint, wenn er in der Konfiguration im `introspectionendpointurl` Knoten angegeben ist
- Speichern des Refresh Token in der Session
- Automatischer Refresh des Access Tokens mit dem Refresh Token
- Automatische Weiterleitung an den IP wenn Access- und Refresh Token abgelaufen sind

Version 1.0.8

- Überprüfung der Token Liftime bei API-Requests
- Zusätzliche Utility Methoden zur Abfrage der Token Lifetime, Auslesen und Validieren des Tokens und Abfrage der User Infos unabhängig vom Login Prozess

Version 1.0.7

- Möglichkeit zur Authentifizierung von API-Requests, die direkt den Access Token enthalten.

5 Weitere Informationen

Nähere Informationen zu den Klassen und Methoden finden sie in der **API Dokumentation** (./apidoc/index.html).

Bei Fragen und Anregungen nutzen Sie unser **inchorus Forum** (https://forum.inchorus.de/).

Dieses Dokument erhalten sie **hier** (./inchorus-oauth2-plugin-doc.pdf) auch als PDF.

6 Ihr Kontakt

G+H Systems GmbH

Professionell, effizient und zuverlässig.

Ludwigstraße 8

63067 Offenbach am Main

Deutschland

Telefon: +49 (0) 69 85 00 02 - 0

Fax: +49 (0) 69 85 00 02 - 51

Email: info@guh-systems.de (mailto:info@guh-systems.de)

Web: www.guh-systems.de

7 Rechtliche Hinweise

Die G+H Systems leistet keinerlei Gewähr bezüglich des Inhaltes oder Gebrauchs dieser Dokumentation. Insbesondere werden keine ausdrücklichen oder stillschweigenden Gewährleistungen hinsichtlich der handelsüblichen Qualität oder Eignung für einen bestimmten Zweck übernommen. Die G+H Systems behält sich weiterhin das Recht vor, diese Dokumentation zu revidieren und ihren Inhalt jederzeit und ohne vorherige Ankündigung zu ändern.

Des Weiteren übernimmt die G+H Systems für Software keinerlei Haftung und schließt insbesondere jegliche ausdrücklichen oder impliziten Gewährleistungsansprüche bezüglich der Marktfähigkeit oder der Eignung für einen bestimmten Zweck aus. Außerdem behält sich die G+H Systems das Recht vor, G+H Software ganz oder teilweise jederzeit inhaltlich zu ändern, ohne dass für die G+H Systems die Verpflichtung entsteht, Personen oder Organisationen von diesen Überarbeitungen oder Änderungen in Kenntnis zu setzen.