Cloud Access Manager 8.1.3

How to Configure Single Sign-On for Native Android Applications

If you have any questions regarding your potential use of this material, contact:

One Identity LLC.

Attn: LEGAL Dept

4 Polaris Way

Aliso Viejo, CA 92656

Refer to our Web site (http://www.OneIdentity.com) for regional and international office information.

Patents

One Identity is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at http://www.OneIdentity.com/legal/patents.aspx.

Trademarks

One Identity and the One Identity logo are trademarks and registered trademarks of One Identity LLC. in the U.S.A. and other countries. For a complete list of One Identity trademarks, please visit our website at www.OneIdentity.com/legal. All other trademarks are the property of their respective owners.

Legend

WARNING: A WARNING icon indicates a potential for property damage, personal injury, or death.

CAUTION: A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.

IMPORTANT, NOTE, TIP, MOBILE, or VIDEO: An information icon indicates supporting information.

Cloud Access Manager How to Configure Single Sign-On for Native Android Applications
Updated - October 2017
Version - 8.1.3

Contents

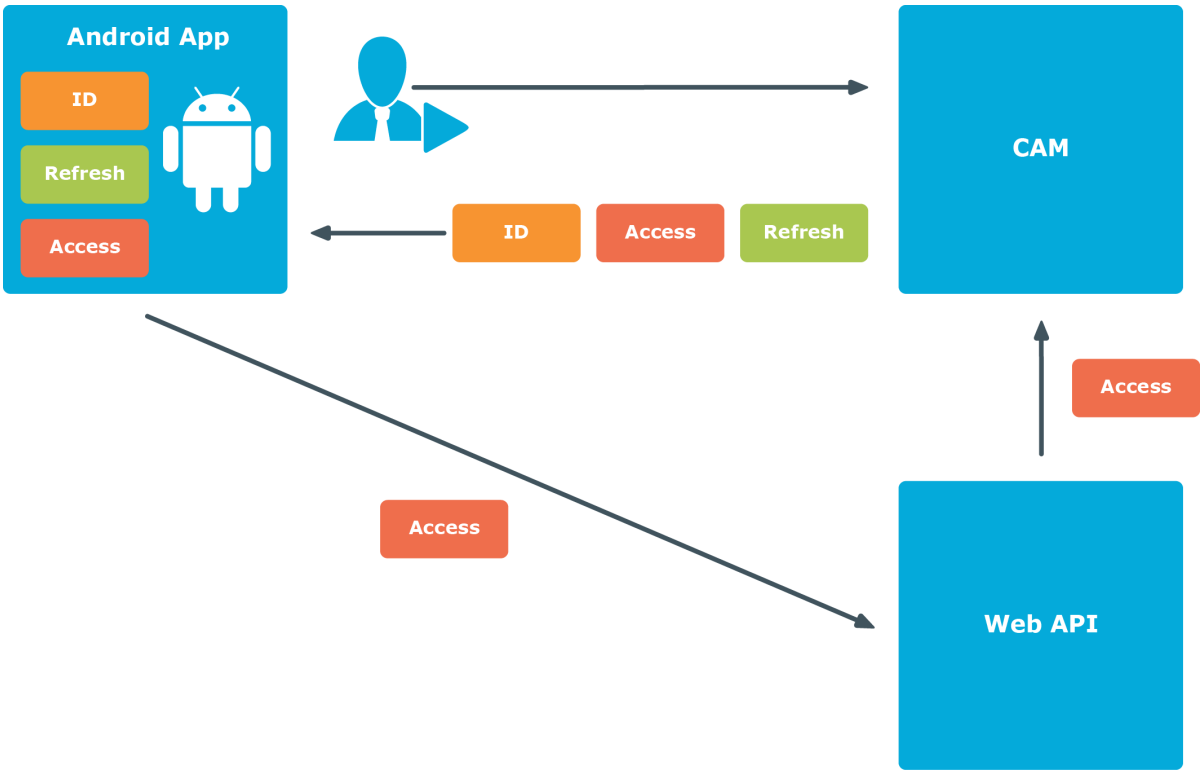
Introduction

This guide describes how to deploy Single Sign-On (SSO) for native Android applications using the OpenID Connect http://openid.net/connect/ protocol.

Overview

Using the OpenID Connect protocol, the Android application authenticates the user against Cloud Access Manager and retrieves a set of three security tokens, as shown in Figure 1. The security tokens are known as the ID Token, Refresh Token and Access Token.

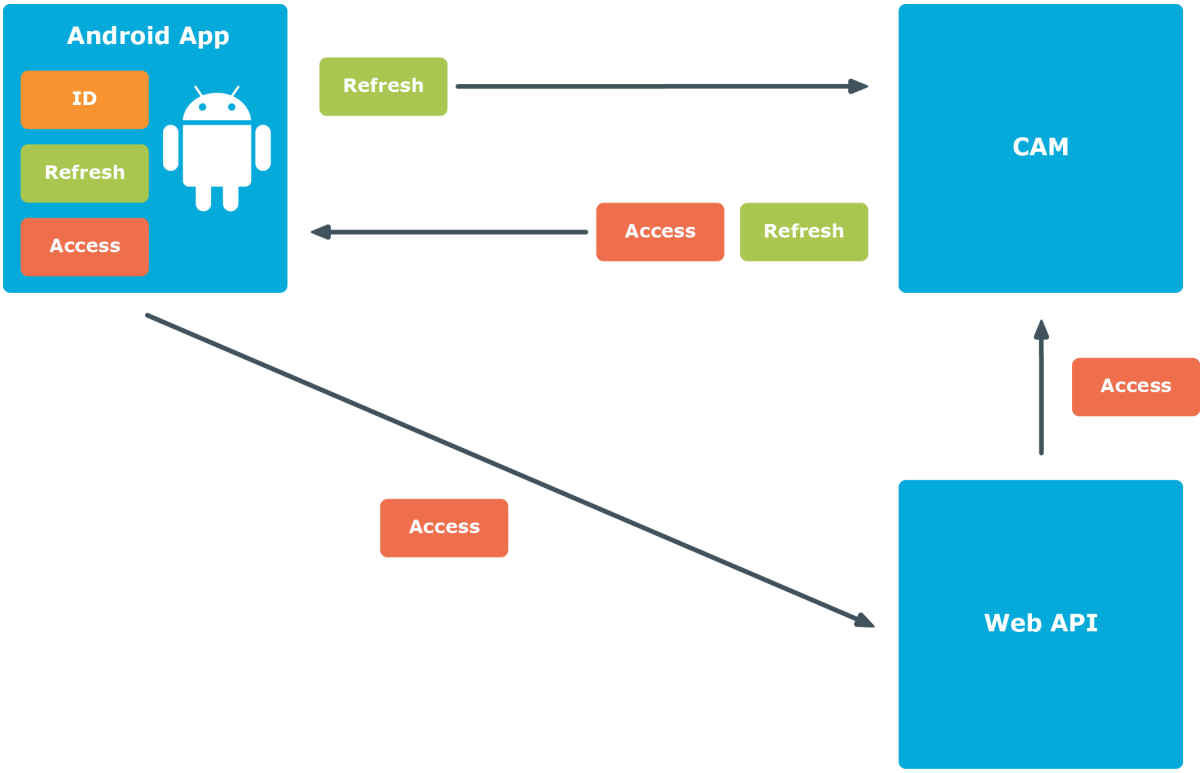
Figure 1: Single Sign-On (SSO) procedure for native Android applications



The ID Token contains a collection of identity claims about the user that can be used by the Android application to identify the user.

The Access Token allows the Android application to securely access OAuth2 protected Web APIs on behalf of the user. When the Access Token expires, the Refresh Token is used by the Android application to obtain a new Access Token, without the need for the user to re-authenticate as shown in Figure 2.

The Web API validates the Access Token by using it to obtain a set of claims about the user from Cloud Access Manager. The claims are then used by the Web API to identify the user and control the user's access.



Application walkthrough

This sample application consists of two components:

- Android OpenID Connect application
- .Net OAuth2 protected Web Application Programming Interface (API).

The sample Android application contains a package called openidconnect which can be used in a standard Android project to authenticate users, using the OpenID Connect Code Flow.

The sample Web API contains a .NET Open Web Interface (OWIN) middleware called CAMBearerTokenAuthentication which can be used in a standard .NET Web API project to authenticate the Android application, using the Access Tokens obtained from Cloud Access Manager.

The function of the sample application components

1. When the application starts it checks for an existing ID Token stored from a previous authentication. If an ID Token does not exist, the application sends an authentication

request, using the system browser, to start the OpenID Connect Authorization Code Flow.

If an ID Token exists, the application skips to step 4.

```
TokenStore tokenStore = SimpleTokenStore.loadDefault(this);
if (tokenStore.getIdToken() == null) {
    // Redirect to login page.
    startActivity(new Intent(this, LoginActivity.class));
}
```

2. The user is then prompted to authenticate to Cloud Access Manager using the system browser. After a successful authentication to Cloud Access Manager, the user is redirected back to the application with an authorization code, using a custom URI scheme.

```
CodeFlow codeFlow = new CodeFlow(this, tokenStore, Config.SETTINGS);
Uri authenticationRequestUrl = codeFlow.authenticationRequestUri();
// Open the Authentication Request URL using the system browser.
startActivity(new Intent(Intent.ACTION_VIEW, authenticationRequestUrl));
```

The application's custom URI scheme is registered in the application's manifest.

```
<activity
    android:name=".LoginActivity"
    android:label="OpenID Connect Client" >
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="cam3uxhvtb3ruls7xk1prr413er4wj1" /> <!-- cam[lowercase clientId] -->
    </intent-filter>
</activity>
```

3. The application uses the authorization code within the redirect URI to obtain an ID Token, Refresh Token and Access Token from Cloud Access Manager. The tokens are stored on the device in an app private area. The Access Token is scoped for use with Cloud Access Manager and the sample Web API. The scope of the Access Token is specified in the authentication request described in step 1.

```
codeFlow.tokenRequestAsync(redirectUri, new CodeFlow.TokenCallback() {
    @Override
    public void onSuccess() {
        // Switch back to the main activity after a successful authentication.
        startActivity(new Intent(LoginActivity.this, MainActivity.class));
    }

    @Override
    public void onError(String errorMsg) {
        loginMsgTextView.setText(errorMsg);
    }
});
```

4. The application can now access the Web API, using the Access Token as

authorization. The Access Token is included in the authorization header of each request.

```
URL resourceServer = new URL("https://sampleapi.company.local/claims");
conn = (HttpURLConnection) resourceServer.openConnection();
String accessToken = codeFlow.getValidAccessToken();
conn.setRequestProperty("Authorization", "Bearer " + accessToken);
```

5. The Web API validates the Access Token by using it to call the Cloud Access Manager User Info Endpoint. The validation is performed using the provided OWIN middleware, which will cache the User Info responses. The OWIN middleware will also verify that the Access Token was scoped for itself by checking that the User Info response contains at least one of its scopes. The claims returned from the User Info Endpoint are used by the Web API to identify the user and control their access.

   The OWIN authentication middleware is registered and configured using:

```
app.UseCAMBearerTokenAuthentication(new CAMBearerTokenAuthenticationOptions
{
    UserInfoEndpoint = "https://cam.company.local/CloudAccessManager/RPSTS/OAuth2/User.aspx",
    RequiredScopes = new[] { "sampleAPI" }
});
```

   The standard Authorize attribute can be used on the Web APIs to restrict access. The Authorize attribute supports restrictions based on role and user claims which, by default, map to the claim names role and preferred_username.

```
[Authorize]
public class ClaimsController : ApiController
{
    // GET: api/Claims
    [Authorize(Roles="read")]
    public IEnumerable<Claim> Get()
    {
        var principal = User as ClaimsPrincipal;
        return principal.Claims;
    }
```

   To utilize other claims, a custom AuthorizeAttribute can be created. For example:

```
public class RequireCustomClaimAttribute : AuthorizeAttribute
{
    protected override bool IsAuthorized(HttpActionContext context)
    {
        var principal = context.Request.GetRequestContext().Principal as ClaimsPrincipal;
        return principal.Claims.Any(c => c.Type == "custom-claim-name" && c.Value == "true");
    }
}
```

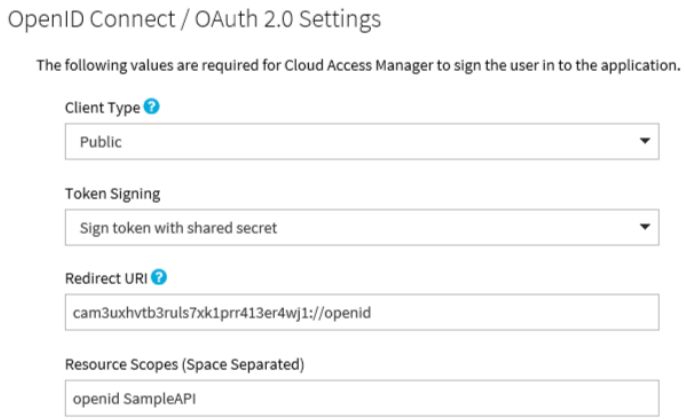
6. The application uses the Refresh Token to pre-emptively obtain a new Refresh Token and Access Token from Cloud Access Manager when the stored Access Token has expired.

Cloud Access Manager configuration

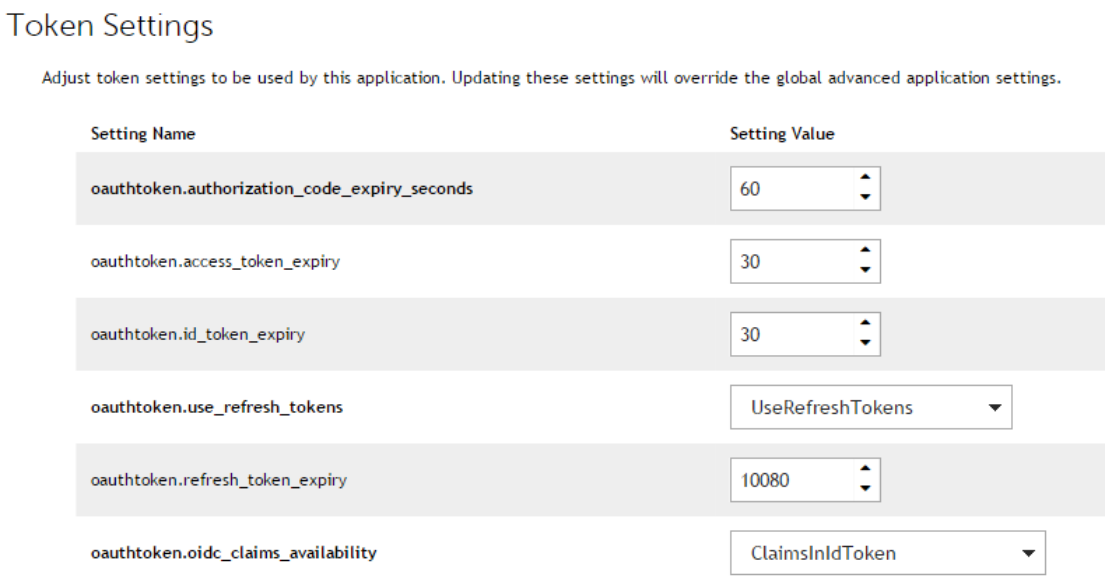
Perform the following configuration steps within Cloud Access Manager to enable single sign-on to native Android applications.

To configure Cloud Access Manager for single sign-on to native Android applications

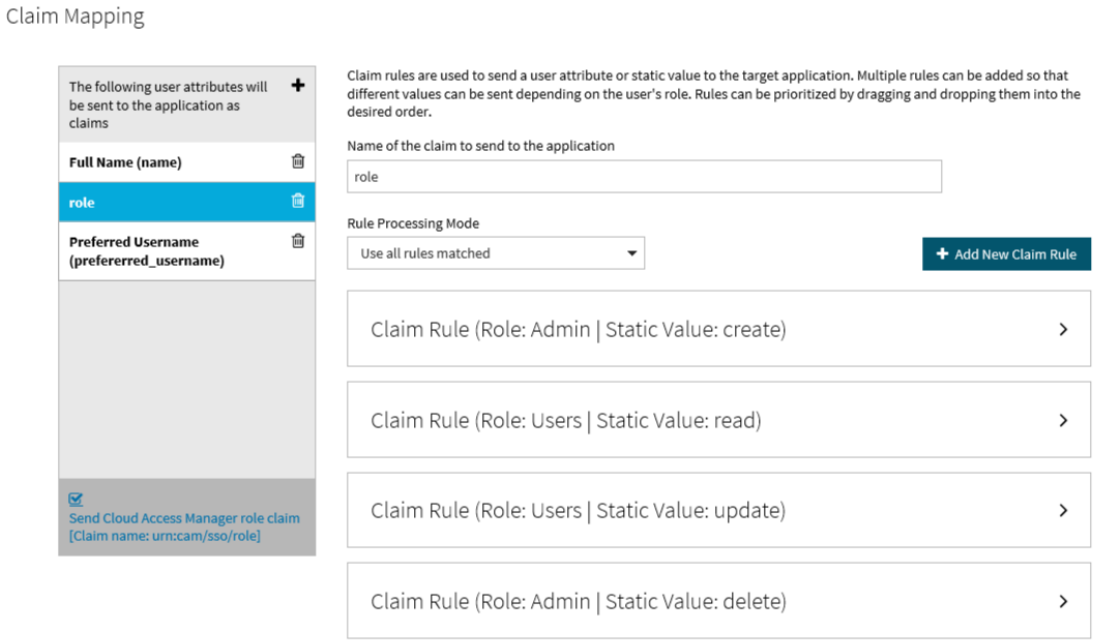
1. Make sure that the settings on the **OpenID Connect / OAuth 2.0 Settings** page are as shown below:



2. Make sure that the settings on the **Token Settings** page are as shown below:



3. Make sure that the settings on the **Claim Mapping** page are as shown below:



About us

Contacting us

For sales or other inquiries, visit https://www.oneidentity.com/company/contact-us.aspx or call +1-800-306-9329.

Technical support resources

Technical support is available to One Identity customers with a valid maintenance contract and customers who have trial versions. You can access the Support Portal at https://support.oneidentity.com/.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request
- View Knowledge Base articles
- Sign up for product notifications
- Download software and technical documentation
- View how-to-videos
- Engage in community discussions
- Chat with support engineers online
- View services to assist you with your product