AWS Web Identity Federation for Mobile Apps - Google (2 of 3 series)

Sivaprasad Padisetty (February 2014)

*This is part two of a three part series. You may want to read about basic introduction and Facebook authentication* [*here*](http://www.padisetty.com/2014/02/aws-web-identity-federation-for-mobile.html)*. In this blog, I will cover the Google authentication in this blog.*

# Prerequisites and Helper function

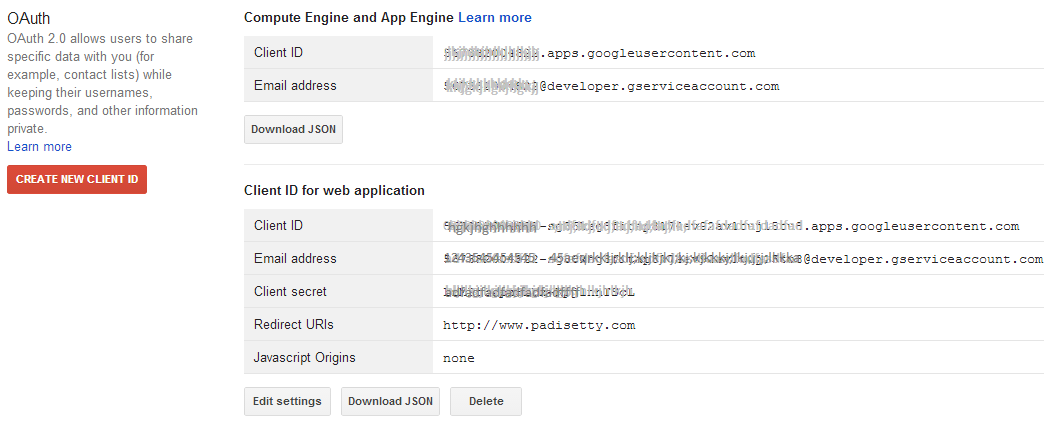
* Sign up for AWS and get the AccessKey & SecretKey. You can find the info about AWS Account and Access Keys [here](http://docs.aws.amazon.com/powershell/latest/userguide/pstools-appendix-signup.html).
* You have Visual Studio installed, I used Visual Studio 2013. Although I did not test it, earlier version should work.
* Install AWS SDK for .Net from [here](http://aws.amazon.com/sdkfornet/) and follow the “Getting Started” instructions.

# Overview

1. Go to the [Google Developers Console](https://cloud.google.com/console), select a project, or create a new one.
2. Select APIs & auth. In the displayed list of APIs, make sure the Google+ API status is set to ON.
3. Register an application and get the client ID and client secret.
4. Create a role in AWS, this is the role the user will be impersonated.
5. The app includes logic to make https request at <https://accounts.google.com/o/oauth2/auth> and to get back a token (or code) from the provider.
6. The app can call AssumeRoleWithWebIdentity without using any AWS security credentials. The call includes the token received from the provider previously.
7. AWS STS is able to verify that the token passed from the app is valid and then returns temporary security credentials to the app. The mobile app's permissions to access AWS are established by the role that the app assumes.

# Register Application

Create a project at [Google Developers Console](https://cloud.google.com/console) and register a web application. Save the client ID and client secret in the app.config as described below. You should also define the redirect URL, it need not be https based. This is the only redirect URL allowed in getting the token. I used <http://padisetty.com>, you can as well use http://google.com.



# Create an AWS Role

This is the role that a Google authenticated user will assume. The role is associated with two things a) trust policy – who can assume this role and b) access policy – what permission does the assumed user have.

C# code below creates a role. Normally, this is manually created once. I chose to write C# code because it is handy for automation. This role can be assumed by any authenticated Google user. The user only has access to their specific key which is located under “federationbucket/Google/<userid>”. Code below is slightly complicated because the same code works for all the identity providers (i.e.) Facebook/Google/Amazon.

providerURL = "accounts.google.com";

providerAppIdName = "aud";

providerUserIdName = "sub";

//identity provider specific AppId is loaded from app.config (e.g)

// FacebookProviderAppId. GoogleProviderAppId, AmazonProviderAppId

providerAppId = ConfigurationManager.AppSettings[identityProvider +

"ProviderAppId"];

// Since the string is passed to String.Format, '{' & '}' has to be escaped.

// Policy document specifies who can invoke AssumeRoleWithWebIdentity

string trustPolicyTemplate = @"{{

""Version"": ""2012-10-17"",

""Statement"": [

{{

""Effect"": ""Allow"",

""Principal"": {{ ""Federated"": ""{1}"" }},

""Action"": ""sts:AssumeRoleWithWebIdentity"",

""Condition"": {{

""StringEquals"": {{""{1}:{2}"": ""{3}""}}

}}

}}

]

}}";

// Defines what permissions to grant when AssumeRoleWithWebIdentity is called

string accessPolicyTemplate = @"{{

""Version"": ""2012-10-17"",

""Statement"": [

{{

""Effect"":""Allow"",

""Action"":[""s3:GetObject"", ""s3:PutObject"", ""s3:DeleteObject""],

""Resource"": [

""arn:aws:s3:::federationtestbucket/{0}/${{{1}:{4}}}"",

""arn:aws:s3:::federationtestbucket/{0}/${{{1}:{4}}}/\*""

]

}}

]

}}";

// Create Trust policy

CreateRoleRequest createRoleRequest = new CreateRoleRequest

{

RoleName = "federationtestrole",

AssumeRolePolicyDocument = string.Format(trustPolicyTemplate,

identityProvider,

providerURL,

providerAppIdName,

providerAppId)

};

Console.WriteLine("\nTrust Policy Document:\n{0}\n",

createRoleRequest.AssumeRolePolicyDocument);

CreateRoleResponse createRoleResponse = iamClient.CreateRole(createRoleRequest);

// Create Access policy (Permissions)

PutRolePolicyRequest putRolePolicyRequest = new PutRolePolicyRequest

{

PolicyName = "federationtestrole-rolepolicy",

RoleName = "federationtestrole",

PolicyDocument = string.Format(accessPolicyTemplate,

identityProvider,

providerURL,

providerAppIdName,

providerAppId,

providerUserIdName)

};

Console.WriteLine("\nAccess Policy Document (Permissions):\n{0}\n",

putRolePolicyRequest.PolicyDocument);

PutRolePolicyResponse putRolePolicyResponse = iamClient.PutRolePolicy(

putRolePolicyRequest);

Above code assumes an app.config file to contain the following values.

<appSettings>

<add key="AWSAccessKey" value="YOUR\_ACCESS\_KEY\_A134" />

<add key="AWSSecretKey" value="YOUR\_SECRET\_KEY\_HERE\_SECRET\_KEY\_HEREndgN" />

<add key="AWSRegion" value="us-east-1" />

<add key="GoogleProviderAppId"

value="123456789012-your\_app\_id\_here\_your\_app\_id\_abc.apps.googleusercontent.com" />

<add key="GoogleProviderAppIdSecret"

value="your\_app\_secret\_here\_abc" />

</appSettings>

Trust Policy document produced by the above code:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": { "Federated": "accounts.google.com" },

"Action": "sts:AssumeRoleWithWebIdentity",

"Condition": {

"StringEquals": {"accounts.google.com:aud":

"123456789012-your\_app\_id\_here\_your\_app\_id\_abc.apps.googleusercontent.com"}

}

}

]

}

Access Policy document (permissions) produced by the above code:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect":"Allow",

"Action":["s3:GetObject", "s3:PutObject", "s3:DeleteObject"],

"Resource": [

"arn:aws:s3:::federationtestbucket/Google/${accounts.google.com:sub}",

"arn:aws:s3:::federationtestbucket/Google/${accounts.google.com:sub}/\*"

]

}

]

}

# Authenticate with Google and get the token

The authorization sequence begins when your application redirects a browser to a Google URL (<https://accounts.google.com/o/oauth2/auth>); the URL includes query parameters that indicate the type of access being requested. Google handles user authentication, session selection, and user consent. The result is an authorization code, which Google returns to your application in a query string.

The code below constructs the query and makes the http call to retrieve the token directly. The GET action is performed in the browser control. If the authentication succeeds, it will be redirected to the URL specified in the query (This redirect URL has to be pre-configured as described above). The C# code below uses the Forms based WebBrowser control to automate this process. As soon as the token is retrieved, the browser control is closed. The structure of the query is pretty straight forward. It can be inferred by looking at the code.

string query = "https://accounts.google.com/o/oauth2/auth?" +

string.Format("client\_id={0}&", client\_id) +

"response\_type=id\_token&" +

"scope=email%20profile&" +

"redirect\_uri=http://www.padisetty.com";

The GetToken helper function, does GET operation and retrieves the token from the redirected URL.

class MyWebBrowser : WebBrowser

{

public string CapturedUrl;

string token;

public MyWebBrowser(string token)

{

this.token = token + "=";

}

protected override void OnDocumentCompleted(

WebBrowserDocumentCompletedEventArgs e)

{

base.OnDocumentCompleted(e);

string st = e.Url.ToString();

if (st.Contains(token))

{

// hack, closing the form here does not work always.

this.Navigate("about:blank");

this.CapturedUrl = st;

Console.WriteLine("Captured: {0}", st);

}

else if (st == "about:blank")

{

((Form)this.Parent).Close();

}

}

}

string GetToken(string token, string url)

{

Form f = new Form();

MyWebBrowser wb = new MyWebBrowser(token);

wb.Dock = DockStyle.Fill;

f.Controls.Add(wb);

wb.Navigate(url);

f.WindowState = FormWindowState.Maximized;

f.ShowDialog();

string st = wb.CapturedUrl;

f.Dispose();

if (st == null)

throw new Exception("Oops! Error getting the token");

int index = st.IndexOfAny(new char[] { '?', '#' });

st = index < 0 ? "" : st.Substring(index + 1);

NameValueCollection pairs = HttpUtility.ParseQueryString(st);

string tokenValue = pairs[token];

Console.WriteLine("TOKEN={0}, Value={1}", token, tokenValue);

return tokenValue;

}

# Authentication with Google: Two Step Process

To further improve security, you can follow the two step process. In the first step you ask for the code, and then exchange the code for the token. When you exchange the code with token, you need to pass the secret key. Note that you should not embed the secret key in your app that defeats the two step process. This exchange has to happen on the server side. This means end user will not have access to the token, they can only see the code.

Your application sends a token request to the Google Authorization Server, receives an authorization code, 
exchanges the code for a token, and uses the token to call a Google API endpoint.

string query = "https://accounts.google.com/o/oauth2/auth?" +

string.Format("client\_id={0}&", client\_id) +

"response\_type=code&" +

"scope=email%20profile&" +

"redirect\_uri=http://www.padisetty.com";

Once you have the code, from the server side, exchange the code for the token.

string id\_token = null;

using (var wc = new WebClient())

{

var data = new NameValueCollection();

data["code"] = GetToken("code", query);

data["client\_id"] = client\_id;

data["client\_secret"] = client\_secret;

data["redirect\_uri"] = "http://www.padisetty.com";

data["grant\_type"] = "authorization\_code";

var response = wc.UploadValues(

"https://accounts.google.com/o/oauth2/token",

"POST", data);

string responsebody = Encoding.UTF8.GetString(response);

dynamic result = JsonConvert.DeserializeObject(responsebody);

id\_token = result.id\_token;

}

# Get Temporary Credentials with AssumeRoleWithWebIdentity

Key concept to grasp here is, you start with anonymous AWS credentials, pass the token received from Google and get the temporary credentials. This is important because the mobile app user will not have any AWS credentials.

AssumeRoleWithWebIdentityRequest assumeRoleWithWebIdentityRequest =

new AssumeRoleWithWebIdentityRequest ()

{

WebIdentityToken = id\_token,

RoleArn = role,

assumeRoleWithWebIdentityRequest.DurationSeconds = 3600,

assumeRoleWithWebIdentityRequest.RoleSessionName = "MySession"

};

var stsClient = new AmazonSecurityTokenServiceClient(new AnonymousAWSCredentials());

AssumeRoleWithWebIdentityResponse assumeRoleWithWebIdentityResponse =

stsClient.AssumeRoleWithWebIdentity(assumeRoleWithWebIdentityRequest);

# References

1. [Scenarios for Granting Temporary Access](http://docs.aws.amazon.com/STS/latest/UsingSTS/STSUseCases.html)
2. [Creating Temporary Security Credentials](http://docs.aws.amazon.com/STS/latest/UsingSTS/CreatingWIF.html)
3. [Sample Identity provider blog](http://blogs.aws.amazon.com/net/post/Tx2KW5KYMRE681I/Web-Identity-Federation-using-the-AWS-SDK-for-NET)
4. [Google OAuth 2.0 Playground](https://developers.google.com/oauthplayground/)
5. [Web Identity Federation Playground](https://web-identity-federation-playground.s3.amazonaws.com/index.html)

You can find the code under “AWS\AWS CSharp Test” folder at <https://github.com/padisetty/Samples>.

Explore & Enjoy!

/Siva