Paper SAS1737-2018

OpenID Connect Opens the Door to SAS® Viya® APIs

Mike Roda, SAS Institute Inc.

ABSTRACT

As part of the strategy to be open and cloud-ready, SAS Viya services leverage OAuth and OpenID Connect tokens for authentication. OpenID Connect and its base technology OAuth 2.0 provide a simple security framework built on the HTTP protocol and are quickly becoming the de facto standard for public APIs. This paper describes how SAS Viya uses these standards and demonstrates how developers and administrators can use simple commands such as curl to authenticate to SAS Viya APIs.

INTRODUCTION

Research on OAuth 2.0 started internally at SAS as early as 2014. Microservice architecture and representational state transfer (REST) application programming interfaces (APIs) were going to form the underpinning of SAS Viya services, and OAuth 2.0 was being widely adopted on the Internet for authentication and authorization of public-facing APIs, led by Google, Facebook, and others. It was a perfect fit. OAuth 2.0 provides a simple and open security protocol built on HTTP, and OpenID Connect extends that for authentication. Around the same time, a lot of research in SAS was happening with Cloud Foundry, the open-source, cloud-independent platform as a service (PaaS). One component of Cloud Foundry is its open-source User Account and Authentication (UAA) Server. UAA is an implementation of an authorization server, supporting both OAuth 2.0 and OpenID Connect. It is a very active project that supports external logins via Lightweight Directory Access Protocol (LDAP) and Security Assertions Markup Language (SAML). This made it a good fit for SAS Viya.

The SAS® Logon Manager (SASLogon, saslogon, or even just "logon" sometimes) is based on Cloud Foundry UAA, with extensive modification for SAS Viya. The modifications include adapting it to run as a Spring Boot application, support for externalized configuration and Consul, single sign-on and single logoff, session replication and clustering, assumable groups, event publishing to the event framework, SAS branding, and many other features. Support was also added for Kerberos (including delegation), pluggable authentication module (PAM), SAS 9 one-time-passwords, and guest access. Although SAS continues to develop new features in the SAS Logon Manager, we also routinely merge in the latest version of UAA to pick up bug fixes, security fixes, and new features.

This paper is organized into four main parts. There are many good sources of information about OAuth 2.0 and OpenID Connect, so rather than provide another overview here, this paper describes some of the key concepts and flows in the context of how they are used in SAS Viya. OAuth 2.0 requires client applications to be registered with the server, so the next section covers this important aspect. Next it explains how clients can obtain tokens and describes the contents of those tokens. Finally, it describes how to get information about SAS Viya APIs and how to authenticate to them.

OVERVIEW

OAuth 2.0 (or just OAuth for short, but not to be confused with version 1.0, which it is incompatible with), is a security framework that facilitates access to resources by applications, either by the resource owner delegating this access to the application through an approval process that doesn't involve the owner's credentials passing through the application, or by allowing the application to access the resource on its own behalf. OAuth protocol messages use the HTTP protocol. The protocol is relatively simple, consisting of only a few endpoints, request parameters passed in `application/x-www-form-urlencoded` encoding, and only BASIC authentication of applications. An important characteristic of OAuth is that the access provided to an application has limited scope. This scope is defined when the application is registered, a mandatory step. Furthermore, when a resource owner delegates access to an application, the owner has an opportunity to approve or deny access for each scope. OAuth 2.0 is formally defined in RFC6749.

OpenID Connect 1.0, or just OIDC, is an extension to OAuth 2.0 for authentication. The specification introduces a new type of token called the ID token and new flows for obtaining ID tokens. ID tokens are essentially an electronic form of identity, carrying claims about the identity of the user along with other metadata. The contents of the access tokens and ID tokens are discussed in more detail later.

ROLES

There are four roles defined in the OAuth 2.0 framework. They are listed below with descriptions in context of SAS Viya:

- Resource owner - Simply the end user.
- Resource server - Service that provides resources. These are the services hosting SAS Viya APIs.
- Client - Referred to earlier as the application, which is directed to access resources with the owner's authorization. Services hosting SAS Viya APIs also are an OAuth client (to themselves). When called without any form of access token, the service must seek delegated access (and authentication) from the resource owner before servicing the request.
- Authorization server - Server that issues access tokens to clients. It also authenticates resource owners and prompts them for approving access to clients. The SAS Logon Manager handles this in SAS Viya.

Browser-client applications are purposely not mentioned above since SAS Viya browser-client applications such as SAS® Visual Analytics do not participate directly in OAuth. The client application running in the browser instructs the browser to make requests against the APIs, and the services hosting those APIs act as the OAuth client, facilitating the orchestration necessary to obtain approval from the user and obtain an access token. This is described below in the authorization code flow.

FLOWS

Section 1.2 of the OAuth 2.0 RFC describes an abstract flow, which is specialized for each type of authorization grant. Below we describe a few of the key flows in the context of SAS Viya. Although OpenID Connect adds additional flows for obtaining ID tokens, and these flows are supported by the SAS Logon Manager, they are not used by SAS Viya applications. Therefore, they are not covered here.

Authorization Code

Very much the preferred flow, the authorization code flow is sometimes referred to as the '3-legged' flow since it involves a 3-way conversation between a client, the user, and the authorization server. The flow facilitates the client obtaining an access token without ever handling the user's credentials. In the context of SAS Viya, this flow is used when a visual application running in the browser (for example, SAS Visual Analytics) calls an API. SAS Viya builds on this flow with session management. Here are the steps:

1. The application running in the browser instructs the browser to make a request to a REST API.
2. The service hosting the API responds with an HTTP session cookie and a redirect to the `/SASLogon/oauth/authorize` endpoint on SASLogon.
3. The browser remembers the session cookie and calls SASLogon with the authorization request.
4. SASLogon receives the authorization request. If the request is coming from a SAS client, the request is automatically approved for all scopes (group memberships) and responds with a new authorization code and redirects the browser back to the URI of the API.
5. The browser sends the request with the authorization code to the API. The request also contains the session cookie.
6. The API service sends a request to the `/SASLogon/oauth/token` endpoint and passes the authorization code.

7. SASLogon issues a new access token and ID token and returns them in the response, along with a refresh token. A special claim is added to the access token to associate it with the login session.
8. The API processes the tokens to authenticate the user and make an authorization decision based on the scopes (group memberships) in the access token. The tokens are stored in the HTTP session and another redirect is returned to the browser to call the API without the authorization code. This last redirect isn't technically necessary but ensures that the code isn't left in the navigation bar where it might get bookmarked.
9. The browser receives the redirect and calls the API again without the authorization code.
10. The API returns the data requested.
11. The browser receives the data from the API and hands it to the visual application.

These steps assume that the end user is already signed in to SASLogon and has an active HTTP session with SASLogon. If not, SASLogon redirects the browser to the login page after step 3 so that the user can sign in and establish an HTTP session. Note that each service returns HTTP session cookies with a path, so the browser maintains individual HTTP sessions with each service it has talked to. Subsequent calls from the browser to the same API use the existing HTTP session without triggering the authorization code flow again. This happens until the session times out from inactivity, which is 30 minutes by default.

If the API call is not an HTTP GET or HEAD request, some additional steps happen since the original HTTP method and any body content are lost after following a 302 redirect. The Location URI returned to the browser in step 2 includes a callback URI to the API plus a parameter indicating the original HTTP method. In step 8 the API returns a special HTTP 449 response instead of 302. HTTP 449 is a Microsoft extension for Retry. The application in the browser gets the 449 and issues a retry of the original request in step 1. This time the request goes through because a session has been established.

Logout

Although not an OAuth 2.0 flow, the logout flow complements the flow described above. Upon logout, all sessions must be destroyed and access tokens revoked. Here are the steps:

1. The end user clicks on the logout button in a visual application.
2. The application instructs the browser to call the logout endpoint on the web application it was launched from (for example, `/SASVisualAnalytics/logout`).
3. The web application destroys the HTTP session, along with the access token in it. The response to the browser unsets the session cookie and redirects to the logout endpoint on SASLogon.
4. The browser forgets the session cookie and calls the logout endpoint on SASLogon.
5. SASLogon destroys the HTTP session and sends out a 'session destroyed' event to all services. The response to the browser unsets the session cookie and displays the logout page.
6. The browser forgets the session cookie and displays a page saying you are signed out.
7. All services receive the 'session destroyed' event from SASLogon. The event contains a hashed code that services use to look up any associated HTTP sessions and access tokens and destroys them.

Resource-Owner Password Credentials

The password flow is used when a client needs to call the API on behalf of a user and has the user's password credentials. This should be used only when the client does not have a way of handling the web interaction used in the authorization code flow.

1. The client requests the user name and password from the user.
2. The user provides his or her user name and password to the client.

3. The client requests an access token by using `grant_type=password` and presenting the user's credentials. The client itself also must authenticate.
4. SASLogon authenticates the client, validates the user name and password, and issues an access token.
5. The client makes a request to the API and authenticates by presenting the access token.
6. The service hosting the API validates the access token and serves the request.

Client Credentials

Clients can obtain an access token using the client's credentials alone. This type of access token is not associated with any user and is scoped according to the authorities registered to the client. This topic is covered in the next section. Since there is no user involved in this flow, there is no resource owner except the client itself. This is shown in the following steps:

1. The client requests an access token by using `grant_type=client_credentials` and authenticating with its own credentials.
2. SASLogon authenticates the client and issues an access token.
3. The client makes a request to the API and authenticates by presenting the access token.
4. The service hosting the API validates the access token and, if valid, serves the request.

CLIENT REGISTRATION

The client that will be calling the SAS Viya APIs must be registered with SASLogon. There is a bit of a 'chicken or the egg' problem here because we need to obtain an access token to register the client, but we need an existing client ID to obtain an access token. Fortunately, the SAS Logon Manager has an endpoint that issues an OAuth access token to anyone who has a valid SAS® Configuration Server (Consul) token.

Consul tokens are located in the directory `/opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default/`. The client token will suffice for this purpose. Output 1 **Error! Reference source not found.**sets the CONSUL_TOKEN environment variable from the value stored in the file:

```
# cd /opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default
# export CONSUL_TOKEN=`cat client.token`
```

Output 1. Set the CONSUL_TOKEN Environment Variable

To use the consul token to obtain an access token, make a POST request to `/SASLogon/oauth/clients/consul`. Table 1 describes the request headers, and Table 2 describes the request parameters.

Name	Description
X-Consul-Token	The client Consul token.

Table 1. Request Headers for the /SASLogon/oauth/clients/consul Endpoint

Name	Description
serviceId	Name of the client that is going to be registered.
callback	Set to 'false' to get an access token back in the response. Otherwise, the token is sent to the service registered in Consul by that name.

Table 2. Request Parameters for the /SASLogon/oauth/clients/consul Endpoint

In Output 2 below, curl is used to obtain an access token using the Consul token stored in the environment variable. The output is piped through `python -m json.tool` to produce formatted JSON. The access token is shortened for readability.

```
# curl -s -X POST
"https://localhost/SASLogon/oauth/clients/consul?callback=false&serviceId=a
pp" -H "X-Consul-Token: $CONSUL_TOKEN" | python -m json.tool
{
  "access_token":"eyJhbGciOiJSxlZ2FjeS...VrWbHFrFOeJtNA",
  "token_type":"bearer",
  "expires_in":43199,
  "scope":"clients.admin",
  "jti":"fd558f0226df43cfbe12e044e3ac1d45"
}
```

Output 2. Use curl to Obtain an Access Token for Client Registration

If the Consul token is valid, SASLogon will respond back with JSON text containing an access token that can be used to register the client. Copy the value of the access token and save it into an environment variable as shown below in Output 3 below. Note this access token is narrowly scoped for registering a new client. You cannot use it to call other SAS Viya APIs.

```
# export TOKEN=eyJhbGciOiJSxlZ2FjeS...VrWbHFrFOeJtNA
```

Output 3. Set an Environment Variable with the Value of the Access Token

The last step registers a new OAuth client by making a POST request to `/SASLogon/oauth/clients`. Table 3 shows the request headers, and Table 4 describes the request parameters.

Name	Description
Content-Type	application/json
Authorization	Bearer $TOKEN

Table 3. Request Headers for the Client Administrative Endpoints

Name	Type	Constraints	Description
client_id	String	Required	Unique client identifier.
authorized_grant_types	Array	Required	List of grant types that can be used to obtain a token with this client. Types include `authorization_code`, `password`, `implicit`, and `client_credentials`.
redirect_uri	Array	Optional	Allowed URI pattern for redirect during authorization. Wildcard patterns can be specified using the Ant-style pattern.
Scope	Array	Optional (defaults to "uaa.none")	Scopes allowed for the client.
resource_ids	Array	Optional	Resources the client is allowed to access.

authorities	Array	Optional (defaults to "uaa.none")	Scopes that tokens will get using the `client_credentials` grant_type.
autoapprove	[Boolean, Array]	Optional	Scopes that do not require user approval. Boolean values true and false apply to all scopes. Otherwise, a list can be provided.
access_token_validity	Number	Optional	Time in seconds to access token expiration after it is issued.
refresh_token_validity	Number	Optional	Time in seconds to refresh token expiration after it is issued.
allowedproviders	Array	Optional	A list of origin keys (alias) for identity providers to which the client is limited. Null implies any identity provider is allowed.
Name	String	Optional	A human readable name for the client.
token_salt	String	String	A random string used to generate the client's revocation key. Change this value to revoke all active tokens for the client.
client_secret	String	Required if the client will use the authorization_code or client_credentials grant types	A secret string used for authenticating as this client.
use-sessions	Boolean	Optional	Access tokens issued to this client should be associated with an HTTP session and revoked when upon logout or time-out.

Table 4. Request Parameters for Client Administrative Endpoints

In Output 4 below, curl is used to register a new client using the access token stored in the environment variable:

```
# curl -s -X POST "https://localhost/SASLogon/oauth/clients" \
       -H "Content-Type: application/json" \
       -H "Authorization: Bearer $TOKEN" \
       -d '{
        "client_id": "app",
        "client_secret": "******",
        "scope": ["openid", "group1"],
        "authorized_grant_types": ["password"]
       }' | python -m json.tool
{
   "scope":[  "openid", "group1" ],
   "client_id":"app",
   "resource_ids":[ "none" ],
   "authorized_grant_types":[ "password" ],
   "autoapprove":[ ],
   "authorities":[ "uaa.none" ],
   "lastModified":1509556135212
}
```

Output 4. Using curl to Register a New Client

TOKENS

Access tokens are issued to a client for a list of scopes that specify the level of access that should be provided. When delegating access to a client application, users approve or deny each scope requested by the application. However, the SAS Logon Manager treats a user's group memberships as scopes and automatically approves these scopes for use by SAS applications, with the exception of the SAS Administrators group. Users that are a member of this group are prompted each time they sign in to see if they want to assume the administrator access for the session. Group memberships are queried from LDAP during authentication.

ID tokens are the cousin of access tokens, used only for asserting identity and authentication. While access tokens might be opaque, ID tokens carry identity information and other metadata, such as the name of the server that issued the token, when authentication took place, the client that obtained the token, and the resource the token is intended for.

By policy, the SAS Logon Manager issues access tokens, which are also ID tokens. They contain a super set of claims about the access authorization as well as the identity. These access tokens are required to access the SAS Viya APIs. The following sections describe how a client can obtain an access token and describe the contents in more detail.

OBTAINING TOKENS

Access tokens are obtained when a client makes a request, authenticates to SASLogon, and passes a form of authorization. The authorization is expressed in the form of an authorization grant. OAuth 2.0 defines the following grant types, flows of which were covered earlier: authorization code, implicit, password, client credentials, and refresh token. SASLogon also supports some additional grant types as an extension. The token endpoint is defined in section-3.2 of RFC 6749. For SAS Viya, clients make a POST request to `/SASLogon/oauth/token`. Table 5 and Table 6 below describe the request headers and parameters.

Name	Description
Content-Type	application/x-www-form-urlencoded
Authorization	Basic credentials of the client.

Table 5. Request Headers for the Token Endpoint

Name	Description
grant_type	One of four grant types specified in RFC 6749 or an extension supported by SASLogon.
username	Required, if the grant_type is "password".
password	Required, if the grant_type is "password".
code	Required, if the grant_type is "authorization_code".
redirect_uri	Required, if the grant_type is "authorization_code" and if the "redirect_uri" parameter was included in the authorization request.
refresh_token	Required, if the grant_type is "refresh_token".

Table 6. Request Parameters for the Token Endpoint

In Output 5 below, curl is used to obtain an access token using a password grant. The access token is shortened for readability.

```
# curl -s -X POST "https://localhost/SASLogon/oauth/token" \
       -H "Content-Type: application/x-www-form-urlencoded" \
```

```
       -d "grant_type=password&username=bob&password=bobspassword" \
       -u "app:secret" | python -m json.tool
{
  "access_token":"eyJhbGciOiJSUzI1N...PWLR9jE_R2cC43e8g",
  "token_type":"bearer",
  "access_token":"eyJhbGciOiJSUzI...oWpREW8V43hXCDOM",
  "expires_in":43199,
  "scope":"openid group1 group2 group3 group4",
  "jti":"e88eeb0c3c5c4432a357810bad5d0c85"
}
```

Output 5. Use curl to Obtain an Access Token

By default, all the scopes the client is registered for will be obtained, corresponding to the user's group memberships, but the scope can be more narrowly specified by adding a 'scope' parameter to the POST data and including a comma-delimited list of scopes. A subset of the scopes must be registered for the client.

Copy the access token into an environment variable, as shown below in Output 6, so it can be used on subsequent requests when calling SAS Viya APIs.

```
# export TOKEN=eyJhbGciOiJSUzI1N...PWLR9jE_R2cC43e8g
```

Output 6. Use curl to Obtain an Access Token

CONTENTS OF TOKENS

ID tokens and access tokens issued by the SAS Logon Manager are usually JSON web tokens (JWT). JWTs offer several advantages over opaque tokens since they can be validated immediately using the digital signature and claims read without making a callback to the authorization server.

The JWT consists of three parts, a JSON header, JSON body, and binary signature. Each part is base64-encoded and separated by a period (.). At the beginning is the header. The header indicates the type of token (always 'JWT'), the algorithm used to digitally sign the token, and an identifier representing the signing key. An example is shown in Output 7 below, and these values are discussed in more detail later.

```
{
  alg: "RS256",
  kid: "legacy-token-key",
  typ: "JWT"
}
```

Output 7. JWT Header

The body contains claims about the user as well as metadata about the token itself. Output 8 below is a decoded example JWT:

```
{
   jti:"c412c1409900406b9687df75dd82ba3e",
   sub:"b6bafd6f-e49a-4fe8-95c9-6f5af20c43bf",
   scope:[
      "openid",
      "group1",
      "group2",
      "group3",
```

```
      "group4"
   ],
   client_id:"app",
   cid:"app",
   azp:"app",
   grant_type:"password",
   user_id:"b6bafd6f-e49a-4fe8-95c9-6f5af20c43bf",
   ext_id:"cn=Bob,ou=User Accounts,dc=EXAMPLE,dc=com",
   origin:"ldap",
   user_name:"bob",
   email:"bob@example.com",
   auth_time:1509455528,
   rev_sig:"8c5f29b3",
   iat:1509455528,
   exp:1509498728,
   iss:"http://localhost/SASLogon/oauth/token",
   zid:"uaa",
   aud:[
      "openid",
      "app"
   ]
}
```

Output 8. JWT Body

Not all fields are included for every grant type, and some fields are redundant. Table 7 below describes each field and indicates from which specification or implementation it originates:

Name	Reference	Description
jti	JWT	Unique identifier for the JWT.
sub	OIDC	Internal identifier for the user (Subject).
scope	OAuth	List of scopes (group memberships) this access token has.
client_id	OAuth	Client ID that requested the token.
cid	OAuth	Client ID that requested the token.
azp	OIDC	The party to which the ID Token was issued (OIDC).
grant_type	OAuth	Type of authorization grant.
user_id	UAA	Internal identifier for the user.
ext_id	SAS	External ID for the user.
user_name	UAA	Canonical user name of the user.
email	OIDC	Email address of the user.
origin	UAA	Identity provider that authenticated the user.
remote_ip	SAS	IP of the user.
session_sig	SAS	Hashed signature of the user's login session, used to revoke tokens upon logout or time-out.

rev_sig	UAA	Hashed revocation signature, calculated on the server to validate that a token has been revoked.
iat	JWT	Time the token was issued (epoch).
exp	JWT	Time the token expires (epoch).
iss	JWT	URI of the authorization server that issued the token.
zid	UAA	Used in multi-tenant environments to identity the tenant.
aud	OIDC	List of protected resources (audiences) for which the token is granted.

Table 7. Claims in the JWT

The last part of the JWT is the JSON web signature (JWS), not shown here since it is just base64-encoded bytes. JWS is formalized in RFC7515. By default, SAS Viya uses RSA Signature with SHA-256. This is reflected in the header as the RS256 algorithm. The public key can be viewed by doing a GET on the `/SASLogon/token_key` endpoint. In Output 9 below (shortened for readability), notice the `kid` (key ID) matches the `kid` in the JWT header shown earlier:

```
{
    "alg": "RS256",
    "e": "AQAB",
    "kid": "legacy-token-key",
    "kty": "RSA",
    "n": "...",
    "use": "sig",
    "value": "-----BEGIN PUBLIC KEY-----
\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCL3uRlAWdXvC6oZwsrbzy0qIgN\n...\n--
---END PUBLIC KEY-----"
}
```

Output 9. JWT Header

Alternatively, SAS Viya can be configured to sign in using hashed message authentication codes from a shared key. In this mode, the algorithm is HS256, and the token_key endpoint requires BASIC authentication using client credentials.

The contents of a JWT can be decoded and verified using the third-party site https://jwt.io/.

REVOKING A TOKEN

When a client is finished using an access token, it might want to revoke it so it is no longer valid even if the expiration time has not been reached. However, currently in SAS Viya, support for revoking tokens is limited. Typically, only tokens associated with a login session are subject to revocation. When a user signs out of SAS Viya, all tokens issued on behalf of the user during the course of the session are revoked. To use this feature, clients must be registered with the `use-sessions` option set to `true`. Access tokens obtained by the client will then be associated with an HTTP session in SASLogon, either in the session from the web-based login, or a session issued directly to the client as part of obtaining the token via the API. Care should be taken when using the latter because the HTTP session in SASLogon will time out after 30 minutes of inactivity. So, clients that want to use sessions need to keep this session active by making periodic requests to SASLogon. The tokens are then revoked when the `/SASLogon/logout` endpoint is called on the corresponding HTTP session.

There are a few other circumstances whereby an access token might be revoked. If a registered client's secret is changed, all tokens previously issued to that client are revoked. Also, if the key used to sign tokens is changed, all existing tokens signed with the old key are no longer valid.

SAS VIYA APIS

OVERVIEW

REST is a style of API that views services as a distributed data model rather than by a set of remote procedure calls. Elements in the data model are referred to as resources, addressed by a Uniform Resource Identifier (URI), which are accessed and manipulated using standard HTTP protocol methods (GET, POST, PUT, and DELETE) and response codes. REST APIs are generally stateless, although a session can be maintained at the security layer to avoid repeated authentication attempts.

Basic API metadata can be obtained from the services' `/apiMeta` endpoint. This endpoint does not require authentication and returns JSON by default. For example, doing a GET on `/folders/apiMeta` returns metadata on the folders service REST API. This is shown below in Output 10:

```
{
    "build": {
        "version":1,
        "timeStamp":"2018-02-15T20:23:57Z",
        "buildVersion":"2.13.7-m.1",
        "sourceId":"d20f21ff81f7e3a6fcbc8707527fe95c963b79e8",
        "sourceTimeStamp":"2018-02-15T20:23:17Z" },
    "version":2,
    "apiVersion":1
}
```

Output 10. API Metadata for Folders Service

Detailed API documentation is available to developers from the developer.sas.com website. At the time of this writing only the CAS API doc is available online but Open API doc is being added. In the meantime, for SAS Viya 3.3, services provide their Open API, or Swagger, specification via their `/apiMeta/api` endpoint. This URI produces JSON that can be parsed programmatically or loaded into the Swagger UI. The Swagger UI is an open-source tool that allows you to visualize and explore detailed information about the API, including required parameters and media types. Although the Swagger UI can be used to execute calls, it doesn't provide any way to authenticate, thereby limiting its usefulness. However, it displays a curl command, and you can copy that command and add an authentication header as described in the following sections.

AUTHENTICATION

Browser-Based Applications

First let's cover how SAS Viya applications such as SAS Visual Analytics handle authentication. These are browser-client applications that are accessed from a URL (for example, `/SASVisualAnalytics/`) and downloaded to the browser as javascript. The web app itself requires authentication and authorization so that it redirects the browser to SASLogon for the authorization code flow. This was described earlier in the overview. Once the application is running in the browser, it makes calls directly to the APIs but does not normally handle access tokens directly. Like the web app that provided the application to begin with, the services hosting the APIs now take on dual roles of client and protected resource. The application in the browser simply makes requests to the API, and the API becomes the client, redirecting the browser again to SASLogon for the authorization code flow. You can try this out yourself with any GET request by simply entering the URI into a web browser.

After the flow is complete and the browser has been authenticated to the API, the API returns an HTTP session cookie so that subsequent requests do not need to be re-authenticated as long as the session is kept active. Note that the HTTP session is used in the context of security, while the APIs themselves are usually stateless, an important characteristic of RESTful services.

Bearer Tokens

Scripts and non-browser based applications typically need to call the APIs with a `Bearer` token. Bearer tokens usage is described in RFC6750. Access tokens become Bearer tokens when they are used by a party (a "bearer") to gain access to protected resources. Typically, that part is the client that obtained the access token in the first place. The token can be passed to the API as a query string parameter, but recommended practice is to pass it on the HTTP `Authorization` header.

The example shown in Output 11 below uses curl to make a request using a Bearer token. In this case it is fetching identity information for the user authenticated by the token:

```
# curl -s -X GET "https://localhost/identities/users/@currentUser" \
       -H "Accept: application/json" \
       -H "Authorization: Bearer $TOKEN" | python -m json.tool
{
   "links": [
       {
           "method": "GET",
           "rel": "self",
           "href": "/identities/users/bob",
           "uri": "/identities/users/bob",
           "type": "application/vnd.sas.identity.user"
       },
       {
           "method": "GET",
           "rel": "alternate",
           "href": "/identities/users/bob",
           "uri": "/identities/users/bob",
           "type": "application/vnd.sas.identity.user.summary"
       },
       {
           "method": "GET",
           "rel": "avatarContent",
           "href": "/identities/users/bob/avatar/content",
           "uri": "/identities/users/bob/avatar/content",
           "type": "image/*"
       },
       {
           "method": "GET",
           "rel": "avatar",
           "href": "/identities/users/bob/avatar",
           "uri": "/identities/users/bob/avatar",
           "type": "application/vnd.sas.identity.avatar"
       },
       {
           "method": "GET",
           "rel": "memberships",
           "href": "/identities/users/bob/memberships",
           "uri": "/identities/users/bob/memberships",
           "type": "application/vnd.sas.collection",
           "itemType": "application/vnd.sas.identity.group.membership"
       },
       {
           "method": "GET",
           "rel": "flatMemberships",
           "href": "/identities/users/bob/memberships",
           "uri": "/identities/users/bob/memberships",
```

```
            "type": "application/vnd.sas.collection",
            "itemType": "application/vnd.sas.identity.group.membership.flat"
        }
    ],
    "version": 1,
    "id": "bob",
    "name": "Bob",
    "providerId": "ldap",
    "creationTimeStamp": "2003-03-05T19:16:22.000Z",
    "modifiedTimeStamp": "2017-10-31T10:25:40.000Z",
    "state": "active",
    "emailAddresses": [
        {
            "type": "work",
            "value": "bob@example.com"
        }
    ],
    "phoneNumbers": [
        {
            "type": "work",
            "value": "5551212"
        }
    ]
}
```

Output 11. Using curl to Make a Request against the Identities Service API

The example above returns information about and links for the current authenticated user (@currentUser). This corresponds to the user represented in the token. Note that services do not normally return an HTTP session for API calls made with a Bearer token, so the token needs to be passed on every request.

SAS HTTP Procedure

PROC HTTP can be used to call SAS Viya REST APIs. Beginning with SAS Viya 3.3, PROC HTTP automatically obtains and attaches a Bearer token when user name and password credentials are specified in the following manner:

```
proc http url=.....
  OAUTH_BEARER=SAS_SERVICES
  username="<username>"
  password="<password>"
;
```

The only requirement is that `SERVICESBASEURL` is set. Note this feature is not supported on Base SAS 9.4.

COMMAND-LINE INTERFACES

While this paper is primarily focused on how to interact directly with the REST APIs, SAS Viya also provides command-line interfaces (CLIs) that simplify many common operations. The CLIs can be run from the machine where SAS Viya is deployed, located under `/opt/sas/viya/home/bin/`, or they can be downloaded to another machine. They are documented in the *SAS Viya: Administration Guide*.

To begin with, an access token can be obtained using password credentials. This is shown in Output 12 below:

```
# sas-admin --sas-endpoint http://localhost auth login
Enter credentials for http://localhost:
```

```
Userid> bob

Password>
Login succeeded. Token saved.
```

Output 12. Authenticating with the CLI

Now other CLIs can be run and they will use the saved token, or the saved token can be used to call the APIs directly. To use the saved token after running `sas-admin auth login`, examine the contents of the file `~/.sas/credentials.json` and copy the `access-token` field. Output 13 below is an example file (contents have been shortened for readability):

```
{
  "Default": {
    "access-token":
"eyJhbGciOiJSUzI1NiIsImtCI6ImxlZ2FjeS10b2tlbi1rZXkiLCJ0eXAiOiJKV1QifQ...",
    "refresh-token":
"eyJhbGciOiJSUzI1NiIsImtCI6ImxlZ2FjeS10b2tlbi1rZXkiLCJ0eXAiOiJKV1QifQ..."
  }
}
```

Output 13. Contents of the Credentials Stored by the CLI

The following command shown in Output 14, similar to the one shown earlier, fetches information about the current user (that is, the user authenticated by the token):

```
# ./sas-admin --sas-endpoint http://localhost identities whoami
{
    "addresses": null,
    "creationTimeStamp": "2003-03-05T19:16:22.000Z",
    "emailAddresses": [
        {
            "value": "bob@example.com"
        }
    ],
    "id": "bob",
    "modifiedTimeStamp": "2017-12-07T11:21:43.000Z",
    "name": "Bob",
    "phoneNumbers": [
        {
            "type": "work",
            "value": "5551212"
        }
    ],
    "providerId": "ldap",
    "state": "active",
    "title": ""
}
```

Output 14. Using the CLI to Call the Identities API

CONCLUSION

SAS made the right choice adopting OAuth 2.0 and OpenID Connect for SAS Viya, as the industry leaders have clearly gravitated that way. Combined with rich RESTful APIs that provide metadata and follow HATEOAS principals, the APIs are more open than ever. End-users can choose to work directly with the APIs using standard protocols, leverage a growing number of command-line interfaces, or even call the APIs from within SAS code. However, client registration is a new concept and can be a stumbling block. There is no UI provided and the many options can be confusing. There also is no simple UI where a user can obtain an authorization code for cut and paste. Perhaps future work will focus on these areas.

REFERENCES

"JSON Web Signature (JWS)." *Internet Engineering Task Force,* May 2015. Available at https://tools.ietf.org/html/rfc7515.

"JSON Web Token (JWT)." *Internet Engineering Task Force,* May 2015. Available at https://tools.ietf.org/html/rfc7519.

"The OAuth 2.0 Authorization Framework." *Internet Engineering Task Force,* October 2012. Available at https://tools.ietf.org/html/rfc6749.

"The OAuth 2.0 Authorization Framework: Bearer Token Usage." *Internet Engineering Task Force,* October 2012. Available at https://tools.ietf.org/html/rfc6750.

"OpenID Connect Core 1.0 incorporating errata set 1." *OpenID Foundation,* November 8, 2014. Available at http://openid.net/specs/openid-connect-core-1_0.html.

SAS Institute Inc. "CAS REST API." Version 3.0.1. Available at https://developer.sas.com/apis/cas/rest/v3.0.1/apidoc.html.

SAS Institute Inc. *SAS® Viya™ 3.2 System Options: Reference.* Available at http://go.documentation.sas.com/?docsetId=lesysoptsref&docsetTarget=p191674k3chjf3n1vze4cfzfn18e.htm&docsetVersion=3.2.

SAS Institute Inc. *SAS® Viya™ 3.3 Administration: Command-Line Interfaces*. Available at http://go.documentation.sas.com/?docsetId=calcli&docsetTarget=titlepage.htm&docsetVersion=3.3.

"What is UAA?" *Cloud Foundry Documentation*. Available at https://docs.cloudfoundry.org/uaa/uaa-overview.html#overview. Accessed December 2017.

ACKNOWLEDGMENTS

Stuart Rogers was especially helpful in reviewing the paper and graciously offering diagrams from the training courses he has created. Joseph Henry provided the information for the HTTP procedure section and has always been more than willing to point out areas that need improvement. Both are employees of SAS Institute.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mike Roda
SAS Institute Inc.
100 SAS Campus Drive
Cary, NC 27513
mike.roda@sas.com