

Article

[Vinicius Maranh...](#) · Apr 2, 2020 5m read

Securing your APIs with OAuth 2.0 in InterSystems API Management – Part 3

In this 3-part series of articles, is shown how you can use IAM to simply add security, according to OAuth 2.0 standards, to a previously unauthenticated service deployed in IRIS.

In the [first part](#), was provided some OAuth 2.0 background together with some IRIS and IAM initial definitions and configurations in order to facilitate the understanding of the whole process of securing your services.

The [second part](#) discussed and showed in detail the steps needed to configure IAM to validate the access token present in the incoming request and forward the request to the backend if the validation succeeds.

This last part of the series will discuss and demonstrate the configurations needed to IAM generate an access token (acting as an authorization server) and validate it, together with some important final considerations.

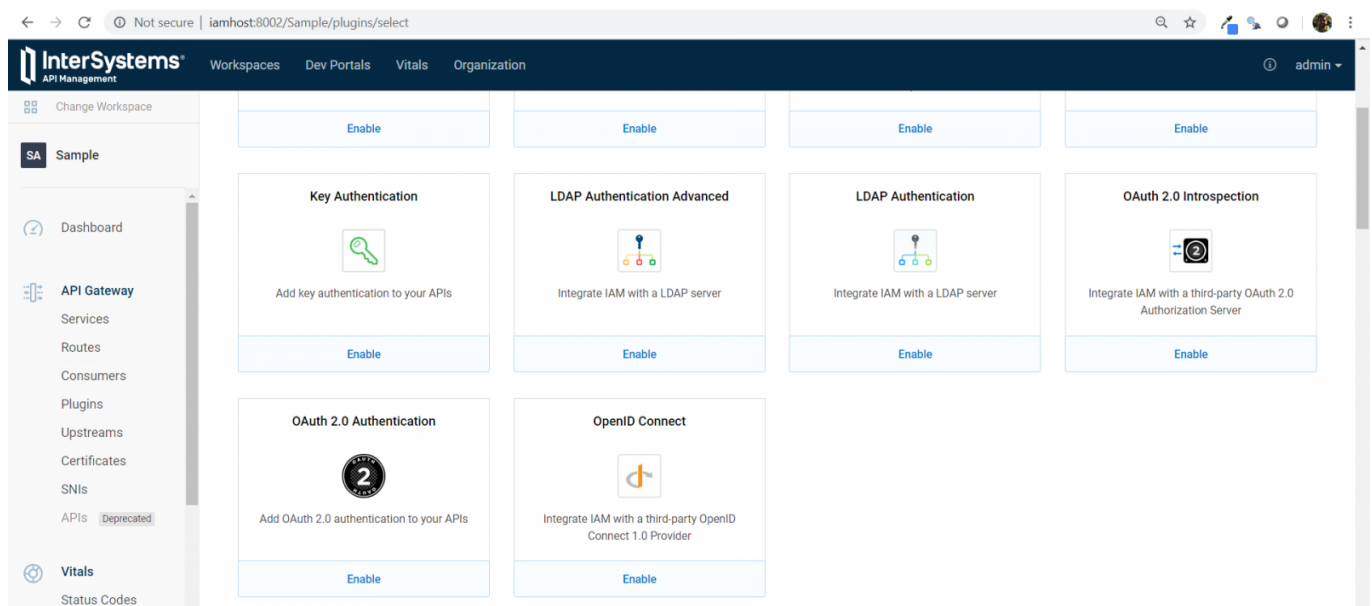
If you want to try IAM, please contact your InterSystems Sales Representative.

Scenario 2: IAM as an authorization server and access token validator

In this scenario, differently from the first scenario, we are going to use a plugin called “ OAuth 2.0 Authentication ” .

In order to use IAM as the authorization server in this Resource Owner Password Credentials flow, the username and password must be authenticated by the client application. The request to get the access token from IAM should be made only if the authentication is successful.

Let ' s start by adding it to our “ SampleIRISService ” . As you can see in the screenshot below, we have some different fields to fill up in order to configure this plugin.



First of all, we are going to paste the id of our “ SampleIRISService ” into the field “ serviceid ” to enable this plugin to our service.

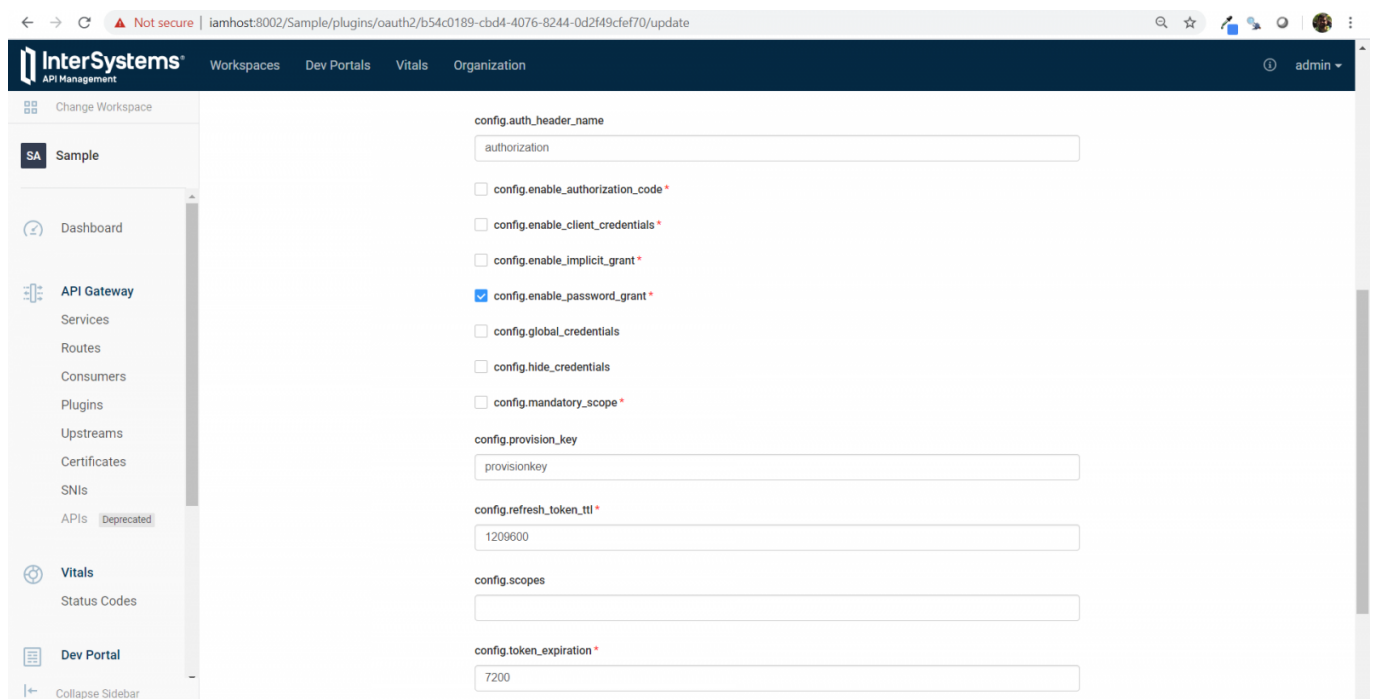
In the field “ config.authheadername ” we are going to specify the header name that will carry the authorization token. In this case, I ’ ll leave the default value of “ authorization ” .

The “ OAuth 2.0 Authentication ” plugin supports the Authorization Code Grant, Client Credentials, Implicit Grant or Resource Owner Password Credentials Grant OAuth 2.0 flows. As we are using the Resource Owner Password Credentials flow in this article, we will check the box “ config.enablepasswordgrant ” .

In the “ config.provisionkey ” field, enter any string to be used as the provision key. This value will be used to request an access token to IAM.

In this case, I left all the other fields with the default value. You can check the full reference for each field in the plugin documentation available [here](#).

Here is how the plugin configuration looks like at the end:



The screenshot shows the InterSystems API Management console. The left sidebar has a navigation menu with 'API Gateway' selected. The main panel displays the configuration for the OAuth 2.0 Authentication plugin. The configuration fields are as follows:

Field	Value
config.auth_header_name	authorization
config.enable_authorization_code *	<input type="checkbox"/>
config.enable_client_credentials *	<input type="checkbox"/>
config.enable_implicit_grant *	<input type="checkbox"/>
config.enable_password_grant *	<input checked="" type="checkbox"/>
config.global_credentials	<input type="checkbox"/>
config.hide_credentials	<input type="checkbox"/>
config.mandatory_scope *	<input type="checkbox"/>
config.provision_key	provisionkey
config.refresh_token_ttl *	1209600
config.scopes	
config.token_expiration *	7200

Once the plugin is created, we need to create the credentials to our “ ClientApp ” consumer.

To do so, go to “ Consumers ” on the left menu and click on “ ClientApp ” . Next, click on “ Credentials ” tab and then on “ New OAuth 2.0 Application ” button.

The screenshot shows the InterSystems API Management console. The top navigation bar includes 'Workspaces', 'Dev Portals', 'Vitals', and 'Organization'. The left sidebar shows a 'Change Workspace' button and a list of items: 'SA Sample' (selected), 'Dashboard', 'API Gateway' (with sub-items: Services, Routes, Consumers, Plugins, Upstreams, Certificates, SNIs, APIs, and a deprecated 'APIs' item), and 'Vitals' (with 'Status Codes' sub-item). The main content area is titled 'ClientApp' and displays the following details:

created_at	February 18th 2020, 3:50:03pm
id	ebf70f90-6342-47dc-877e-a2f4b45657cd
type	proxy
username	ClientApp

Below the details are three tabs: 'Activity', 'Credentials', and 'Plugins'. The 'Activity' tab is active, showing the heading 'OAuth 2.0'. The main content area under this tab is empty, displaying the message 'No OAuth 2.0 Applications Yet' and a button labeled 'New OAuth 2.0 Application'.

On the following page, enter any name to identify your application on the field “ name ” , define a client id and a client secret, respectively, on the fields “ clientid ” and “ clientsecret ” and lastly, enter the URL in your application where users will be sent after authorization on the field “ redirecturi ” . Then, click on “ Create ” .

The screenshot shows a web browser window with the InterSystems API Management interface. The browser's address bar shows the URL: `iamhost:8002/Sample/consumers/ebf70f90-6342-47dc-877e-a2f4b45657cd/oauth2/create`. The page title is "Create OAuth 2.0 Application". The left sidebar contains a navigation menu with options: "Change Workspace", "Sample", "Dashboard", "API Gateway", "Services", "Routes", "Consumers", "Plugins", "Upstreams", "Certificates", "SNIs", and "APIs" (marked as "Deprecated"). The main content area contains the "Create OAuth 2.0 Application" form with the following fields:

- name**:
- client_id**:
- client_secret**:
- redirect_uri**:

At the bottom of the form are two buttons: "Create" (in blue) and "Cancel" (in grey).

Now, you are ready to send requests.

The first request that we need to make is to obtain the access token from IAM. The “ OAuth 2.0 Authentication ” plugin automatically create an endpoint appending the path “ /oauth2/token ” to the already created route.
Note: Make sure that you use HTTPS protocol and IAM ’ s proxy port listening to TLS/SSL requests (the default port is 8443). This is an OAuth 2.0 requirement.

Therefore, in this case, we would need to make a POST request to the URL:

<https://iamhost:8443/event/oauth2/token>

In the request body, you should include the following JSON:

```
{
  "clientid": "clientid",
  "clientsecret": "clientsecret",
  "granttype": "password",
  "provisionkey": "provisionkey",
  "authenticateduserid": "1"
}
```

As you can see, this JSON contains values defined both during “ OAuth 2.0 Authentication ” plugin creation, such as “ granttype ” and “ provisionkey ”, and during Consumer ’ s credentials creation, such as “ clientid ” and “ clientsecret ”.

The parameter “ authenticateduserid ” should also be added by the client application when the username and

password provided are successfully authenticated. Its value should be used to uniquely identify the authenticated user.

The request and its respective response should look like this:

The screenshot shows a REST client interface with a POST request to `https://iamhost:8443/event/oauth2/token`. The request body is a JSON object with the following fields:

```
{
  "client_id": "clientid",
  "client_secret": "clientsecret",
  "grant_type": "password",
  "provision_key": "provisionkey",
  "authenticated_userid": "1"
}
```

The response status is 200 OK. The response body is a JSON object with the following fields:

```
{
  "refresh_token": "E50m6Yd9xkly6lybgo3DOvu5ktZTjzkwF",
  "token_type": "bearer",
  "access_token": "hCviOzuhHTUdFxpCM6z5KUf4IRnV1IsD",
  "expires_in": 7200
}
```

With that, we can now make a request to get the event data including the “`access_token`” value from the response above as a “`Bearer Token`” in a GET request to the URL

<https://iamhost:8443/event/1>

The screenshot shows a REST client interface with a GET request to `https://iamhost:8443/event/1`. The request is configured with the authorization type set to “`Bearer Token`”. A warning message states: “Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)”. The token value `hCviOzuhHTUdFxpCM6z5KUf4IRnV1IsD` is entered in the “Token” field.

The response status is 200 OK. The response body is a JSON object with the following fields:

```
{
  "id": 1,
  "Name": "Global Summit",
  "Location": {
    "id": 1,
    "City": "Boston",
    "Country": "United States of America"
  }
}
```

If your access token expires, you can generate a new access token using the refresh token that you received

together with the expired access token by making a POST request to the same endpoint used to get an access token, with a slightly different body:

```
{
  "clientid": "clientid",
  "clientsecret": "clientsecret",
  "granttype": "refresh_token",
  "refresh_token": "E50m6Yd9xWy6lybgo3DOvu5ktZTjzkwF"
}
```

The request and its respective response should look like this:

The screenshot displays a REST client interface with a tab titled "POST Refresh IAM Token". The request is a POST to `https://iamhost:8443/event/oauth2/token`. The body is a JSON object with the following fields: `client_id`, `client_secret`, `grant_type` (set to "refresh_token"), and `refresh_token` (set to "E50m6Yd9xWy6lybgo3DOvu5ktZTjzkwF"). The response status is 200 OK, with a time of 44ms and a size of 381 B. The response body is a JSON object containing `refresh_token`, `token_type` ("bearer"), `access_token` (a long alphanumeric string), and `expires_in` (7200).

```
POST Refresh IAM Token
Refresh IAM Token
POST https://iamhost:8443/event/oauth2/token
Send Save
Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code
none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify
1 {
2   "client_id": "clientid",
3   "client_secret": "clientsecret",
4   "grant_type": "refresh_token",
5   "refresh_token": "E50m6Yd9xWy6lybgo3DOvu5ktZTjzkwF"
6 }
7
8
Body Cookies Headers (7) Test Results Status: 200 OK Time: 44ms Size: 381 B Save Response
Pretty Raw Preview Visualize JSON
1 {
2   "refresh_token": "9CgRksqh80uz881zBMtOm6dNsu01HhYt",
3   "token_type": "bearer",
4   "access_token": "7ACfJXvLwFuJrd2WfuCX15P9qHDeT9cA",
5   "expires_in": 7200
6 }
```

One interesting feature of the “ OAuth 2.0 Authentication ” plugin is the ability to view and invalidate access tokens.

To list the tokens, send a GET request to the following endpoint of IAM ’ s Admin API:

<https://iamhost:8444/{workspacename}/oauth2tokens>

where {workspacename} is the name of the IAM workspace used. Make sure to enter the necessary credentials to call IAM ’ s Admin API in case if you have enabled RBAC.

The screenshot shows the InterSystems API Management console. At the top, there's a tab labeled "GET List IAM tokens". Below it, the request method is "GET" and the URL is "https://iamhost:8444/Sample/oauth2_tokens". The "Send" button is visible. Below the request bar, there's a "Headers (1)" section with a table:

KEY	VALUE	DESCRIPTION
Kong-Admin-Token	sysAPI	
Key	Value	Description

Below the headers, there's a "Temporary Headers (7)" section. The main body of the console shows the response status "200 OK", time "27ms", and size "5.98 KB". The response body is displayed in JSON format:

```
{
  "authenticated_userid": "1",
  "refresh_token": "MpnwawpGLx8B8z4cfAvTctNQeTYj8hCD",
  "token_type": "bearer",
  "access_token": "15dKBDdqNH3RCoSNOQvZGAM6JKUundXy",
  "expires_in": 7200,
  "id": "41156679-33b8-43b5-a4a2-f0c9a5f45b7e",
  "service_id": "24f38db6-c989-439b-ac25-d449353b64ce"
},
{
  "created_at": 1582900874000,
  "credential_id": "c0c3fddb-6bac-4701-804f-cdd722864c41",
  "scope": "",
  "authenticated_userid": "1",
  "refresh_token": "9CgRksqh80uz881zBMtOm6dNsu01HhYt",
  "token_type": "bearer",
  "access_token": "7ACf3XvLwFuJrd2wFuCX15P9qHDeT9c4",
  "expires_in": 7200,
  "id": "e92f5963-760f-43f4-8d82-b29a6215e3b6",
  "service_id": "24f38db6-c989-439b-ac25-d449353b64ce"
}
```

Note that “credentialid” is the id of the OAuth application that we created inside the ClientApp consumer (in this case is called SampleApp) and “serviceid” is the id of our “SampleIRISService” which this plugin is applied to.

To invalidate a token, you can send a DELETE request to the following endpoint

<https://iamhost:8444/Sample/oauth2tokens/{tokenId}>

where {tokenId} is the id of the token to be invalidated.

The screenshot shows the InterSystems API Management console. At the top, there's a tab labeled "DEL Invalidate Token". Below it, the request method is "DELETE" and the URL is "https://iamhost:8444/Sample/oauth2_tokens/e92f5963-760f-43f4-8d82-b29a6215e3b6". The "Send" button is visible. Below the request bar, there's a "Headers (1)" section with a table:

KEY	VALUE	DESCRIPTION
Kong-Admin-Token	sysAPI	
Key	Value	Description

Below the headers, there's a "Temporary Headers (8)" section. The main body of the console shows the response status "204 No Content", time "24ms", and size "291 B". The response body is empty.

If we try to use the invalidated token, we get a message saying that the token is invalid or expired if we send a GET request containing this invalidated token as a Bearer Token to the URL:

<https://iamhost:8443/event/1>

The screenshot displays the InterSystems API Management console interface. At the top, a tab for 'getEvent IAM OAuth plugin' is active. The main area shows a GET request to 'https://iamhost:8443/event/1'. The 'Authorization' tab is selected, showing a 'Bearer Token' type. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)'. The 'Token' field contains the value '7ACfjXvLwfuJrd2WfUCXl5P9qHDeT9cA'. Below this, the 'Body' tab is selected, showing a JSON response with an error:

```
{  "error_description": "The access token is invalid or has expired",  "error": "invalid_token"}
```

. The status bar at the bottom indicates 'Status: 401 Unauthorized', 'Time: 34ms', and 'Size: 423 B'.

Final Considerations

In this article was demonstrated how you can add OAuth 2.0 authentication in IAM to an unauthenticated service deployed in IRIS. You should keep in mind that the service itself will continue to be unauthenticated in IRIS. Therefore, if anyone calls the IRIS service endpoint directly, bypassing the IAM layer, will be able to see the information without any authentication. For that reason, it is important to have security rules in a network level to prevent unwanted requests to bypass IAM layer.

You can learn more about IAM [here](#).

If you want to try IAM, please contact your InterSystems Sales Representative.

[#API](#) [#OAuth2](#) [#REST API](#) [#Security](#) [#InterSystems IRIS](#)

Source

URL: <https://community.intersystems.com/post/securing-your-apis-oauth-20-intersystems-api-management-%E2%80%93-part-3>