Using OAuth 2.0 to Access VSA APIs

A Programming Primer

Introduction

VSA APIs use the OAuth 2.0 protocol for authentication and authorization. Kaseya supports common OAuth 2.0 scenarios to permit access for web server, installed and client-side applications.

To start, you will need to obtain OAuth 2.0 client credentials from the Kaseya Virtual System Administrator (VSA) “Server Management” Console. Then your client application needs to link itself with the VSA so trust can be established. Finally, as users need access to data and APIs in the VSA for the first time, they will need to authorize access through a traditional “consent” code flow.

This document outlines the registration and authorization scenarios Kaseya supports, and provides guidance on how to build your first OAuth client that communicates with the VSA.

> **!** **Note:** Given the security implications of getting the implementation correct, we strongly encourage you to use commercial or well-supported open source OAuth 2.0 libraries when interacting with Kaseya’s OAuth 2.0 endpoints. It is a best practice to use well-tested code provided by others, and it will help you protect yourself and your users.

Basic steps

All applications follow a basic pattern when accessing a VSA API using OAuth 2.0. At a high level, you follow five steps:

1. Obtain OAuth 2.0 credentials from the VSA Server by registering your application.
2. Authenticate to the VSA to obtain a temporary authorization code.
3. Exchange the authorization code for an access token.
4. Send the access token to the VSA API when needed.
5. Refresh your access token, if necessary.

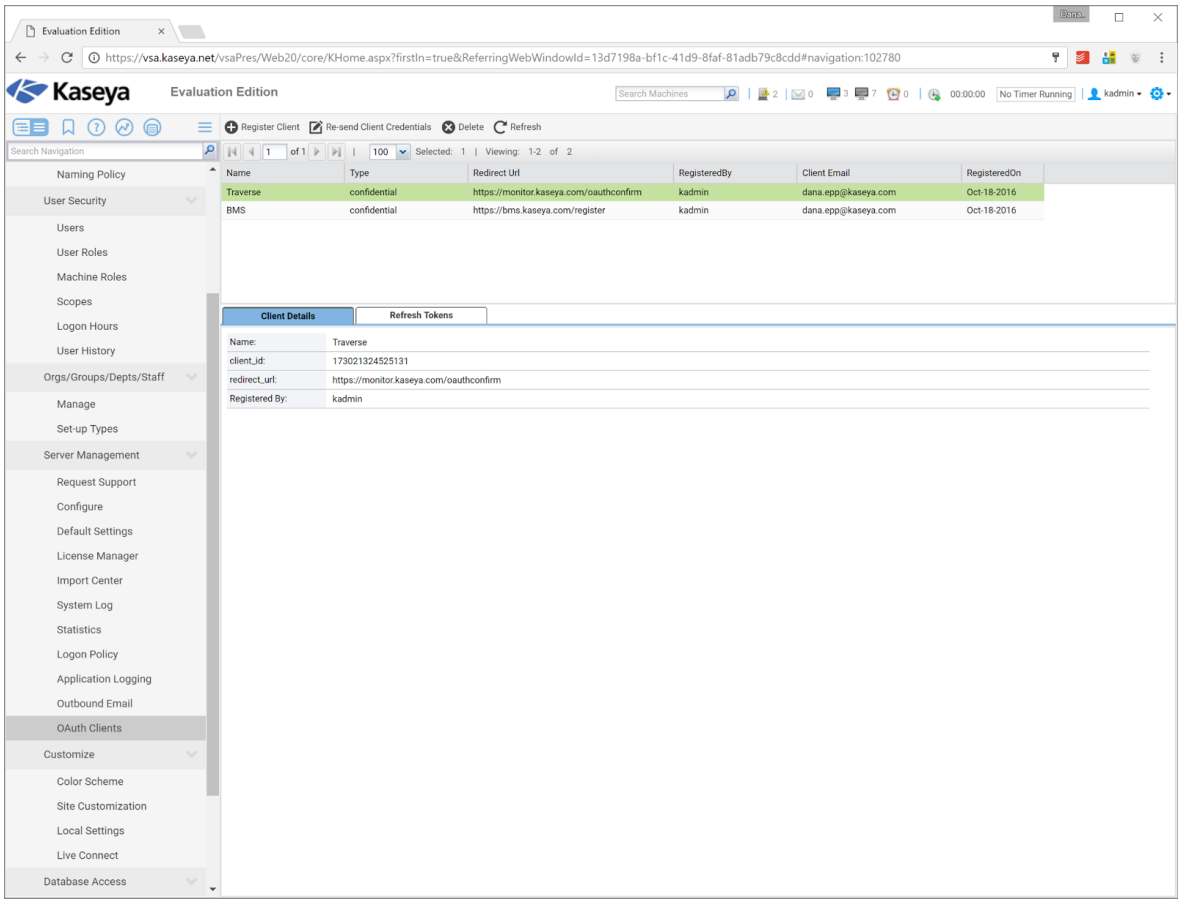
© 2016 Kaseya. All Rights Reserved.

1. Obtain OAuth 2.0 credentials from the VSA Server

Visit the VSA server to obtain OAuth 2.0 credentials such as a client ID and client secret that are known to both the VSA and your application. This can be done from `System > Server Management > OAuth Clients`. Once the client is registered, a `client_id` and `client_secret` are generated by the system. The `client_id` is shown on the UI post registration, and both the `client_id` and `client_secret` are sent to the email address provided at the time. The `client_secret` is confidential and must be stored securely by the application.

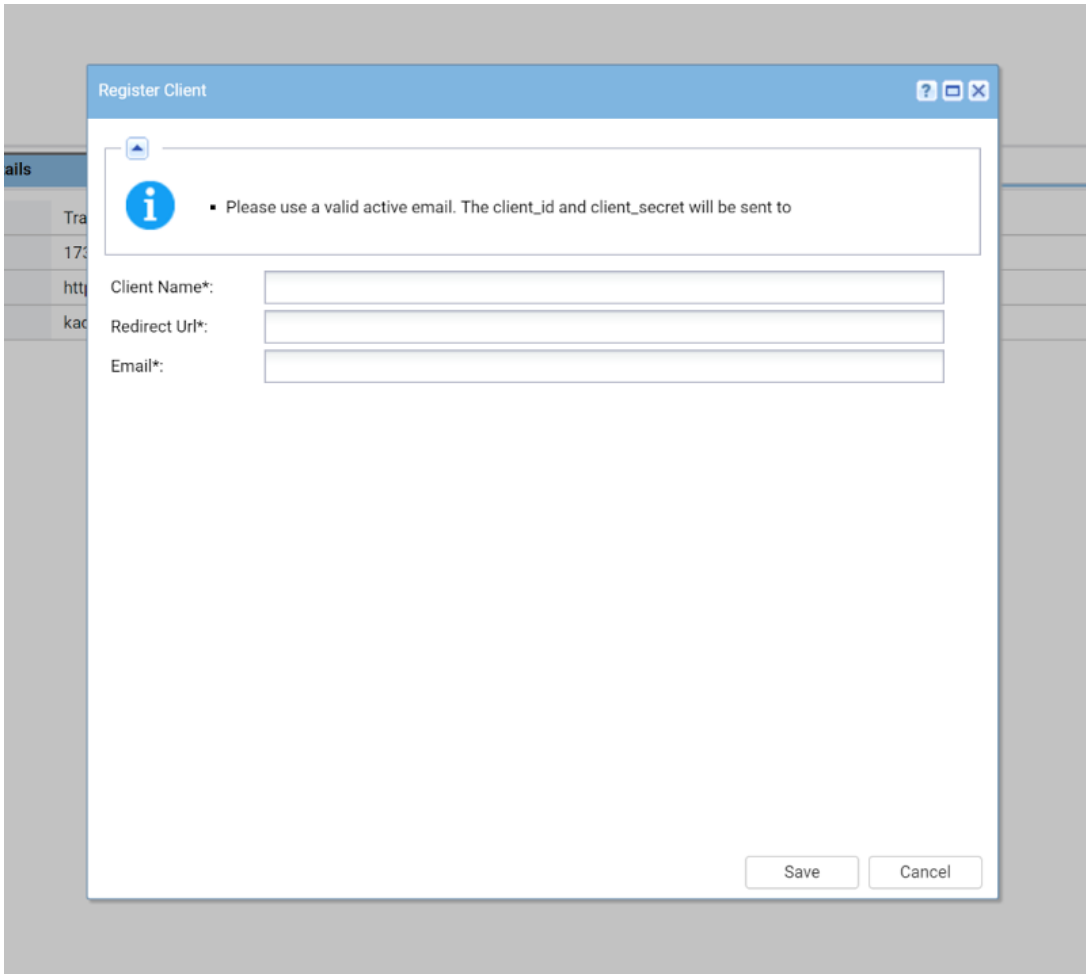
> **Note:** Support for OAuth 2.0 clients is available in Kaseya VSA v9.4 and above.

The following screenshot shows the user interface for all registered client applications. Here you can register clients, re-send client credentials and revoke refresh tokens for existing clients.



© 2016 Kaseya. All Rights Reserved.

The following screenshot shows the user interface to actually register a client application:



2. Authenticate to obtain a temporary authorization code

To allow a client access to a user's protected resource, the client application must open an HTTP/S session pointing to the following location:

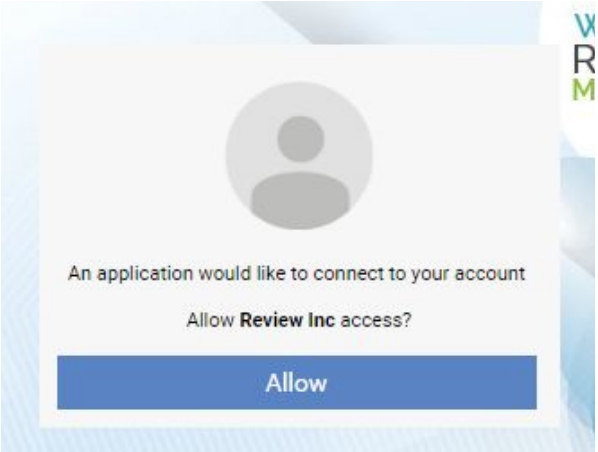
```
https://{vsa_url}/vsapres/web20/core/login.aspx?response_type=code&redirect_uri={redirect_uri}&client_id={client_id}
```

- {vsa_url} - the url of the VSA that the client application registered on
- {redirect_uri} - the redirect_uri provided during registration, url encoded
- {client_id} - the client ID issued to the client application during registration

If the client_id and redirect_uri are invalid, the login page will fail and show an "Invalid Request"

© 2016 Kaseya. All Rights Reserved.

error. If the parameters are valid, the login page will prompt for credentials. After the user logs on, they will be prompted for consent to authorize the client application to communicate with the VSA, on behalf of them. It will look something like this:



If the user clicks **Allow**, they are essentially giving the client application access to their protected resources. Clicking Allow will cause the HTTP/S session to redirect to the `redirect_uri` for the client application, with the following uri:

```
{redirect_uri}?code={auth_code}
```

The `auth_code` passed to client application has a lifetime of 5 minutes and must be used to make a token request within that timeframe.

> **Note:** This OAuth 2.0 code flow process requires a proper HTTP/S session. If you are building a native application that wishes to use the VSA API you will need to support a browser control or otherwise connect via HTTP/S to properly conduct this transaction.

3. Exchange the authorization code for an access token

Once you have received an `auth_code` you must exchange it within 5 minutes for an `access_token` and `refresh_token`. To do this the client application will need to make a request to the url:

```
POST https://{vsa_url}/api/v1.0/authorize
```

...with the following `x-www-form-urlencoded` parameters in the request body.

© 2016 Kaseya. All Rights Reserved.

- `grant_type` - must be set to '`authorization_code`'
- `code` - must be set to the `auth_code` obtained in the previous step
- `redirect_uri` - the url encoded redirect uri of the client application
- `client_id` - the `client_id` of the client application
- `client_secret` - the `client_secret` of the client application

> **Note:** Kaseya's implementation of OAuth 2.0 on the VSA will NOT permit authorization to API endpoints if SSL has not been configured for the server.

If the request is invalid, an appropriate response according to the RFC is returned. A valid request will produce the following response:

```
{
  "access_token": "21269455",
  "token_type": "Bearer",
  "expires_in": 1800,
  "refresh_token": "83fedffdb7ec44b586925b78f3bf76648ea45c95cbf7484189d2e1739e120ed2"
}
```

The response contains an `access_token`, it's lifetime in seconds defined by the `expires_in` parameter, the token type of 'Bearer' in the `token_type` parameter, and a `refresh_token`. The `refresh_token` is confidential and should be stored securely by the client application.

> **Note:** It is the responsibility of the client application to track the expiration time of the access token and utilize the refresh token to request a new access token as required. It is a best practice to conduct the token exchange before it actually expires, except on initial connection on application restart, where a new access token should be fetched immediately anyways.

4. Send the access token to the VSA API when needed

After a client application obtains an access token, it sends the token to the VSA REST API in an HTTP authorization header as the 'Bearer' token. It is possible to send tokens as URI query-string parameters, but we don't recommend it, because URI parameters can end up in log files that are not completely secure. Also, it is good REST practice to avoid creating unnecessary URI parameter names.

Access tokens are valid only for the set of operations and resources described in the scope of the token request. At the time of this writing, all VSA APIs honor the roles and scopes within the VSA to limit access to data automatically, and not that of the scopes within an access token.

© 2016 Kaseya. All Rights Reserved.

> **Confused?** A scope in OAuth 2.0 is NOT the same thing as a scope within the VSA. Where a "scoped" access token limits what APIs can be called through permissions stored in the JSON Web Token (JWT), a VSA scope limits how the APIs filter access to the data on the backend. The result? While the VSA authorization service can use permissions in the JWT, today we use the security model inside of VSA instead which our customers better understand and have already configured for limited user access.

5. Refresh your access token, if necessary

Access tokens have a limited lifetime, typically 30 minutes. If your application needs access to a VSA API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows your application to obtain new access tokens as required.

To do this, post an HTTP/S request to the following endpoint:

```
POST https://{vsa_uri}/api/v1.0/token
```

...with the following `x-www-form-urlencoded` parameters in the request body.

- `grant_type` - must be set to '`refresh_token`'
- `refresh_token` - the refresh_token stored by the client
- `redirect_uri` - the url encoded redirect uri of the client application
- `client_id` - the client_id of the client application
- `client_secret` - the client_secret of the client application

If the request is invalid, an appropriate response according to the RFC is returned. A valid request will produce something like the following response:

```
{
  "access_token": "12429176",
  "token_type": "Bearer",
  "expires_in": 1800,
  "refresh_token": "c7dd3673589c42f4ad215340e89ce5feed5b2241d4ab4c98bd3d9a1c0317f878"
}
```

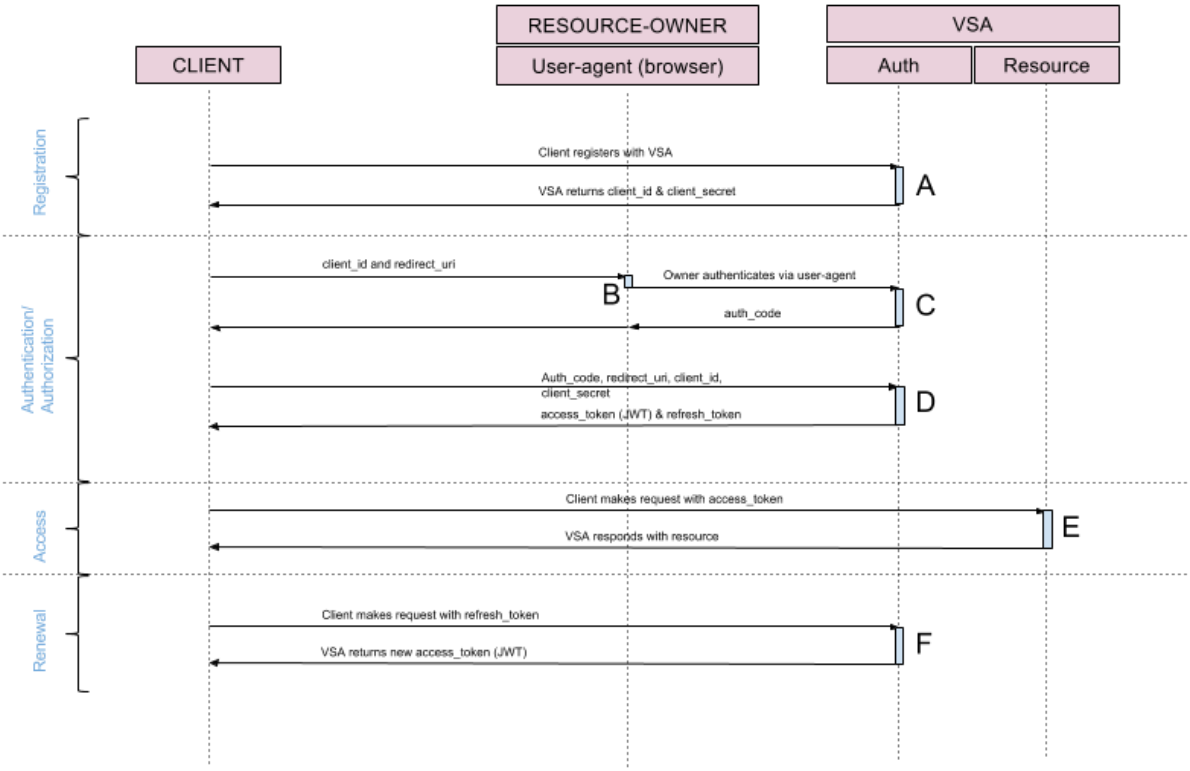
Again, the response contains an `access_token`, it's lifetime in seconds defined by the `expires_in` parameter, the token type of 'Bearer' in the `token_type` parameter, and a `refresh_token`. Note that the refresh_token has been re-generated and replaces any previously issued `refresh_token` to the client. The client must now replace the previously

© 2016 Kaseya. All Rights Reserved.

stored token with this one.

> **Note:** Save refresh tokens in secure long-term storage and continue to use them as long as they remain valid. By default, a refresh token is good for 60 days. If a refresh token expires, the client application needs to follow the OAuth 2.0 code flow authorization process to re-establish trust between systems.

Authorization Sequence Diagram



© 2016 Kaseya. All Rights Reserved.