Widget 2.0 JWT authentication

Last Modified on 08/27/2025 5:27 pm EDT

When using the widget with private knowledge bases, or knowledge bases with a mix of public and private content that should all be shown in the widget, SSO authentication is required.

To implement SSO authentication for your readers in the widget, choose between **OAuth2** or **JSON Web Tokens (JWT)**.

With JWT authentication, you'll need:

1. A JWT client secret to retrieve the token.
2. To update your widget embed script to request the token to authenticate and display knowledge base content to your readers.

Get your client secret

First, generate the JWT Client Secret for your knowledge base:

1. Go to **KB settings > Widget**.
2. In the **Admin Settings** section, under **JWT Authentication**, copy the **JWT Secret**. If no secret is displayed, select **Generate Client Secret** to generate one.

Now that you have your client secret, use it to request an access token from the JWT token endpoint.

We recommend modifying the default embed script to include a section that calls the `_ko19.onReady` method and get the token within that method. Review the sections below for overall workflow examples and a sample call to this method. Refer to the Widget methods and functions dictionary for more information on working with this method.

Working with JWT authentication

For information about what JWT is and how you can use it, refer to this JWT introduction article. For use with our contextual help widget, use the following requirements:

JWT header

Your JWT header must specify the HS256 encryption algorithm:

```
// Encryption algorithm must be HS256
{
  "alg": "HS256",
  "typ": "JWT"
}
```

JWT payload

No sensitive information
JWT payloads are not encrypted and should **NOT INCLUDE** any sensitive information about the reader.
If you need to pass sensitive information about your readers, use the server side OAuth2 workflow.

Your JWT payload must include these fields and values:

```
{
  "iss" => "app.knowledgeowl.com", // Issuer - *required
  "aud" => "YOUR.KNOWLEDGEBASE.URL", // Audience - *required
  "iat" => unixTimestamp, // Issued At - *required
  "nbf" => unixTimestamp - 1000, // Not Before - *required
  "exp" => unixTimestamp + 1000, // Expires - *required
  "reader_ssoid" => endUserUID, // Unique ID - *required
  "reader_username" => endUserEmail, // Email or Username - *required
  "reader_groups" => "Support,Admin" // Comma separated list of reader group names - optional
}
```

Example authentication code

Here's a sample authentication code:

```
var _ko19 = _ko19 || {};
_ko19.__pc = '123xxxxxx-123xxxxxxx';
_ko19.base_url = '//yourkb.knowledgeowl.com';
!function() {
  var ko = document.createElement('script');
  ko.type = 'text/javascript';
  ko.async = true;

  ko.src = `${_ko19.base_url}/widget-app/assets/js/load.js`;
  document.head.appendChild(ko);

  /**
   * The widget is loaded and ready to receive commands
   */
  _ko19.onReady = function () {
    // Get the JWT token and pass it to the widget
    getToken();
  }

  /**
   * Form the authentication object and pass it to the widget for authentication
   */
  getToken = function() {
    /** Example of hitting an endpoint to get the JWT token **/
    /*
      $.get('my.jwtendpoint.url', function(response){
        if(response.token) {
   var JWT = response.token;
 }
      });
    */

    /** Example where the JWT already exists on the page **/
    var JWT = preDefinedVar; // 'eyJ0.eyJpc3Mi.dKSb41xr' - Example of shortened value for JWT
    var authToken = {'type': 'jwt', 'token': JWT};

    // Use the formed object with the JWT to authenticate
    authWidget(authToken);
  }

  /**
   * The widget will pass the response back from the authentication call here.
   * This can be useful for debugging purposes if any errors occur
   */
  _ko19.onAuthResp = function(response) {
    // Take some kind of action if the authentication fails
    if(response.error)
      console.log(response);
  }

  /**
   * The widget will call this function whenever authorization fails
   * This can occur before the user is logged in, and after the authentication expires
   * With most workflows, you will want to re-authorize the end-user
   * when this event occurs
   */
  _ko19.onAuthError = function () {
    // Take an action when the user is not logged in
    getToken()
  }

  /**
   * Pass your token as JSON into the widget's authenticate function
   * the JSON format should be as follows:
   * {'type': 'jwt|oauth', 'token': token_value}
   */
  authWidget = function(authToken) {
    _ko19.authenticate(authToken);
  }
}();
```

Turn on JWT widget authentication

When your script is ready for testing/usage, set the widget to use JWT authentication:

1. **Go to KB settings > Widget.**

2. In the **Admin Settings** section, under **JWT Authentication**, check the box to **Secure widget access using JWT**.
3. Be sure to **Save** your changes.

Generate a new JWT secret

For security reasons, you may need to periodically generate a new JWT secret. You can do this directly yourself in **KB settings > Widget**.

Once you regenerate the JWT secret, the existing JWT secret won't work, so your widget won't authenticate until you update it to use the new secret.

To regenerate your secret:

1. Go to **KB settings > Widget**.
2. In the **Admin Settings** section, under **JWT Authentication**, select **Regenerate JWT Secret**. The **Regenerate JWT Secret** modal opens.
3. As the message states, regenerating the secret will invalidate the old secret (which breaks your existing widget authentication until you update it to use the new secret). To proceed with regenerating the secret, select **Regenerate**.
4. The modal disappears and your new JWT secret will be displayed.

Update your authentication scripts/processes to use the new secret.