OAuth

OAauth authentication must be used by all user interfaces to authenticate a user. It can also be used to connect third party services like Amazon Alexa. Authentication is handled by C1 Auth There is no authentication information passed when the module is opened. Instead the module must redirect the user to the C1 Auth login page. If the user is logged in already, the login page immediately redirects back to the module page and the user is logged in without seeing a login form.

OAuth flow

C1 Auth requires a standard Authorization Code Flow. See https://datatracker.ietf.org/doc/html/rfc6749 for details. The chapter Endpoint usage applies, so please read it before continuing reading here.

URLs

The C1 Auth seed servers are as follows:

Branch	URLs	Port
Production	https://n0r0c0.auth.sensaru.net, https://n1r0c0.auth.sensaru.net, https://n2r0c0.auth.sensaru.net	4002
Staging	https://n0r0c0.auth-staging.sensaru.net, https://n1r0c0.auth-staging.sensaru.net, https://n2r0c0.auth-staging.sensaru.net	3002
Development	https://n0r0c0.auth-dev.sensaru.net, https://n1r0c0.auth-dev.sensaru.net, https://n2r0c0.auth-dev.sensaru.net	2002

The login URLs are:

Branch	URLs
Production	https://login.sensaru.net
Staging	https://login-staging.sensaru.net

Branch	URLs
Development	https://login-dev.sensaru.net

The C1 Core seed servers are as follows:

Branch	URLs	REST and JSON-RPC port	Binary RPC port
Production	https://n0r0c0.core.sensaru.net, https://n1r0c0.core.sensaru.net, https://n2r0c0.core.sensaru.net	4001	4000
Staging	https://n0r0c0.core-dev.sensaru.net, https://n1r0c0.core-dev.sensaru.net, https://n2r0c0.core-dev.sensaru.net	3001	3000
Development	https://n0r0c0.core-dev.sensaru.net, https://n1r0c0.core-dev.sensaru.net, https://n2r0c0.core-dev.sensaru.net	2001	2000

Redirect to login page

When the user is not logged in, he needs to be redirected to the login page.

The following `GET` parameters need to be passed to the login page:

Parameter	Description
`client_id`	The client ID assigned to the module's UI. The client ID and client secret are randomly generated and must be registered to C1 Auth in order to work.
`response_type`	Only `code` is supported.
`state`	See description of `state` below.
`code_challenge`	See description of `code_challenge` below.

Parameter	Description
`code_challenge_method`	Only `S256` is supported.
`sp`	The ID of the system provider.
`sd`	The ID of the system distributor.
`bp`	The ID of the business partner.

`state`

The state variable enforces, that a login really is initiated by the module's UI. It prevents cross login requests. Either a random string can be used as state variable. This string needs to be stored and verified when the user is redirected back from the login page. Alternatively we can use a signed random value. This enables us to just check the signature when the user is redirected back and we don't have to store the state anywhere.

We can also sign additional metadata in the state string. The recommendation is:

Metadata	Description
Time	A time stamp to know when the state was generated. The state should only be valid for a short amount of time (e. g. 10 minutes)
Random data	Random data to make hacking more difficult
Principal	The principal ID to prevent logging in as a different principal than requested. The principal IDs must be passed to the module's UI as `GET` query parameters.

An example implementation in PHP might look like this:

```
function getState() : string
{
    global $systemProvider;
    global $systemDistributor;
    global $businessPartner;
    $keyspace = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';

    $pieces = [];
    $max = mb_strlen($keyspace, '8bit') - 1;
    for ($i = 0; $i < 64; ++$i)
    {
```

```
        $pieces[] = $keyspace[random_int(0, $max)];
    }
    $randomString = implode('', $pieces);
    $data = time().','.$randomString.','.$systemProvider.','.$systemDistributor.','.$businessPartner;

    $keyId = openssl_pkey_get_private('file://'.__DIR__.'/../stateCerts/key.pem');
    if($keyId === false) die('Could not read private key.');
    $signature = '';
    if(openssl_sign($data, $signature, $keyId, OPENSSL_ALGO_SHA512) === false)
    {
        openssl_pkey_free($keyId);
        die('Could not generate state.');
    }
    openssl_pkey_free($keyId);
    return base64_encode($data.';'.base64_encode($signature));
}
```

code_challenge

Sensaru Cloud uses "Proof Key for Code Exchange" (PKCE) to mitigate the risk of interception attacks. Read https://datatracker.ietf.org/doc/html/rfc7636 for details. The code verifier is just a random string. Here's an example in PHP on how to generate a code verifier:

```
function getCodeVerifier()
{
    $keyspace = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-._~';

    $pieces = [];
    $max = mb_strlen($keyspace, '8bit') - 1;
    for ($i = 0; $i < 128; ++$i)
    {
        $pieces[] = $keyspace[random_int(0, $max)];
    }
    return implode('', $pieces);
}
```

The generated code verifier is hashed and base64-encoded to generate `code_challenge`:

```
$codeChallenge = base64_encode(hash('sha256', $codeVerifier, true));
```

This hashed code challenge is sent to the login page. The code verifier needs to stored (e. g. in the session). When the user is redirected back and the authorization code is exchanged for the access token, the (unhashed) code verifier must be sent to the authorization server.

Process redirect from login page

When the user is redirected back to the module page, two `GET` parameters are passed: `code` and `state`. `state` matches the one you passed to the login page. When the data is signed, the signature needs to be verified. Otherwise it is required to check it `state` matches the stored one. It is also

required to check if `state` has expired. Here's an example on how to check the signature and for expiration in PHP:

```
list($data, $signature) = explode(';', base64_decode(urldecode($_GET['state'])));
   if(strlen($data) > 0 && strlen($signature) > 0)
   {
      list($time, $randomString, $stateSystemProvider, $stateSystemDistributor, $stateBusinessPartner) =
explode(',', $data);
      if($stateSystemProvider === '') $stateSystemProvider = '0';
      if($stateSystemDistributor === '') $stateSystemDistributor = '0';
      if($stateBusinessPartner === '') $stateBusinessPartner = '0';
      $timeDifference = time() - $time;
      if($timeDifference >= 0 && $timeDifference < 600)
      {
         $keyId = openssl_pkey_get_public('file://'.__DIR__.'/../stateCerts/cert.pem');
         $result = openssl_verify($data, base64_decode($signature), $keyId, OPENSSL_ALGO_SHA512);
         openssl_pkey_free($keyId);

         if($result === 1)
         {
            .
            .
            .
         }
      }
   }
}
```

When the signature is valid, the access token can be requested. For this a HTTP `POST` needs to be sent to C1 Auth to path `/api/v1/oauth/token` with the following data using `application/x-www-form-urlencoded` as content type:

Parameter	Description
`grant_type`	`authorization_code`
`code`	The authorization code just received as `GET` parameter `code`.
`client_id`	The client ID to identify the module.
`client_secret`	The client secret to authenticate the module.
`code_verifier`	The previously generated code verifier which was sent as a hashed value to the login page.

On success the access and refresh tokens are returned as a JSON. Both need to be stored in the user's session. The validity typically is 1 hour. Before this, the access token should be refreshed using the refresh token.

The access and refresh token contain information that must be checked. With PHP this looks like this:

```
$keyParts = explode(',', base64_decode($accessToken));
$keyParts2 = explode(',', base64_decode($refreshToken));
if(count($keyParts) >= 8 &&
   count($keyParts2) >= 8 &&
   $keyParts[7] === $clientId &&
   $keyParts[2] === $stateSystemProvider &&
   $keyParts[3] === $stateSystemDistributor &&
   $keyParts[4] === $stateBusinessPartner &&
   $keyParts2[7] === $clientId &&
   $keyParts2[2] === $stateSystemProvider &&
   $keyParts2[3] === $stateSystemDistributor &&
   $keyParts2[4] === $stateBusinessPartner)
{
    //Save REST object state to session.
    $rest->sessionSave();
    header('Location: '. $_SERVER['PHP_SELF']);
    die();
}
else
{
    $rest->logout();
    die('Unauthorized');
}
```

In particular it is mandatory to verify that the following access token and refresh token fields match the information within the module:

Field name	Field index	Must match
Client ID	7	The client ID of the module
System provider	2	The system provider in `state`
System distributor	3	The system distributor in `state`
Business partner	4	The business partner in `state`

Warning

These checks are mandatory to have a secure system! Otherwise the system might be hacked.

Verify with C1 Core that user is authenticated and access token belongs to your module

At last the access token must be used to query the user from C1 Core. The response must be checked for validity.

To query the user information, send a `GET` request to `/user` on one of the C1 Core endpoints.

An example check in PHP looks like this:

```
isset($result['success']) && $result['success'] === true &&
isset($result['result']['authenticated']) && $result['result']['authenticated'] === true && //Check if the access token is valid
isset($result['result']['clientId']) && $result['result']['clientId'] === $clientId //Check if the access token belongs to our service
```

When the response is valid, the user is successfully logged in.