Integrating VPP Patient Portal with Health Gateway

Step-by-step approach to integrating the VPP Patient Portal with Health Gateway. The integration ensures a seamless user experience while securely handling user information and tokens, leveraging BC Services Card authentication.

**Overview**

* **Objective**: Enable users to access the VPP Patient Portal from Health Gateway, securely passing user information (e.g., PHN, birthdate) and tokens.
* **Workflow**:
  1. User logs into Health Gateway using BC Services Card authentication.
  2. User clicks the "VPP Patient Portal" button.
  3. Health Gateway submits a POST request to the VPP Patient Portal with tokens and user data.
  4. VPP Patient Portal verifies tokens, creates a session, and provides access to health records.
  5. Portal handles token refreshing internally when the access token expires.

**Implementation Steps**

**Step 1: Health Gateway Configuration**

**1.1. Create a Form to Submit Data via POST**

* **Purpose**: Use a form to securely send user information and tokens to the VPP Patient Portal via a POST request.
* **Action URL**: https://yourvppportal.com/hgw\_launch (Update to your actual portal URL)
* **Method**: POST
* **Hidden Input Fields**:
  + access\_token
  + refresh\_token
  + phn
  + birthdate

**Code Example (HTML):**

<!-- Health Gateway Side -->

<form id="portalForm" action="https://yourvppportal.com/hgw\_launch" method="POST">

<input type="hidden" name="access\_token" value="ACCESS\_TOKEN">

<input type="hidden" name="refresh\_token" value="REFRESH\_TOKEN">

<input type="hidden" name="phn" value="USER\_PHN">

<input type="hidden" name="birthdate" value="USER\_BIRTHDATE">

</form>

<button type="button" onclick="document.getElementById('portalForm').submit();">

VPP Patient Portal

</button>

* **Notes**:
  + Replace placeholders (ACCESS\_TOKEN, REFRESH\_TOKEN, USER\_PHN, USER\_BIRTHDATE) with actual values obtained after user authentication.
  + Ensure that the form is served over HTTPS to protect data in transit.
  + Optionally, encrypt the data before including it in the form for added security.

**Step 2: VPP Patient Portal Setup**

**2.1. Set Up the /hgw\_launch Endpoint**

* **Purpose**: Receive the POST request from Health Gateway, extract the user information and tokens, verify the access token, and create a user session.
* **Route**: /hgw\_launch
* **Methods**: POST (Reject GET requests to enhance security)
* **Function Name**: hgw\_launch

**Code Example (Python Flask):**

# VPP Patient Portal Side

from flask import Flask, request, redirect, url\_for, session

app = Flask(\_\_name\_\_)

app.secret\_key = 'your\_secret\_key' # Replace with a secure, random key in production

@app.route('/hgw\_launch', methods=['POST'])

def hgw\_launch():

access\_token = request.form.get('access\_token')

refresh\_token = request.form.get('refresh\_token')

phn = request.form.get('phn')

birthdate = request.form.get('birthdate')

# Verify the access token

if verify\_access\_token(access\_token):

# Store tokens and user info securely in the session

session['access\_token'] = access\_token

session['refresh\_token'] = refresh\_token

session['phn'] = phn

session['birthdate'] = birthdate

# Redirect to the dashboard or main page

return redirect(url\_for('dashboard'))

else:

return "Invalid access token", 401

* **Notes**:
  + Ensure that only POST requests are allowed to this endpoint.
  + Validate all incoming data to prevent injection attacks.
  + Store sensitive information securely in the session.

**2.2. Implement the verify\_access\_token Function**

* **Purpose**: Validate the access token using BC Services Card's public key to ensure it's authentic and unaltered.

**Code Example (Python):**

import jwt # PyJWT library for decoding JWT tokens

def verify\_access\_token(token):

BC\_SERVICES\_PUBLIC\_KEY = "-----BEGIN PUBLIC KEY-----\n...\n-----END PUBLIC KEY-----"

try:

decoded\_token = jwt.decode(

token,

BC\_SERVICES\_PUBLIC\_KEY,

algorithms=['RS256'],

audience="yourvppportal.com"

)

# Token is valid

return True

except jwt.ExpiredSignatureError:

# Token has expired

return False

except jwt.InvalidTokenError:

# Token is invalid

return False

* **Notes**:
  + Replace "-----BEGIN PUBLIC KEY-----\n...\n-----END PUBLIC KEY-----" with the actual public key provided by BC Services Card.
  + Ensure the audience parameter matches your Portal's domain.

**2.3. Secure Session Management**

* **Store Tokens and User Information Securely**:
  + Use server-side sessions (session in Flask) to store sensitive data.
  + Set appropriate session expiration times based on token validity.
  + Configure sessions to use secure cookies (SESSION\_COOKIE\_SECURE = True, SESSION\_COOKIE\_HTTPONLY = True).

**Flask Configuration Example:**

app.config.update(

SESSION\_COOKIE\_SECURE=True,

SESSION\_COOKIE\_HTTPONLY=True,

SESSION\_COOKIE\_SAMESITE='Lax',

PERMANENT\_SESSION\_LIFETIME=timedelta(hours=1)

)

**Step 3: Create the Dashboard Endpoint**

* **Purpose**: Display the user's health records after successful authentication.

**Code Example (Python Flask):**

from flask import render\_template

@app.route('/dashboard')

def dashboard():

if 'access\_token' in session:

phn = session.get('phn')

birthdate = session.get('birthdate')

# Render dashboard template with user data

return render\_template('dashboard.html', phn=phn, birthdate=birthdate)

else:

return redirect(url\_for('hgw\_launch')) # It should be either Login with BC Services Card

* **Notes**:
  + Ensure that only authenticated users can access the dashboard.
  + Use templates to render the user interface securely.

**Step 4: Implement Token Refresh Mechanism**

**4.1. Store Refresh Token Securely**

* **Purpose**: Keep the refresh token secure to allow obtaining new access tokens without re-authentication.
* **Storage**: Store in server-side session or secure, HTTP-only cookies.

**4.2. Create Token Refresh Endpoint**

* **Route**: /api/refresh-access-token
* **Methods**: POST
* **Purpose**: Use the refresh token to obtain a new access token when the current one expires.

**Code Example (Python Flask):**

import requests

from flask import jsonify

@app.route('/api/refresh-access-token', methods=['POST'])

def refresh\_access\_token():

refresh\_token = session.get('refresh\_token')

if not refresh\_token:

return jsonify({"status": "error", "message": "No refresh token available"}), 401

response = requests.post(

'https://auth.bcservicescard.gov/token',

data={

'grant\_type': 'refresh\_token',

'refresh\_token': refresh\_token,

'client\_id': 'your\_client\_id', # Replace with your client ID

'client\_secret': 'your\_client\_secret' # Replace with your client secret

}

)

if response.status\_code == 200:

new\_tokens = response.json()

session['access\_token'] = new\_tokens['access\_token']

# Update refresh token if provided

session['refresh\_token'] = new\_tokens.get('refresh\_token', refresh\_token)

return jsonify({"status": "success"}), 200

else:

return jsonify({"status": "error", "message": "Failed to refresh token"}), 401

* **Notes**:
  + Replace 'your\_client\_id' and 'your\_client\_secret' with actual credentials provided by BC Services Card.
  + Handle errors and exceptions appropriately.

**Step 5: Client-Side Handling**

**5.1. Check and Refresh Token Before API Calls**

* **Purpose**: Ensure that the access token is valid before making any API requests that require authentication.

**JavaScript Code Example:**

async function checkAndRefreshToken() {

try {

const response = await fetch('/api/refresh-access-token', {

method: 'POST',

credentials: 'include'

});

const result = await response.json();

if (result.status !== 'success') {

// Tokens are invalid; prompt re-authentication

alert('Session expired. Please log in again through Health Gateway.');

window.location.href = 'https://www.healthgateway.gov.bc.ca/';

}

} catch (error) {

console.error('Error checking token:', error);

alert('An error occurred. Please try again later.');

}

}

* **Usage**: Call checkAndRefreshToken() before making any authenticated API requests.

**5.2. Error Handling**

* **Display User-Friendly Messages**:
  + Inform users when their session has expired.
  + Provide clear instructions for re-authentication.
* **Example**:

function handleError(error) {

if (error.response && error.response.status === 401) {

alert('Your session has expired. Please log in again through Health Gateway.');

window.location.href = 'https://www.healthgateway.gov.bc.ca/';

} else {

alert('An unexpected error occurred. Please try again later.');

}

}

**Step 6: Security Considerations**

**6.1. Use HTTPS Everywhere**

* **Purpose**: Protect data in transit between the client and server.
* **Action**:
  + Obtain and install SSL/TLS certificates on your servers.
  + Redirect all HTTP traffic to HTTPS.

**6.2. Secure Cookies and Sessions**

* **Configure Cookies**:
  + Set Secure, HTTPOnly, and SameSite attributes.
* **Example (Flask Configuration):**

app.config.update(

SESSION\_COOKIE\_SECURE=True,

SESSION\_COOKIE\_HTTPONLY=True,

SESSION\_COOKIE\_SAMESITE='Lax'

)

**6.3. Implement CSRF Protection**

* **Purpose**: Prevent Cross-Site Request Forgery attacks.
* **Action**:
  + Use CSRF tokens in forms and verify them on the server.
  + In Flask, you can use the Flask-WTF extension for CSRF protection.

**6.4. Input Validation and Sanitization**

* **Purpose**: Prevent injection attacks and ensure data integrity.
* **Action**:
  + Validate all incoming data on the server side.
  + Use data validation libraries or write custom validation functions.

**6.5. Logging and Monitoring**

* **Purpose**: Detect and respond to security incidents.
* **Action**:
  + Log authentication attempts and errors.
  + Ensure logs do not contain sensitive information like tokens or personal data.
  + Monitor logs regularly for suspicious activities.

**Step 7: User Experience Enhancements**

**7.1. Navigation Back to Health Gateway**

* **Provide a Link**:
  + Include a button or link in the Portal to allow users to return to Health Gateway.
* **Example (HTML):**

<a href="https://www.healthgateway.gov.bc.ca/">Back to Health Gateway</a>

**7.2. Inform Users About Session Timeouts**

* **Display Notifications**:
  + Let users know when their session is about to expire.
  + Offer an option to extend the session if appropriate.
* **Example (JavaScript):**

function startSessionTimer() {

// Set a timeout for session expiration warning (e.g., 5 minutes before expiry)

setTimeout(() => {

alert('Your session will expire soon. Please save your work.');

}, SESSION\_DURATION - WARNING\_TIME);

}

**Step 8: Testing and Deployment**

**8.1. Thorough Testing**

* **Test Each Component**:
  + Verify that tokens are correctly passed and validated.
  + Ensure that the session management works as expected.
  + Test token refresh functionality.
* **Test Error Handling**:
  + Simulate expired or invalid tokens.
  + Ensure appropriate error messages are displayed.

**8.2. Security Testing**

* **Perform Security Audits**:
  + Conduct penetration testing to identify vulnerabilities.
  + Use automated tools to scan for common security issues.
* **Review Code**:
  + Conduct code reviews focusing on security aspects.

**8.3. Compliance Checks**

* **Ensure Compliance**:
  + Verify that the handling of personal health information complies with relevant regulations (e.g., PIPEDA in Canada).
  + Implement necessary privacy policies and user consent mechanisms.

**Step 9: Deployment Considerations**

**9.1. Use Production-Ready Servers**

* **Configure Servers Securely**:
  + Disable unnecessary services.
  + Keep software up to date with security patches.

**9.2. SSL/TLS Certificates**

* **Obtain Certificates**:
  + Use certificates from a trusted Certificate Authority (e.g., Let's Encrypt).

**9.3. Backup and Recovery**

* **Implement Backup Strategies**:
  + Regularly back up databases and important data.
  + Test recovery procedures.

**Summary Workflow**

1. **User logs into Health Gateway** using BC Services Card authentication.
2. **User clicks the "VPP Patient Portal" button**, which submits a POST request to https://yourvppportal.com/hgw\_launch with tokens and user data.
3. **VPP Patient Portal receives the POST data**, verifies the access token using the BC Services Card's public key, and securely stores tokens and user information in the session.
4. **User is redirected to the VPP Portal dashboard**, where they can access their health records.
5. **Portal implements token refresh mechanism**, using the refresh token to obtain new access tokens when needed.
6. **Client-side code checks token validity** before making API calls, refreshing tokens as necessary.
7. **Error handling and user notifications** ensure users are informed of session expirations and can re-authenticate if needed.
8. **Security measures** are implemented throughout to protect user data and comply with privacy regulations.

**Additional Notes**

* **Consistency**: Ensure that the route names, function names, and variable names are consistent across the application for clarity.
* **Documentation**: Document your code and processes for future maintenance and auditing.
* **Collaboration with Health Gateway Team**:
  + Coordinate with the Health Gateway development team to ensure compatibility and address any integration challenges.