CS 6500 Network Security

Assignment 4 – 25th April 2025 (discussed in class)

The Goal of this assignment are as follows:

a) Gain familiarity and experience with OAuth2.0, Flows
b) Use Cisco Packet Tracer tool to establish IP Sec Tunnel and verify communication across two enterprises over an ip sec tunnel.
c) In Cisco Packet Tracer, we cannot use Wireshark, i.e. we cannot see the contents of the packets. By using GNS3 simulator, can possibly recreate the same situation and see the contents. However getting the Cisco IOS for the GNS3 is a little step to be performed. (you may explore it later at your free time).

Assignment Submission date: 2nd May 12.00 Noon IST.

Please submit the assignment by email to me, copy to TAs. The submission is individual. You may work together discuss, however each one makes separate submission.

The following Resources will be of help to you, and kindly use them:

[1] Cisco Networking Academy: Learn Cybersecurity, Python & More
[2] Creating an IPSec Tunnel using CISCO Packet tracer is detailed in Setting up an IPSec VPN using Cisco Packet Tracer – CyberSecFaith

Part 1: Understanding OAuth 2.0 "Authorization code flow "in a step-by-step manner using a simple project.

We'll create a basic web application that uses OAuth 2.0 to authenticate users via a third-party provider like Google.

Project Overview

We'll build a web application that allows users to log in using their Google account. This project will help you understand the key components of OAuth 2.0, including authorization grants, access tokens, and scopes.

Prerequisites

- Basic knowledge of web development (HTML, CSS, JavaScript)
- Node.js and npm installed on your machine
- A Google account

Step 1: Set Up a Google OAuth 2.0 Client

1. Go to the Google Cloud Console.
2. Create a new project.
3. Navigate to **APIs & Services > Credentials**.
4. Click **Create Credentials** and select **OAuth 2.0 Client IDs**.
5. Configure the OAuth consent screen with the necessary details.
6. Set the application type to **Web application** and add the authorized redirect URI (e.g., http://localhost:3000/auth/google/callback).
7. Save the client ID and client secret.

Step 2: Set Up the Project

1. Create a new directory for your project and navigate into it:

mkdir oauth2-demo

cd oauth2-demo

2. Initialize a new Node.js project:

npm init -y

3. Install the necessary dependencies:

npm install express passport passport-google-oauth20

Step 3: Create the Server

1. Create a file named server.js and add the following code:

```
const express = require('express');
const passport = require('passport');
const GoogleStrategy = require('passport-google-oauth20').Strategy;

const app = express();

passport.use(new GoogleStrategy({
    clientID: 'YOUR_GOOGLE_CLIENT_ID',
    clientSecret: 'YOUR_GOOGLE_CLIENT_SECRET',
    callbackURL: 'http://localhost:3000/auth/google/callback'
  },
```

```
  function(accessToken, refreshToken, profile, done) {
    // Here you would save the user profile to your database
    return done(null, profile);
  }
));

passport.serializeUser((user, done) => {
  done(null, user);
});

passport.deserializeUser((user, done) => {
  done(null, user);
});

app.use(passport.initialize());
app.use(passport.session());

app.get('/auth/google',
  passport.authenticate('google', { scope: ['profile', 'email'] })
);

app.get('/auth/google/callback',
  { failureRedirect: '/' }),
  (req, res) => {
    res.redirect('/profile');
  }
);

app.get('/profile', (req, res) => {
  res.send(`Hello, ${req.user.displayName}`);
});

app.listen(3000, () => {
```

```
    console.log('Server is running on http://localhost:3000');
});
```

Step 4: Run the Server

1. Start the server:

node server.js

2. Open your browser and navigate to http://localhost:3000/auth/google.

Understanding OAuth 2.0 Components

-Authorization Grant**: The user grants permission to the application to access their Google profile.

- **Access Token**: The application receives an access token from Google, which it can use to access the user's profile information.
- **Scopes**: The scope parameter in the authentication request specifies the level of access requested (e.g., profile, email).

This project should give you a solid understanding of how to implement OAuth 2.0 authentication in a web application.

=== BACKGROUND MATERIAL ====

(To run the above necessary code, you'll need to ensure you have Node.js and npm installed on your Windows system. Here are some additional steps to help you get started on Windows:

Additional Steps for Windows

1. **Install Node.js and npm**:

- Download and install Node.js from the official website.

- Follow the installation instructions to install Node.js and npm.

2. **Set Up Your Project Directory**:

- Open Command Prompt or PowerShell.

- Navigate to the directory where you want to create your project:

```bash
```

cd path\to\your\project\directory

```

3. **Follow the Project Steps**:

- Follow the steps provided in the initial project setup, starting from creating the project directory and initializing the Node.js project.

If you encounter any issues or need further assistance with the setup on Windows, feel free to ask!

===

## Part 2: Understanding OAuth 2.0 additional flows using the project.

This part is to understand further the following flows.

1. **Implicit Flow**
2. **Client Credentials Flow**
3. **Resource Owner Password Credentials Flow**

### Enhanced Project Overview

We'll extend the existing web application to support multiple OAuth 2.0 flows, allowing you to understand different scenarios and use cases.

### Prerequisites

- Basic knowledge of web development (HTML, CSS, JavaScript)
- Node.js and npm installed on your machine
- A Google account (for Authorization Code and Implicit Flows)
- A mock OAuth 2.0 server (for Client Credentials and Resource Owner Password Credentials Flows)

### Step 1: Set Up a Mock OAuth 2.0 Server
```

For simplicity, we'll use a mock OAuth 2.0 server like Auth0 or OAuth2orize for testing purposes.

Step 2: Update Dependencies

Ensure you have the necessary dependencies installed:

```bash
npm install express passport passport-google-oauth20 passport-oauth2
```

Step 3: Update the Server Code

Modify `server.js` to include additional OAuth 2.0 flows.

Authorization Code Flow (Already Covered)

This flow is already implemented in the initial project.

Implicit Flow

Add a new route for the Implicit Flow:

```javascript
app.get('/auth/google/implicit', (req, res) => {
  const authUrl = `https://accounts.google.com/o/oauth2/v2/auth?response_type=token&client_id=YOUR_GOOGLE_CLIENT_ID&redirect_uri=http://localhost:3000/auth/google/implicit/callback&scope=profile email`;
  res.redirect(authUrl);
});

app.get('/auth/google/implicit/callback', (req, res) => {
  const accessToken = req.query.access_token;
  res.send(`Access Token: ${accessToken}`);
```

```
});
```

Client Credentials Flow

Add a new route for the Client Credentials Flow:

```javascript
const request = require('request');

app.get('/auth/client-credentials', (req, res) => {
  const options = {
    url: 'https://mock-oauth2-server/token',
    method: 'POST',
    auth: {
      user: 'YOUR_CLIENT_ID',
      pass: 'YOUR_CLIENT_SECRET'
    },
    form: {
      grant_type: 'client_credentials'
    }
  };

  request(options, (error, response, body) => {
    if (error) {
      return res.send(error);
    }
    res.send(`Access Token: ${JSON.parse(body).access_token}`);
  });
});
```

```

#### Resource Owner Password Credentials Flow

Add a new route for the Resource Owner Password Credentials Flow:

```javascript
app.get('/auth/password', (req, res) => {
  const options = {
    url: 'https://mock-oauth2-server/token',
    method: 'POST',
    auth: {
      user: 'YOUR_CLIENT_ID',
      pass: 'YOUR_CLIENT_SECRET'
    },
    form: {
      grant_type: 'password',
      username: 'user@example.com',
      password: 'user_password'
    }
  };

  request(options, (error, response, body) => {
    if (error) {
      return res.send(error);
    }
    res.send(`Access Token: ${JSON.parse(body).access_token}`);
  });
});
```
```

Step 4: Run the Enhanced Server

Start the server:

```bash
node server.js
```

Step 5: Test the OAuth 2.0 Flows

1. **Authorization Code Flow**: Navigate to `http://localhost:3000/auth/google`.
2. **Implicit Flow**: Navigate to `http://localhost:3000/auth/google/implicit`.
3. **Client Credentials Flow**: Navigate to `http://localhost:3000/auth/client-credentials`.
4. **Resource Owner Password Credentials Flow**: Navigate to `http://localhost:3000/auth/password`.

Understanding Additional OAuth 2.0 Flows

- **Implicit Flow**: Used for client-side applications where the access token is returned directly in the URL fragment.
- **Client Credentials Flow**: Used for server-to-server communication where the client application requests an access token using its own credentials.
- **Resource Owner Password Credentials Flow**: Used when the client application has the user's credentials and requests an access token directly.

This enhanced project should give you a comprehensive understanding of various OAuth 2.0 flows and their use cases. If you have any questions or need further assistance, feel free to ask!