INF-2310: Computer Security

Assignment 4: OpenID Authentication

Amund Harneshaug Strøm

UiT id: ast283@uit.no

April 29, 2024

1 Introduction

This report describes the design and implementation of an web application that uses the Microsoft EntraID OpenID service. The service is used to authenticate and authorize users which tries to enter the domain of the web application. The web application will also be deployed on the virtual machine in this course, which requires a certificate to enable HTTPS encryption

2 Background

This assignment covers various topics, and this section aims to provide descriptions of some of them.

- OAuth2 stands for Open Authorization 2.0 and is an open standard for secure access delegation, meaning a user can grant limited access to the resources of a third-party without sharing their passwords. The OAuth2 protocol was developed as a solution for allowing third-party services to access a user's resources without sharing their credentials [1].

  To get a good understanding of how OAuth2 works, it is important to understand the terminology involved in the protocol. The *Client* refers to the application obtaining access to the protected resources. The *resources* often refers to API-endpoints, and are protected by the OAuth2 protocol. The *authorization server* authenticates the user and asks them to grant or deny the client's request. If the user consents, the server generates an authorization code or access token [1].

  There are generally 4 steps involved in the protocol, and would look something like this:

  1. The application needs access to the API
  2. Authorization server asks end-user for permission
  3. Authorization server generates a token containing the permissions granted by the end-user
  4. Application uses this token to access the resources of the API

- A JSON Web Token (JWT) is a standardized format for securely exchanging information as JSON objects between parties. The format makes it compact and small, allowing for swift transmission via URL, POST parameter, or HTTP header. With all the necessary entity information included, JWTs eliminate the need for multiple database queries. Additionally, JWT recipients can validate tokens without requiring server calls[2].

- OpenID Connect is an authentication layer built on top of OAuth 2.0, and introduces new concepts such as identity and session management, making it more suitable for authentication purposes. It uses scopes during the authentication process of an application. Which determine what kind of access an application has to a user's information, such as their name, email or profile picture. Each scope provides a specific set of user attributes, known as claims. The scopes that an application requests depend on which user attributes the application needs[3].

3 Design and Implementation

To complete this assignment it involved 3 main steps; setup the Microsoft Azure Portal and EntraID and integrate our web application with EntraID, create the web application functionality and UX, and finally deploy the web application on the virtual machine given through this course.

To integrate the web application with EntraID you first have to register a new app within EntraID, this will allow our application to access the functionalities within EntraID. The new app is registered for web platforms and requires a redirect URI, which specifies where the authorization server should send the user's browser after the user completes the authentication or authorization process. This URI will be used later within the web application. Now that the app is registered it needs a client secret which is used for proving its identity when it requests for tokens[4]. The client secret is a string of symbols and will also be used within the web application. The last step is to add scopes to the client, allowing it to obtain certain attributes about the users. In this case the application needs 3 scopes; *User.Read* which allows the client to get basic information about the user, *User.ReadBasic.All* which allows the client to get basic information about all users, and *User.ReadWrite* which allows the user to edit its own information.

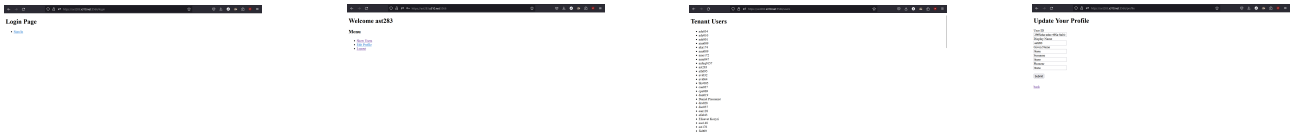
The web application was mostly complete but was missing the API. The API was built on top of the *flask* framework which provides tools, utilities, and abstractions to simplify the backend development. The web application supports 7 API calls which uses the *identity.web* package to interact with the Microsoft OpenID service to authenticate its users. Each call has its own purpose and works as follows:

- The '/login' route initiates the authentication process. It renders the login page and prepares the necessary parameters for authentication such as a list scopes the user should consent to, and a redirect URI that matches the URI in the app within the EntraID service. To complete the login the user is redirected to the given URI[5].
- The 'redirect_path' route completes the authentication process and is called by the login route. It handles the authentication response sent by Microsoft's authentication service. If the response was successful it redirects the user to the homepage together with their validated information[5].
- The '/logout' route signs the user out by invoking a method within the flask framework, an redirects the user to the login page[5].
- The '/' route is the root URL and acts as the homepage for the user. The homepage should however not be rendered if the user is not logged in, and the route should therefore verify whether the user is authenticated or not. If it is not authenticated it redirects the user to the '/login' route, otherwise it renders homepage and passes the user object[5].
- The '/profile' route with the GET method renders the profile page for the current user. However the page should not be rendered if the user is not logged in. The page checks if the user has any credentials, if not it will be redirected to login, otherwise the page uses the credentials to get information about the current user such as UserID and Display Name.
- The '/profile' route with the POST method updates the database with the changes the user did to its profile and renders the new changes onto the screen. However the user should not be able to do changes to its profile if it is not logged in. The page checks if the user has any credentials, if not it will be redirected to login, otherwise the page uses the credentials to allow the user to do changes it its profile and render it on the page.
- The '/user' route retrieves a list of users from Microsoft Graph API and renders them on screen. However the page should not be rendered if the user is not logged in. The page checks if the user has any credentials, if not it will be redirected to login, otherwise the page uses the credentials to get the information about the users registered at Microsoft Graph API.

To deploy the web application on the virtual machine and enable HTTPS encryption, you need a TLS certificate. These certificates are issued by a Certificate Authority (CA), there exists many different CA's this report however uses "Certbot". After obtaining the TLS certificate, import it into the web application. The certificate requires a higher privilege to open and you must therefore open it using the *sudo* command, this is possible using the *subprocess* library. The certificate requiring higher privileges caused some issues since the file path will be important later. A workaround would be to create a temporary file using the *tempfile* library, making it easily accessible. Now that all the preparations are done we can create an SSL context using the *OpenSSL* library, which allows the server to authenticate itself to clients during TLS connections. The SSL context requires the certificate, which we have saved temporarily, and we can then pass the SSL context to the app when it launches. The web application is then running on HTTPS encryption on `https://ast283.x310.net:5566`.

4 Evaluation

My web application is able to run on HTTPS encryption while deployed on the virtual machine. The web application is running on this URL `https://ast283.x310.net:5566`, whenever the app is running. The following images shows that the application running on HTTPS.



(a) Login page (b) Homepage (c) Users page (d) Profile page

Figure 1: Different pages in the web application running on HTTPS.

5 Conclusion

This report has given a high-level view of how to implement an web application that uses the Microsoft EntraID OpenID service. It has described how and why we register an app within EntraID, the implementation of different API calls, and how to deploy the web application with HTTPS encryption on the virtual machine given through this course. The report has proven that the web application works on the virtual machine by providing images showing the web application active in a browser.

References

[1] Albert Starreveld. Understanding OAuth2. Medium.com. Retrieved, April 24th, 2024, from https://medium.com/web-security/understanding-oauth2-a50f29f0fbf7

[2] Auth0 contributers. JSON Web Tokens. In Auth0.com. Retrieved, April 24th, 2024, from https://auth0.com/docs/secure/tokens/json-web-tokens

[3] Auth0 contributers. OpenID Connect Scopes. In Auth0.com. Retrieved, April 24th, 2024, from https://auth0.com/docs/get-started/apis/scopes/openid-connect-scopes

[4] Dickson-Mwendia. Quickstart: Sign in users and call Microsoft Graph from a Python Flask web app. In Microsoft.com. Retrieved, April 24th, 2024, from https://learn.microsoft.com/en-us/entra/identity-platform/quickstart-web-app-python-flask?tabs=windows

[5] Dickson-Mwendia. Tutorial: Add sign in to a Python Flask web app. In Microsoft.com. Retrieved, April 24th, 2024, from https://learn.microsoft.com/en-us/entra/identity-platform/tutorial-web-app-python-sign-in-users