"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER SYSTEM SECURITY

FINAL PROJECT

Proxy server with OpenID Connect authentication

Student:
Darko Filipovski, 161096

Supervisor:
Dr. Vesna Dimitrova

January 22, 2020

1 Introduction

This project is concerned with providing an implementation of an HTTP forward proxy server with OpenID Connect authentication. This means that a user can access the internet only if he is authenticated, otherwise he will be forwarded to an authentication page of the OpenID Connect provider.

When it comes to the proxy server implementation, besides forwarding the requests and responses, the server should also be modular and easily extensible. To achieve this the server features will be implemented in separate handlers through which the requests and/or responses will be passing. In other words, we will have handlers for decoding and encoding HTTP messages, providing authentication, file encoding, as well as providing the ability to easily add features such as logging or blacklisting.

The proxy authentication flow will be built from the ground up since the traditional HTTP Proxy-Authenticate header does not support OpenID Connect as an authentication type. As a result, a number of workarounds were needed, which are explained in the following sections. Finally, users' safety and security is important, so the best security practices were followed as recommended by the authentication provider.

2 Proxy Server

The proxy server was implemented using Netty. Netty is an efford to provide an asynchronous event-driven network application framework for the development of maintainable high-performance and high-scalability protocol servers and clients.

In Netty, a list of channel handlers in a channel pipeline handles or intercepts inbound events and outbound operations of a Channel. Each channel has its own pipeline which is created automatically when a new channel is created. This pipeline implements an advanced form of the Intercepting Filter pattern to give a user full control over how an event is handled and how the channel handlers in a pipeline interact with each other.

The handlers in a channel pipeline are visualized and described below.

HttpServerCodec

This handler handles both inbound and outbound messages. HttpServerCodec is responsible for decoding requests and encoding responses, thus enables easier server side HTTP implementation.

HttpObjectAggregator

A channel handler which aggregates an HTTP message and its content in a single FullHttpRequest object. It takes care of HTTP messages whose transfer encoding is chunked.

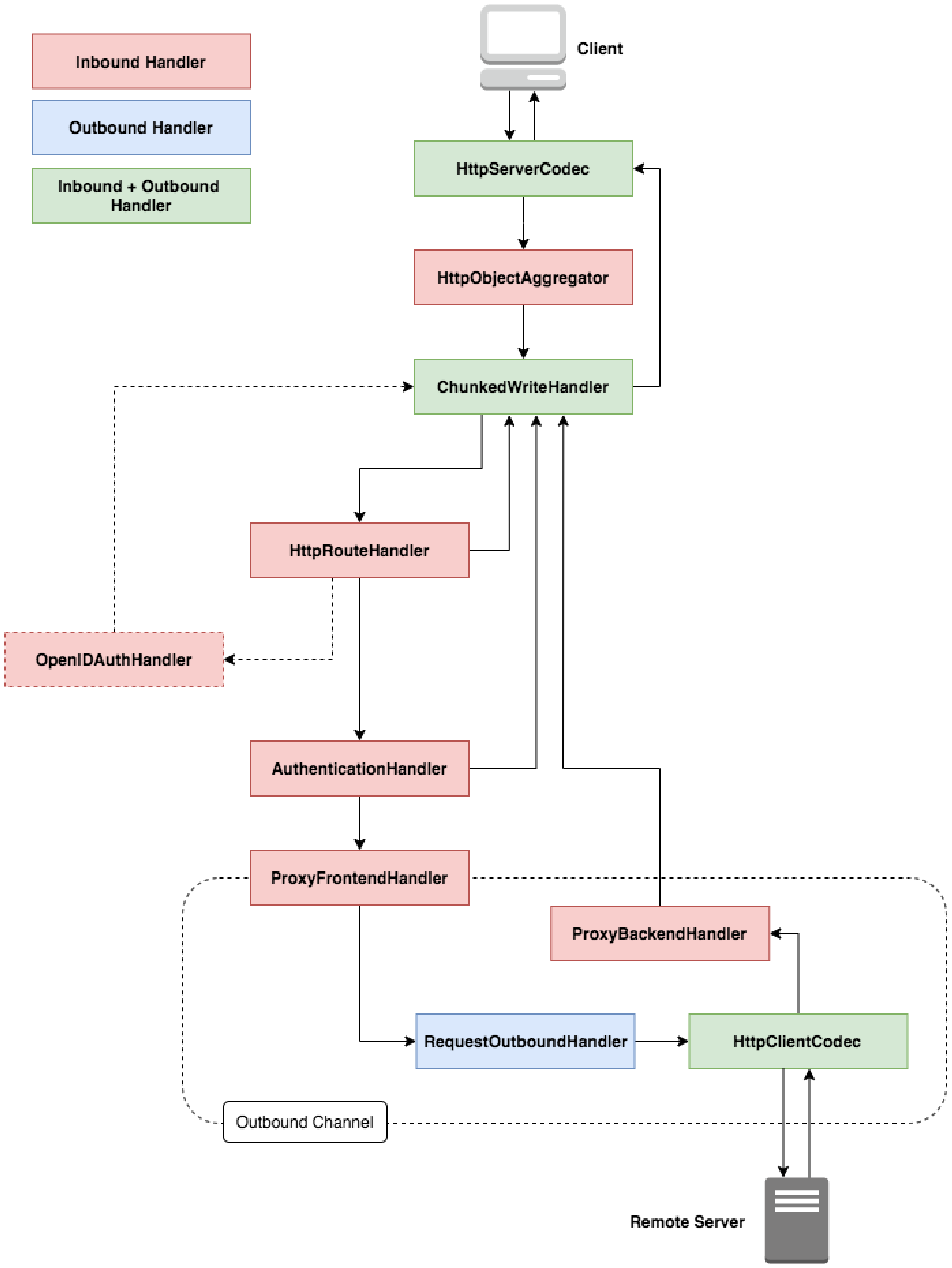


Figure 1: Proxy server's channel pipeline and its handlers

ChunkedWriteHandler

A channel handler that adds support for writing large data streams asynchronously. ChunkedWriteHandler manages the complicated states for sending large data streams, such as file transfers, without difficulty.

HttpRouteHandler

HttpRouteHandler handles the request routing. It is responsible for sending the static HTML files and making the necessary redirects for the proxy authentication flow. Additionally, this handler is responsible for creating a new or obtaining an already existing session. The sessions are managed by the means of HTTP cookies which are then attached to the responses. Another important thing is that this handler dynamically adds the OpenIDAuthHandler when necessary.

OpenIDAuthHandler

This channel handler is responsible for asynchronously exchanging the one-time authorization code, provided by the OpenID Connect provider, for an access token which is then used to obtain user information. After obtaining user information, the user is authenticated and redirected to the initially requested URL.

AuthenticationHandler

AuthenticationHandler checks whether the user is authenticated. If the user is authenticated the request is forwarded to ProxyFrontendHandler, otherwise it redirects the user to the appropriate endpoint for authentication.

ProxyFrontendHandler

ProxyFrontendHandler establishes a TCP connection to the remote server(connects a Channel to the remote peer), which is the destination of the initial user request.

RequestOutboundHandler

This channel handler queues and flushes requests meant for the remote server.

HttpClientCodec

HttpClientCodec is responsible for encoding requests and decoding responses, thus enables easier client side HTTP implementation.

ProxyBackendHandler

This handler receives the remote server response and flushes it through the channel, through the HTTP handlers responsible for wrapping the request, to the user.

3 OpenID Connect Authentication Flow

OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol. The difference between the two is that OAuth 2.0 is about resource access and sharing, while OIDC is all about user authentication. OIDC allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner.

OpenID Connect lets app and site developers authenticate users without taking on the responsibility of storing and managing passwords in the face of an Internet that is well-populated with people trying to compromise your users' accounts for their own gain. Additionally, for the app builder, it provides a secure verifiable answer to the question: "What is the identity of the person currently using the browser or native app that is connected to me?"

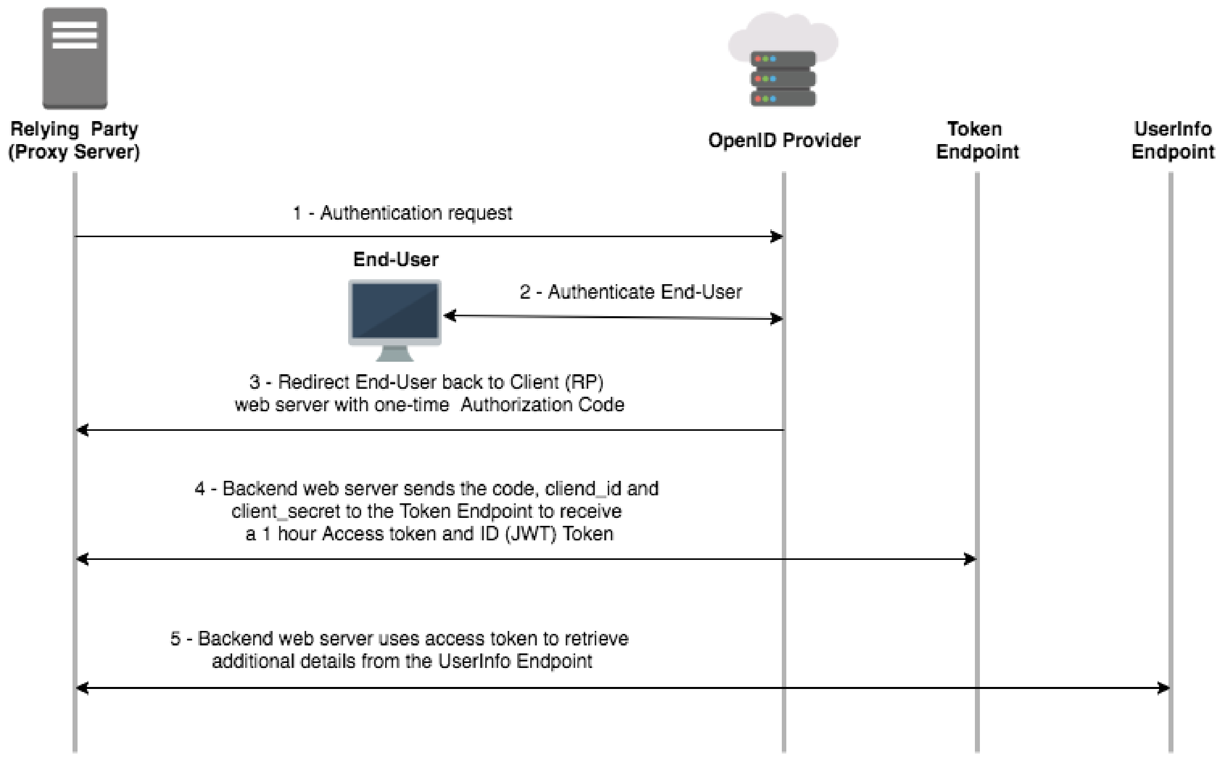

Relying Party (Proxy Server)
OpenID Provider
Token Endpoint
UserInfo Endpoint
1 - Authentication request
End-User
2 - Authenticate End-User
3 - Redirect End-User back to Client (RP) web server with one-time Authorization Code
4 - Backend web server sends the code, cliend_id and client_secret to the Token Endpoint to receive a 1 hour Access token and ID (JWT) Token
5 - Backend web server uses access token to retrieve additional details from the UserInfo Endpoint

Figure 2: OpenID Connect authentication flow

Figure 2 shows the steps taken to authenticate the end user. These steps are described below.

Google was used as the OpenID Connect provider.

1. The Relying Party sends a request to the OpenID Provider to authenticate the End-User. The request must include the Relying Party's identity and the openid scope, it may optionally include other scopes e.g. the email scope if the Relying Party wishes to obtain the user's email address. More specifically, in this project, the user is redirected to an endpoint provided by the OpenID Provider where the user is authenticated.

   Authentication endpoint

   ```
   GET https://accounts.google.com/o/oauth2/v2/auth
   ?client_id={CLIENT_ID}
   &response_type=code
   &scope=openid email profile
   &redirect_uri=http://localhost:6555/code
   &nonce=0394852-3190485-2490358
   &state=@{redirect-url}
   ```

2. The OpenID Provider authenticates the End-User using one of the methods available to it and obtains authorization from the End-user to provide the requested scopes to the identified Relying Party. For this purpose, Google provides a consent screen that describes the information that the user is releasing and the terms that apply.

3. Once the End-User has been authenticated and has authorized the request the OpenID Provider will return an authorization code to the Relying Party's server component. This code is passed to the server through a preconfigured redirect url on the proxy server.

   Endpoint to which the authorization code is sent

   ```
   GET http://localhost:6555/code
   ?state=url=http://www.apache.org/oid-proxy.oid/proxy?target_url=
   http://www.apache.org/
   &code=4wAH6MXADe4Jp_-ZQe1HUzG2ZQzu4ssEoIEYsH8FSE2julA-
   ed18TOrjh8tbLGg70d_KCJAzT2m5P4HCr7j-R4JE
   &scope=email+profile+https://www.googleapis.com/auth/userinfo.profile+
   https://www.googleapis.com/auth/userinfo.email+openid &authuser=0
   &prompt=none
   &session_state=1be462af5a4c310236706d6811a4cb2d3630ecce..39f3
   ```

4. The Relying Party's server component contacts the token endpoint and exchanges the authorization code for an access_token identifying the End-User and optionally access and refresh tokens granting access to the Userinfo endpoint.

Exchange code for access token and ID token

```
POST https://oauth2.googleapis.com/token
?code=4wAH6MXADe4Jp_-ZQe1HUzG2ZQzu4ssEoIEYsH8FSE2julA-
ed18TOrjh8tbLGg70d_KCJAzT2m5P4HCr7j-R4JE
&client_id={CLIENT_ID}
&client_secret={CLIENT_SECRET}
&redirect_uri=http://localhost:6555/code
&grant_type=authorization_code
```

5. Optionally the Relying Party may request the additional user information (e.g. email address) from the UserInfo endpoint by presenting the access token obtained in the previous step. In relation to obtaining user information, the proxy server obtains the user's name, email and profile picture.

Obtaining user profile information

```
GET https://openidconnect.googleapis.com/v1/userinfo
?scope=openid email profile

Authorization: {token_type} {access_token}
```

4 Proxy Server Authentication Flow

Proxy authentication is done through the HTTP Proxy-Authenticate response header, which defines the authentication method that should be used to gain access to a resource behind a proxy server. It authenticates the request to the proxy server, allowing it to transmit the request further. However, this header is limited to certain authentication types. This means that if a company has a single sign-on log-in system in place, that is different than the supported authentication types, then we would not be able to use the same system for proxy authentication. Furthermore, the most commonly used authentication types have security risks, so introducing a modern authentication system would prevent future security issues.

This project focuses on implementing a proxy server which requires an authentication using the OpenID Connect authentication protocol. This protocol is not supported by the traditional HTTP proxy authentication, so a new authentication flow was necessary for achieving the goal. This authentication flow uses cookie based authentication in combination with three way redirection.

The authentication flow is shown in Figure 3 and described in the steps below.

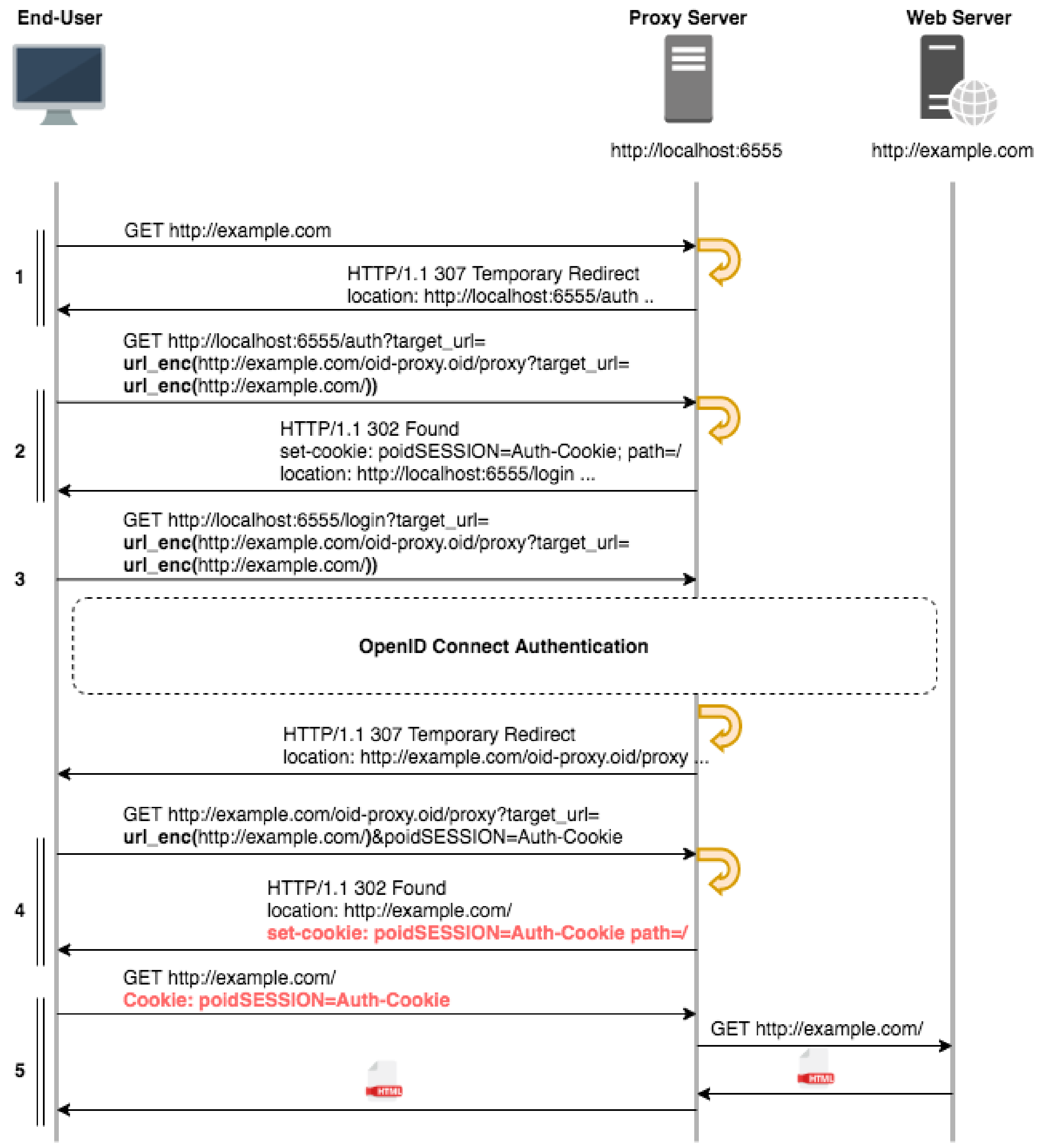

End-User
Proxy Server
Web Server
http://localhost:6555
http://example.com
1
GET http://example.com
HTTP/1.1 307 Temporary Redirect
location: http://localhost:6555/auth ..
GET http://localhost:6555/auth?target_url=
url_enc(http://example.com/oid-proxy.oid/proxy?target_url=
url_enc(http://example.com/))
2
HTTP/1.1 302 Found
set-cookie: poidSESSION=Auth-Cookie; path=/
location: http://localhost:6555/login ...
3
GET http://localhost:6555/login?target_url=
url_enc(http://example.com/oid-proxy.oid/proxy?target_url=
url_enc(http://example.com/))
OpenID Connect Authentication
HTTP/1.1 307 Temporary Redirect
location: http://example.com/oid-proxy.oid/proxy ...
4
GET http://example.com/oid-proxy.oid/proxy?target_url=
url_enc(http://example.com/)&poidSESSION=Auth-Cookie
HTTP/1.1 302 Found
location: http://example.com/
set-cookie: poidSESSION=Auth-Cookie path=/
5
GET http://example.com/
Cookie: poidSESSION=Auth-Cookie
GET http://example.com/
HTML
HTML

Figure 3: The proxy server authentication flow. The yellow arrows indicate that the next request will be a result of a redirection. i.e the response is a location to which the user will be redirected. url_enc(..) means that the enclosed content is url encoded. OpenID Connect Authentication block represents the OpenID authentication flow which finishes with a redirect to a specific URL.

Most of the steps in Figure 3 go in pairs of request and response and they are described below.

Step 1. When a user attempts to access some location on the web he is redirected to the /auth endpoint of the proxy server with a target_url query parameter.

Step 2. Once a request to the /auth endpoint is made, the appropriate endpoint handler checks whether the user is authenticated. If the user is not authenticated he is redirected to the /login endpoint with the target_url query parameter set to the same value. To be able to identify the user a session cookie is set for the proxy server host.

Step 3. The /login endpoint returns a response containing an HTML document. This document contains a URL to which the user will be redirected to authenticate himself to the OpenID Connect Provider. The URL is custom made so that the provider will know where to forward the user upon finishing the authentication.

Step 4. After the OpenID Connect authentication flow is completed the user is redirected to the initially requested host with an endpoint /oid-proxy.oid/proxy and the target_url and poidSESSION (with user's session cookie as the value) query parameters. This endpoint enables the proxy server to appropriately handle the request. When the proxy server sees the previously mentioned endpoint for a foreign host it redirects the user to the location specified in the target_url param and also sets the poidSESSION cookie using the poidSESSION parameter. This step is the most important one because the cookie allows the request to pass the AuthenticationHandler and retrieve the content. This is possible because the cookie identifies the session and enables the proxy server to check if the user is authenticated.

Step 5. The user is then redirected to the initially requested URL with an additional poidSESSION cookie. This cookie allows the proxy server to retrieve the user's session and check whether he is authenticated (this is checked by the AuthenticationHandler). If the user is authenticated, the AuthenticationHandler passes the request to the ProxyFrontendHandler which then retrieves the URL contents and passes it back to the user.

Future requests. Once the user is authenticated there are two cases to be covered. The first one is when the user has previously accessed the web server, thus future requests have a cookie, and when the host has not yet been accessed.

1. Requests sent to a previously accessed web server contain a poidSESSION cookie, so the AuthenticationHandler will just forward the request to the proxy handlers which will retrieve the content. No further redirects are needed.
2. In the case where the request is sent to a web server which has not been previously accessed, and does not have a poidSESSION cookie, the flow is different. First the AuthenticationHandler will realize that there is no session, so the user will be forwarded to the /auth endpoint with the appropriate parameters. The request to the /auth endpoint will contain a session cookie, so the endpoint handler will see that the user is authenticated and continue with the authentication flow from **Step 4**.

This sums up the proxy server authentication flow. The source code can be found on GitHub at https://github.com/filipovskid/ProxyServer-OpenID

Images of the authentication process

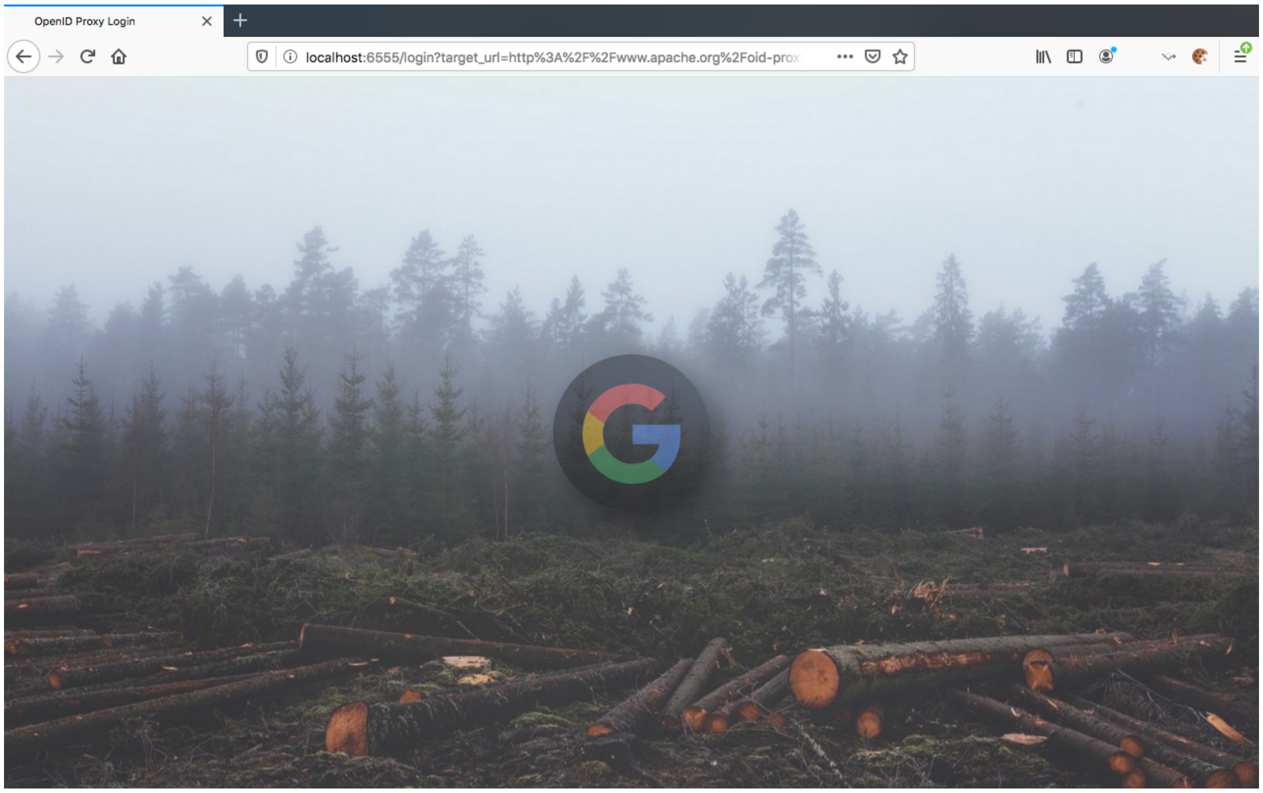
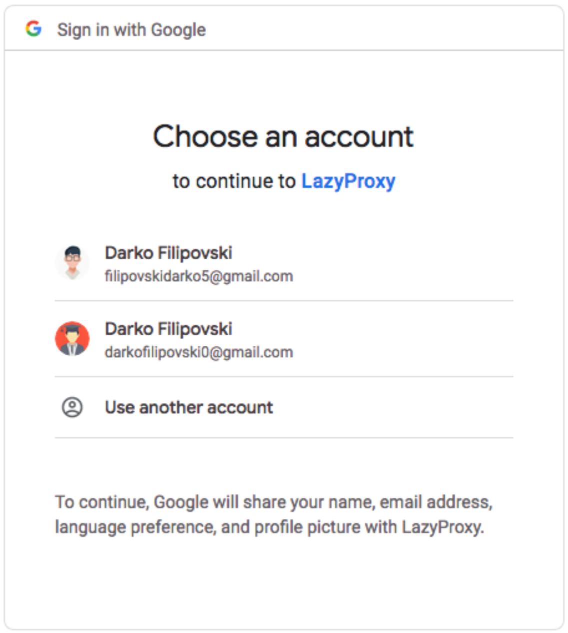

OpenID Proxy Login
localhost:6555/login?target_url=http%3A%2F%2Fwww.apache.org%2Foid-prox

Figure 4: Login endpoint with a large button which redirects to the Google consent screen


Sign in with Google

Choose an account

to continue to LazyProxy

Darko Filipovski
filipovskidarko5@gmail.com

Darko Filipovski
darkofilipovski0@gmail.com

Use another account

To continue, Google will share your name, email address, language preference, and profile picture with LazyProxy.

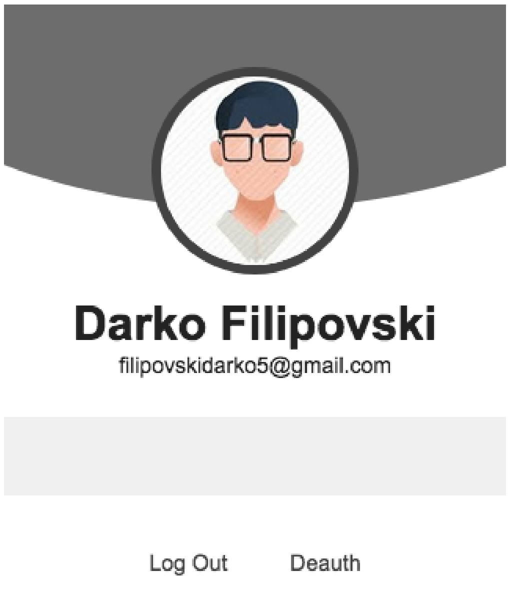

Darko Filipovski
filipovskidarko5@gmail.com

Figure 5: **Left:** Google login screen. **Right:** Profile on the proxy server after login.