Solid-cURL

Abstract
We implemented Solid-cURL, a tool that extends cURL be adding Solid-OIDC authentication. We described the different components of Solid-cURL and how they interact with each other.

Keywords
Solid, Tool

1. Introduction

In their daily work, Solid developers and researchers often have to manually send specific HTTP request to Solid servers. Doing this with a web browser is often not very practical as one first has to login with Solid-OIDC using some Solid app. The requests that can be send afterwards are only those that are allowed by the app.

Web developer and researchers in this situation used the widespread tool cURL. cURL is a command line program that allows to specify the parameters of an HTTP request (method, headers, etc.) in every detail. cURL executes this request and displays the result to the user.

As cURL allows to specify arbitrary headers for a request it could theoretically be used to send authenticated requests to Solid servers by copying the Solid-OIDC specific headers (access token and DPoP) to the cURL command line parameters. This approach, however is very impractical, as the DPoP header changes every time the request method or URI changes; the access token automatically expires after some time.

In this poster we present Solid-cURL. Solid-cURL is a program that tries to mimic the command line interface and behaviour of cURL. Additionally it allows users to log in via Solid-OIDC. The credentials can be stored and used for future HTTP request by specifying a command line parameter.

In the remainder of this paper, we will give an overview of the different components of Solid-cURL, introduce our implementation of Solid-cURL and give a short outlook.

2. Components

In Figure 1 a UML interaction diagram showing the principal components can be seen and their interaction.

The Command Line Parser is responsible for parsing and interpreting the command line input of the user and giving it to the Main component of Solid-cURL. Based on the parameters (more specifically, the Solid user specified as parameter) the Main component uses the Credential Storage component to retrieved previously saved credentials. The credentials are then given to the Authenticator component which carries out the login process (i.e. it retrieves and access

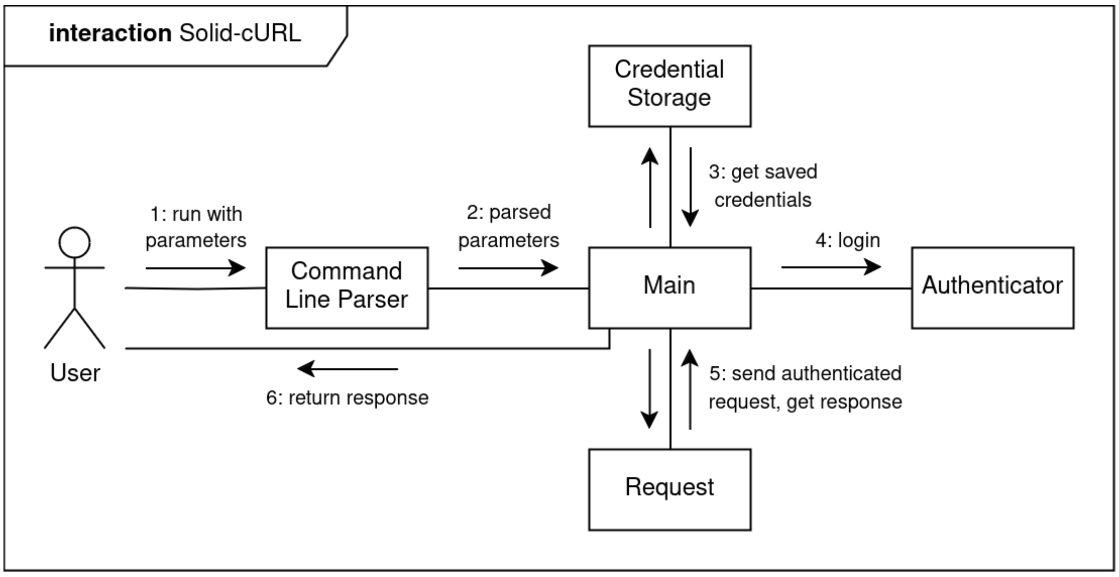

Figure 1: UML interaction diagram showing the different components of Solid-cURL and their interaction

token and creates a DPoP). Access token and DPoP are given to the Request component who carries out the actual request according to the command line parameters and with the correct Solid-OIDC headers set. The response is then printed to the user on the command line.

3. Implementation

We implemented Solid-cURL in Javascript. The source code is available via GitHub[1] and an executable script can be installed using the Node Package Manager[2].

For the Command Line Parser component we used the Commander.js library[3]. For the Credential Storage component we used the operating system's keychain accessed using the keytar library[4]. For the authenticator component we used the solid-client-authn-node library[5] from Inrupt. For the request component we also used the solid-client-authn-node library or the fetch provided by it, respectively.

4. Outlook

For the future we want to extend the capabilities (regarding the possible options and parameters) of Solid-cURL to match those of the real cURL. For this we plan to rewrite Solid-cURL in C++ to be able to use the libcurl library. cURL is also directly based on this library which makes it

[1]https://github.com/wintechis/solid-curl
[2]https://www.npmjs.com/package/solid-curl
[3]https://github.com/tj/commander.js?
[4]https://github.com/atom/node-keytar
[5]https://github.com/inrupt/solid-client-authn-js

easier to mimic cURL capabilities using this libraries. Implementing Solid-cURL in c++ could also positively impact its performance.

For C++, however, there exists to our knowledge no library for Solid-OIDC which means we will have to handle all the different tokens used in the process manually.