Technical Overview

This document gives a brief overview on the technologies used in this project.

Frameworks and Technologies

The user interface is built using React and TypeScript as a language. React has a clear and easy-to-debug structure and data flow, that scales for all future needs of the application. The built web page will be relatively simple so there is no need to learn and use more structured front-end technologies. React also supports compilation errors which will radically speed up the development. TypeScript is a trending, statically typed superset of JavaScript, and it was chosen for the benefits static typing brings to the development process, e.g. compile time errors and superior auto-completion possibilities compared to JavaScript.

Packages are handled with yarn and npm. Webpack is used with TypeScript compiler for building and bundling. For testing we use a combination of Karma, Mocha and Chai. Tests are run in a headless browser with PhantomJS.

Architecture

The prototype version of Cloud Native Security Solution is a client-side only web application. This architecture has the advantage that the state of the application exists in a single Javascript context, and there is no need to transfer commands or parts of the state back and forth between the user's browser and an application server. Development is also easier, as the app can be served as a single, static web page, instead of needing an application server. Main drawback of this architecture is that it can not support any scheduled or long-running tasks independent of user navigating away from the web application. User files to be analyzed must also be downloaded to user's browser to be processed, and the client must receive API keys to F-Secure and cloud storage providers to access them. For this first prototype version, we consider the ease and quickness of development to outweigh these drawbacks.

Data flows

The Cloud Native Security Solution architecture allows for supporting an array of different cloud storage providers. Because the application can not access any of them without the user's authorization, the first step is to prompt the user to select which provider they want to authorize the application to access. The application then redirects the browser to that provider's OAuth 2.0 endpoint, requesting an access token with the capabilities it needs. User completes the OAuth flow on the provider's web page, and the provider redirects the browser back to the application, with an access token included in the URL hash.

Upon being loaded again, the application detects and extracts the access token from the URL hash. With this token it can now access that particular cloud storage on the user's behalf, and proceeds to fetch a list of the files that exist in the storage. Then the application downloads the content of those files, calculates an SHA-1 hash in the browser, and queries the F-Secure Security Cloud for the reputation of that file. The application could also upload the file to F-Secure Security Cloud for further analysis. The response from F-Secure Security Cloud for each file is then displayed to the user in the application interface.

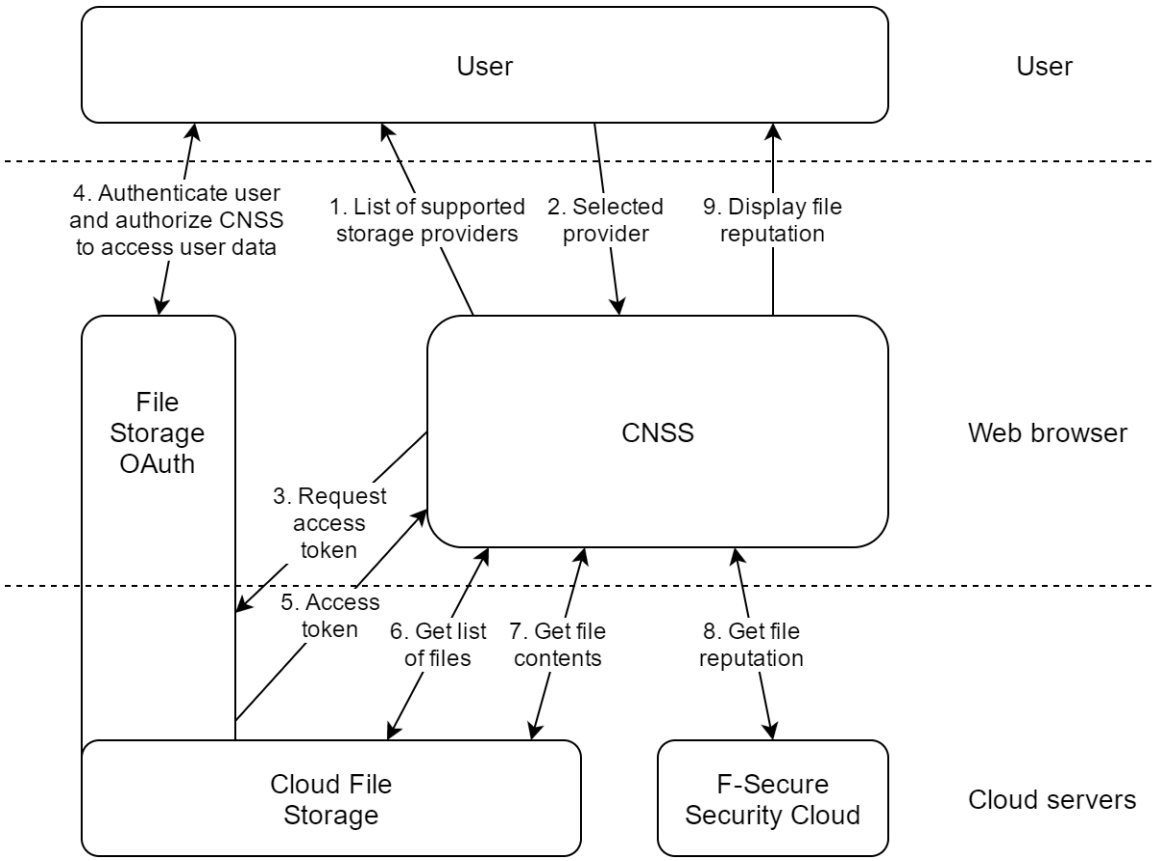

Figure 1. Architectural data

Design

Internally, the most complex part of the Cloud Native Security Solution is a user interface built around the React UI rendering library. To make accessing various remote servers easier, we have created various wrapper modules for them.

The Security Cloud module presents simple methods that take files or hashes in, and returns the analysis results. It handles internally things such as HTTP requests and Security Cloud Doorman authentication.

The cloud storage module consists of two interfaces. The initialization interface represents available storage providers, and it can initiate the OAuth flow for selected storage provider, extract the access token returned from successful OAuth flow, and construct an implementation of the access interface. The access interface, then, represents a storage provider user has already authenticated for. It provides simple methods that fetch a list of the user's files, or content of a specific file, from the cloud storage servers. The cloud storage interfaces are designed to be implementable for multiple different cloud storage providers.

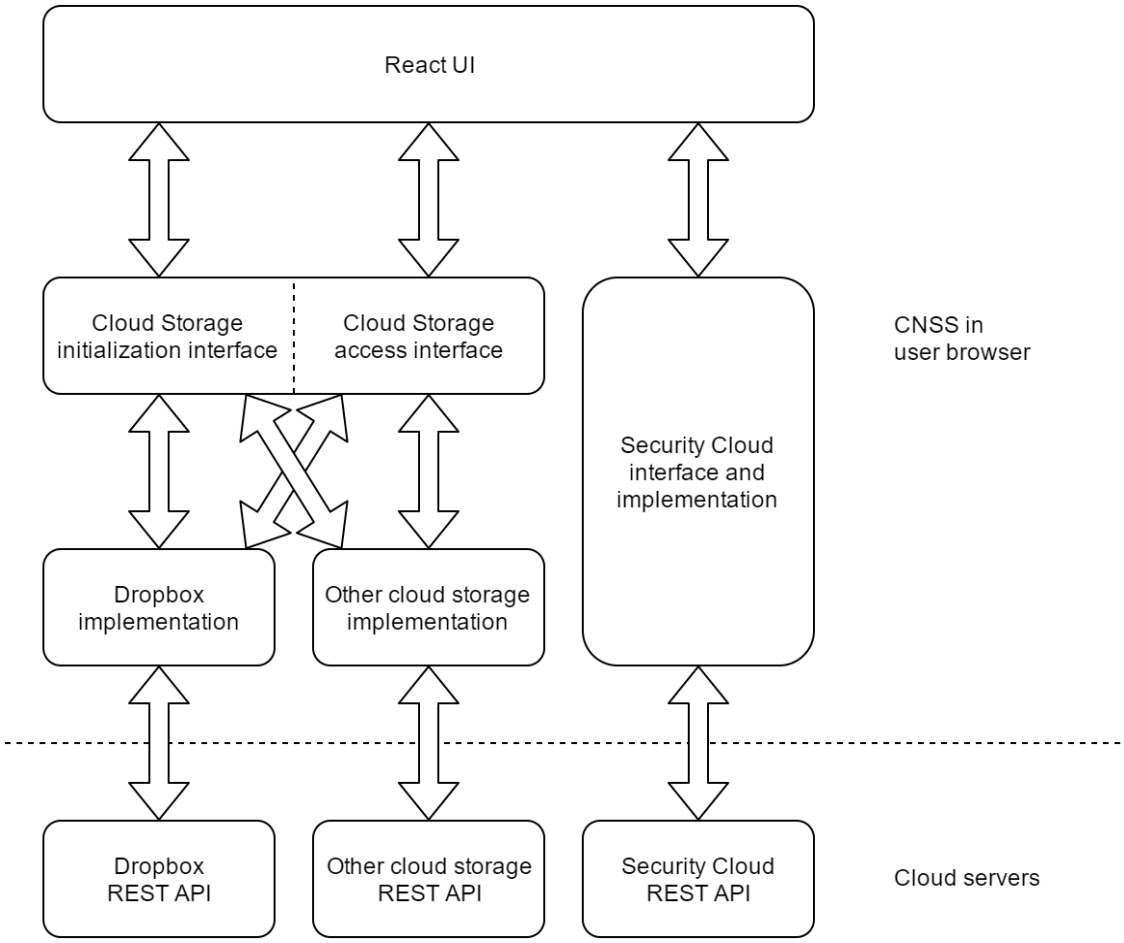

Figure 2. Internal