Zap!

Faster Authentication

Overview

In project Zap, we're trying to explore alternatives ways for authentication. From the surveys and user meetings conducted in January, we set out to implement three ways: fingerprint, voice via speaker recognition, and magic link via email. We showcase these three implementations by creating an mobile application on Android platform, which is supported via RESTful services with a Ruby on Rails server, deployed on Heroku. Hence there are 2 separate codebases which are the Server side and the client side which is the Android app.

Server

The server acts as a centralized hub to handle all the real authentication process. Underneath the three different implementations, we still use password to authenticate the user. This password is chosen by the user on account creation, regardless of which method he uses afterwards to login. We chose Ruby on Rails as our Web framework, which has several plugins and functionality for authentication. Inside, we have RESTful interfaces created for API calls, such as create user profile, password matching, password update, authorization token verification, magic link.

For user management, we chose the Devise gem. Devise is a flexible authentication solution for Rails based on Warden, it is Rack based and based on modularity, so we can chose to use what we need and get rid of the rest. The latest version of Devise deprecated the authorization token, but since we need to use the authorization token for magic link, we manually generate an authorization token and assign it to the user. When the user registered with the server, it will assign an authorization token to the user, once the server has authenticate the user using password matching, it will send the authorization token to the user, and the user can keep it as their pass to access services on the server. Once the user send a logout request, the server will assign a new authorization token to the user.

When the user send a magic link sign in request to the server, the server will send the authorization token to the user's email address in the form of a web link. This is done with Rails ActiveMail service and Gmail, we configure the smtp service for Gmail. Once the user receive the email and click on the link, it will re-open the application on user's Android device and parse the authorization token in the link. Then the application will send the authorization token along with their email address to verify the authorization token. If it is successfully verified, then we determine that the user has successfully signed in.

Since we want to eliminate the possibility of user input the wrong email address, so they can use the magic link authentication without any trouble, we implement a verification process, which will send an verification email to the email address after user registration, once the user clicked on the link inside the email, the registration process is complete.

The server is deployed on Heroku, the database which Rails ActiveRecord is accessing is PostgreSQL, which is free to use on Heroku.

Android Application

In the application there are 2 major actions a user can take -

- **Register** - requires an email and a password.
- **Login**
    - **Password**: required initially
    - **Voice**: requires 3 voice samples
    - **Fingerprint**: requires fingerprint reader in hardware
    - **Magic link**: requires verified email address

In the implementation, the application communicates with the server via REST API calls with JSON body. To do this, we are using okHttp and gson libraries, which provide a simplified interface and better parsing capabilities than those built in Android. For the login part, the user needs to login at least once via Password before being able to use three alternate methods. The reason for this is that for simplicity, we store the username and password in the app's private storage (shared preferences). When the user is authenticated via the below three methods, in the backend, the user's email and password are sent and the user is logged in (except for the magic-link).

Fingerprint

This method of authentication requires a fingerprint enabled android device. The user is also required to register his/her fingerprint into the device before using this method. Since android stores its fingerprints inside hardware, we have no way of 'storing' user's fingerprints in any form at the backend and therefore have to rely on the methods provided by Android to verify fingerprints.

The user enters his/her email (username) and clicks on the button 'login via fingerprint'. This takes the user on a new page where he/she is prompted to place his/her fingerprint on the sensor. Once the user is verified, a backend API call sends the user's email (username) and password to the server and takes him/her to their profile page on successful verification of the credentials.

Done:

- Read fingerprint from hardware and verify it.
- Log in the user with the server.

Doing:

- Improve the layout, with instructions
- Better Error handling and messages.

Roadblocks:

- Working with Callback function after fingerprint verification
- Ensuring a device has fingerprint feature.

Voice

This method of authentication verifies the user by his/her voice. We have used Microsoft's Cognitive (Speaker Recognition) API. Authentication via voice requires the user to follow a set of steps to set it up.

Step 1. Profile creation and Enrollment: Profile creation is done automatically for the user. For the enrollment we require 3 voice samples. These enrollments depend on a phrase which is provided by the Microsoft service, which means the user enrolls and verifies using the same phrase.

Step 2. Verification: Here the user records the same phrase with his voice to authenticate. The sample first goes through to Microsoft which matches it against the previously provided samples. After it is successfully validated we send a request to our server to login the user.

Done:

- Create profile via API call to Microsoft
- Implement voice recording inside the app
- Enroll via 3 voice samples.
- Verify via 1 voice sample.

Doing:

- Provide multiple Phrases for enrollment
- Better layout and instructions

Roadblocks:

- Using a 3rd party Voice recorder inside the app
- Encoding the voice sample to right format for Microsoft service.

Magic link

A magic link is a unique url, which when clicked signs the user into his account. This link is sent to a user's email, which is a users primary identifier and needs to be verified before using this method). This type of authentication is currently in use at platforms like Slack(chat), and Medium(blogging), and the authors were inspired by them. Here, the user only needs access to his email to login. Magic links need to secure such that they shouldn't be guessable, and once a link is used to login, it should not be reusable; and the link should expire after some amount of time.

To login via this method, the user enters his/her email and selects 'login via magic-link' button. This event triggers an API call to ZapServer prompting it to send the 'magic-link' on the user's email address. The core functionality of magic-link is a unique authorization token generated by the server, for a user account. When the user clicks on this link, he is directed inside our App where the auth-token is parsed from the link. A subsequent API call is then made to verify that auth-token with the server. On successful verification, the user is logged in.

Done:

- Implement server side auth token generation.
- Sending the mail from the server to user.
- Open and validate link inside the app with API call.

Doing:

- Improve layout, better error messages.

Roadblocks:

- Parsing the JSON properly
- Opening the link inside the app