

Technical Report: Voice Crack

Rick Boks s1862979 & Rens Dofferhoff s1664042

January 2020

1 Introduction

On a daily basis, we use many online services. For security reasons it is important to have decently long passwords with high entropy, and passwords should not be reused among multiple services. These factors make it difficult to remember the multitude of passwords we use. A prominent replacement of conventional passwords is biometric authentication. Biometric authentication relies on unique biological characteristics of an individual to verify the identity of the user. Examples of such biometric technologies are facial recognition, fingerprint scanning, and voice identification. Biometric authentication is more user friendly as it does not require the user to remember or store something. The major weakness of biometrics lies in the fact that they are not hidden or secret. Biometrics are out in the ether and can be copied. This weakness is most prominent in voice authentication due to proliferation of microphones in our surroundings that can record our speech. With a simple replay of the recorded audio, complex Automatic Speaker Verification (ASV) systems can be fooled. Such an attack is known as a replay attack. To guard against replay attacks many defences have been proposed (e.g. [3],[4]).

In this project we wish to develop an easy to use authentication mechanism that uses speaker verification and implements a mechanism to defend against basic replay attacks. The replay attack defence will be based around the recognition of background audio played while the user executes an authentication attempt. The audio will effectively act as a watermark for an authentication attempt. This idea was proposed in [1] and patented by Amazon [2].

This report will give technical details on the various components of our project, which future work can use as a foundation.

2 Project Overview

An overview of our project is depicted in Figure 1. Our project consists of two major parts: 1) an Android app using which users will authenticate and 2) a server which verifies the identity of the user. When a user requests authentication using the app, a request is sent to the server. The server generates a piece of audio which will be used for the replay defence. The audio is sent back to the user and also stored in the server database. When the user speaks their password in order to authenticate the app will play the sent audio simultaneously. The generated audio acts as a watermark of this authentication attempt and the server will attempt to recognise it in any future authentication attempts in order to detect replay of the audio. The recorded authentication audio will be sent to the server which checks if it is a replay and forwards it to an existing speaker verification system in order to verify the identity of the user. Based on the results from the ASV and our replay attack recognition, a decision is made on the validity of the authentication attempt, which the user is notified of through the app.

The server is capable of handling multiple users simultaneously. The server's functionality is provided using a RESTful API: each instance of the server program is stateless, making it easy to scale the server when more users need to be supported. The ASV used is Microsoft Speaker Verification, this accessible via an API provided in the Azure platform [5].

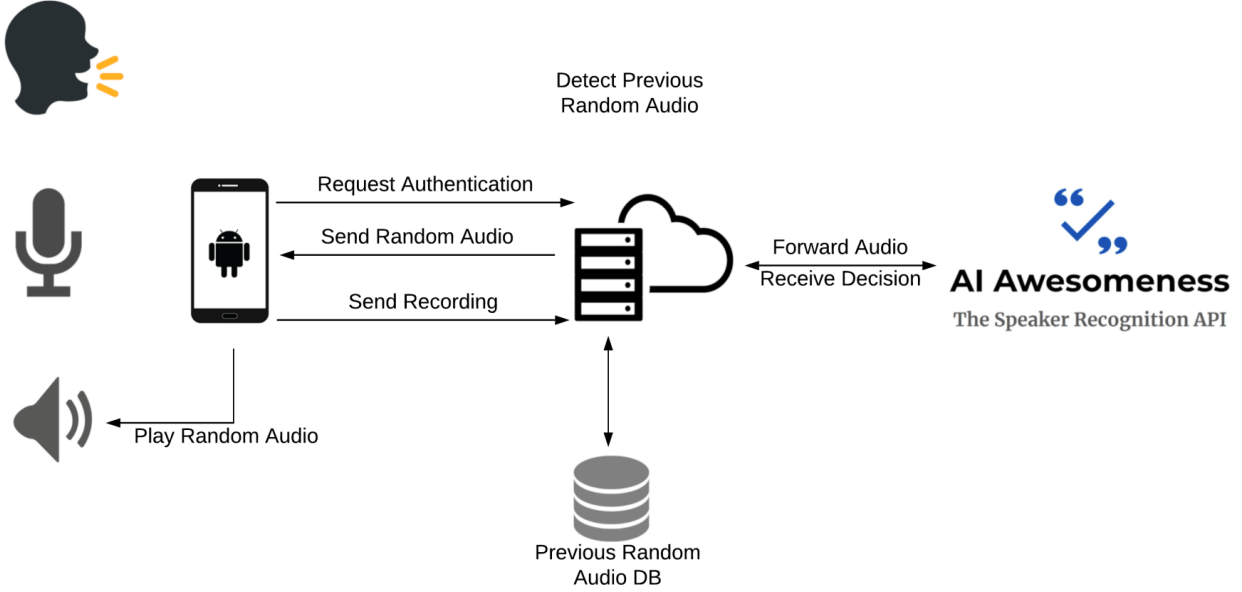


Figure 1: High-level structure of the project

3 Server

Figure 2 gives an overview of the various server components. In order to safely communicate data, the client communicates over an secure connection using TLS 1.2 or higher. This prevents weak ciphers of the older SSL and TLS being used. The secure connection is a necessity since the audio recording used to authenticate must be properly encrypted. To prevent man in the middle attacks, the TLS certificate used by the server is signed by a proper certificate authority (Let’s Encrypt [6]). The communication fronted is provided by a lighttpd (1.4.53) webserver which handles the TLS connections and decrypts incoming requests. The lighttpd webserver is configured to forward the decrypted requests to one of the worker programs which contain the authentication logic. Lighttpd will loadbalance the requests using a least connection algorithm, providing proper scalability. The configuration file for the lighttpd server is provided with the rest of the project code. Requests are forwarded to workers using the FCGI protocol, the workers are contained within a flup server providing the FCGI interface.

The workers are written in Python 3 using the Flask framework in order to setup a RESTful API for the users. Flask allows easy binding of HTTP routes to Python functions from which the various server tasks are executed. We can arbitrarily scale up and loadbalance between the different workers since they are stateless. All state is stored in a shared SQLite3 database running on the server. This database stores the users and previously used audio watermarks in two separate tables. The code of the workers is contained within the `app.fcgi` and `app.py` files. The workers are automatically started by the lighttpd webserver. The API of the service implemented by the server workers is shown in Section 3.3.

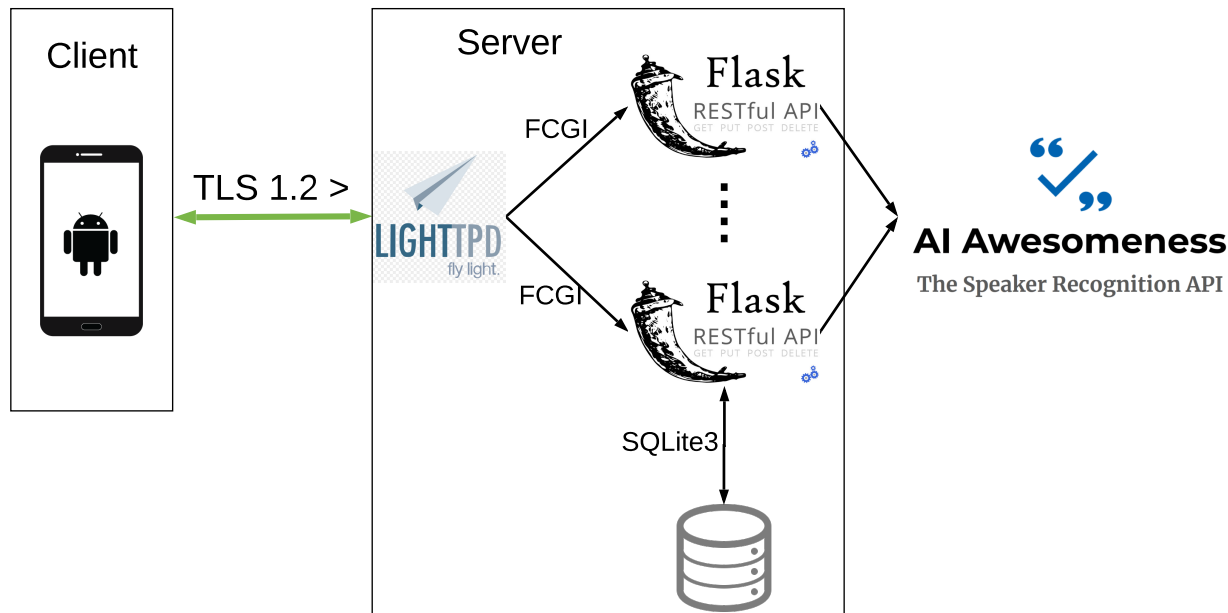


Figure 2: Overview of the server

3.1 Speaker Recognition

We use the Microsoft Speaker Recognition API as our ASV. This API provides both speaker recognition and speaker verification. The speaker verification functionality is accessible through a RESTful API on the Azure platform. We use Python 3’s build-in HTTP client to make request to the ASV, which includes user registration, enrollment and verification calls.

A new user needs to be registered and enrolled. The enrollment requires a new user to send over at least three audio fragments of one of the ten available phrases for which the ASV has been trained by Microsoft [5]. Audio fragments need to be mono 16-bit PCM WAV-files with a sample rate of 16K. The app and other future user types will probably send audio in a different format. To remedy this we use ffmpeg to automatically detect and convert to the needed format. The server workers must simultaneously maintain the local account of a user as well as the created account on the Microsoft service.

3.2 Replay Detection

We did some preliminary experiments with different types of audio watermarks to be played over authentication attempts. One idea was to use song recognition of background music. We realised quickly that the background music is hard to separate from the user’s voice. Additionally, song recognition systems do not have great accuracy on the short audio fragment provided by an authentication attempt ($< 5s$) and is easily disrupted by the user speaking over the song in close proximity. Furthermore, the ASV’s accuracy is also impacted by the music playing in the background, resulting in an exorbitant false rejection rate, making the authentication system unusable. Without deep integration of the defence into a purposefully trained ASV the the following trend seems to take shape: ‘The more difficult the separation of the watermark and voice, the higher the false rejection rate of the overall authentication system’. Amazon has a patent of a replay attack defence very similar to our project [2]. Of course, Amazon has the capabilities to custom build their ASV to work alongside the defence, instead of adding it to an existing ASV, like we do in this project.

We chose to go with a watermark based on a constant tone between 4000 and 14000 Hz. These high frequency constant tones are easily filtered out by the feature extraction phase of the ASV, meaning that there is little

interference with the users voice. The 4000 to 14000 Hz range is divided into buckets of 300 Hz in order to deal with the imprecision of measurements. The server workers generate a constant tone with a frequency associated to one of the yet unused buckets using librosa, and send the resulting audio file to the user upon a login request. The workers will store the already used tone for a user in the database so they can be checked in the future to detect replay attacks. We use a pitch tracking algorithm provided by librosa (piptrack function) to detect all major tones in the audio and check if any of the previously used frequencies are present. The pitch tracking is based on the thresholded parabolically-interpolated STFT algorithm described in [7]. The tones are easily separated from the user's voice, allowing the ASV and pitch detection to function properly, yet this also allows any clever attacker to do the same, weakening the defence. Librosa's piptrack and tone generation respectively require and generate audio in formats that might not be compatible with user input. For conversion to/from the needed formats, ffmpeg is used.

3.3 Server API

The API can be used by any HTTPS capable device/software. We choose to develop an Android app, but a website containing JavaScript could also use the API. The API is described below, for the implementation see the `app.py` file.

```
/register/<username>  
Used to create a new user  
Method: GET  
Arguments:
```

```
    -Username part of path
```

```
Returns:
```

```
    -1 DB failure  
    0 Success  
    1 MS API failure  
    2 Username already in use
```

```
/enroll/<username>  
Used to register new password sentence.  
The audio is send to the MS ASV, and results are returned  
Method: POST  
Arguments:
```

```
    -Username part of path  
    -File upload containing enrolment audio data
```

```
Returns:
```

```
    1 Missing audio file  
    MS JSON result structure containing num of enrolments left to go
```

```
/login/<username>  
User login request. Generated audio and send back to user  
Method: GET  
Arguments:
```

```
    -Username part of path
```

```
Returns:
```

```
    Code 404 user not found  
    -1 Error  
    Audio data file named sound.wav
```

```
/authenticate/<username>
Authentication attempt. Check attached audio for replay and
Analyse MS ASV results.
Method: POST
Arguments:
    -Username part of path
    -File upload containing authentication audio data
Returns:
    -1 Error
    0 Successful authentication
    1 Correct user not recognised by ASV
    2 watermark tone not detected
    3 Replay attack detected
```

4 App

The Android app was created in Java using Android Studio and the default Gradle build system. The app should work on any Android phone running Android 8.0 or higher. The app provides users basic access to the server API. It allows the user to register a username, enrol passphrases and attempt authentication. The app is capable of simultaneously playing and recording audio to facilitate an authentication attempt. The app will display the result of the authentication. If the authentication attempt was denied, the reason for this is provided. This can either be 1) denied by the ASV, 2) a replay attack was detected or 3) missing background tone. Figure 3 shows various screenshots of the app.

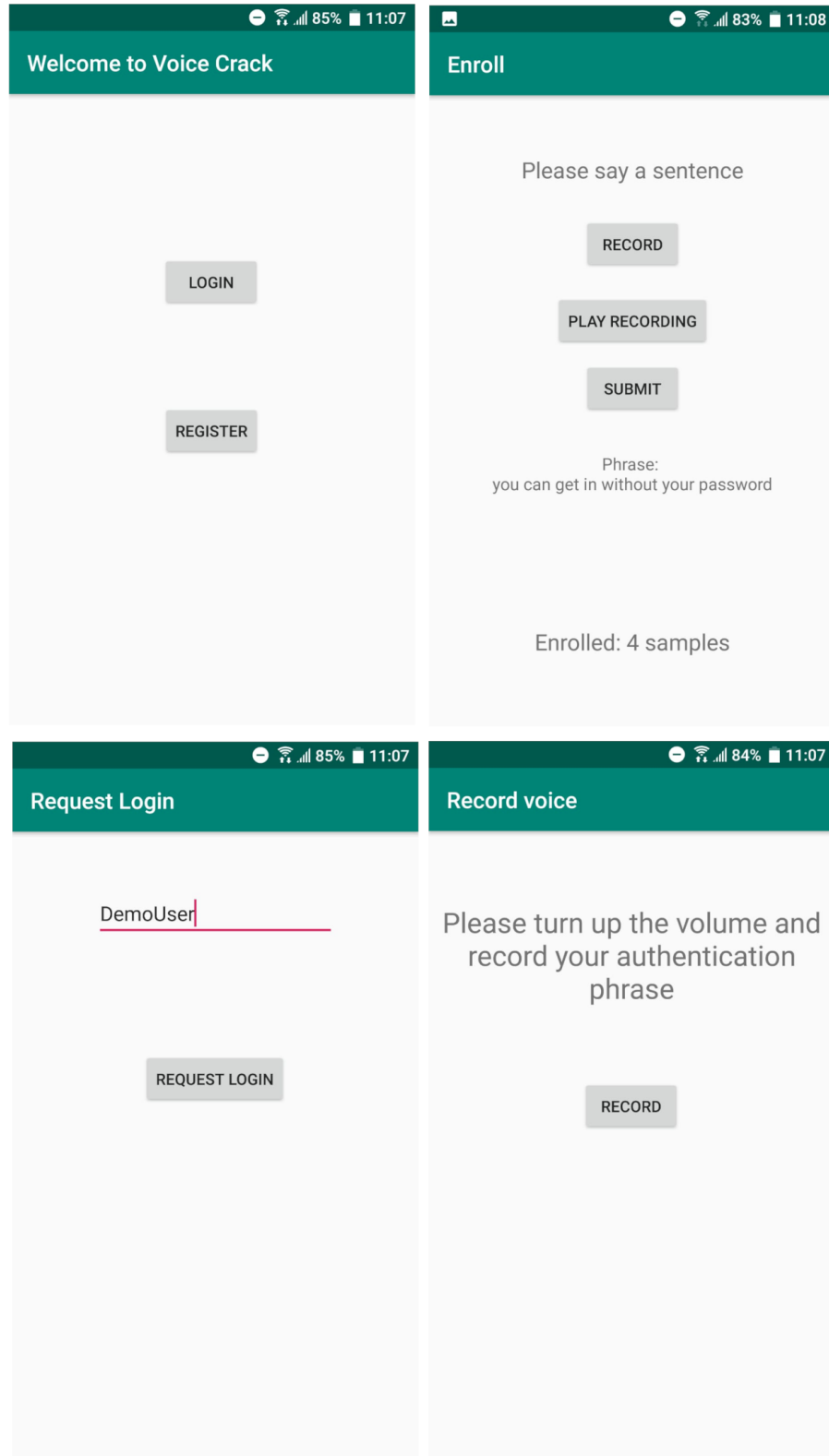


Figure 3: Various screenshots of the app

5 Code & Setup

This section will describe the code, setup and dependencies of our project. All code can be found at [8].

5.1 Backend Server

Version 1.4.53 of lighttpd was used (default Debian 10 at time of writing). The configuration file for the lighttpd webserver is provided along with the code. Alter the following lines such that they point to your TLS certificate:

```
ssl.privkey= "/etc/letsencrypt/live/domain_name/privkey.pem"  
ssl.pemfile= "/etc/letsencrypt/live/domain_name/cert.pem"  
ssl.ca-file= "/etc/letsencrypt/live/domain_name/chain.pem"
```

The resulting configuration file should be copied to:

```
/etc/lighttpd/lighttpd.conf
```

The lighttpd server can be started stopped and analysed using the systemctl command.

The worker program has the following software dependencies that must be installed on the system:

- Flask
- librosa
- sqlite3
- flup
- urllib3
- numpy

The program logic is contained within the app.py file. The worker start up script using flup is contained in the `app.fcgi` file. Both these files should be copied to `/var/www/app/`. In this folder, also create a sub-folder named `content` and make sure the www-data group has full permissions on it. The workers will create the database in this new folder. The lighttpd server will start one or more instances of the worker upon startup, the amount can be configured. Communication happens over the standard HTTPS port 443. If you are running behind a NAT-system, strict firewalls or routing mechanisms, make sure to setup your system such that this port is reachable. We run our backend on a Scaleway DEV1-L cloud instance but a Raspberry Pi 3 would also be sufficient for modest amount of users.

5.2 App

The app can be built in Android studio using the default configuration at time of writing. The app is build for minimum SDK 26 (Android 8.0 or higher). The resulting apk has been tested on 4 different phones and is provided along with the code[8].

6 Conclusion

Our replay defence works well in quiet spaces, but as was evident in the live demo, high frequency noise can disturb the system leading to erroneous results. We counteracted this by raising the threshold value at which a tone is seen as significant. The Microsoft speaker API delivers excellent performance even in noisy environments. From our experience, this audio watermark type defence as an addition to an existing ASV as proposed in [1] is not very effective, since a trade-off between separability of the watermark audio and false rejection rate is evident. Tighter integration of such a defence with a custom ASV is necessary to make this defence effective. We speculate this is Amazon's approach in the implementation of their patent [2].

References

- [1] Shoup A, Talkar T, Chen J, Jain A. An Overview and Analysis of Voice Authentication Methods.
- [2] Bhimanaik BK, Hitchcock DW, inventors; Amazon Technologies Inc, assignee. Detecting replay attacks in voice-based authentication. United States patent US 10,510,352. 2019 Dec 17.
- [3] Patil HA, Kamble MR. A survey on replay attack detection for automatic speaker verification (ASV) system. In 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC) 2018 Nov 12 (pp. 1047-1053). IEEE.
- [4] Chen S, Ren K, Piao S, Wang C, Wang Q, Weng J, Su L, Mohaisen A. You can hear but you cannot steal: Defending against voice impersonation attacks on smartphones. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) 2017 Jun 5 (pp. 183-195). IEEE.
- [5] <https://azure.microsoft.com/en-us/services/cognitive-services/speaker-recognition/> , referenced at 12 January, 2020.
- [6] <https://letsencrypt.org/> , referenced at 12 January, 2020.
- [7] https://ccrma.stanford.edu/~jos/sasp/Sinusoidal_Peak_Interpolation.html , referenced at 12 January, 2020.
- [8] https://github.com/RensDofferhoff/VoiceCrack_API_2019 , referenced at 15 January, 2020.