EasyAuth – Implementation of a Multi-Factor Authentication Scheme based on Sound, Fingerprint and One Time Passwords (OTP)

Levent Ertaul, Ishita Thanki
CSU East Bay, Hayward, CA, USA.
levent.ertaul@csueastbay.edu, ithanki@horizon.csueastbay.edu

***Abstract*—In this research paper, a multi-factor authentication scheme is facilitated in the form of an android application called EasyAuth that will improve the Login process of Twitter via three schemes: Voice/Sound Based Authentication, Fingerprint Authentication, and One-Time Password (OTP). There is a well-established and known 2-factor authentication scheme, however, the EasyAuth application is designed as a one-step advancement authentication scheme that integrates three types of authentication methods making use of multi-factor authentication with a random selection for those methods. The authentication process is explained step-by-step with help of code snippets for each of the schemes used in the EasyAuth application. This paper explains and gives detailed description on the Vigo library used in the voice authentication as well as SPass API used for fingerprint authentication. In addition to voice and fingerprint authentication, a randomly generated one-time password generation is used which is produced by a web-service. Finally, test results are shown for all three authentication systems used in the EasyAuth application which makes the login process much easier for end users by using the provided schemes that can be randomized further to improve security.**

I. INTRODUCTION

Mobile users are nowadays highly frustrated because they need to remember distinct passwords for different websites as well as for their Android applications that include a variety of requirements for setting a password. For example, common requirements include minimum length or combination of: upper case letters, lower case letters, special characters and numbers. Users also need to remember those passwords which can be an irritating process and every end user wants very high security but without bearing the stress and pain of creating and remembering lengthy passwords [1].

The first authentication scheme is voice based authentication using the Vigo library [2] that is provided by a company based in California named Voice Vault Inc. The Vigo library is a standardized approach to mobile voice biometrics that is based on the simple and overriding idea that mobile use cases are inherently the same: people are using the same devices, in the same environments to achieve the same goals. The Vigo approach means that building and deploying voice biometrics in a mobile app is as simple as possible and can be achieved in the shortest possible time and with a minimum of resources [23]. It is a mobile voice biometrics service which is working on two major guiding principles: Simplicity and Standardization. Vigo is hosted by Amazon Web Service (AWS) and there is no local infrastructure to deploy or maintain [3]; hence, the data is stored on the cloud and the Voice Vault manages and maintains it for you and assures you that it is secured.

The second type of authentication scheme used in EasyAuth is fingerprint authentication – the basic algorithm used here compares two fingerprints and upon finding the match, it grants access to the system. It is proved that every human has unique fingerprints and hence it is useful for authentication process [4]. EasyAuth uses Samsung Pass API to authenticate a user before providing access to a Twitter account. In order to make this feature work, the system requires a fingerprint sensor to digitally capture the image of users' fingerprints which is called "live scan". This live scan is digitally processed to make a biometric template out of it, which is stored and used to perform matching. There are various technologies used in the sensors which are piezoelectric, piezo resistive, ultrasonic etc. are used in the sensors.

To add another security layer to the above schemes, the idea of adding OTP is explored. OTP works like a token which changes every time or on periodic time intervals [5]. This is used to add an additional layer of security as it has been used in other 2-way authentication schemes and has been successful. A huge example of this scheme is Google Authenticator which provides users with this functionality but has some limitations too for third party applications [6]. OTPs have been proved better than a user selected password since passwords selected by users are relatively weak and can be guessed or cracked easily; contrary OTP's are not easily guessed since they change every 30 or 60 seconds.

In EasyAuth, a special provision is made so whenever users select from the above three schemes; a random scheme will be shown to them for authentication which is totally random and very hard to guess too. This is the one twist introduced for EasyAuth for making the system significantly secured. Further, users will be able to choose the different kind of authentication which may be either one of the users' choice or they can select 2-way or multi-factor as per their security requirements. This makes EasyAuth best fit and simple to use as based on users need.

This mechanism will solve the password frustration problem of the users since every scheme adopted in EasyAuth is making use of users' voice/finger biometrics that they do not need to remember. Section II of this paper addresses the comparison between 2-factor and multi-factor authentication.

Section III will highlight the details of how the multi-factor authentication method was implemented in EasyAuth, and provide the system architecture for the same. Section IV will present the test results obtained during testing EasyAuth application. Section V gives the final conclusion.

II. 2-WAY AUTHENTICATION VS MULTI-FACTOR AUTHENTICATION

There are different types of technologies used to provide authentication. Some of them include: 2-way authentication (2FA) or 2-step verification or multi-factor (MFA), and 2 or more step authentication.

The 2-factor authentication is basically based on two things: something you have and something you know. If a system provides authentication that fulfills the above two things, then this can be called a two-factor authentication system. For instance, a user can have his username and password and user will be provided by a token, which is a random number and is independent of the system. Once he enters his credentials to login to system and if the token which user entered matches then only he will be allowed to login otherwise the authentication fails.

The 2-factor authentication is very unpopular because it involves extra steps that the user must complete in order to log into the system. Traditionally, hardware tokens like RSA Secure ID [7] and dongles [8] were used which complied with FIDO U2F [9] standard for universal two factor authentication. These methods may not be cost effective and need a physical token that is provided by a service provider to each user. Nowadays software tokens are being used instead of hardware tokens to make it easy and affordable cost wise. Google 2-Step Verification [10] is one such example, which allows users to enter verification codes.

But, these systems are giving a hard time to users because they need to wait for a code and sometimes it would take a long time to authenticate. One more flaw in Google Authenticator is that it is based on the local time of the system so if your time is incorrect then it will not authenticate you since the algorithm fails to work. Hence, it is important that your phone's time is accurate [10] [11]. Most of time 2-factor authenticator and multi-factor authentication are considered the same but they are not similar. One example is a corollary of mathematics which says "Every square is a rectangle but not every rectangle is a square". This means the criteria defining one object doesn't encompass the criteria for the second object in all situations [12]. Hence as both of these sounds too similar it is very common for many companies to market their products as having multi-factor authentication.

Basically, multi-factor authentication comprises of three criteria's – something you know, something you have, and something you are. These three factors are needed, the last factor "something you are" plays a very significant role since it will be very unique and independent from the system used. Further, there's no way an attacker can guess or have any idea of that authentication factor as it is based on something you are. For instance, the "something you are" factor can be a biometric of the user which may require them to give their voice biometric, fingerprint, retinal scanning, or facial recognition [13].

One can design a system where users' needs to go through multiple layers of security requiring them to enter their username and password followed by a verification code and finally authenticate with biometrics. But in spite of designing such systems, which make use of 2-factor authentication or multi-factor authentication, most of the users prefer password-only authentication for services where 2-factor authentication is not mandatory [14] [15]. This is probably due to the extra burden that 2 factor authentication causes to the user [16] [17].

The best example of multi-factor authentication currently is Microsoft Azure multi-factor authentication. They provide many different ways by which user can authenticate themselves and hence called multi-factor authentication. The Microsoft Azure has one additional fraud reporting feature in which a user gets a message asking to proceed for step-2 verification using OTP. At that time user can press a Fraud Report button, which will help in detecting frauds [18].

There are different vendors who provide MFA solutions like EMC RSA Authentication Manager which is part of its SecureID technology, Symantec Verisign VIP, CA Strong Authentication and Vasco Identikey Digipass [29]. There is one more alternative to this that is offered by LastPass [30] which gives 2-factor authentication. Also PCI will make use of MFA which has been proposed recently [31].

On further research, one can make up a point that it is impossible to use such methods which provide high security but are frustrating to users where users need to remember passwords and again enter some more information to secure their systems. One such system which makes use of multi-factor authentication, which gives users the freedom to authenticate themselves very easily as well as with their biometrics, could be one solution to reduce the password frustration and make the authentication process easy for users which is discussed in the next section.

III. EASYAUTH APPLICATION - IMPLEMENTATION

In order to provide ease to users to authenticate themselves we came up with an idea of building and designing an android application called "EasyAuth". The main objective behind creating this application was to provide users a very simple mechanism to make the login process easier. Twitter accounts are too much likely as well as relatively easy to hack. History shows that approximately 250,000 Twitter account passwords have been compromised by hackers [20] and Twitter suffered high-profile spate of hacks in 2013 [21].

To overcome this, Twitter implemented 2-factor authentication but it was not mandatory for users and thus asked users to create strong passwords. However, recent news shows that "Twitter is emailing users whose account security was compromised by a bug last week, exposing email addresses and phone numbers linked to "a small number of accounts." The company also said that fewer than 10,000 accounts were affected [22]. The above news clearly shows that Twitter accounts are very much targeted by hackers, and it would be difficult for twitter users to secure their accounts if the accounts get hacked even after implementation of 2-factor authentication by the company.

We figured out how to build such an application which uses authentication factor which is “something you are” and came up with an idea of providing enhancement to Twitter users by giving them an option of multi-factor authentication with random selection. The system will only be able to decide on the basis of selected options provided by users which will make the login process of Twitter very convenient and difficult for hackers to guess and crack too.

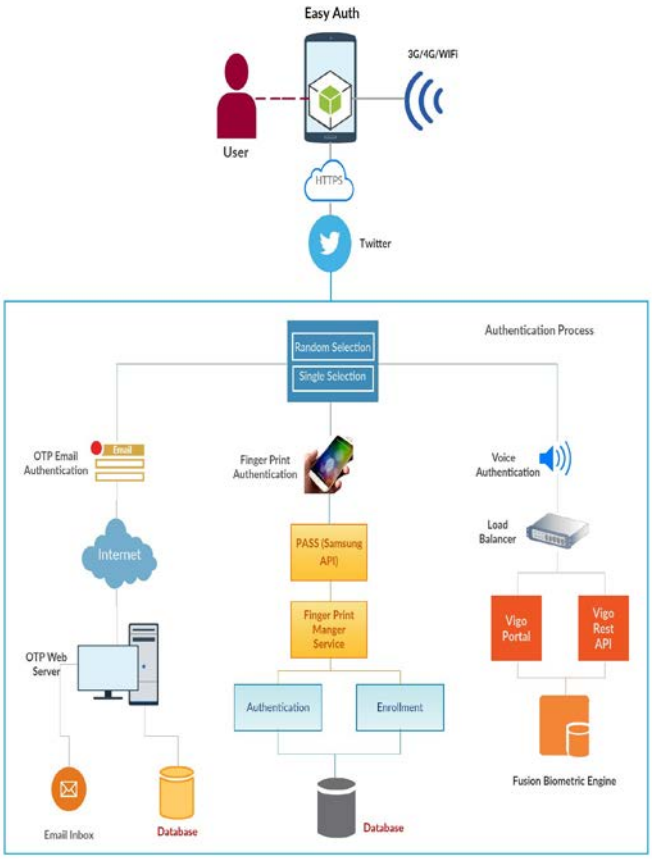

Fig. *1 System Architecture Diagram for EasyAuth Application*

Figure 1 above depicts the system’s architecture of EasyAuth and gives an idea about how the functioning of the EasyAuth application and various components used to build this application. This includes various APIs used to integrate Twitter with the different types of authentication schemes provided by the application in Android OS. As shown in Figure 1, the user needs to login to Twitter.com via EasyAuth; we have used Twitter -4j API [18] to integrate Twitter login with EasyAuth which is done securely using https connection. The users need a smart phone and the internet to download and install EasyAuth.

Furthermore, users will login to Twitter using their Twitter ID and password which is a single type of authentication. In integrating Twitter login to EasyAuth, whenever user logs in Twitter – an Auth token is generated and sent back which is further used to retrieve user’s information for user Tweets. Once they login, users will be provided three types of authentication schemes which acts here as multi-factor authentication. These includes authentication with voice, fingerprint, and OTP. We have given flexibility to users to select any of one, two, or all the three schemes based on their requirements as well as the level of security they need.

Once they select the authentication scheme, they will be asked for registration with the preferred scheme. For instance, if a user selects voice based authentication, then in the next phase, a user will be asked to register his/her voice. This will be done with the help of Vigo Library where a user will be asked to speak phrases composed of 4-digit codes, i.e., “1234”. This process is repeated until the user’s voice is successfully stored in the cloud server used by Vigo [23]. Next time when users logs in, they will be asked to speak a random 4-digit code and the spoken phrases will be matched with the voice samples stored in the Fusion Biometric Engine whenever users attempts to log in to Twitter.com using EasyAuth.

The second authentication scheme is Fingerprint Authentication which asks users to log in to Twitter using their fingerprint. We have used Samsung Pass API [24] to implement this in EasyAuth where users’ needs to first register their fingerprints using the enrollment module, and later they will verify their fingerprint at the time of authentication by placing their finger on the hardware sensor which is provided by most recent Samsung Galaxy cell phones including Samsung Galaxy S6 Edge, Samsung Galaxy S7 etc. In this way, users will be able to use the inbuilt fingerprint sensor for authentication making this cost effective, fast and secure.

The third scheme is Time-Based One Time Passwords authentication scheme. To use this, we make use of a web-service which is called to send an OTP to the email of a user which he/she will enter when asked for by this type of authentication during the login to Twitter. The only reason we ask users to enter the email address again during login is due to security issue of Twitter’s API which will not allow the storing of credentials for security reasons. Also, we believe that user credentials are highly confidential and must never be stored anywhere in the application.

In EasyAuth, there is a twist to enhance the security to its pinnacle because if a user selects all three schemes, next time whenever a user wish to login to Twitter, the system itself will give any one of the three schemes to provide authentication. The scheme chosen will be random, making it very hard for an attacker to guess what kind of authentication scheme will be asked by system. This is very significant and this is the most important aspect of our project.

The benefit of random selection is tremendously useful to users since they do not need to go through several steps in order to authenticate themselves. This will definitely avoid many problems which lead users to use multi-factor authentication schemes and help them login to their Twitter account easily without any issues of entering any extra information.

This project was implemented on a personal computer with a 2.5GHz Intel i7 processor, 8 GB of RAM, and running on a Windows 8.1 OS. Android Studio 1.5.1 Build 141.2456560 [25] was used to develop the project. The minimum SDK version required for this application development is 21; the minimum complier version required to build this application is 23 and the build tool version is 22.0.1. Apart from the GUI-related code, there are four prominent code portions categorized as 1) Twitter Integration with EasyAuth, 2) Voice authentication using Vigo and its operations, 3) Fingerprint

Authentication using SPass API and, 4) OTP Generation using web-service. The section below describes about the important code snippets of the EasyAuth application.

A. Twitter Integration with EasyAuth

1) Code for integration with Twiiter.com – Twitter Login

```
private void loginToTwitter() {
    boolean isLoggedIn =
mSharedPreferences.getBoolean(PREF_KEY_TWITTER_LOGIN,
false);
    if (!isLoggedIn) {
        final ConfigurationBuilder builder = new
ConfigurationBuilder();
        builder.setOAuthConsumerKey(consumerKey);
        builder.setOAuthConsumerSecret(consumerSecret);
        final twitter4j.conf.Configuration configuration =
builder.build();
        final TwitterFactory factory = new
TwitterFactory(configuration);
        twitter = factory.getInstance();
        try {
            requestToken =
twitter.getOAuthRequestToken(callbackUrl);
            final Intent intent = new Intent(this,
WebViewActivity.class);
            intent.putExtra(WebViewActivity.EXTRA_URL,
requestToken.getAuthenticationURL());
            startActivityForResult(intent,
WEBVIEW_REQUEST_CODE);
        } catch (TwitterException e) {
            e.printStackTrace();
        }
    } else {
        loginLayout.setVisibility(View.GONE);
        shareLayout.setVisibility(View.VISIBLE);
    }
}
```

This function contains functionality regarding twitter's login. It is using Twitter4j library to perform the login. The first line checks the applications' local shared preferences to check if a user of twitter is already logged in. We have stored the flag regarding twitter login as true or false to app's private context. So, if we get that false, it will attempt for the twitter login with a consumer key and secret provided by twitter developer console. In case of requesting twitter's API, developers need to request twitter API with consumer key and consumer secret. To get consumer key and consumer secret, developers need to create an app here [26] and fill the details regarding the app. Once it is successfully created it will obtain Consumer Key and Consumer Secret which will be used to send request to twitter's API. The above method creates the instance of twitter library and sends request for authentication token to twitter's API using the provided consumer key and secret. Once the Auth token is obtained, the user login will be successful.

2) Initialize Twitter Configuration

```
private void initTwitterConfigs() {
    consumerKey =
getString(R.string.twitter_consumer_key);
    consumerSecret =
getString(R.string.twitter_consumer_secret);
    callbackUrl = getString(R.string.twitter_callback);
    oAuthVerifier =
getString(R.string.twitter_oauth_verifier);
}
```

To request twitter's API, we need for authentication, we need to use a Consumer Key and Consumer Secret, which can be obtained by creating application at twitter's developer console [27]. We have created new app at twitter's developer console, completed the app creation process and obtained the Consumer Key and Consumer Secret to integrate in our Android app. Also, to use these configuration details throughout the application, we have stored the values to the string.xml file. From there we can get values by calling a getString(int ID) function. In this way, we have initialized the local String variables before calling loginTwitter( ) function.

3) Save Twitter Configuration to local shared preferences

```
private void saveTwitterInfo(AccessToken accessToken) {
     long userID = accessToken.getUserId();
     User user;
     try {
         user = twitter.showUser(userID);
         String username = user.getName();
         SharedPreferences.Editor e =
mSharedPreferences.edit();
         e.putString(PREF_KEY_OAUTH_TOKEN,
accessToken.getToken());
         e.putString(PREF_KEY_OAUTH_SECRET,
accessToken.getTokenSecret());
         e.putBoolean(PREF_KEY_TWITTER_LOGIN, true);
         e.putString(PREF_USER_NAME, username);
         e.commit();
     } catch (TwitterException e1) {
         e1.printStackTrace();
     }
 }
```

This function is used to save the twitter auth token and auth secret as well as other profile information to the application's private shared preferences. Once the twitter authentication is completed successfully, the API will return the **oAuthToken** and **oAuthSecret,** which will be used to fetch other details regarding logged-in users. Once we get all details, we call a saveTwitterInfo function to save all details to private context. Those can be used globally throughout the application in case of fetching details regarding logged-in user's details. We also need to display logged in user's tweets once the authentication is completed successfully. So, in that case we can use oAuthToken and oAuthSecret to request twitter's API and fetch user's tweets and display to the screen. In case of this, we use these values globally in the app; we have stored the information to application's shared preference using this function.

B. Voice Authentication using Vigo and its operations

1) Code for initialization using Vigo credentials

```
ViGoLibrary.getInstance().init(
        VIGO_CREDENTIAL_ID,
        VIGO_CREDENTIAL_PWD,
        VIGO_SERVER_URL,
        VIGO_APP_ID);
```

In case of voice authentication, we need to use Vigo Library. The above code is used for initialization of the library. To request Vigo API, we need CREDENTIAL_ID, CREDENTIAL_PASSWORD, URL and APP_ID. To get these details we need to complete registration at Voice Vault [28]. Once we get all the details, we need to integrate them to

application and using these we can send request to Vigo Library. It is important to note that the Vigo Library is paid but provides a 45 days' trial version, which we have used for our development.

2) Code for Register User using Vigo library

```
public void startRegistrationClick(View button) {
    button.setEnabled(false);
    if (mClaimantId == null) {
        ViGoLibrary.getInstance().registerClaimant(this)
    }
    else if (!mIsClaimantRegistered) {
        registerClaimantCallback(mClaimantId);
    }
}
```

In case of using Voice Authentication, we need to register the voice with Vigo Library before using login. To register the voice with Vigo Library the above code snippet will be used. Also, when registration is completed successfully, the registered call back event will get notified and the details will be passed to a new recording screen. Once the user is registered with Vigo Library, the claimant ID will be provided to users and that will be used to record an audio phrase with Vigo Library.

3) Code to record voice using Vigo library

```
public void recordClick(View recordButton) {
    findViewById(R.id.buttonRecord).setEnabled(false);

findViewById(R.id.buttonRecord).setSoundEffectsEnabled(false)
;
mTextViewStatus.setText(getString(R.string.status_recording))
;
    ViGoLibrary.getInstance().startRecording(
            VIGO_RECORD_TIME_MILLISECS,
isAudioRecordVoiceRecognitionOptionEnabled, this);
}
```

In the case of voice authentication, the audio should be registered to Vigo API. The above displayed function is used to record a user's voice with Vigo Library. The method is useful to both REGISTER YOUR VOICE and LOGIN WITH YOUR VOICE cases. There is a method with Vigo Library to record user's voice, startRecording () which will require 3 arguments: Time in millisecond (VIGO_RECORD_TIME_MILLISECS), in or case 4000 milliseconds (4 seconds) in our case; a flag to check whether the voice recognition feature is available to the device or not; and context of activity class. Vigo Library has an interface VoiceVaultAPIVoiceCallback, which contains the callback methods like onRecordCompleted (), which will be notified when a recording is completed after 4 seconds. Thus, this way voice authentication is established as an authentication scheme in EasyAuth application.

C. Fingerprint Authentication using SPass API

```
private SpassFingerprint.IdentifyListener mIdentifyListener
= new SpassFingerprint.IdentifyListener() {
    @Override
    public void onFinished(int eventStatus) {
        int FingerprintIndex = 0;
        String FingerprintGuideText = null;
        try {
            FingerprintIndex =
mSpassFingerprint.getIdentifiedFingerprintIndex();
        } catch (IllegalStateException ise) {
        }
        if (eventStatus ==
SpassFingerprint.STATUS_AUTHENTIFICATION_SUCCESS) {
            Toast.makeText(DashBoardFingerPrint.this,
"Success",Toast.LENGTH_SHORT).show();
            Intent intent = new
Intent(DashBoardFingerPrint.this , DashBoard.class);
            startActivity(intent);
            finish();
        } else if (eventStatus ==
SpassFingerprint.STATUS_AUTHENTIFICATION_PASSWORD_SUCCESS) {
        } else if (eventStatus ==
SpassFingerprint.STATUS_OPERATION_DENIED) {
        } else if (eventStatus ==
SpassFingerprint.STATUS_USER_CANCELLED) {
        } else if (eventStatus ==
SpassFingerprint.STATUS_TIMEOUT_FAILED) {
        } else if (eventStatus ==
SpassFingerprint.STATUS_QUALITY_FAILED) {
            needRetryIdentify = true;
            FingerprintGuideText =
mSpassFingerprint.getGuideForPoorQuality();
            Toast.makeText(DashBoardFingerPrint.this,
FingerprintGuideText, Toast.LENGTH_SHORT).show();
        } else {
            needRetryIdentify = true;
        }
        if (!needRetryIdentify) {
            resetIdentifyIndex();
        }
    }
    @Override
    public void onReady() {
    }
    @Override
    public void onStarted() {
    }
    @Override
    public void onCompleted() {
        onReadyIdentify = false;
        if (needRetryIdentify) {
            needRetryIdentify = false;
            mHandler.sendEmptyMessageDelayed(MSG_AUTH, 100);
        }
    }
};
```

In case of the fingerprint authentication, users need to register the fingerprints with SPass API. After registration is successful, it will start identifying the fingerprint and compare with a registered one. The above code snippet is showing the listener for the identification process. There are various methods that are called in various cases which are explained below. 1) onReady - This will notify when the SPass library is ready for the identification process. 2) onStart - This will notify when the user starts the identification process.3) onFinished - This will notify when the user finishes the identification process. 4) onCompleted - This will get notified once the whole process finished. Here we are attempting to retry if identification is not successful. The requirement for fingerprint authentication is a smart phone with an inbuilt sensor to make use of this authentication scheme.

D. OTP Authentication using web service

```
private void startTimer() {
    countDownTimer = new
```

```
CountDownTimer(totalTimeCountInMilliseconds, 500) {
        @Override
        public void onTick(long leftTimeInMilliseconds) {
            long seconds = leftTimeInMilliseconds / 1000;
            if (leftTimeInMilliseconds <
timeBlinkInMilliseconds) {

timer.setTextAppearance(getApplicationContext(),
R.style.blinkText);
                if (blink) {
                    timer.setVisibility(View.VISIBLE);
                } else {
                    timer.setVisibility(View.INVISIBLE);
                }
                blink = !blink;
            }
            remainingSecond = leftTimeInMilliseconds;
            Log.i("", "++" + remainingSecond);
            timer.setText(String.format("%02d", seconds /
60)
                    + ":" + String.format("%02d", seconds
% 60));
        }
        @Override
        public void onFinish() {
            Intent intent = new
Intent(CodeVerificationActivity.this,
ResendCodeVerification.class);
            intent.putExtra("emailId", emailId);
            startActivity(intent);
            finish();
        }
    }.start();
}
```

In case of using OTP Authentication, we need to generate One Time Password to complete the process. Once the OTP is generated, it will be sent to our server and will also be sent to users via email; valid for 2 minutes. The above method is used to display the timer for the OTP lifetime on screen. Once the timer is over, the user will be redirected to another screen with the message “The OTP which was sent to you in email is expired”. If desired, users can resend the OTP again by clicking on Resend OTP.

```
Random rnd = new Random();
n = String.valueOf(100000 + rnd.nextInt(900000));
```

The above code snippet is used to generate a new random number between 100000 and 900000. We can change the range also and generate more complex codes that contain alpha numeric characters and special symbols randomizing it as required to make the guesswork difficult for hackers.

IV. Test Results For EasyAuth

We took several test cases and tested the EasyAuth Application in various different environments and the entire performance analysis is shown on graphs. They are plotted by taking fixed number of counts and later on we are able to derive which authentication scheme is accurate and which one fails when the environment changes.

Fig. 2 Test Results in various environments for Voice and Fingerprint

Figure 2 depicts pass as well as fail scenarios for all the different test environments we choose for testing this application. We tested this application in environments which include: authentication with voice based authentications schemes at a coffee shop, while driving a car, while driving a car with loud music, while driving a car with slow music, while walking, while cooking and other noise heavy environments. We have observed that the voice authentication takes a longer time during loud music and also if one is using a mobile network like 4G/LTE, it will sometimes fail due to the lack of network availability. For fingerprint authentication, we took test cases like touching the sensor with a wet finger, applying little oil on the finger, applying talcum powder on finger and so on. We have observed that it fails to recognize your fingerprint if we consider such scenarios.

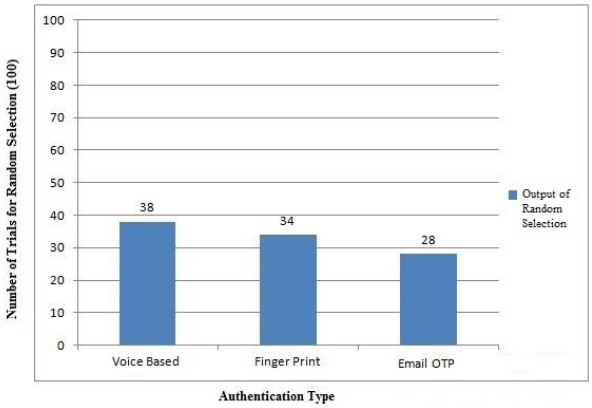

Fig. *3 Test Results which shows Random Selection given by System for three authentication schemes*

Figure 3 depicts the relationship between the number of trials (Total 100), which we selected for testing versus different authentication schemes. Whenever a user selects all three authentication schemes, the system would give any one out of three whenever a user tries to login to Twitter. We tried to login 100 times, out of which 34 times the system gave us fingerprint authentication, 28 times OTP and 38 times voice. Thus, the voice authentication scheme topped among the three during our testing.

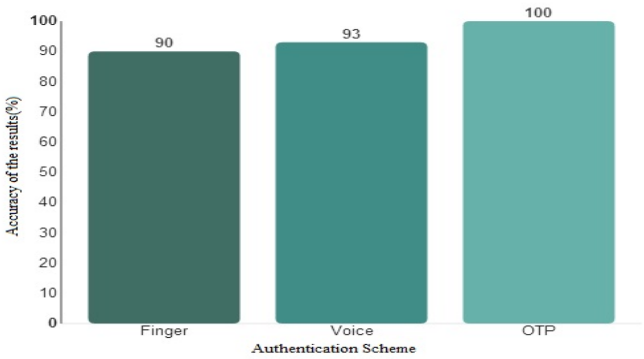

Fig. *4 Comparison of Accuracy of Voice, Fingerprint and OTP Scheme*

Implementing three authentication schemes along with a single factor authentication gave us multi-factor authentication systems especially for Twitter.com but it was quite challenging to figure out which one is more accurate. For this testing, we chose fixed counts of trails to login to Twitter but took several test cases to measure the accuracy of EasyAuth.

From the Figure 4, one can make out that of all three mechanisms, OTP is always successful if user enters correct code unless a wrong code or wrong email address is provided. To conclude, for voice it failed during loud music environments specifically and thus is 93% accurate and for fingerprint we intentionally tested by applying talcum powder as well as water on finger which resulted in low accuracy specifically for our testing around 90%.

V. Conclusion

We have shown that by implementing multi-factor authentication, we can surely make the login process of Twitter.com easier and also provide users a hassle-free login to Twitter by using "something they are" as well as "something they know", like voice, fingerprint and OTP. We also provide users the flexibility to choose the level of desired security as per their preference so users can be flexible. Our results show that multi-factor authentication is much better; and by randomizing it with different authentication schemes, can be used effectively which makes the system sustainable and secured against the guesswork of hackers. Many issues were discovered with the Android fingerprint scanner which comes inbuilt so we have aimed to resolve the problem of unrecognized fingerprints. We also target to develop an iOS version of the EasyAuth application in future.

VI. References

[1] Password Rage. http://www.information-age.com/technology/security/123459599/do-you-have-password-rage-third-people-admit-tantrums-over-password-frustration

[2] Vigo Architecture and Principles. http://voicevault.com/wp-content/uploads/2014/03/ViGo-Architecture-and-Principles.pdf

[3] Vigo Introduction & Security. http://voicevault.com/wp-content/uploads/2014/03/ViGo-Introduction_secured.pdf

[4] What is so unique about your fingerprint? http://wonderopolis.org/wonder/what-s-so-special-about-your-fingerprints

[5] One-Time Passwords – htop and totp. http://blogs.forgerock.org/petermajor/2014/02/one-time-passwords-hotp-and-totp/

[6] Google Authenticator Support Answers. https://support.google.com/accounts/answer/185833?hl=en

[7] EMC INC. RSA Secure ID. http://www.emc.com/collateral/data-sheet/h13821-ds-rsa-securid-hardware-tokens.pdf

[8] Yubi Key Hardware. https://www.yubico.com/faq/yubikey/

[9] Fido U2F Specifications. https://fidoalliance.org/specifications/overview/

[10] Google 2-Step Verification. http://www.google.com/landing/2step/

[11] Google Authenticator – Product Forum Topic. https://productforums.google.com/forum/#!topic/gmail/4-D-0lXGtwc

[12] 2-factor authentication v/s Multi-Factor Authentication. http://mi-token.com/2fa-vs-multi-factor-authentication/

[13] Multi-factor Authentication Explanation with Authentication Factors. http://searchsecurity.techtarget.com/definition/multifactor-authentication-MFA

[14] Impermium study unearths consumer attitudes toward internet security. http://goo.gl/NsUCL7, 2013Lagrange polynomial interpolation. http://www2.lawrence.edu/fast/GREGGJ/Math420/Section_3_1.pdf

[15] PETSAS, T., TSIRANTONAKIS, G., ATHANASOPOULOS, E., AND IOANNIDIS, S. Two-factor authentication: Is the world ready? Quantifying 2FA adoption. In 8th European Workshop on System Security (2015), EuroSec '15

[16] GUNSON, N., MARSHALL, D., MORTON, H., AND JACK, M. A. User perceptions of security and usability of single-factor and two-factor authentication in automated telephone banking. Computers & Security 30, 4 (2011), 208–220.

[17] WEIR, C. S., DOUGLAS, G., RICHARDSON, T., AND JACK,M. A. Usable security: User preferences for authentication methods in e-banking and the effects of experience. Interacting with Computers 22, 3 (2010), 153–164.

[18] Multi-factor authentication using Microsoft Azure. https://azure.microsoft.com/en-us/documentation/articles/multi-factor-authentication/

[19] Twitter 4j API. http://twitter4j.org/en/powered-by.html

[20] Twitter Accounts Hacked in 2013- Report in Abcnews. http://abcnews.go.com/blogs/technology/2013/02/250000-twitter-accounts-hacked-dont-panic-heres-what-to-do/

[21] Twitter Accouts Vulnerable in 2013. https://www.thewrap.com/twitter-warns-users-about-hacked-accounts/

[22] Bug in Twitter. https://blog.twitter.com/2016/fixing-a-recent-password-recovery-issue

[23] Vigo Rest API Guide. http://voicevault.com/wp-content/uploads/2014/03/ViGo-REST-API-Guide.pdf

[24] Samsung Pass API Documentation. http://img-developer.samsung.com/onlinedocs/sms/pass/index.html

[25] Android Studio. http://developer.android.com/tools/studio/index.html

[26] Developer Twitter for new application. dev.twitter.com/apps/new

[27] Create Application at Twitter Console. https://apps.twitter.com/

[28] Vigo Free Trail Registration Sign up Link. http://voicevault.com/for-developers/#signup

[29] Comparison of Top MFA products. http://searchsecurity.techtarget.com/feature/The-fundamentals-of-MFA-Comparing-the-top-multifactor-authentication-products

[30] Last Pass. https://lastpass.com/support.php?cmd=showfaq&id=375

[31] MFA Heads PCI's List of Change. http://www.paymentssource.com/news/retail-acquiring/multi-factor-authentication-heads-pcis-list-of-changes-3023992-1.html