March 2018

Mobile Behavior Based Authentication

Benjamin Huchley
Worcester Polytechnic Institute

Bryan Joshua Benson
Worcester Polytechnic Institute

Derrek R. Rueger
Worcester Polytechnic Institute

Mobile Behavior Based Authentication

A Major Qualifying Project

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree in Bachelor Science

In

Computer Science

By

_____

Bryan Benson

_____

Ben Huchley

_____

Derrek Rueger

_____

Date: March 7, 2018

_____

Prof. Emmanuel Agu, Advisor

Abstract

Traditional methods of authentication used in smartphones have been proven to be both inconvenient for users and inadequate to prevent attackers from accessing stolen devices. Many users use a four-digit PIN as their method of authentication or no authentication at all, because more secure forms of authentication are too time-consuming to use. To attempt to make authentication more convenient so that users would choose more secure methods, we designed a system, LOBS (Location, Orientation, Battery and Screen), that would authenticate users continuously and automatically by recognizing their behavior patterns. LOBS constructed this behavioral signature by examining data gathered from the phone's sensors during normal phone usage. Specifically, it recorded WiFi networks observed, GPS location, accelerometer readings, battery usage over time, and the times that the screen was turned off and on. Then, it used a neural network that was trained on the user's historical data to analyze the latest data and determine a trust score, which measured how likely that data was to be from the same person as the historical data. LOBS authenticated the user if the trust score passed a threshold without them consciously entering any identity verification. We evaluated LOBS using the F0.1 score, which is a numerical evaluation of classifiers that weights false positives (people being authenticated when they shouldn't be) ten times as highly as false negatives, since authenticating an impostor has deeper ramifications than locking out the user. After conducting an initial study to fine-tune some parameters of the system, we evaluated LOBS with a study that used data gathered from six people over the course of a week. The scores we obtained were too low for LOBS to be commercially marketable (we achieved F0.1 score of 50.1%, well short of our goal of 90%), but much higher than random chance (which would get 16.7% with half the positive cases and half the negative cases being correct), which is promising.

Contents

Table of Figures

1. Introduction

1.1 Current Smartphone Authentication Patterns

In the United States, 77% of people owned a smartphone in 2015, according to Murmuria et al [23]. With constant access to a mobile device, more than half of the population enjoys the convenience of having countless services available at their fingertips. From social media to games, news outlets, dating services, and navigation tools, the majority of smartphone users use those devices as part of their daily routines. One especially important case is that, among those that own smartphones in the United States, 51% of adults used their smartphones to make online purchases in 2015, and 12% used their devices to make purchases within physical stores. These users have to store their credit card information on their phones to perform checkouts. Even if only a small percentage of smartphone users store this data on their device to expedite checkouts, the users who practice these behaviors place themselves at considerable risk should the phone and its contents not be properly secured.

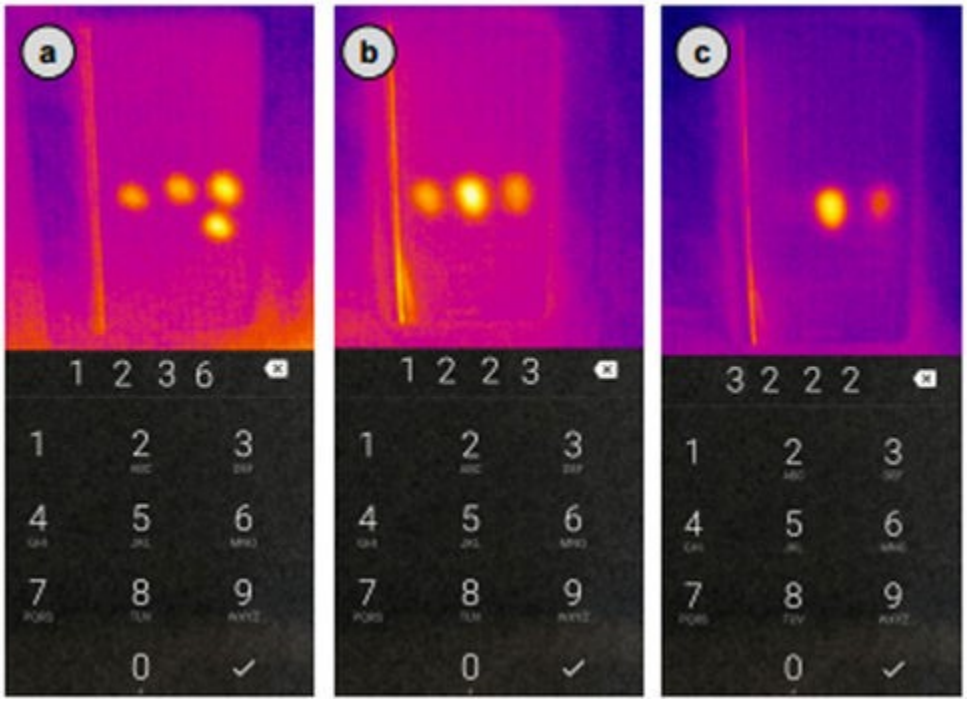
According to the Pew Research Center [20], as of 2015 the most popular means of securing a smartphone is a password in the form of a four digit PIN, used by 25% of users. In addition, an even larger percentage (28%) of the smartphone owners use no means of securing their device whatsoever. When compared to other, more secure, methods such as fingerprint scanning, facial recognition, and lengthier password options, one can reasonably assume that the more common and less secure options owe their popularity to their convenience. However, using a more convenient method of authentication invariably makes the device and any information on it less secure. Therefore, the 53% of all smartphone users in the United States that opt for no password or a four-digit PIN are sacrificing the security of their devices in exchange for convenience.

1.2 Smartphone Security versus Convenience

Today's world of omnipresent smartphones is increasingly vulnerable to security flaws in one of the most common modes of authentication: passwords. More secure passwords are difficult for the user to remember and time-consuming to enter, while simpler ones are easily cracked. Many people use the same password for multiple devices and accounts, subjecting them to greater risk should any account be compromised. In addition, users tend to store personal information, such as credit card information, on their mobile devices, much of which they need to be secured against outside attack. Common modes of biometric authentication, such as face recognition or fingerprint scanning, are potentially more secure but have a different major issue, aside from the aforementioned lack of convenience: if the attacker finds a way to impersonate the user's biometric information, such as with a photo of their face or a mold designed to match their fingerprint, the user can never change that information, so the thief will always be able to access both the user's current device and any devices they acquire in the future.

In fact, two of the most popular means of authentication among smart devices, namely four digit PINs and swipe patterns, are among the easiest of methods for hackers to crack. Highlighted in an article published by Express [19], potential thieves with access to thermal imaging only need to take a picture of the screen of a smartphone device, either in person or over surveillance equipment. The numbers which are touched to enter a four digit PIN will give off residual heat that can be picked up in the image, with the amount of heat emitted corresponding to how frequently and how recently each button was pressed to enter the code. Thereafter, there it is a trivial matter of determining in which order to press the buttons in, which can even be done by simply trying every possibility with a four-digit PIN. Consequently, hackers can breach a mobile device simply through the use of a thermal image. Swipe-pattern based authentication, while it may seem more complex, is

at even greater risk according to that article, as the sliding of one's finger over the screen creates a much more obvious trail of hotspots. In order to test this, the Express group arranged an experiment in which thermal images of touch screens were provided to participants, who were then asked to predict the four digit PIN for the device. Figure 1 below illustrate how thermal imaging can reveal the password used by the user to the would-be hackers, and the predictions made by the participants.



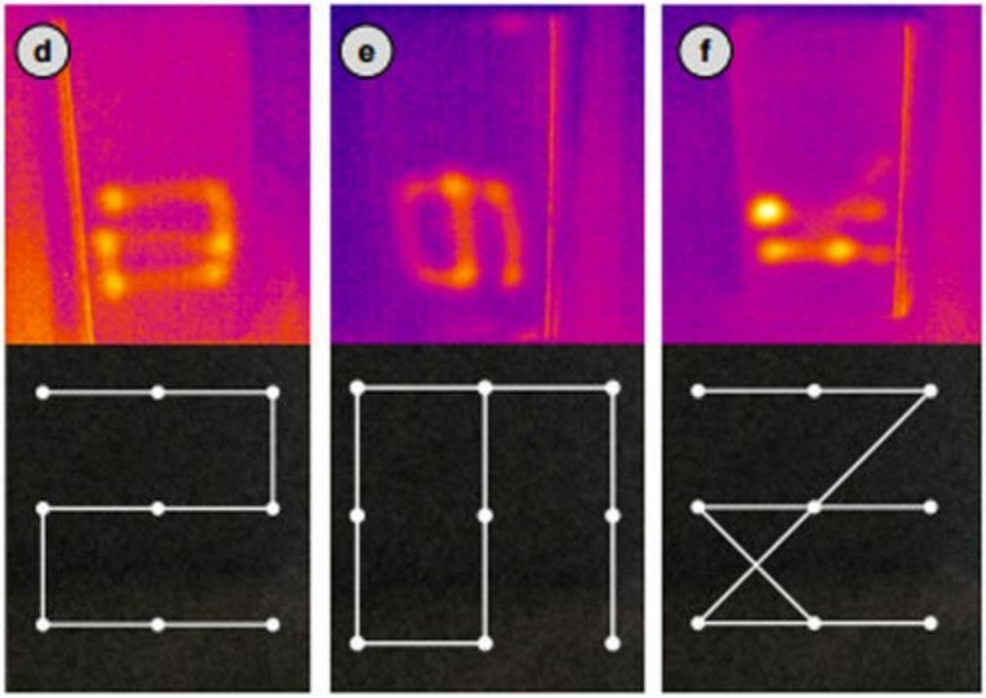


Figure 1: Password predictions made by participants when provided with thermal images of a phone. [19]

Passwords involving use of the touch screen also leave behind other forms of evidence besides heat signatures. In another study by Aviv et al [3], it was shown that the oils left behind by a finger on the touch screen can put a smart device at risk of being breached even without specialized equipment. Under certain light conditions, such as tilting the device on its side, residual oils left behind from taps and swipes become visible on the surface. Particularly visible patterns of taps or swipes are very likely to correspond to the password used to authenticate the device. In fact, the study performed by Aviv et al determined that passwords could be partially guessed in this manner 98% of the time, and fully discovered 68% of the time, under optimal lighting. Even under poor lighting conditions, these methods for guessing and full discovery still worked for a worrying 37% and 14% of attempts respectively. As a result of this unfortunate reality, even securing one's mobile device in the most popular and convenient authentication methods still leaves the user at

considerable risk of having their device breached. Figure 2 below illustrates the visibility of the residual oils from swipe data.

Figure 2: Residual oil patterns on smartphone screens are visible, even without optimal lighting [3]

A potentially more convenient alternative to physical biometric authentication methods, such as facial or fingerprint scanning, is behavior based authentication. Behavior based authentication is a system by which user behavior is observed by the device being accessed, and this information is used to authenticate them [10]. Mobile users who often forget their password would likely find use in an application which could accurately use their behavior, such as locations visited and apps used, to replace the need for their other forms of authentication, without storing data about their physical biometrics. Mobile users who store personal data on their phone would find use in an app that identifies intruders and forces them to manually authenticate; such a system could be used as a backup authentication method, and typically used in conjunction with more

secure methods such as a password or biometric, especially in situations with more pressing security concerns. Most importantly, by creating a system which allows users to be automatically authenticated based on their behavior, we hope that they might be more willing to use more secure forms of passwords or physical biometrics due to not having to enter them as frequently.

1.3 Other Methods of Smartphone Authentication

Even within the emerging field of behavior based authentication, there are still several factors to consider when selecting a method of authentication. When a mobile device needs to determine whether or not to authenticate a user, there are two different strategies to do so. The most common method is referred to as explicit authentication. To be authenticated by an explicit method, the user is prompted to enter a password, swipe a finger over a scanner, or enter some other form of identifying information in order to authenticate themselves [15]. A simple way to think of explicit authentication is to consider it a form of active authentication, which means to say that it requires manual input from the user of the device.

Implicit authentication, on the other hand, relies in the behavior of the user to determine whether or not they should be unauthenticated instead of the use of passwords, PIN numbers, or fingerprint data [15]. When using implicit authentication, the user is granted access to the device or feature until the authentication system detects behavior which contrasts sufficiently enough from the usual behavior that the system is no longer confident that the user is in fact the owner of the device. In contrast to explicit authentication, implicit authentication can be thought of as a passive method in that the user is not required to deliberately enter any input. Generally, implicit authentication is paired with explicit authentication by having the system fall back to explicit authentication if the user is deauthenticated, in case the true user's behavior changes suddenly for

some reason, but this is less secure than either method individually, as an attacker could access the device by cracking either.

Another key factor to consider when authenticating users is to decide when or how frequently a user is to be prompted to provide identifying information. On most computers and mobile devices that use explicit authentication, if the user needs to be authenticated, it is usually done each time the device has been shut down, put to sleep, or become inactive for a specified period of time. Under systems such as these, the device is considered to have episodic authentication: a form of authentication in which the user is required to re-authenticate periodically in order to continue using the device. When a user sets their mobile device to require a password input after a set time of inactivity, this is a form of authentication which is explicit, due to the fact that manual input is expected from the user, and episodic in that the user is periodically logged out and must be authenticated once more.

The goal behind implicitly authenticating users with their behavioral profiles is to make the process of authenticating continuous. To authenticate continuously as opposed to episodically, instead of requiring that the user login each time the device is used after a period of inactivity, the user remains authenticated until a situation arises which causes the user to no longer to be trusted, such as behaving in a way which does not match their behavioral profile. Our ideal use case is to authenticate users continuously and implicitly. More specifically, for as long as the user's recent behavior is found to be similar to their behavioral profile, the device will be authenticated with no input required, and the user will remain authenticated unless their behavior significantly changes.

1.4 Prior Work

In 2016, Donti et al. [8] created a behavioral model system, called BMS, for implicit mobile authentication. This system was designed to record location, WiFi SSIDs observed, device tilt, and application usage. They stored their data on a Google Drive server. From there, they used the simplest type of neural network, known as feedforward, to generate behavioral profiles for each user, based on the first six days of the prior week. They then used the trained network to generate scores indicating how likely the network thought the last day's worth of data was to be produced by the same user as the prior six days'. If this score was high, the user would hypothetically be authenticated automatically, and if not then they would be deauthenticated. To evaluate the effectiveness of the system, they also generated scores by running each trained network on each other person's latest data as well as that of the person it trained on [8]. In order to process this information, they built a supervised feed-forward artificial neural network (ANN) using TensorFlow [28]. Donti et al. used the WiFi BSSID, the WiFi level, and the timestamp of the recording as the features that were fed into the ANN to produce a trust score. In order to increase the accuracy of these trust scores, Donti et al. supervised their network by labeling the behavioral data with the correct expected output. For example, when data from a specific subject was collected, the data was labeled to be true for that particular subject, and false for all other users. Each time the behavioral data was collected, it was uploaded to a folder on Google Drive via a custom Android application. In order to be accessible by TensorFlow and any servers, the WiFi data was transmitted in JSON format. Once the data was stored in the Google Drive, it could then manually be transferred to a folder to be sent as input to the ANN generated via TensorFlow.

With access to their code, research, and reports, this project was meant to build off of this previous work. Our goal was to improve upon their system and hopefully reach the point where it could be used commercially alongside conventional forms of authentication.

1.5 The Goals of This MQP

In this MQP, we designed and implemented a system, called LOBS (which stood for Location, Orientation, Battery, and Screen, as those were the behavioral metrics we eventually decided to use), that authenticated or deauthenticated users continuously and implicitly.

Convenience was the primary goal of LOBS, since it was always intended to run alongside conventional means of authentication. If LOBS deauthenticated the user, the system would fall back to conventional explicit authentication methods, and it was therefore impossible for LOBS to increase security beyond what those means of authentication provide. This meant that our app would only come into play occasionally, and therefore its role was to provide a way for the user to sometimes not need to put in a password.

However, it was vitally important for LOBS not to compromise security. We aimed to achieve the same confidence of authentication as passwords provide before we allowed our app to actually replace them. Like our predecessor BMS, we chose to represent this sense of confidence as a percentage of how closely recent sensor data from the user matches the behavioral profile of the device owner [8].

In this paper, a system, LOBS, is described, researched and implemented, in which behavioral biometrics, such as phone tilt and location, are used to authenticate a user based on their behavior, falling back to other authentication methods if the user cannot be authenticated based on behavior. To use LOBS, a user only needed to install a mobile application, which would then gather

information from the built in sensors of the Android device. Once these data were sent to a server, the server could then generate a trust score by comparing the user's recent behavior against their historical behavioral profile, which was developed via deep learning. In order to build the user's historical behavioral profile, LOBS trained on user data gathered over an extended period of time. Future snapshots were compared to this behavioral profile, and a trust score generated which predicted how likely the current user wass to be the owner of the phone The app then authenticated the user if its confidence that the user is indeed the owner was above a certain threshold. In order for this application to work, it operated on the notion of trust. Whether or not the user would be trusted was determined by whether or not their data generated a trust score above or below a specified amount, or trust score threshold. A trust score is generated by a neural network such as the one used by BMS or LOBS to represent the likelihood of recent user test data matching the model from the training data set [8].

In this case, much like placing trust in another person, trust—in the case of our application—is the measure of whether or not LOBS was confident that the current user of the device was the owner of the device. In LOBS, trust is derived from a trust score. If the resulting trust score generated by comparing recent behavior seen by the device to previous behavior exceeded a certain threshold, then the user was trusted. If this was not the case, then the user was not trusted, and was de-authenticated. We used a trust score that ranges from 0 to 1, a number which represented the level of confidence LOBS has that the user is the owner of the device. In general, higher trust scores indicated higher trust that the data observed indeed belonged to the actual owner of the device or smartphone, with one being absolute confidence.

Using the work of Donti et al. [8] as the backbone of our project, we wanted to improve upon this application in several ways. To streamline the application workflow, we wanted to make

data upload and TensorFlow integration happen automatically. We also wanted to increase the number of metrics used to develop LOBS. Specifically, we wished to use GPS location, WiFi networks seen, device orientation, battery charge information, and screen mode (on/off) as our list of metrics. In doing so, we hoped to create a more integrated system that resulted in more accurate trust scores.

Our overarching goal was to investigate what metrics worked to authenticate users and to present results of such analyses. As part of determining how well our metrics worked, we also wanted to change the neural network hyperparameters of LOBS such the activation function used by the neural network, as well as the trust score threshold required for authentication. Modifying the Android operating system to actually use our app as a means of authentication was beyond the scope of the project, which can be pursued as future work.

Our goals can be summarized as follows:

- Automate the smartphone data gathering and uploading process used by Donti et al. [8]
- Re-architect and re-implement the smartphone data gatherer to use a more comprehensive third party library, in order to have access to more sensors without needing to implement each sensor ourselves
- Automate the process they used by which their neural network trains on the data and generates trust score predictions
- Conduct sufficiently large studies (longer time frame and more subjects) that we could evaluate the effectiveness of LOBS
- Optimize the accuracy and performance of LOBS by modifying its parameters

2. Literature Review

2.1 Summary

In order to evaluate which metrics would be the most useful to include in our implicit authentication app, in this chapter, we explored existing research on alternate methods of authentication without use of conventional passwords. In particular, we looked at research into constructing behavioral profiles based on data that can be gathered in the background, without the user needing to explicitly provide any information or input, since our goal was to create an app that could continuously authenticate users without interrupting their experiences to ask them for passwords.

2.2 Sensor Data Gathering Libraries

We looked into finding libraries that could collect various sensor data, since it would have been time-consuming to write the data-gathering code ourselves. We eventually found several possibilities, outlined below.

2.2.1 Funf

The first, Funf [12], was developed by MIT, and covers most, if not all, of the sensor data available on Android devices. It also had a built-in capability to save the data in the form of JSON files, which can then be uploaded to any given server. Thus, we could easily build a server that would receive that data and feed it to the neural network. However, the codebase for Funf had not been updated in over four years, so we needed to update the codebase before we could use it. Despite that, we eventually decided to use it to gather our data, as there were no others that would

be easier to integrate and Funf is straightforward to extend to use additional sensors. The sensors available in Funf are outlined below in Table 1.

Type of Sensor	Data retrieved	Why we used it (or not)
AccelerometerSensorProbe	Accelerometer data	Yes, the user's hand movements were part of the behavioral signature
AccountsProbe	Accounts data	No, because most phones only have one account
ActivityProbe	Steps walked	No, because it would have drained the battery too much
AndroidInfoProbe	Operating system information	No, because it doesn't change on any given phone
ApplicationsProbe	Applications being installed or uninstalled	No, because we didn't expect this to be a routine occurrence
AudioCaptureProbe	Audio data	No, because background applications can't access this anymore
BatteryProbe	Battery status	Yes, because how much the user uses the phone and how low the battery gets were part of the behavioral signature
BluetoothProbe	Data on nearby bluetooth devices	No, it would be used as a proxy for location, similar to WiFi in that it looks at nearby devices
BrowserBookmarksProbe	Browser bookmarks	No, phone bookmarks do not change frequently enough to be used for authentication.
BrowserSearchesProbe	Browser searches	No, because we couldn't access this anymore
CallLogProbe	Call logs	No, because it will have to wait for intruders to either make calls or not, and would

		only trigger LOBS if the true user usually does the opposite.
CellTowerProbe	Nearby cell tower information	No, because this was another location probe that worked outdoors
ContactProbe	Phone contacts	No, because we didn't expect this to change routinely
GravitySensorProbe	Gravity data	No, gravity changes very slightly by location, making this a proxy for location
GyroscopeSensorProbe	Gyroscope data	No, because we used orientation instead of angular acceleration, and the gyroscope sensor also tended to have a lot of random noise
HardwareInfoProbe	Device hardware information	No, because it didn't change for any given device
ImageCaptureProbe	Camera data	No, because background applications couldn't access this anymore
ImageMediaProbe	Stored image data	No, because it wasn't expected to change over a relatively short time
LightSensorProbe	External light data	No, because this only works if the camera is on, and background applications couldn't keep the camera on
MagneticFieldSensorProbe	Magnetic field data	Yes, because we needed it to calculate orientation
OrientationSensorProbe	Device orientation data	No, because this just calculated orientation based on accelerometer and magnetic field, so it had no new data
PressureSensorProbe	Air pressure data	No, this is a measure for air pressure which changes slightly by location, making it another location proxy

ProcessStatisticsProbe	Running process data	No, because background applications couldn't get this data anymore
ProximitySensorProbe	Sensor distance in cm	No, because usually nothing is close enough to pick up during normal phone usage
RotationVectorSensorProbe	Device rotation data	No, because it was derived from the gyroscope
RunningApplicationsProbe	Running application data	No, because the functionality this sensor relies on was deprecated and no longer returned the actual list of running applications
ScreenProbe	On/off state of screen	Yes, because when the user used their phone was part of the behavioral profile
ServicesProbe	Running services	No, because this was deprecated, similar to running applications
SimpleLocationProbe	An estimate of device location	No, because we used the more precise location estimate
SmsProbe	SMS data	No, much like phone calls, this relies on waiting for a thief to create usage patterns which do not match the user
TelephonyProbe	Mobile network information	No, this relies on a thief changing phone service providers
TemperatureSensorProbe	Temperature of environment or phone, depending on the device	No, device temperature is a proxy for battery usage, and external temperature trends are a proxy for location
TimeOffsetProbe	Time offset between device and major NTP servers	No, because this had nothing to do with the user
VideoCaptureProbe	Captured video data	No, like call and text history this relies on waiting for a thief to develop usage habits

		contrasting with the owner's
VideoMediaProbe	Stored video data	No, for the same reason as video capture
WifiProbe	Nearby WiFi network data	Yes, because the visible WiFi networks was part of the behavioral signature

Table 1: Sensors available in the Funf library and reasons for and against their use

2.2.2 AndroSensor

The second app we found for gathering smartphone sensor data, AndroSensor [11], also had not been updated for a few years. It supported several programming languages and could export the data to CSV files, but had fewer sensors than Funf (mostly related to location and orientation) and a less convenient uploading mechanism, so we found no reason to use it over Funf.

2.2.3 SensingKit

The third library, SensingKit [26], was the only one that had been updated within the last two years, so it would have been the easiest to integrate. It also supported iOS, which could have been a benefit in the future, although we were not planning to include that in LOBS. However, like AndroSensor, its sensors were mostly related to location and phone orientation, which was insufficient for some of the metrics we wanted to implement. Therefore, we once again decided to use Funf to gather sensor data on the smartphone.

2.3 Related Work

We also looked at several previous works to determine which combination of metrics to select for LOBS.

2.3.1 Lists of Metrics

One previous work that describes various possible metrics that can be used in behavioral authentication is 'Behavioral biometrics: a survey and classification' by Roman Yampolskiy and Venu Govindaraju [29]. As the title suggests, it surveyed all of the biometrics people had experimented with using for authentication at the time and listed the uniqueness and coverage of each one. It also classified them into groups based on what sort of data they gather about the user, e.g. physical biometrics like a fingerprint, passive usage patterns such as phone orientation and keystroke patterns (which are what we were primarily interested in using in our app), and more active usage patterns like app usage and call history (which we could also have used but decided not to since they are easier to imitate). We used this information primarily as a reference when trying to figure out what metrics were actually worth using from the list of types of data we can gather on Android phones.

2.3.2 Related Systems

Several previous works describe alternative authentication systems. One such system, ActivPass, is not an implicit authentication system, but is still useful for comparison to this system described in this paper. ActivPass is a system by which users can be authenticated based on questions about rare events in their recent activity, such as SMS history, described in Dandapat et al. [27]. While the method described showed some promise in being able to authenticate users

without needing passwords, our goal was to create an app that would authenticate users without requiring them to consciously provide any information at all, so the findings there were of little use.

Khan et al. [15] describes Itus, a framework for developers to build upon for behavioral biometric authentication. It provides a framework for various biometrics to be added for analysis on Android devices, to be used by Android developers, and machine learning algorithms for the training and analysis of these metrics. Unlike LOBS, Itus was an overarching framework for building authentication systems, not an authentication system itself.

2.3.3 Touchscreen, Keystroke, and Orientation Metrics

This section covers work with goals similar to ours, especially those which would authenticate users based on data gathered in the background. Several of these papers used touchscreen or orientation data as their metrics, described below.

The first of these, a system by Sitova et al. [30], was the paper describing a system called HMOG, which stands for hand movement, orientation, and grasp. As the name suggests, the system would use the accelerometer and gyroscope sensors built into the phone to measure tiny movements in how the phone was held, which generally result from how the person holds the phone. Upon reading this paper, we decided that the device orientation metrics they used would definitely be worth including in our app.

In Biedert et al. [4], touch screen data was collected from subjects using an Android app to read and compare images. While it was in use, the app collected various touchscreen related metrics, including touch pressure, trajectory, distance, orientation, and velocity. The data from these metrics were then used k-nearest-neighbors (kNN) and support-vector machine (SVM) neural nets, which resulted in 0-4% EER (Equal Error Rate) for authentication, with some outliers at 10%.

The EER is a metric for determining accuracy of behavioural authentication systems, where the False Rejection Rate (FRR) is equal to the False Acceptance Rate (FAR). The FRR is a measure of how many true users were rejected incorrectly, while the FAR is a measure of how many imposters were authenticated incorrectly. A lower EER means a higher accuracy (Cheng et al [7]).

While touchscreen data seemed promising at first, unfortunately, the Android operating system no longer allowed the required touchscreen data to be collected in the background by third party applications like ours, so this turned out to be impossible to implement in our app.

In Murmuria et al [23], touchscreen data was also used as a metric. The preliminary results showed that direction and duration of taps, as well as diameter of the finger, returned the best results for tapping data. For swiping data, direction, arc-length, pressure, speed, and finger diameter worked best as metrics. However, this was also largely useless for us, as we could not gather any of the data described in that paper either.

Bo et al. [5] used touchscreen data as well as device feedback from vibrations to determine when different users were using the device. The researchers assigned a trust score to the likelihood of the device having changed hands. This trust score increased with multiple consistent judgements. While the researchers were able to show 100% accuracy for 5 to 10 consecutive observations, they do not consider external noise, which would be important in real-world applications.

In Crawford et al. [13], two different behavioral biometrics, specifically voice patterns and keystrokes, were used to verify users on an Apple iPhone. This data was gathered in the background. The keystroke metrics had an EER of less than 10% and an FRR of around 2.5%. The keystroke data is fairly unique, but some users in this study had concerns about their keystroke data being used elsewhere. Voice patterns had an EER of 25% in Crawford et al. [13], but had an EER

of 10-12% in previous papers. We decided to not use voice as a metric, as we couldn't gather voice data with the Android permissions that we had access to.

In Kwapisz et al. [16], accelerometer data from 36 users was used as a metric for authentication. Each subject was given an Android smartphone and told to walk, jog, and climb up and down stairs for specific time periods with the phone in their front pants pocket. The results had a wide range of accuracy, from 42% to 100% between the 36 users. This wide range of accuracy suggests that gait-based authentication may not be the best metric to use, especially given potential battery drain due to constant accelerometer sensor use. As such, LOBS did not use gait as a metric.

2.3.4 Location Metrics

We found several papers describing authenticating users based on the user's location history. Donti et al. [8] was the paper written by the group that was working on the same MQP last year, so we started with that. From their paper, we learned that timestamped location data (gathered using visible WiFi networks while inside and using either GPS or visible cell tower signals while outside) is an easy to gather and fairly effective metric for constructing users' personality profiles, although not sufficient on its own to distinguish people. They also looked at the orientation of the phone while it was in use and the apps that were used, but although both of those metrics were possible to gather and construct profiles from, neither proved as useful as location for distinguishing the four people they used in their study. They used TensorFlow [28] to construct the profiles from the raw user data, consisting of GPS location, WiFi networks seen, device orientation, and app usage, and then test the effectiveness of the authentication by producing trust scores based on new data from each participant.

Their code was loosely based on the TensorFlow tutorial designed to work with Boston housing data [9], but they modified the input function and features to accommodate their data set. They created a script to set up the neural network with attributes for each sensor type, in order to have each sensor type easily accessible to the neural network as a whole. The feed-forward neural network was fed the data as its input function, with the training output stored in a pickle file, a way of storing Python objects efficiently via serialization [21], in each iteration. Once the training set was stored in a pickle file, the neural network produced a trust score based on the test data they provided. They used the last week's worth of data for each user, with the first six days being training data and the last day being used for testing.

Since we had access to all the code of the Donti et al. [8] MQP, we decided that reusing some of their TensorFlow code would be the most straightforward way to test for authentication. However, we followed a suggestion from our advisor to use a more complete and extensible third-party library rather than trying to reuse their data-gathering code. We used version 1.3.0 of TensorFlow for our study.

In Rachuri et al. [14], various behavioral biometrics were used by a Naive Bayes classifier to determine suggestions for everyday user activities, such as cooking dinner or waiting for a bus, for a social check-in app. These metrics included accelerometer, GPS, WiFi, application usage, calls, SMS messages, battery, and network information. This resulted in a false positive rate of less than 3%. However, it was determined that the battery cost of physical sensors, such as GPS, was significantly higher than soft sensors, such as application usage. Not using a GPS metric resulted in a 6% loss of accuracy, but a significant increase in battery life. Since these papers generally agreed that location is an effective metric, we decided that it was worth using in LOBS.

2.3.5 History Content Metrics

Another group of metrics is history content: text messages, phone history, and app usage. In Fridman et al. [17], stylometry (literary style), web browsing history, app usage, and location were used as metrics for authentication, in part due to the low battery usage involved in collecting them. Location was determined via both GPS and WiFi. The experiment resulted in an EER between 1 and 5%.

One of the relevant papers found by last year's group was the paper "Soft Authentication with Low-Cost Signatures" by Bupithya et al. [25], which uses data that the phone already gathers in the background, such as SMS history and location (via cell tower placement), to detect anomalies in users' behavior. While they were able to detect anomalies with a higher degree of accuracy than previous methods that only looked at the GPS signal, this method could not detect anomalies without at least half an hour and preferably over an hour of anomalous data to analyze: it achieved 42% coverage (the fraction of the time it could confidently say whether or not the data was anomalous) with an hour's worth of data. However, our initial goal was to use metrics that could detect anomalies within half an hour, in order to minimize the amount of time the attacker had to do damage before being locked out. While the metrics used in Bupithya et al. [24] were not within this threshold, they were still useful.

In Shi et al. [10], SMS, phone call, and browser history, together with location data, were used on 50 Android devices to authenticate the owner of the device and distinguish them from an impostor. The users in this study could typically use their device about 100 times before suffering an incorrectly failed authentication, whereas an adversary had a 50% probability of being locked out after two uses of the device, and a 95% chance after 16 uses. The SMS, phone call, and browser history data were hashed with a key generated on app install and stored only on the device to

protect user privacy. We decided against using any of these history content metrics, since none of them are implemented in Funf [12].

2.3.6 Battery Metrics

A few papers recommended using battery usage as a metric. In Murmuria et al. [22], four behavioral biometrics, specifically battery usage, movement, apps usage, and touchscreen data, were used to authenticate a group of 73 subjects using Facebook and Google Chrome on Android devices. Two machine learning algorithms were used to analyze the data: Strangeness-based Outlier Detection (StrOUD) and the Discord Algorithm. The experiment obtained an EER between 6.9% and 16.9%.

In Zhang et al. [31], researchers monitored only power consumption of applications used on HTC smartphones. They used this data to form a model of the users of the phone, an effort which resulted in an average error rate of 4.1%.

We eventually decided to use battery usage as one of our metrics, as Funf had a sensor to gather it, and the systems that used it achieved sufficiently low error rates that it was evidently worth including.

3. Methods

3.1 System Architecture

The system we designed that authenticated users based on their behavior is outlined below. First, a data collecting service on the device continuously collected behavioral biometric data from the device. This data was then sent periodically to a server, where a machine learning algorithm trained on the given user's data. The device could send a trust score request to the server, which sent a predicted trust score back to the device indicating how likely the user most recently using the device was the one who usually uses the device. A graphical overview of LOBS architecture is outlined in Figure 3 below.

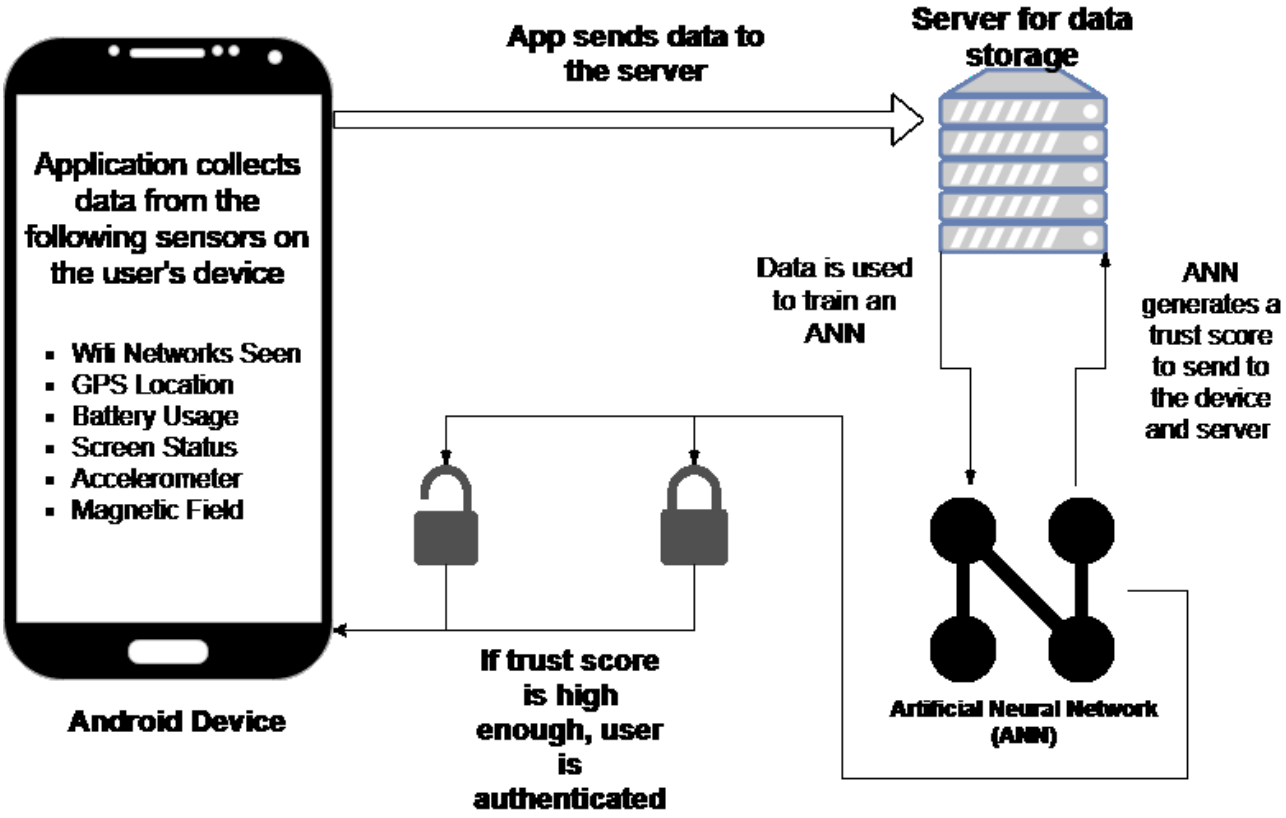


Figure 3: The high-level architecture of LOBS.

3.2 Selecting Metrics and Establishing Criteria for the Application

After examining the prior work of other researchers who have attempted to make applications which serve similar purposes, we came up with a list of criteria for the metrics that we would be using in our app.

Ideally, all metrics recorded by our application would gather information from sensors which are constantly in use by Android devices during normal operation. By developing the behavioral profiles of users using metrics that are constantly being recorded by their device already, our application would not only be less invasive of the user's privacy, but also present minimal burden on the user and put little to no additional strain on the battery life of the device.

Secondly, while we intended to use as many metrics as possible to develop behavioral profiles for our users, we speculated that using every metric available would not only be redundant, but would also make the application less convenient: the more data that gets fed to our neural network for machine learning, the longer it would take to train the network to develop a user profile with confidence, and the longer it would take to actually determine a trust score for the user. For example, the Android operating system includes sensors for GPS, WiFi networks seen, and Cell Tower location alongside many others. Each of these sensors can be used to determine the location of the user with varying degrees of accuracy, but to use them all together would result in more data being sent to our neural network, which would slow down the training process in exchange for recording potentially overlapping information. In order to avoid redundancy, we opted to use sensors that gathered different types of data.

Finally, out of all of the metrics initially considered for the applications, some could not be used due to limitations of third-party applications on Android devices. Based on findings in prior work, we initially intended keystrokes and swipe data recorded from the user to be examined by our application, because of their uniqueness from individual to individual. However, Android devices only provide this data to applications which run in the foreground. While this was technically a possibility for our application, requiring our app to run constantly in the foreground would have violated our goal of developing a convenient alternative to passwords. The sensor types we used are as follows: WiFi networks seen, GPS location, battery usage, screen mode (on or off), and device orientation (using accelerometer and magnetic field sensors as a proxy for orientation). All of these sensors could run in the background, which made them more ideal for LOBS.

Once our metrics had been selected, we moved on to look at the means available to gather information from Android sensors. We had already decided that TensorFlow would be the library that we would use for building the neural network that our application would use, so we then needed a way to streamline the process of sending sensor information to the neural network. As it turned out, Funf [12] already had a built-in mechanism to convert the data to JSON and send it to a server, so we only needed to build a server that would forward the data to our TensorFlow application and upload the data to there. The code that used TensorFlow was written in Python. We used the metrics listed in Table 2 as inputs to our neural network in order to correctly authenticate a genuine smartphone user against an impostor.

Metric	Pros	Cons
WiFi	Easy to measure location while indoors	If user's phone is stolen, the phone would automatically connect anyway as the thief and the user would initially be

		in the same location, rendering this useless. Could also circumvent simply by being near where the person usually uses their phone, depending on how it is set up. This would be true for any location metric. Could be confused by people who are in similar places at similar times i.e. class, work.
GPS	Relatively precise location detection that works almost anywhere	Very battery intensive, also could be confused by mimicking the phone's owner
Orientation	Difficult to impersonate as it's largely an unconscious behavior	Information content was reduced when phone is used while placed on a surface instead of in the user's hand
Screen Status	No or low battery cost	Only records data when screen is turned on/off, which many users do not do often
Battery Usage	Always tracked and surprisingly accurate (95.9%) Zhang et al. [31]	Battery life degrades over time, and user can change habits

Table 2: Selected Metrics for the application and reasons for and against their use

3.3 Modification of Prior Software

We began by copying the TensorFlow code written by Donti et al. [8] to a WPI research cluster. We then modified the code to run with the Funf sensor data that would be sent to the cluster via the previously mentioned Node.js server. The provided code already had the functionality of using sensor data to build a behavioral profile of users based on which WiFi SSIDs were seen by the sensors. With these profiles the code was capable of creating a trust score by evaluating recently

collected data against the behavioral profile of the user generated over the course of the training period.

Additionally, in order to construct a user behavioral profile, the provided code for BMS also had the functionality of being able to convert the WiFi SSID data recorded from the user to a form that could be used to train the neural network, and the other sensors could use the raw data. The strategy employed by the previous group consisted of training the neural network on user data collected over the course of a week. Once the training period was complete, the network used in the application would then be able to generate a trust score in the form of a percent value of how closely recent data matched the training data. With a trust score generated, the application was then able to compare that score to a threshold in order to determine whether or not the user of the device was trusted to be the owner of the device.

Initially upon receiving code for BMS, there was no threshold set in place, but selecting a value for the purpose of our application was a simple process. Based on the research of other similar applications, we decided to start with a threshold of a ninety percent trust score or higher, and from there we needed only to weigh the security benefits of having a greater threshold against the inconvenience of potentially deauthenticating an actual user with a lower-than-usual trust score. For example, while our application would ideally not authenticate an impostor who takes a Massachusetts resident's mobile device to New York City, it would also be ideal to not lock out that Massachusetts resident should they take a day trip to New York City. We decided, however, to err on the side of security, and set the trust score threshold to something higher than we initially thought necessary so that it would lock out people doing unusual things like that even if they likely were the real owner. This was not a problem because the actual owner of the device could regain

access by entering their usual password if they happened to be locked out. We determined the value of the threshold to use with the data from our initial study.

3.4 Repairing Funf

Perhaps the largest influence of the selections of metrics for the application was data could be gathered using the Funf library [12]. The library itself had code prepared with the ability to read data from more than thirty sensors within Android devices. Each sensor on the list was therefore considered for our application, and each metric that fit our criteria, as stated in the previous section, ended up being used in the final product. However, the developers of Funf had stopped maintaining the library in 2013, while our application was being developed in 2017. In order to make use of the library, it had to be updated. Fortunately for us, Android always maintained backward compatibility of its hardware features, so most of the code in the library still worked. However, within the last four years, Google had created an IDE (integrated development environment, or an application that helps you write and then compiles code) called Android Studio specifically for Android development, and all Android development had to be done using that by the time we created our app. Since Funf had stopped being updated before Android Studio existed, the project was structured in a way that was compatible with Eclipse, the IDE that was used for Android development at the time, but not with Android Studio. Therefore, the one change that we did have to make to Funf was modifying the project structure to be compatible with Android Studio.

In order to make it easy to migrate projects from Eclipse, Android Studio used the same programming language and file organization system as Eclipse did, so that the only change that had to be made in order for projects made in Eclipse to work in Android Studio was the addition of several configuration files for its new build system. Specifically, Android Studio uses the Gradle

build system, which looks at a set of predetermined configuration files to find some information it needs to build the app. This included information like the minimum and targeted versions of Android, the dependencies the app had and what versions of those should be used, and the location of the source files. In order to migrate Funf to Android Studio, we essentially had to create the files necessary for Gradle to build it.

The Gradle build system required several configuration files to be present in the project in order to run, which Funf did not have. While these files can be lengthy, most of the contents are the same for almost all projects. Android Studio creates these files whenever a new project is made, so we started by creating a blank project for Funf and copying the existing code into it, and then figuring out what needed to be changed from the automatically generated Gradle files. As it turned out, the default settings were almost all correct; the only thing that had to be added was the third-party dependencies that Funf itself required.

We ran into one unexpected problem while adding Funf's dependencies to the Gradle files. The Gradle file that determined dependencies had to contain a list of libraries by package name, and if no version number is provided, it defaulted to the latest one. By looking at the compilation errors in the Funf code with no dependencies provided, we determined that it only had two external dependencies, one of which was the Android support library necessary to ensure that the old sensor code still worked, and the other of which was Google's Gson library. The most recent version of the Android support library worked, since the main function of the library was to allow backwards compatibility, but the most recent version of Gson did not.

Rather than try to update Funf to use the most recent versions of Gson, which would have required us to dive deep into the Funf source code in order to figure out what calls no longer worked and how to fix them, we simply decided to use that Gradle file to set the version of Gson to

the version that Funf would have used when it was in development. At first, we assumed that that would be the latest version released before Funf development stopped. Setting the version only required adding a version number to the Gradle file, but that version did not turn out to be the correct one either.

We ended up determining the correct version of Gson by trial and error: since the most recent version of Gson released before Funf development stopped didn't work, we tried the prior version, and when that also didn't work we tried the one before that, and so on until it worked. While this was time-consuming, we eventually found a version that the Funf code compiled with, and set the Gradle file to always use that version in the future so that later updates to Gson would not cause the updated Funf to break again. Once that was done, we could start using Funf as a library in our project.

The data produced by the Funf-based data collector application was uploaded in the JSON format, with each sensor having its own fields, as shown in Table 3.

Sensor Type	Example JSON Data Sent By Funf	Features Used By Neural Net
WiFi Sensor Data	{"BSSID": "e0:88:5d:9a:0c:7d", "BSSLoadElement": "0500610000", "ChannelMode": "11n_HT20(45.0)", "SSID": "HOME-0C77-2.4", "SamsungVsie": "null", "autoJoinStatus": 0, "blackListTimestamp": 0, "capabilities":"[WPA-PSK-CCMP+TKIP][WPA2-PSK-CCMP+TKIP][WPS-PBC][ESS][BLE]", "centerFreq0": 0, "centerFreq1": 0, "channelWidth": 0, "distanceCm": -1, "distanceSdCm": -1, "Flags": 0, "Frequency": 2437, "is5GHzVsi": false,	BSSID, distanceCm, operatorFriendlyName, venueName

	"is80211McRTTResponder": false, "isAutoJoinCandidate": 0," Level": -30, "numConnection": 0, "numIpConfigFailures": 0, "numUsage": 0, "operatorFriendlyName": "" ,"seen": 0, "Timestamp": 1510733136.151,"tsf":1124860494, "Untrusted": false, "venueName": "", "wifiSsid": {"octets":{"buf":[72,79,77,69,45,48,67,55,55,45,50,46,52,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0], "Count": 13}}} }	
Accelerometer and Magnetic Field Sensors (Orientation)	{"accuracy": 3, "Timestamp": 1510733197.866043, "X": -0.7278373, "Y": 7.469909, "Z": 9.481039}	Timestamp, x, y, z
GPS Location	{"mAccuracy": 10.0, "mAltitude": 233.0, "mBearing": 0.0, "mElapsedRealtimeNanos": 1133051794991, "mExtras": {"satellites":8}, "mHasAccuracy": true, "mHasAltitude": true, "mHasBearing": false, "mHasSpeed": true, "mIsFromMockProvider": false, "mLatitude": 39.79432262, "mLongitude": -85.77671905, "mProvider": "gps", "mSpeed": 0.0, "mTime": 1510733145000, "Timestamp": 1510733144.40}	mLongitude, mLatitude, mAccuracy, mBearing, mElapsedRealtimeNanos, mTime
Battery Sensor	{"capacity": 280000, "Charge_type": 0, "Current_avg": -13, "Current_now": 0, "Health": 2, "Hv_charger": false,	Timestamp, Level

	"Icon-small": 17303412, "Invalid_charger": 0, "Level": 100, "Max_charging_current": 0, "Online": 6, "Plugged": 2, "Pogo_plugged": 0, "Power_sharing": false, "Present": true, "Scale": 100, "Self_discharging": false, "Status": 5, "Technology": "Li-ion", "Temperature" :284, "Timestamp" :1512328712.715, "Voltage": 4335}	
Screen Sensor	{"screenOn":true, "timestamp":1516648604.32}	Timestamp, screenOn

Table 3: Sensor types and features used and examples of their JSON data

3.5 Initial Data Gathering Study

In order to test LOBS, we designed two separate studies. The goal of the initial study was to find a threshold for how many days worth of data would need to be examined from any given user before the application began to return consistent trust scores when a sample of said user's data was compared to their behavioral profile.

In this study, we pushed the Android app we had developed onto the Google Play store as an open beta, then asked friends and family to forward the link to the app to people they might know. We were able to recruit fifty participants from coworkers of family and friends to download the app this way, and acquired nearly 8 gigabytes of data over the course of a single night. When running this data through the training and prediction algorithms it became apparent that the additional data those functions generated filled up several terabytes worth of memory, and that our initial setup for storing data was inefficient. If the data was going to be properly analyzed for the purpose of the

study, LOBS would have to train and run predictions on one user at a time. In order to determine just how many days of data would be needed, a Python script was created which would compare samples of each user's data against several behavioral profiles for that user. For example, the first behavioral profile consisted of a full day of data from the user, the second consisting of two full days, continuing up to two weeks of data.

However, since all the participants in this study were recruited by word of mouth and often secondhand, participants did not know how to keep the data collector running after it was installed, so none of them successfully collected two full weeks' worth of data. In fact, most participants had less than a day of data, so the script varying the amount of training for them produced no results. We eventually determined, using data from two of the researchers, that two weeks' worth of data was the maximum that LOBS could train and predict for in reasonable time, but that was too small a sample size for us to determine how the accuracy of the system was affected.

3.6 Post Study Changes

While conducting the first study, several issues were discovered with that iteration of the data collection application. The first issue was observed during the initial attempts at running the training and predicting algorithms on the data from the fifty initial study subjects: early in the process, the storage of the virtual machine we were using in the WPI research cluster was completely full of the resulting data. After investigating where all of the extra data came from, it became obvious that the recently added accelerometer and magnetic field sensors were being sampled more frequently than the other sensors and generating far too much data. While the other sensors polled every twenty to thirty minutes, the accelerometer was polling five times each second for the first minute out of every hour. When compared to the other data collected by the application, the vast majority of it came from the accelerometer.

The server becoming full due to this abundance of data was not the only consequence of this overly frequent polling. While LOBS was being tested, the device running it would noticeably rise in temperature whenever the device was beginning to poll the accelerometer and magnetic field data. Additionally, LOBS was reported to draw a substantial amount of battery power. As a result of the extremely rapid sampling, it was drawing ten to a hundred times more power than other applications on the device. Even beyond the issues of storage and battery consumption, users without unlimited data available to their mobile devices were at risk of running out of data. Due to LOBS periodically uploading the information polled from the sensors to our servers, it required the use of network data whenever the user was not connected to a WiFi network. Whenever the application began to sample the new data and send it to the server each hour, if the user was not on WiFi, they would use up their cellular data allotment very quickly. Even if the user was on WiFi most of the time, the original version of the app could easily use hundreds of megabytes of mobile data in less than a day in a few data uploads that happened to occur while the user wasn't connected. In order to recruit more participants to future studies, keep the interest of the current participants, maintain convenience of the application, and improve its scalability, something needed to be done to fix these issues.

The first step in doing so was to adjust the intervals at which the sensor data recorded by the data gathering application sent the information to the servers. However, in order to ensure that intruders could be detected by the device relatively quickly, we attempted to avoid waiting longer than half an hour between each upload. Also, in order to reduce the data usage of the application, LOBS was modified to first check if the device was connected to a WiFi network before uploading data to the server. In the event that the device was not connected to the Internet when it was

scheduled to upload its sensor data, the application would simply discard the data, as we found that we could gather plenty of data that way anyway.

Then, in order to make LOBS more efficient, we wrote a Python script to convert the multi-file data format we already had to a single-file-per-subject format. Instead of having each individual data point sent from a subject's device stored in a separate file, which is what we did in our original version, we stored all data points for each subject in a single array in a single file, with each index in the array being a JSON object containing a single data point. Once we had converted the data from the initial study to the new format, we determined that the new data system was able to train at an average rate of one data point every 0.296 seconds, which was an improvement to performance by roughly a factor of four. For example, our initial trial only made it through testing the information of 3 users after running for two weeks; however, after changing to the new format for the information in order to perform less file I/O, the same task was accomplished for all subjects in a matter of hours. With this change, the amount of time to process data increased in a linear relationship with the amount of data points to process. This relation is depicted below in Figure 4.

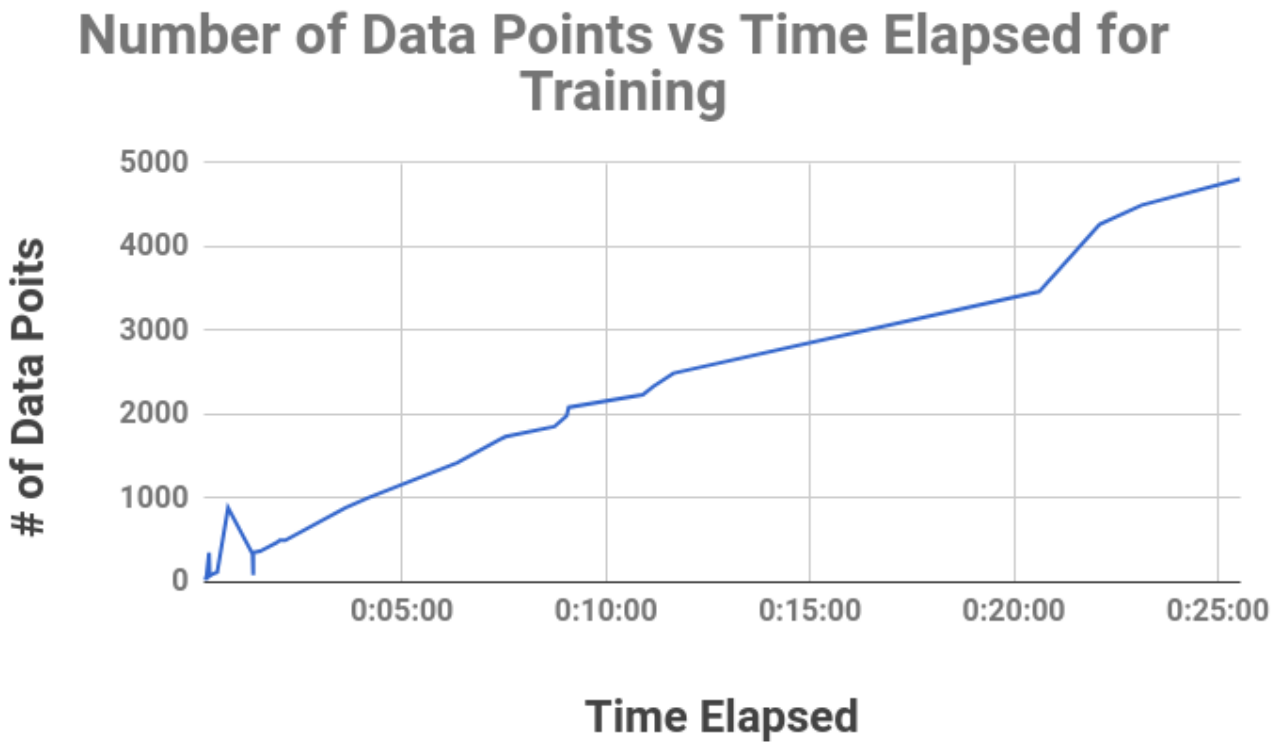


Figure 4: Training duration versus data points after minimizing file I/O.

After converting the data format of LOBS, we modified the activation function of the neural network from Sigmoid to TanH. This decision was motivated by the popularity of the TanH function in neural network applications. In order to get an estimate as to whether this change would be beneficial to our particular neural network, we compared the training time of several test neural networks using Sigmoid and TanH provided by an online test suite by the name of "a Neural Network Playground" [6]. Unfortunately, LOBS was unable to train quickly enough to adjust parameters as often as desired, so we used the online sandbox to develop an educated guess as to the accuracy and time efficiency of training a neural network with our intended parameters. In all cases, running networks with TanH resulted in greater accuracy, and results were achieved in roughly one third of the time [6]. After this change in the activation function was implemented, the efficiency of the neural net improved again to training at an average rate of one data point every

0.171 seconds. These training times were measured on a RedHat virtual machine with 11 gigabytes of RAM and a four-core Intel processor with 2097.570 mHz speeds. These training times are depicted below in Figure 5.

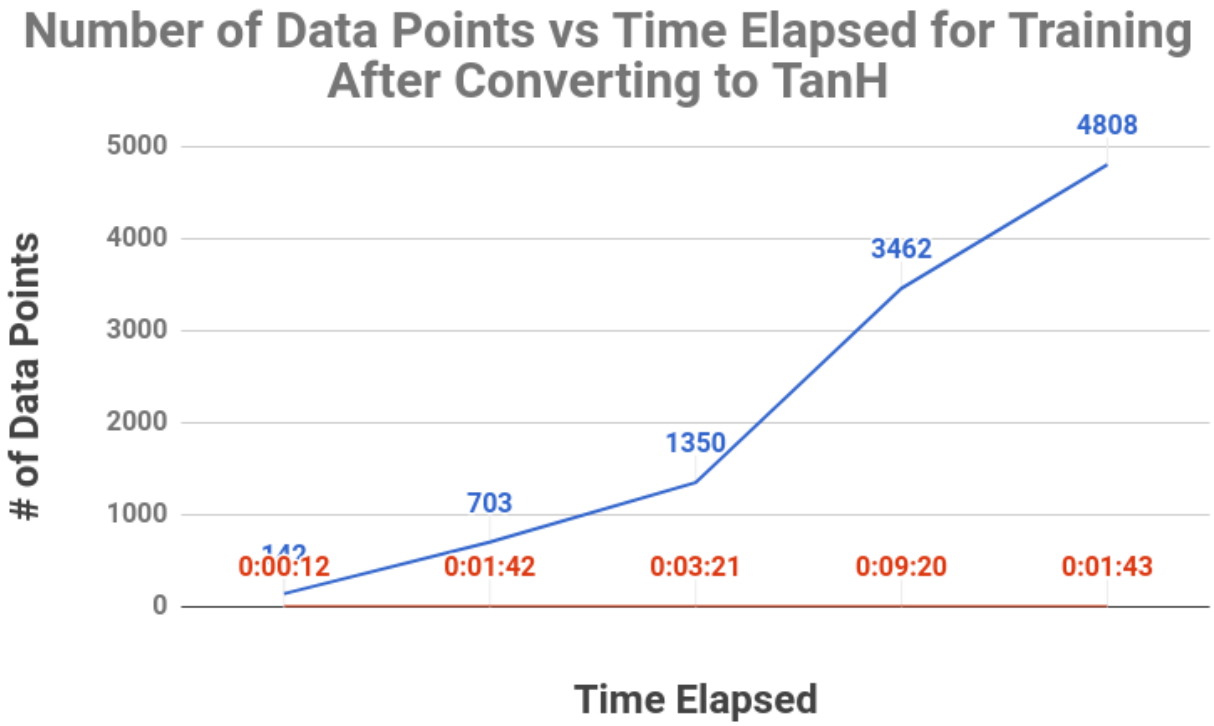


Figure 5: Training duration versus data points after minimizing file I/O and converting activation function to TanH.

After converting the neural net to use TanH as the activation function, we decided to try to improve other hyperparameters of the neural net. The two parameters we modified were the size of the two hidden layers in the neural network and its learning rate. In order to determine the optimal values for these parameters, we wrote and ran a script to randomly select fifty combinations of parameters and then train and evaluate the loss on a particular subject's data. We would have liked to test more in order to get more precise results, but did not have time, as it was very time-consuming to evaluate the different parameters. In order to ensure that the effects of changes in hyperparameters were not confounded with those from differences in the data, we picked a subject

before making any changes and used that subject to evaluate all of them. We picked a subject that had enough data to get meaningful results but not so much that the network would train very slowly, in order to try as many combinations of parameters as possible in the time we had. The range for the randomly selected sizes of hidden layers was two layers of between two and sixteen nodes each. The range of values investigated for the learning rate was between 10^{-6} and 10^{-2}. The script determined that the best combination of parameters was seven nodes per layer and a learning rate of 0.0082. The shape of the new feedforward neural network with seven nodes per layer, instead of the initial eight, is displayed in Figure 6 below.

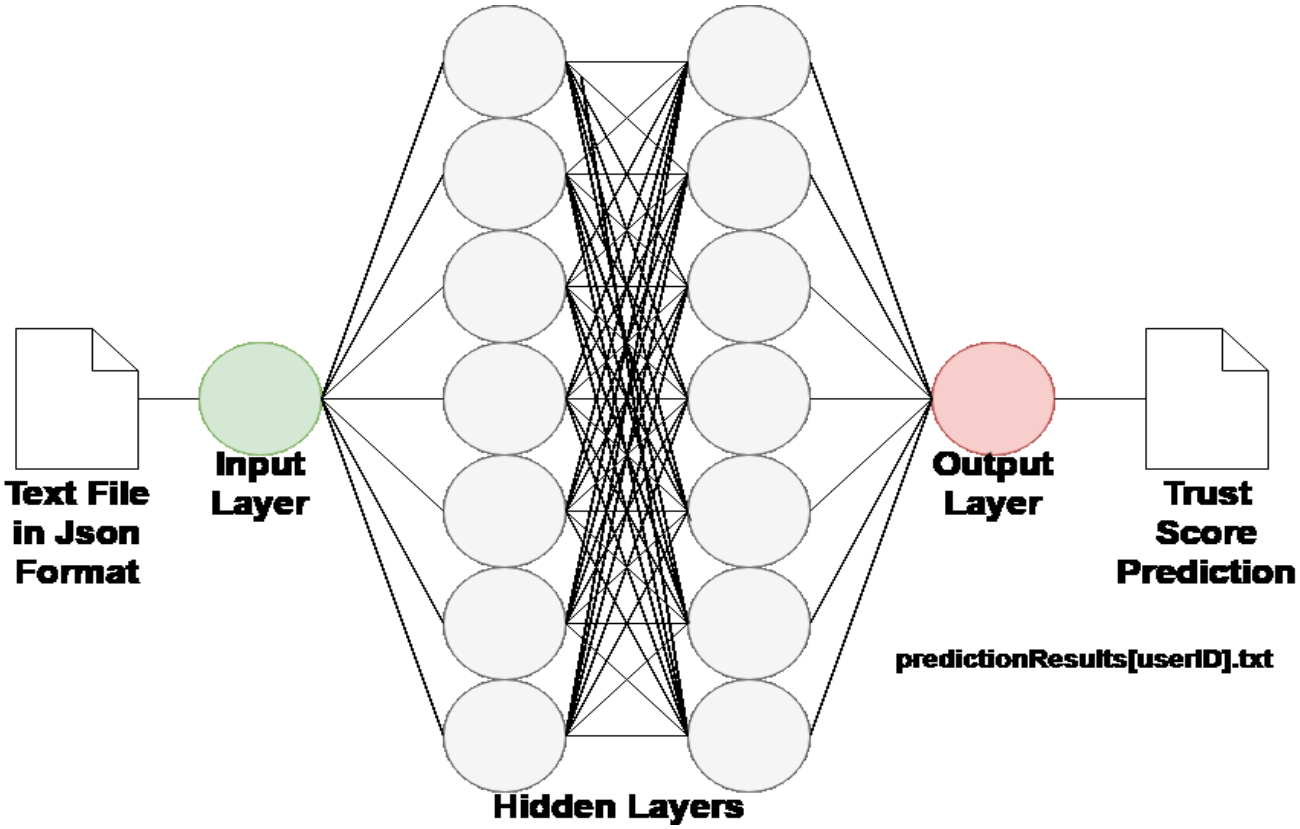


Figure 6: The reduced size, seven nodes per layer shape of the new neural network.

3.7 SONA Study

In our second study, we posted our app as a study on WPI's SONA system, in which students can participate in studies for credit. The participants in the study were students that signed up through that system, and were given credit for their participation once we got enough data from them.

In this study, we tried to collect some additional personal information about the participants: age, gender, and occupation. We did this on the basis that this information may help provide insights on which metrics work best for accurately authenticating different groups of individuals. The personal information was anonymized and linked to the gathered data of the participant. However, no participants recruited by word of mouth filled out the form where they provided that information. The participants recruited by the SONA study did fill out the form, but the entire demographic listed their occupation as student, and their age ranging from 19-22, so we decided against trying to draw any conclusions from it.

As with the first study, once installed, the Android app collected and sent data to our server on the WPI research cluster. Once two weeks of data were gathered from the subjects, we ran the trust score prediction Python scripts previously mentioned on their data to generate trust scores for analysis.

4. Implementation

Once we had decided on a sensor data-gathering library (Funf) and a machine learning library (TensorFlow), we set up the Node.js server to receive data from the Funf-based application on users' devices and pass it to the TensorFlow library. We had three paths on the server, one of which received the data, a second which ran the TensorFlow prediction algorithm and sent back a trust score, and a third which collected anonymous data about the subject for data analysis (age, gender, and occupation). The three paths defined in the Node.js server are outlined in Table 4 below.

Server Path	Description	Example JSON
/getData	Received Funf data and wrote it to a file.	{"datetime": "Nov0919:29:22017", "data": { "Accuracy": 3, "Timestamp": 1510284566.034270, "X": 10.002563, "y" :0.24931335, "z": 0.87023926}}
/getTrustScore	Ran the TensorFlow prediction algorithm and sent back a trust score.	{"response": "Data Read Successfully.", "trustScore": 0.993, "trusted":trusted}
/writeAdditionalInfo	Collected anonymous data about the subject for data analysis.	{"Age": "21", "Occupation": "Student", "Gender": "Male"}

Table 4: The three paths on the Node.js server and example JSON for each path

The first path received as its parameters a unique encrypted device UUID, the date and time on which the data was sent, and the sensor data generated by Funf in JSON format.

It then wrote that data as a new entry in a JSON array contained in a JSON file associated with the UUID of the subject. The second path took in the device UUID as a parameter, and ran the TensorFlow prediction algorithm using the data for that UUID as input. This algorithm returned a trust score, which the Node.js server then sent back to the Android device, along with whether or not the user of the device met the threshold. The third path took in the user input provided in the Android application, along with the device's UUID, and then wrote that information in a JSON file on the server. Figure 7 below illustrates the workflow between the three paths on the Node.js server and the Android application.

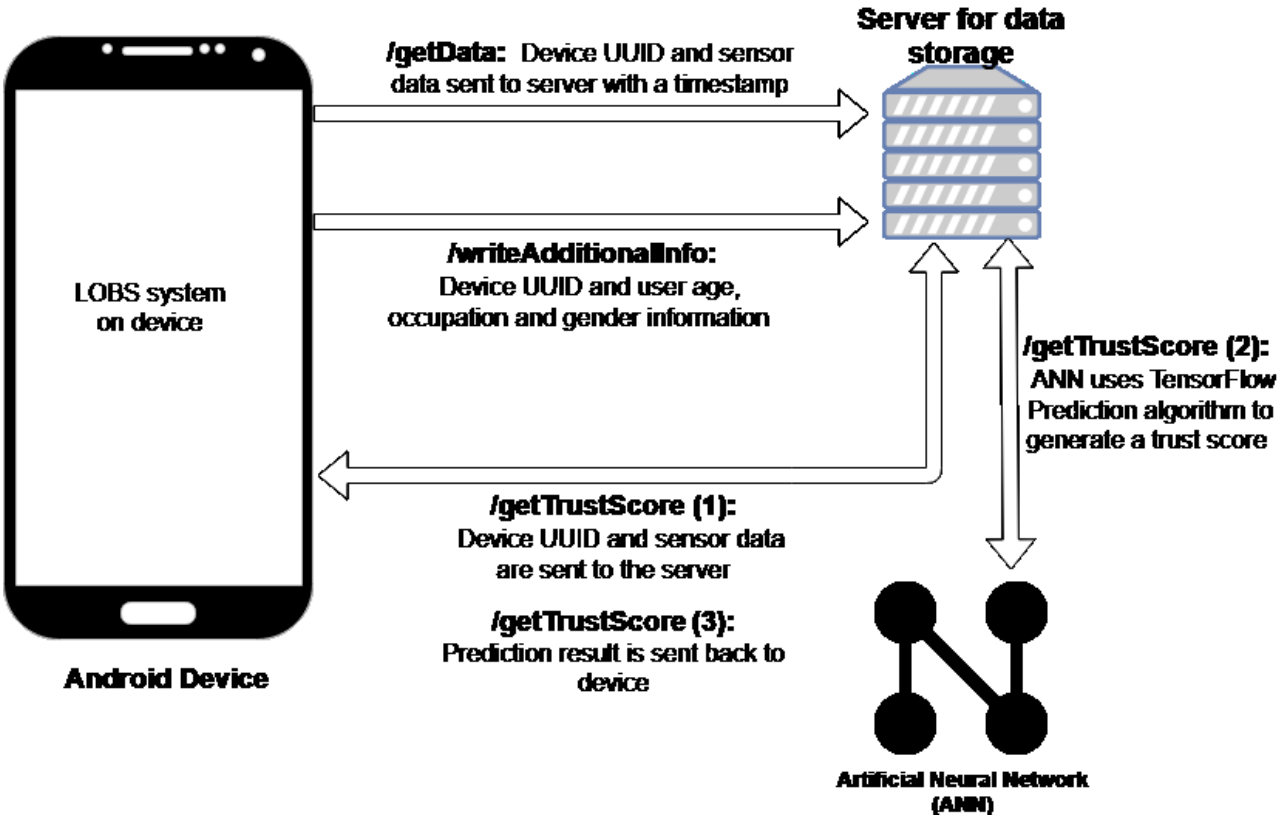


Figure 7: The workflow between the paths on the server and the Android application.

Once we updated Funf to work with current Android systems, we created a simple Android application, using Funf as a dependency, which periodically sent the data Funf collected to the Node.js server. We then modified the TensorFlow code to work with this data, and used it to

compute prediction trust scores for the data. A separate Python file was set up for each sensor type, with TensorFlow code to train and predict for that sensor, in order to separate the TensorFlow training by sensor type. To train the TensorFlow neural net, the training algorithm was run on each sensor separately with sample data gathered for that user in the past.

The original neural net we used had two layers and eight nodes per layer. It took in JSON files containing data sent from Android devices using Funf as input, then trained on those files individually for each sensor type. Once trained, the neural net used as input all JSON data for the appropriate sensor taken from the date range specified for prediction (one to fourteen days), with the training data being the first three quarters of that and the test data being the last quarter, then computed a trust score for each sensor type. The original layout of the neural network with eight nodes per layer is outlined in Figure 8 below.

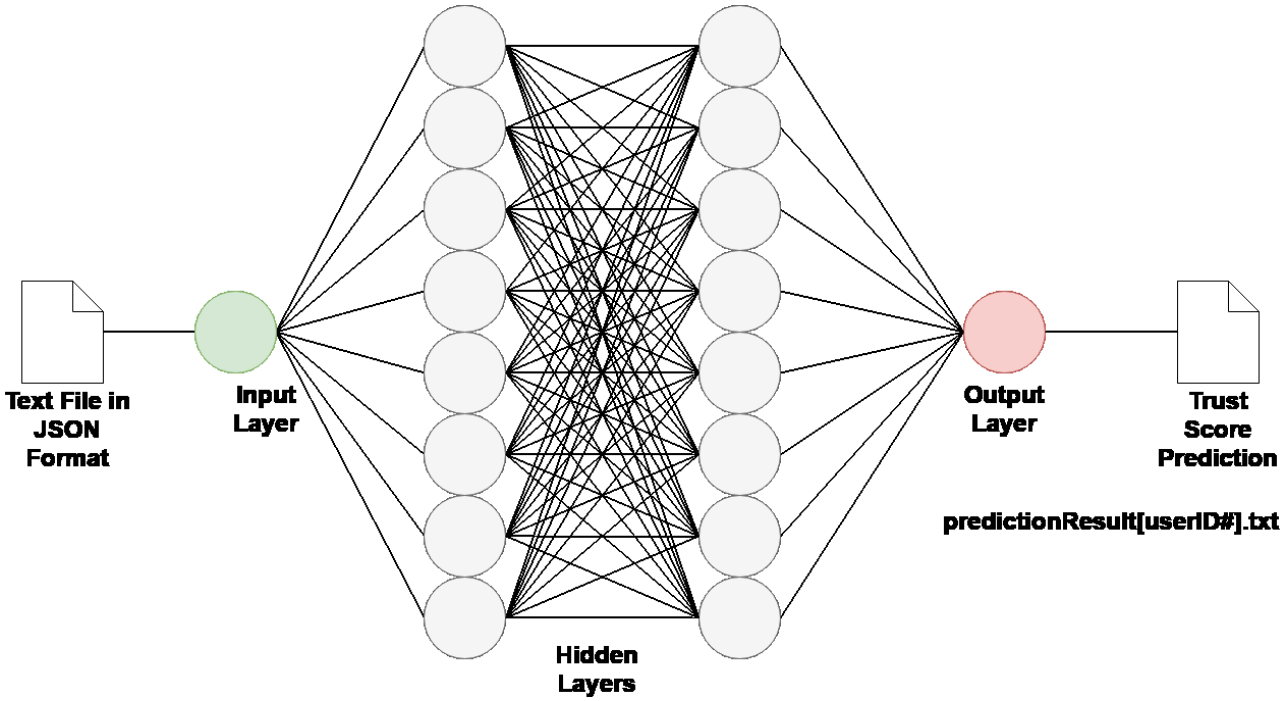


Figure 8: Eight node per layer structure of our neural network

Once the neural network had produced scores for each sensor individually, a final trust score was produced by averaging together the reported trust score for each sensor. The final trust score was then written to a text file associated with that subject's UUID.

These trust scores could then be read by the trust score request API call from that subject's Android device, and sent back to the device for authentication. The workflow between Funf on an Android device, the Node.js server, and the Python TensorFlow scripts thus served as a way for devices to be authenticated via behavioral biometrics. All of this code, as well as the trust scores, were stored in a private Github repository as well as on the WPI research cluster, both as a backup and as a way of managing access to the code by multiple programmers for the purpose of maintaining the codebase.

The code used in our TensorFlow implementation is outlined below.

```
regressor=tf.contrib.learn.DNNRegressor(feature_columns=feature_cols,
                    hidden_units=hidden_units,
                    model_dir=networkSetup.MODEL_DIR,
                    activation_fn=tf.nn.tanh,
optimizer=tf.train.GradientDescentOptimizer(
                    learning_rate=learning_rate ),
                    config = tf.contrib.learn.RunConfig(
                    save_summary_steps = 10000000))
```

The above code snippet created a neural network with the specified parameters, which are the ones that LOBS used. Feature_cols was a variable containing the components of the device sensor data we wished to feed into our neural network. The hidden_units variable was a list of numbers—one for each layer in LOBS—which determined how many 'neurons' we wished to put into that layer. Model_dir was a variable file path which contained the ID of the user whose data were being trained on and the corresponding sensor type. The activation_fn parameter corresponded to the activation function used in the neural network. In general, activation functions are used to

determine whether or not each “neuron” will fire a “synapse” to other nodes when information is passed to it.

```
def train(filepath, isAuthentic, hidden_units=[8, 8], learning_rate=0.001):
 regressor = getRegressor(hidden_units, learning_rate)
 regressor.fit(input_fn=lambda: input_fn(filepath, isAuthentic), steps=1)
```

In order to train a neural network for future use, we used the above function. This function took in a file path, a format for number of layers and how many neurons should be used in each layer, and a learning rate which corresponded to how much the training network would change with every new piece of data it was trained on. Inside the train function, we called getRegressor to build our neural network to train using the specified variables.

```
def evaluate(filepath, isAuthentic, hidden_units=[8, 8], learning_rate=0.001):
 regressor = getRegressor(hidden_units, learning_rate)
 ev = regressor.evaluate(input_fn=lambda: input_fn(filepath, isAuthentic), steps=1)
 loss_score = ev["loss"]
 print("Loss: {0:f}".format(loss_score))
 return loss_score
```

The evaluate function computed the average loss that resulted from training the neural network. It used the same variables as train in order to do so. For each data point fed into the network, we developed a loss score, which was a measure of error in a neural network’s ability to classify the training input. Once the neural network was finished training, we obtained the output of this function which was an average of all loss values recorded.

```
def predict(filepath, hidden_units=[8,8], learning_rate=0.001):
 regressor = getRegressor(hidden_units, learning_rate)
 y = regressor.predict(input_fn=lambda: input_fn(filepath))
 predictions = list(itertools.islice(y, networkSetup.getNumRows()))
 sum = 0
 for p in predictions:
  sum+= p
 average = float(sum)/len(predictions)
 print('average prediction over wifi sensor objects: %f' %(average))
 return average
```

The predict function was similar to its predecessors train and evaluate. In the same way, it received a format for hidden layers and a learning rate, and used those inputs to compare a user's testing data (typically the last quarter of their total data) against a neural network which had trained on the rest of their data. This process iteratively worked through the testing data set, and returned an average prediction score which represented how confident LOBS was that the testing data and training data were provided by the same person.

5. Results

5.1 Initial Study Results

In our initial study of fifty participants, we ran a Python script that would generate trust scores for each participant based on their own profile, to evaluate the effectiveness of LOBS in authenticating the correct person, against those of four other randomly selected participants, to test locking out the wrong ones. The impostors we tested against were the same for everyone.

With the data from that study, we experimented with varying some parameters of our authentication algorithm to see what produced the best results. Initially, we measured results by calculating the F1 score of the classifications based on various required trust score thresholds for authenticity. The F1 score is the harmonic mean of precision, which is the fraction of the time that cases labeled as authentic by the classifier being evaluated are actually authentic, and recall, which is the fraction of the authentic cases that are labeled as such by the classifier. It is impossible to draw conclusions from comparing F scores computed with differently sized data sets, so we used the F score primarily to compare the results LOBS obtained with different parameters for the same data set. For the authentic cases in our data set, we generated trust scores for every participant, using the first three quarters of their data as the "historical" data for training and the remaining quarter as the "new" data that LOBS would determine the authenticity of. For the inauthentic cases, we matched the "historical" data of each participant with the "new" data of each other participant. This meant that we had many more inauthentic cases than authentic ones (skewed data), which artificially lowered all our F scores, but did not make the comparisons between our scores less effective. In order to streamline the process of experimenting with multiple trust score thresholds, we generated all the trust scores only once, and then wrote a script that would generate F scores by

classifying those trust scores into “authentic” or “not authentic” for any of several given trust score thresholds.

We observed how frequently LOBS authenticated users and impostors with values of trust score thresholds ranging from our initial guess of 90% up to a cutoff of 99%. We tested every integer number of percentage points, as we found simply by looking at the data that many of the trust scores produced were above 90%, in order to find which cutoff point yielded the highest F1 score. A higher F1 score indicates a higher degree of accuracy for LOBS when using the associated trust score threshold. The F1 scores for every trust score threshold in that range are shown in Figure 9, which shows that the optimal trust score cutoff for us was 93%.

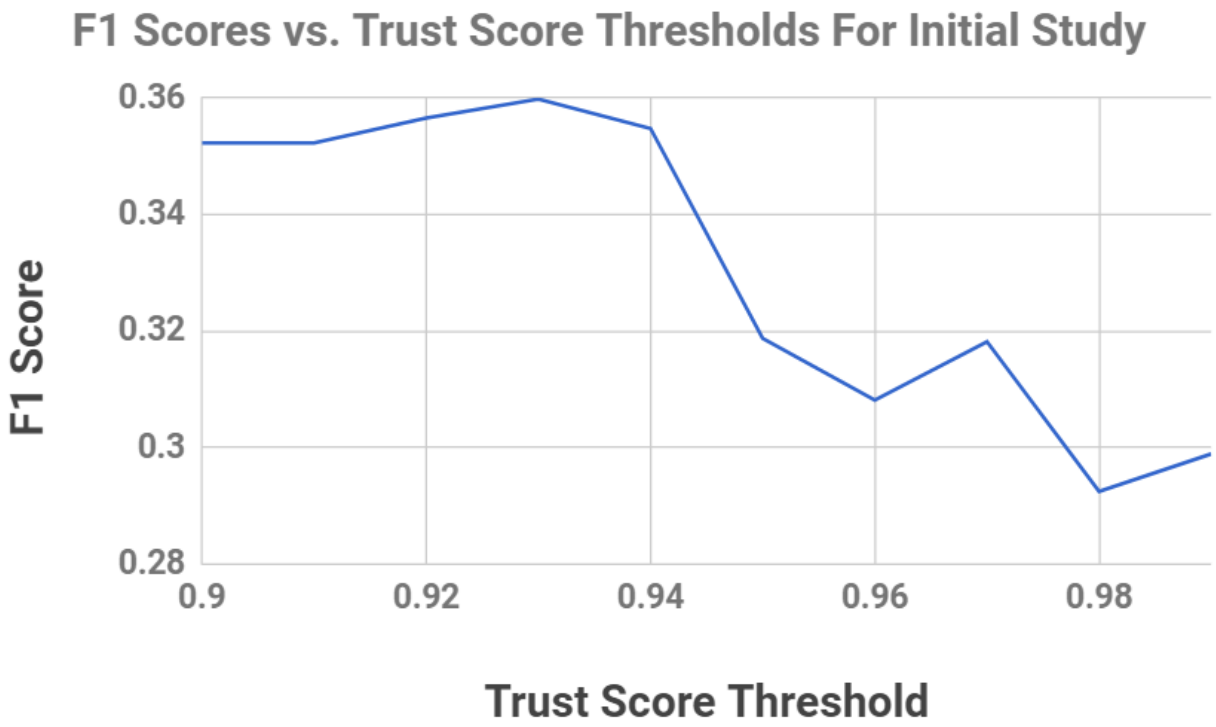


Figure 9: F1 Scores at varying trust score thresholds for authenticity for our initial study.

After some discussion from our team, we decided that the F1 score was not the ideal measure of accuracy for our network, since for our application, incorrectly authenticating the wrong

person was a much more serious error than failing to authenticate the right one. Consequently, we decided to weight false positives more than false negatives in our evaluation by a factor of 10, by using the F0.1 score instead of the F1 score. Other than the different weighting of false positives and false negatives, the F0.1 score is the same as the F1 score. The F0.1 scores LOBS obtained for the same set of trust score thresholds are shown in Figure 10. When evaluated with the F0.1 score, the best trust score threshold still turned out to be 93%.

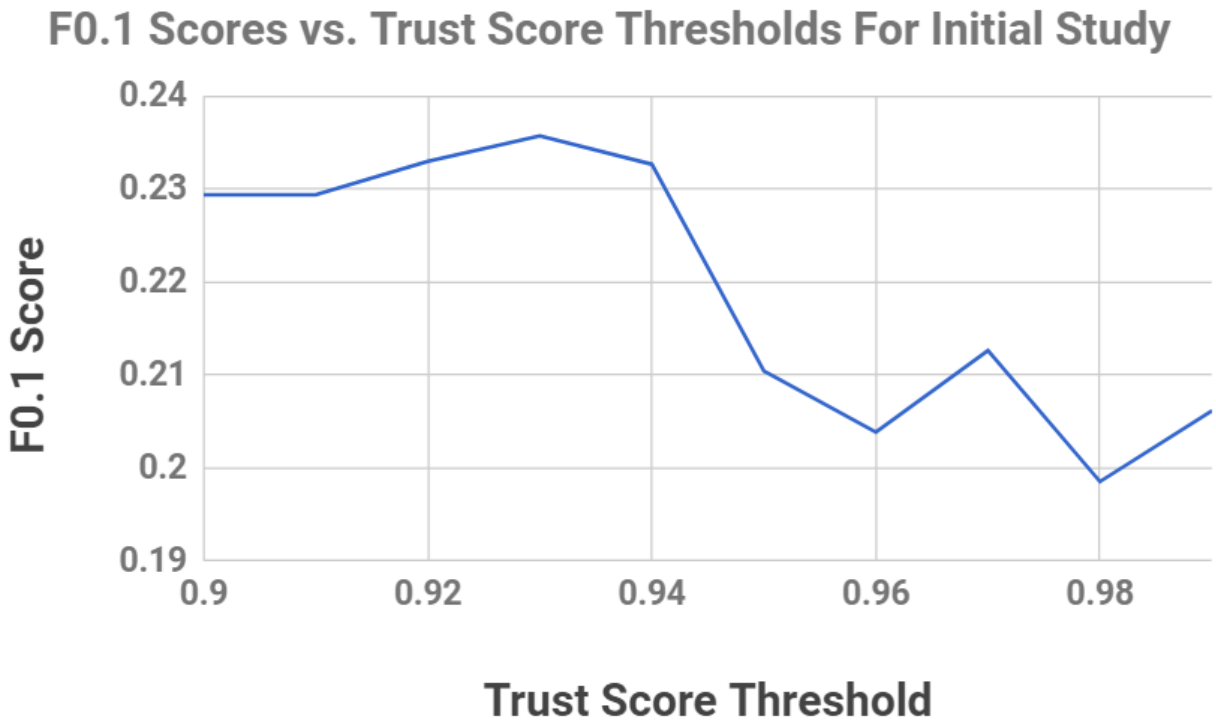


Figure 10: F0.1 scores at varying trust score thresholds for our initial study.

In an effort to determine how many days' worth of data we needed to gather to get useful data for any given subject, we also tried to generate predictions with the network being trained on various numbers of days of data, ranging from 1 to as many as we had for any given subject. However, this did not turn out to be very useful, as most of our subjects had too little data for varying the number of days to make a difference. The only conclusion we could draw from that

experiment was that having more than two weeks' worth of data for any one subject slowed the system down without affecting the results. Consequently, our results were generated by using all the data produced by the subject, unless it was over two weeks' worth, in which case we only used the last two weeks.

5.2 SONA Study Results

When the data from the SONA study had been gathered, we ran the modified neural network to generate trust scores for each of the participants when trained on their own and when trained on each other participant's data. Strangely, the trust scores generated by the modified neural network were universally very high: none of them fell below 0.9, and almost all were above 0.99999. Consequently, although we had originally intended to use the same trust score threshold as we had found to work in the prior study, it was apparent that this would produce very poor results. Instead, we determined what trust score threshold was appropriate for the SONA data separately. We set the trust score threshold for authentication to values of $1 - 10^{-X}$ for values of X from 1 to 15. In other words, we tried 0.9, 0.99, etc. until we reached $1 - 10^{-15}$.

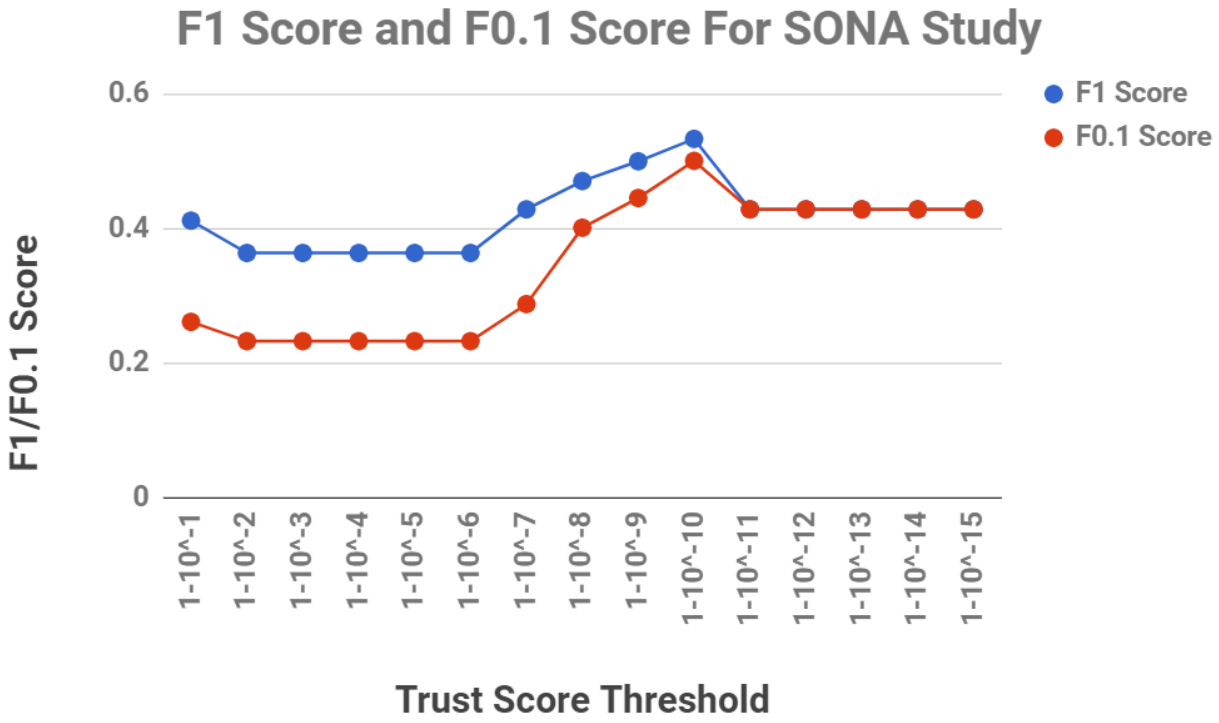


Figure 11: F1 and F0.1 scores of LOBS with various trust score thresholds on the SONA study data.

We found that, for both the F1 score and the F0.1 score, the best-performing trust score threshold for the SONA data was $1 - 10^{-10}$, as shown in Figure 11. Interestingly, although the trust scores were exceedingly high for the SONA data, the F scores we got with a similarly high threshold were substantially better than those we got with the older data: the F0.1 score for this trust score threshold was 0.501, whereas the best we achieved with the older data was 0.236. However, since F scores cannot necessarily be compared with different data sets, that does not mean that the SONA system actually outperformed the older one.

To determine if the new parameters actually outperformed the old ones on the same data set, we tried running the network with the original hyperparameters on the SONA data, in order to get F scores that we could compare. The best trust score threshold with the old parameters was 90%, with a F0.1 score of 0.288, which was much lower than the F0.1 score of 0.501 that we obtained with the

new parameters. Since that was using the same data, the F scores were comparable, which indicated that the network with the new parameters did outperform the network with the older parameters on the SONA data.

6. Discussion

6.1 Study Accuracy

For both our early study and our later SONA study, the F0.1 scores we produced were substantially better than what a random classifier would generate with that distribution of positive and negative cases. A random classifier might seem easy to beat, but it was the only comparison we had, since there is no general definition of a "good" F score. With the original study's data, where we had 50 people, and generated predictions for each person with both their own data and each other person's, a random classifier would produce an average F0.1 score of 1.98%. Our classifier obtained an F0.1 score of 23.6%, which is a significant improvement. With the smaller SONA study, which only had 6 participants, a random classifier would generate an average F0.1 score of 16.7%. Our classifier, with the newest learning rate, shape, and activation function, produced a F0.1 score of 50.1%. While we had no data for competing algorithms that we could compare our results to, LOBS was clearly an improvement over random guessing. Therefore, this method of authenticating users was recognizing data that distinguishes people, if imperfectly. This meant that the prototype version of LOBS was working, at least to some degree.

We also tried running the older study's data in the network with the new hyperparameters and the SONA study's data in the network with the original ones, in order to evaluate the effect of changing the hyperparameters. Using the neural net with the new parameters, we achieved a F0.1 score of 5.06% on the larger study. This was lower than we obtained with the original parameters. However, running the network with the original parameters on the SONA data produced a maximum F0.1 score of 28.8%, which is lower than the new parameters as well. From that, we concluded that the most likely explanation was that, because the new parameters were optimized

with a subject with a substantial amount of data, those parameters were most appropriate for other subjects with a similar amount of data. Since the subjects in the SONA study generally had more data than those in the earlier study, they might have been closer to the test subject that those parameters were optimized for, and consequently the new parameters could have performed better with the SONA study's data than with that of the original study.

Additionally, despite the lack of success of the day-by-day analysis, we did determine, by looking at the trust scores for subjects with differing amounts of total data, that subjects with more data were generally more likely to be correctly authenticated and less likely to authenticate impostors. While we could not determine a specific threshold at which more data collected consistently produced better results, we found that, when trained on subjects with at least two weeks of historical data, LOBS' accuracy at verifying the correct user and detecting impostors stabilized, and having more historical data slowed the network down greatly without noticeably improving the results further. This was a very rough estimation, due to the fact that the amount of data we gathered for any given subject ranged from a couple of hours to more than two months' worth of data. Unfortunately, we had very few subjects who had at least two weeks' worth of historical data to train on, so although we could find out how much of that data was needed to produce consistent results for each of them, we could not precisely determine the amount of data required to produce consistent results in general. All users generated consistent results with at least two weeks of historical data, though, so we concluded that at most two weeks' worth of data were generally required to obtain good results.

It is also worth noting that that amount of time was what was required for all metrics to produce effective classifications, and individual metrics might do so faster. Most notably, the accelerometer could produce consistent results in its trust score evaluations with a few minutes'

worth of data, but LOBS as a whole would not produce accurate results until each metric did so individually, since the final trust score produced was the average of those produced by each metric.

6.2 Energy Consumption

The data collecting app itself consumed about 332.1 mW during an accelerometer sensor polling session. In comparison, the data collecting app used in Rachuri et al. [14] consumed 561 mW. Notably, the Rechuri application used the same hard sensors as our application, which would typically result in similar power consumption. The device used in Rachuri et al. [14] ran the Android 4.4 operating system, while most devices used in our study used the Android 7.0 operating system. Therefore, it is likely that the much newer operating system, coupled with newer hardware is more energy efficient, which may have impacted these results.

6.3 Authentication Speed

Our original goal for the speed of LOBS was to have a trust score be available to authenticate or deauthenticate the user within half an hour of starting to use the device, once the user had been using LOBS and collecting historical data for at least two weeks. The goal for the speed of returning trust scores was slower than would have been ideal, since we wanted to be able to deter intruders almost immediately, but we decided to choose half an hour in order to allow LOBS to collect enough data and process it.

In addition, while we were able to reduce the amount of time to train and predict a trust score from a user's data by roughly a factor of eight from our initial prototype, LOBS still took at least 10 minutes to train on the already collected data and produce a trust score. This further limited

the timeframe for data collection. Once we built LOBS, however, we realized that in order to accurately predict a trust score, at least 300 points of test data were required. While we could produce that much data with under a minute of collection, almost all of that data would come from the accelerometer, resulting in trust scores based almost entirely on accelerometer data.

Consequently, we concluded that LOBS could be made to run fast enough to meet our goal, but we could only do so by effectively ignoring all metrics we used other than accelerometer. This indicates that, if LOBS is to reach our goal for performance, we need to use only sensors that gather data as fast or nearly as fast as the accelerometer does, which none of our other metrics did. Future work on LOBS might therefore include finding other metrics that produce data at that pace, in order to have more than one sensor gather useful amounts of data within our target timespan for authenticating or deauthenticating the user.

7. Future Work

7.1 Integration with Android Operating System

Our ability to gather distinguishing data for our users was heavily limited by the fact that our data gathering application was a third party application and therefore did not have access to some types of data. Specifically, the Android operating system gathered data on how the user uses the touch screen and the keyboard, which prior studies had demonstrated were very useful for distinguishing users, but did not give third party applications access to this data at the time of this study. Additionally, as a third party application, it was impossible for us to actually modify the login flow for the phone.

Therefore, further research would be much more effective if done by a group that is associated with a mobile device company, which would give them access to touchscreen and keyboard data, and also permit them to incorporate LOBS into the device at the operating system level. With this association, our application could be improved to capture a user's usage of the touch screen for taps and swipes, and typing tendencies through use of the keyboard. Furthermore, with access to the operating system, the application would be integrated as a backup authentication method, prompting the user to perform their typical authentication only when LOBS does not sufficiently trust the user. In contrast, the final version of LOBS only calculated this level of trust, but had no means of acting upon it. It is also worth mentioning that implementing a similar system as part of the operating system will ensure that sensor data can be gathered at all points of the day, instead of relying on users interacting with an application on a regular basis.

7.2 Faster Metrics

When trying to optimize the performance of LOBS, we found that, of the metrics we used, only the accelerometer actually produced enough data to authenticate or deauthenticate people with any degree of accuracy by the time that we wanted to have LOBS return a trust score. This effectively meant that, if we tried to authenticate people as quickly as we originally were trying to, we would only be using one metric. We would like to have more metrics that gather data fast enough to produce meaningful results within that time frame in order to authenticate people more accurately at that speed.

7.3 Additional Devices

7.3.1 Other Smartphone Operating Systems

In addition to adding different sensor types and access levels, another way to potentially improve data and prediction quality is to include different devices. Our studies only used Android phones to collect data, but there are other devices that could be used. For mobile phones, both iPhones and Windows Phones are valid options for a mobile behavior-based authentication system. In fact, the only limitation for a device being able to use a system such as this is that it gather enough data in the course of normal interactions with the user to verify the user's identity, which is true of all modern smartphones that we could find.

7.3.2 Other Device Types

Along with phone data collection, there are a variety of other device types that could be used to collect behavioral data. For example, smart home devices have had a rise in popularity in

recent years. Since these devices are constantly listening to their users' voices, and occasionally being interacted with in other ways, as the users go about their daily activities, they collect valuable behavioral data that can be later used for authentication. Similarly, smartwatches were an accessory with growing popularity at the time of this study. Like phones, they collect large amounts of data from their wearers throughout the day, including heart rate, gait, GPS, and nearby WiFi and Bluetooth devices. Future studies can take advantage of these various devices, using the framework provided by this study, in order to create a wider range of functionality for data collection. This could be used either to more accurately authenticate people into their phones or to authenticate them separately into their other devices.

7.4 Other Neural Networks

Additionally, our study only used a feedforward neural network to create trust score predictions. There are many other types of neural networks, such as a recursive neural network, or RNN. Future studies could take advantage of the various types of neural networks and attempt to generate more accurate trust scores than we did by using them.

7.5 Additional Experiments

There are some other experiments we could perform in order to determine the optimal amount and types of data used to predict the best trust scores. One such experiment would be to run training and testing on smaller subsets of each subject's data, starting with the subject's complete data. In each iteration, the amount of data used by the neural network would be smaller, and the

network's accuracy would be evaluated for that amount of data. This experiment would help determine the minimum amount of historical data required to accurately identify impostors.

Another possible experiment would be to determine what metrics are the most effective for classification by running LOBS on data that excludes one or more of the metrics and observing whether or not the accuracy of LOBS is impacted as a result. This experiment would help determine if any metrics were unnecessary, and therefore could be taken out of LOBS without making it any less effective. If any metrics were determined to be not worth using and consequently removed, this would improve the efficiency of LOBS in two ways. The data collection app would not need to collect and send the data for those sensors to the server, thereby saving battery power and taking less time to collect data. In addition, the neural network would not need to train on or predict trust scores for those sensor types, which would reduce the time it takes for LOBS to produce a trust score for the user.

8. Conclusions

In this paper, we designed and implemented a system, called LOBS, to implicitly authenticate or deauthenticate users based on their behavior, without the user needing to consciously provide any information. If incorporated into the normal login flow of smartphones, it would enhance phone security by requiring the user to enter their password less, which would make users more willing to use more secure methods of explicit authentication even if those are less convenient to enter. LOBS gathered data on the phone's location, using the GPS sensor and the observed WiFi networks; orientation, using the accelerometer and magnetic field sensor; battery usage, using the battery sensor; and the times that the screen turns on and off. It then trained a feedforward neural network, created using TensorFlow, on the data that was gathered for each user. When the user needed to be authenticated or deauthenticated, LOBS would produce a trust score by running the neural network that was trained on the user's older data on the latest data. This trust score would reflect how confident the network was that the newer data was from the same user as the older data. If it was above a certain threshold, the user was authenticated; if not, the user was deauthenticated. Modifying the phone's login flow to incorporate LOBS was beyond the scope of this project, but if we had been able to do that, the user would have been logged out and required to log in with a password or some other form of explicit authentication.

While LOBS did not perform well enough to be commercially marketable as an addition to normal passwords, it performed much better than random chance. We measured the effectiveness of our system using the F0.1 score, which is the harmonic mean of the precision and recall of the system's classification of users, with false positives (impostors being authenticated) being weighted ten times as highly as false negatives (authentic users being locked out), since false positives are more of a security issue than false negatives. Our goal was to reach a score of 90%. With our first

large study, which we used to fine-tune the system, we achieved an F0.1 score of 23.6%, whereas a random classifier with the same data would have achieved a score of 1.98%. With our second one, which we used to test the improved system, we achieved a score of 50.1%, whereas a random classifier on that data would have achieved a score of 16.7%.

Although we did not achieve our targeted level of accuracy for LOBS, the fact that we could distinguish people's actual data from that of an impostor with some degree of reliability with this method is promising. Considering that we had less data to work with than we would have liked and that we did not have access to several of the metrics that had been demonstrated to be the most effective, such as keyboard usage and touch screen data, the fact that it works at all is an accomplishment. Our results, while not exemplary, showed a decent proof of concept for a behaviour based mobile authentication system. We were able to create an automated, fully-functional system that gathered sensor data and analyzed it to authenticate subjects based on their recent data. The F0.1 scores we determined from the results of our experiments were lower than we would have liked, but much higher than randomly guessing, which indicates that LOBS was detecting and analyzing real differences between the participants' data. Consequently, while LOBS needs work, an improved version could be commercially marketable.

Bibliography

[1] 1843161695945955. “Log Analytics With Deep Learning And Machine Learning.” *Hacker Noon*, Hacker Noon, 13 May 2017 hackernoon.com/log-analytics-with-deep-learning-and-machine-learning-20a1891ff70e.

[2] Anony-Mousse. “Best Empirical Macro/Micro F1 Score?” Machine Learning - Best Empirical Macro/Micro F1 Score? - Cross Validated, Https://Stats.stackexchange.com, 12 June 2015, 11:42, stats.stackexchange.com/questions/139002/best-empirical-macro-micro-f1-score.

[3] Aviv, Adam J, Gibson, Katherine, Mossop, Evan, Blaze, Matt, Smith, Jonathan M. “Smudge Attacks on Smartphone Screens.” pp. 1–10.

[4] Biedert, E. Ma, I. Martinovic, and D. Song, “Touchalytics: On the Applicability of Touchscreen Input as a Behavioral Biometric for Continuous Authentication,” IEEE Trans. Inf. Forensics Secur., vol. 8, no. 1, pp. 136–148, Jan. 2013

[5] Bo, C., Zhang, L., Jung, T., Han, J., Li, X.-Y., Wang, Y.: Continuous user identification via touch and movement behavioral biometrics. In: Proc. 2014 IEEE International Conference on Performance Computing and Communications (IPCCC), pp. 1–8. IEEE (2014). http://onlineconfusionmatrix.com/

[6] Carter, Daniel Smilkov and Shan. “Tensorflow - Neural Network Playground.” *A Neural Network Playground*, http://playground.tensorflow.org/. Accessed on *Jan 5th, 2018.*

[7] Cheng, Jyh-Min, and Hsiao-Chuan Wang. “A Method of Estimating the Equal Error Rate for Automatic Speaker Verification.” SympoTIC '04. Joint 1st Workshop on Mobile Future & Symposium on Trends In Communications (IEEE Cat. No.04EX877), 15 Dec. 2004, doi:10.1109/chinsl.2004.1409642.

[8] Donti, Arun, et al. “A Behavioral Model System for Implicit Mobile Authentication.” Worcester Polytechnic Institute, Apr. 2017.

[9] “DNNRegressor with Custom input_fn for Housing Dataset.” https://github.com/, Google Brain Tree, 16 June 2017, github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/input_fn/boston.py. Accessed on *Feb 26th, 2018.*

[10] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, “Implicit Authentication through Learning User Behavior,” in Information Security, 2010, pp. 99–113.

[11] Asim, Fiv. “AndroSensor.” AndroSensor, Google Play, 23 Jan. 2015, play.google.com/store/apps/details?id=com.fivasim.androsensor. Accessed on *September 20th, 2018.*

[12] “Funf Open Sensing Framework for Android.” Funf Open Sensing Framework for Android, Funf.org, 11 Jan. 2016, github.com/funf-org/funf-core-android.

[13] H. Crawford, K. Renaud, and T. Storer, “A framework for continuous, transparent mobile device authentication,” Computer Security, vol. 39, Part B, pp. 127–136, Nov. 2013.

[14] K. K. Rachuri, T. Hossmann, C. Mascolo, and S. Holden, “Beyond location check-ins: Exploring physical and soft sensing to augment social check-in apps,” in Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on, 2015, pp. 123–130.

[15] H. Khan, A. Atwater, and U. Hengartner, “Itus: an implicit authentication framework for android,” in Proceedings of the 20th annual international conference on Mobile computing and networking, Maui, Hawaii, USA, 2014, pp. 507–518.

[16] Kwapisz, J.R., Weiss, G.M., Moore, S.A.: Cell phone-based biometric identification. In: 2010 Fourth IEEE International Conference on Biometrics: Theory Applications and Systems (BTAS), pp. 1–7. IEEE (2010).

[17] L. Fridman, S. Weber, R. Greenstadt, and M. Kam, “Active Authentication on Mobile Devices via Stylometry, Application Usage, Web Browsing, and GPS Location,” IEEE Syst. J., vol. PP, no. 99, pp. 1–9, 2016.

[18] Momoh, Osi. “Deep Learning.” *Investopedia*, Investopedia, 30 Oct. 2016, www.investopedia.com/terms/d/deep-learning.asp.

[19] Moore, Michael. “Thieves Can Steal Your Smartphone PIN Code in SECONDS.” *Express.co.uk*, Express.co.uk, 13 Mar. 2017, www.express.co.uk/life-style/science-technology/778637/mobile-phone-security-steal-pin-code-pattern-lock.

[20] “Mobile Fact Sheet.” Pew Research Center: Internet, Science & Tech, Pew Research Center, 12 Jan. 2017, www.pewinternet.org/fact-sheet/mobile/.

[21] “Pickle — Python Object Serialization.” Documentation » The Python Standard Library, The Python Software Foundation, 3 Feb. 2018, docs.python.org/2/library/pickle.html.

[22] R. Murmuria, A. Stavrou, D. Barbará, and D. Fleck, “Continuous Authentication on Mobile Devices Using Power Consumption, Touch Gestures and Physical Movement of Users,” in International Workshop on Recent Advances in Intrusion Detection, 2015, pp. 405–424.

[23] R. Murmuria and A. Stavrou, “Authentication Feature and Model Selection using Penalty Algorithms,” in Symposium on Usable Privacy and Security (SOUPS), 2016.

[24] Rainie, Lee, and Andrew Perrin. “10 Facts about Smartphones as the IPhone Turns 10.” Pew Research Center, Pew Research Center, 28 June 2017, www.pewresearch.org/fact-tank/2017/06/28/10-facts-about-smartphones/.

[25] Senaka Buthpitiya, Anind K. Dey, Martin Griss. 'Soft Authentication with Low-Cost Signatures'. *2014 IEEE International Conference on Pervasive Computing and Communications*, pp.172-180. http://www.cs.odu.edu/~cs752/papers/advance-019.pdf

[26] "SensingKit Mobile Sensing Framework." SensingKit, Queen Mary University of London, www.sensingkit.org/. Accessed on *September 20th, 2018.*

[27] Sourav Kumar Dandapat, Swadhin Pradhan, Bivas Mitra, Romit Roy Choudhury, and Niloy Ganguly. 'ActivPass: Your Daily Activity is Your Password'. Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, April 18-23, 2015, Seoul, Republic of Korea, pp.2325-2334.

[28] "TensorFlow." TensorFlow, Google Brain Team, www.tensorflow.org/. Accessed on *Feb 26th, 2018.*

[29] Yampolskiy, R.V. and Govindaraju, V. (2008) 'Behavioural biometrics: a survey and classification', *Int. J. Biometrics*, Vol. 1, No. 1, pp.81-113.

[30] Z. Sitova, J. Sedenka, Q. Yang, G. Peng, G. Zhou, P. Gasti, and K. Balagani. Hmog: New behavioral biometric features for continuous authentication of smartphone users. Information Forensics and Security, IEEE Transactions on, PP(99):1–1, 2016. M. Frank, R.

[31] Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R.P., Mao, Z.M., Yang, L.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp. 105–114. ACM (2010).