CONTINUOUS USER AUTHENTICATION AND IDENTIFICATION USING USER INTERFACE INTERACTIONS ON MOBILE DEVICES

By

Vaibhav Bhushan Sharma

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science - Master of Science

2015

ABSTRACT

CONTINUOUS USER AUTHENTICATION AND IDENTIFICATION FROM USER INTERFACE INTERACTIONS ON MOBILE DEVICES

By

Vaibhav Bhushan Sharma

We investigate whether a mobile application can continuously and unobtrusively authenticate and identify its users based on only their interactions with the User Interface of the application. A unique advantage that this modality provides over currently explored implicit modalities on mobile devices is that every user who uses the mobile application is automatically enrolled into the classification system. Every user must interact with the User Interface of an application in order to use it and therefore this modality is always guaranteed to have sufficient number of inputs for training and testing purposes. Using different types of input controls available on the Android platform, we collected interactions from 42 users in five different sessions. We created base classifiers from each type of input control and combine them into an ensemble classifier in order to authenticate and identify users. We find Support Vector Machine based ensemble classifier achieves a mean equal error rate of 5% in case of authentication and a mean accuracy of 90% in case of identification. We find Support Vector Machine based ensemble classifiers outperform other techniques in both cases. While the ensemble classifier performance for authentication and identification is not found to be sufficient for it to replace current primary authentication mechanisms used in mobile applications, its truly continuous nature provides motivation for it to be used in combination with primary mechanisms.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Richard J. Enbody for guiding me in the right direction when I was working towards a solution for the research problem presented in this thesis. I wish to express my gratitude to Dr. Arun Ross for teaching a wonderful course on pattern recognition. I would also like to thank two friends, Sunpreet Singh Arora and Sorrachai Yingchareonthawornchai for their helpful suggestions during discussions about this research. I would also like to thank all the volunteers who provided their usage data. This work would not have been possible without their support. Finally, I wish to thank my friends and family for always encouraging me and believing in me during difficult times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

With the widespread use of mobile devices, a large number of mobile users use their devices for accessing sensitive information. Mobile devices which have been compromised can cause theft of such critical information and be harmful to the mobile user's wellbeing. Examples of such information include usage of mobile banking applications, critical information saved into email accounts and private contact information stored for friends and family. In order to prevent attackers from getting access to such information, mobile device manufacturers typically program devices to explicitly authenticate the mobile user before allowing access to the device.

1.1 Explicit Authentication

Typically used explicit authentication mechanisms include asking the user the setup a pass code or a pattern on first boot of the device. However, a variety of issues manifest themselves in these authentication schemes. Users must remember the pass codes or patterns at all times. This form of explicit authentication also diverts the user's attention away from the purpose he wishes to use the device for. However, the most critical issue with such authentication schemes is that these schemes provide all or nothing access to the device. In other words, the user either gets access to all the applications on the mobile device or gets no access at all. In the context of a convenience vs. security tradeoff, such schemes compromise heavily on user convenience. A critical assumption made by such schemes is that only the

legitimate user will use the device after having unlocked it and the device will be locked again after the usage session. However, mobile users often encounter situations when the device has to be put away for a brief period of time due to another situation requiring their urgent attention leaving the device vulnerable to attackers. As reported by Lookout Security[21], one in 10 mobile users in the United States are victims of mobile theft which can lead to further information theft. Such scenarios necessitate other authentication schemes[19] which are less obtrusive, more continuous while remaining accurate enough to be useful in practical situations.

1.2 Implicit Authentication

Implicit Authentication(IA) schemes have been an active area of research in the past few years. The general idea when doing implicit authentication is to only use inputs which are given by users indirectly and make a decision on whether any such input or group of inputs were provided by a legitimate user that we have previously seen or an unknown user. Investigation has been carried out to check if modalities derived from implicit input contain information which is discriminant enough to distinguish between a known user and a set of unknown users. It has been found that users are surprisingly receptive[12] to using such non-intrusive schemes for authentication even though IA schemes continue to have security limitations. The criteria which makes IA more desirable is that it is strictly non-intrusive and continuous providing users a feeling of security and at the same time, not creating any barriers from providing desirable functionality to users. Some of the IA modalities that have been explored are described briefly in the next section.

1.2.1 Explored Implicit Authentication Schemes

1.2.1.1 Touch-based Authentication

The most prominent form of capturing the user's implicit inputs to a mobile device would be via the touch screen since all of a user's interactions with the mobile device occur via the touch screen. An example of such a modality would be touch-based authentication[8]

wherein the authors gathered vertical and horizontal fling gestures performed by 41 users while using different mobile phone applications. However, this modality requires users to provide scroll gestures as inputs to the device, thereby creating a limitation of the modality not getting inputs fast enough on devices with larger screens.

1.2.1.2 Keystroke-based Authentication

Feng et al.[5] explored the possibly of authenticating users via their behavior observed when they used the soft keyboard available on mobile devices. While this modality has the advantage of being resilient to attack, it is limited to being useful only when attackers are forced to use the soft keyboard for their purposes. In scenarios when malicious tasks can be accomplished without providing any keystrokes, this modality has limited value.

1.2.1.3 Sensor-based Authentication

Kayacik et al.[15] devised a sensor-based authentication mechanism which created a user profile from sensor data during the mobile device's usage, measured profile stability and switched to a authentication phase once a user profile stabilized. However, this technique requires invasive access to a user's private information such as location inferred from cell towers several times a day, access to wifi signals and accelerometer readings. In situations where a user may not be entirely comfortable with sharing this information with a authentication mechanism, this technique would have limited value.

1.2.2 UI Interaction-based Authentication

In this report, we propose a new modality for implicitly authenticating users to their mobile devices. Every mobile application will always have a User Interface(UI) to interact with its users. We observe user behavior as seen through interactions with the application's UI and use it for not only authenticating, but also identifying users. This technique has the advantage of being available on every mobile device regardless of screen size. Every popular mobile application will have a UI which users will interact with and thereby provide implicit inputs for this modality. These inputs can be captured in a non-obtrusive manner and do not impose any specific requirements on the application such as requiring the application

to gather inputs from the user via a soft keyboard. Also, this modality can capture inputs without requiring invasive access to the mobile device's sensors. Thus, this modality is almost always guaranteed inputs regardless of the sensors present on the device.

As part of this investigation, we use every available form of UI interaction in the Android OS[10] to authenticate users based on their UI interactions during application usage sessions. We propose that it is possible to use a combination of all kinds of available input controls on the Android OS to perform implicit authentication and identification on an Android device.

We hypothesize that an interaction of a user with the UI of a mobile application contains some distinguishing information which when put together with every piece of information extracted from UI interactions in the same app usage session can help determine the identity of the user. The intuition behind this hypotheses is illustrated in Figure 1.1. This figure shows the coordinates where two different users touched in order to bring the keyboard's focus to the edittext. It can be seen clearly that these two users focused on different parts of the edittext and therefore shows how the edittext input control is able to capture discriminant information from these two users. Figure 1.2 demonstrates the separability between touch features extracted between two different users when they were interacting with radiobuttons. It shows how the features naturally lend themselves to a clear classification and also how feature values for different users occupy different spaces in the feature space. The objective of this study is to evaluate if such UI interactions are discriminant enough to provide an implicit form of user authentication instead of explicit forms such as secret pins or passwords.

1.3 Implicit User Identification

While modalities providing authentication mechanisms help determine if a test input belongs to the legitimate user or not, the problem of user identification assigns every test

Figure 1.1: Touch gestures on Android EditText for two users. It can be seen that these two users focused on two different parts of the EditText input control to bring keyboard focus to it.

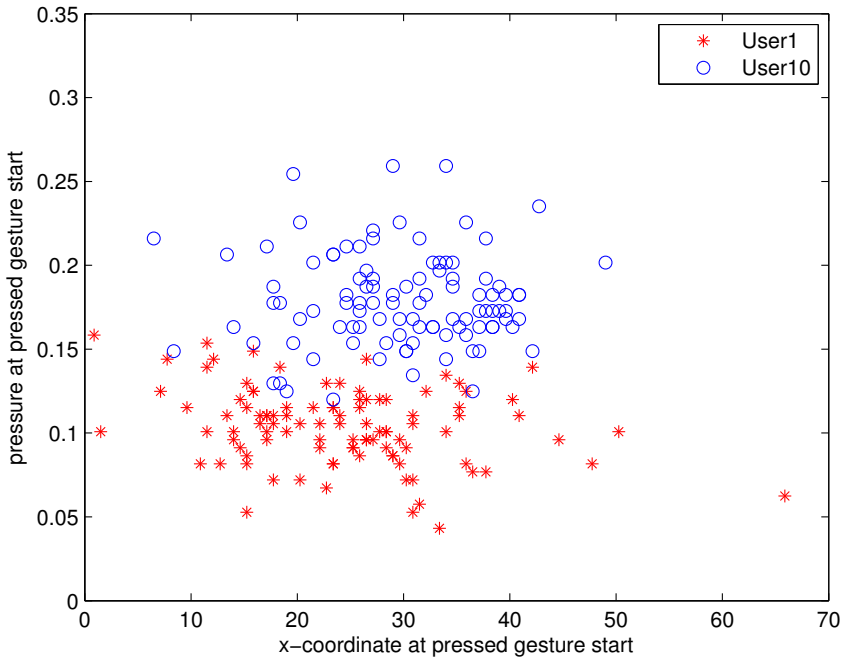

Figure 1.2: Radiobutton usage for two users from our data collection show separability. It can be seen that these two users exerted different finger pressures when attempting to select a Radiobutton for the first time.

input to one user in a set of already known users. Thus being able to not only authenticate but also identify a user based on his test inputs provides a stronger guarantee on the classification of a test input.

1.3.1 Sensor-based Implicit User Identification

Shi et al.[23] makes use of voice, location, touch and accelerometer sensor inputs for establishing user identity and triggers explicit authentication when user identity is found to have changed. A major limitation of this approach is its dependence on multiple sensors being available on the mobile device.

1.3.2 UI Interaction-based Implicit User Identification

We hypothesize that a user's interactions with the UI of a mobile application are discriminant enough to establish the user's identity. We collect UI interaction data from 42 users by asking volunteers to use 5 mobile applications with the same UI. We then train on 4 app usage sessions for every user and try to classify the 5th app usage session as belonging to 1 of the 42 known users. Being able to identify users in this manner has the advantage of not imposing the requirement of having multiple sensors available on the mobile device while also not having to invasively capture any of the user's private information via any other sensors apart from its touchscreen.

The remainder of this report details a literature survey of this research area, provides a description of the data collection performed and classification framework used and provides an evaluation of the classification framework. We discuss some limitations of this technique and provide concluding remarks on this modality.

Chapter 2

Related Work

The hypotheses stated above lies in the general area of implicit authentication on mobile devices. The general idea of implicit authentication is to authenticate the user based on a modality or a combination of different modalities for which the user provides samples in an implicit way, e.g. gestures that the user performs on the screen of a mobile device. Different schemes make different assumptions about the mobile device, e.g. Shi et al.[23] assume the accessibility of four sensor modalities including voice, location, multitouch and locomotion.

Frank et al.[8] extracted features from scrolling gestures performed by users during interaction with their data collection applications and showed users 'touch interactions to be a practical modality for authenticating users. However, this modality makes use of an assumption that the mobile applications that the user or attacker uses will provide sufficient inputs to this technique. Shahzad et al[22] demonstrate explicit touch gestures performed by users as another possible modality that can be used to authenticate users.

Feng et al.[4] demonstrate another touch-based modality and evaluate it on a large data set of test users. These techniques suggest authentication mechanisms that capture data from the entire device irrespective of the application being used. However, both Feng et al. and Frank et al. suggest that taking application context may improve the overall au-

thentication accuracy. Khan et al.[18] argue for a more application-centric approach and use the same touch-based classification methods on datasets collected from the same application.

A framework named *Itus* for doing app-centric implicit authentication has also been proposed by Khan et al.[17]. This framework proposes greater flexibility, extensibility and control be provided to mobile application developers for performing implicit authentication. Other proposed implicit authentication techniques include using sensor-based modalities[14],[26], recognizing users from their gait[7] and using the device picking-up motion as a modality for user authentication[6]. Since the most recent work in this area has been in the direction of making implicit authentication more app-centric, the next logical step seemed to be to try to use app-specific User Interface interactions for authenticating the user.

The most comprehensive evaluation of different authentication schemes has been done by Khan et al.[16]. Six different IA schemes were evaluated against four independently available data sets containing data for over 300 participants. However since none of the existing techniques had performed authentication using only the UI of a mobile application, there were no publicly available datasets on this modality, thereby prompting us to collect a dataset of our own.

One proposed authentication scheme most similar to ours would be LatentGesture[20]. However, this scheme uses only a subset of the input controls[1] available on the Android OS. The evaluation of this scheme for both authentication and identification is done on a much smaller set of users than our evaluation. In this study, the users were chosen on the basis of their prior experience with the mobile device and were also given the opportunity to test the applications before the data collection procedure was begun. This method of data collection contrasts heavily with our data collection procedure wherein we did not select users based on any criteria, apart from asking them to volunteer their time for our data

collection and did not offer users any opportunity to get trained to interact with our data collection applications. We think this method of data collection closely mimics a practical attack scenario wherein an attacker may not have been previously trained to interact with the mobile application but would still be able to successfully interact with the applications.

Chapter 3

General Idea and Goals

In this section, we provide intuition for using UI interactions as a unique modality and describe how we modeled this modality for our investigative purposes.

3.1 Motivation

Several existing approaches to IA have demonstrated that every user behaves differently while interacting with a mobile device. The overall performance of an IA scheme depends on how well the scheme extracts discriminant features from its modality and the classification algorithm used to authenticate and identify test inputs. The intuition behind this modality can be summarized as follows : *As long as the User Interface for a mobile application remains consistent, user behavior while interacting with the User Interface also remains consistent.* For example, the way a user types email into the text box of an email application is different than the way a user types the password into a mobile banking application. Such observations motivated an investigation into checking if this behavior is consistent enough for one user while also being discriminant enough to distinguish among different users. We divided this investigation into two separate problems described in the following subsections.

3.1.1 Authentication

This problem can be phrased as follows :

> Given a training set containing samples for a legitimate user and a set of impostors, classify UI interactions extracted from every app usage session as belonging

to the legitimate user or an imposter.

3.1.2 Identification

This problem can be phrased as follows :

> Given a training set containing samples for a set of enrolled users, classify UI interactions extracted from every app usage session as belonging to a user among the set of enrolled users.

3.2 User Interface Design using Android Input Controls

Android considers all UI elements that allow users to interact with them while dynamically providing visible feedback to users as *input controls*[1]. For the purposes of this study, nine different types of input controls were used to capture interactions with users.

As shown in Figure 3.1, nine different types of input controls were used for this study. These along with their corresponding interactions recorded for each user are shown in Table 3.1.

Input Control	Functionality of Input Control
button	tap to start next activity
checkbox	tap to select value
radiobutton	tap to select value
switch	tap or horizontal fling from left to right to toggle
togglebutton	tap to toggle value
picker	vertical fling to pick a value
edittext	tap to bring keyboard focus
spinner item	tap to select item
spinner button	tap to show drop down

Table 3.1: Android Input Controls used for data collection

As shown by Table 3.1, three different kinds of button controls were used viz. button(shown as the *Next* button in Figure 3.1), togglebutton and the spinner button used to initiate the spinner drop down. The *Next* button's function was to suspend the current

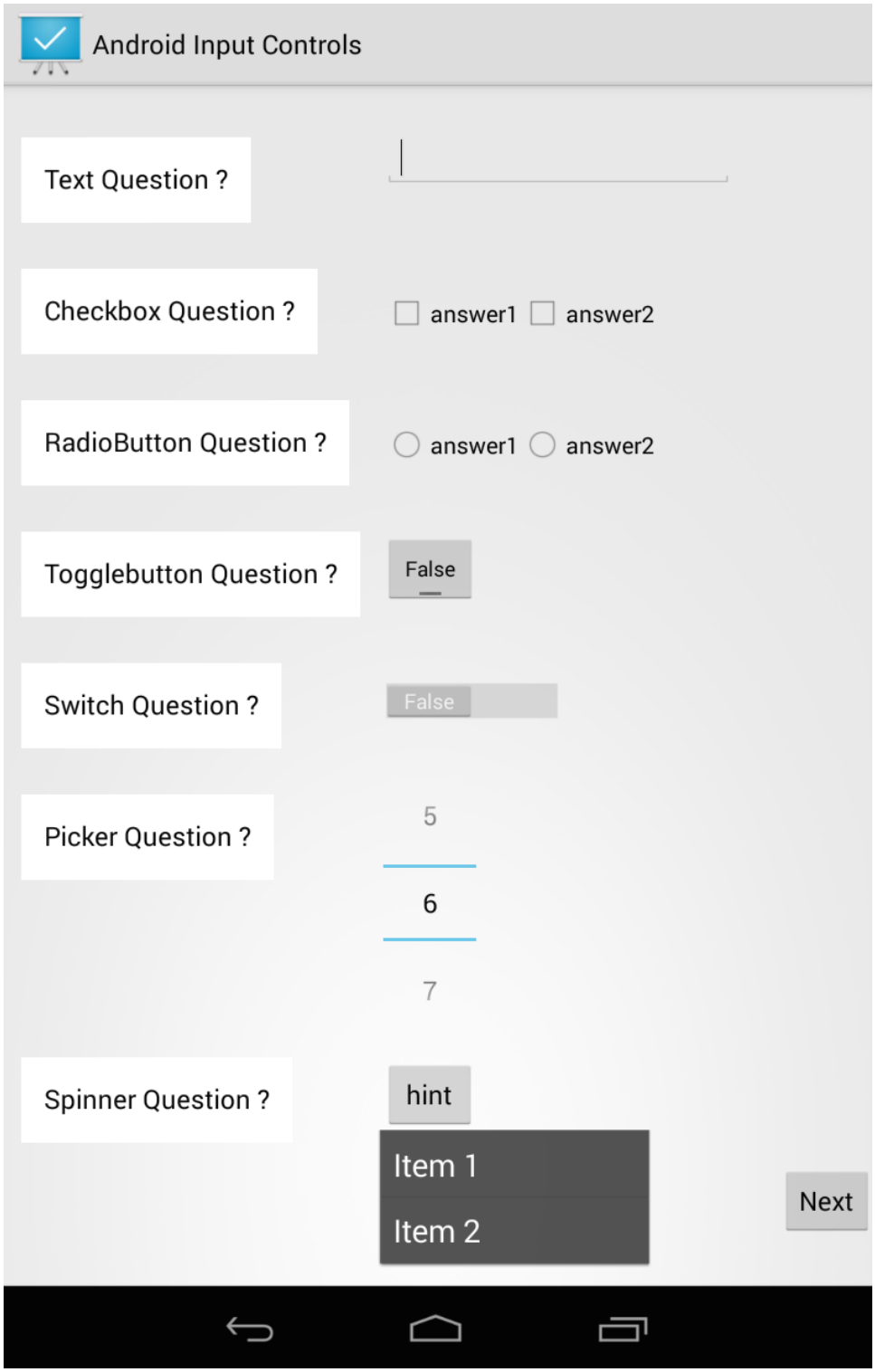

Figure 3.1: Android Input Controls used for data collection

Activity[9] of the app and start the successive *Activity*, the togglebutton was used to change a value from *True* to *False* and vice versa, the spinner button was used to trigger a drop

down menu from which the user could select a value. Since all these three controls performed three different functions, all interactions performed with these three types of buttons were treated separately.

3.3 Continuous Authentication and Identification using UI Interactions

The general idea behind verifying UI interactions for authenticating and identifying users as a new modality is to train on a user's behavior with the UI of an application during the initial sessions when the application is being used and then during later usage sessions, authenticate or identify the user.

3.3.1 Enrollment Phase

During this phase, all interactions of a user with any element that is part of the UI of an applications are logged. As many features that can be derived from every interaction are logged. The number of features that can be derived from every interaction depends on the API exposed by the underlying operating system. Most of the features we logged for every interaction could be derived from the Android GestureDetector[11] class. The number of usage sessions to be used for enrollment can be determined by the application developer. In Section 7, we will discuss the effect of changing the number of input controls on authentication and identification accuracy.

3.3.2 Authentication / Identification Phase

During this phase, every interaction of a user with any element of the UI is classified. The classification task depends on whether the final decision is one of authentication or identification. At this stage one critical assumption is made : *Every interaction performed during the same session of usage of an application must have been performed by the same user*. Hence, all interactions performed during the same session can be combined to get one final classification decision. In case of authentication, the entire session is classified as having been performed by either the legitimate user or the imposter. In case of identification, the

entire session is classified as having been performed by one user among the set of known users.

Chapter 4

Data Acquisition

In this section, we describe the data acquisition procotol followed when collecting data for UI interactions.

4.1 MSU SIRB Approval for Data Collection

The protocol followed for collecting data for this study can be found in the documents approved by the Michigan State University - Social Science Behavioral/Education Institutional Review Board (SIRB) for IRB# 15-277. The data collection for this study began on April 23, 2015 and ended on May 22, 2015. As part of this protocol, users were first asked if they wished to volunteer for this data collection procedure. Willing volunteers were then explained the data collection procedure and the three levels of authorization. The data collection as stated in Section 4.3 was then performed from every consenting volunteer. One of the objectives of the design of the data collection applications was to prevent anyone with access to the data from identifying any of the volunteers who participated in this study.

4.2 Design of Android Application for Data Collection

An Android device was chosen for this data collection exercise. Given the nine types of input controls shown in Table 3.1, it was determined that every user should interact with 15-20 instances of each type of input control during every application usage session. In addition, it was determined that data from five such usage session should be obtained from every user.

This value of collecting data for five sessions was set to find the right balance between obtaining sufficient training and test data and keeping user frustration within reasonable limits. The questions were created in a manner which prevented any personal or private information being entered by the users while still provoking some thought in users before an interaction with an input control was begun. For example, one of the questions asked was : "Does March precede April ?". These design principles were used to closely mimic usage in practical scenarios wherein users will typically stay aware of their interactions with the UI of an application. An attempt was made to keep the type of questions similar in every *Activity* of the application in order to reduce the time required to complete each session of data collection.

4.2.1 Intra-session Consistency of User Interface

Users interacted with 15-20 instances of the nine different types of input controls during every app usage session. An attempt was made to maintain consistency in user interactions with every type of input control.

For every input control, an attempt was made to keep all instances of the same input control vertically aligned. Also, all instances of input controls of the same type were kept of the same length and breadth to provide the same amount of area for each user to interact with the input control. We describe the design decisions made to maintain consistency among all interactions with the same type of input control.

- *Edittext*: We captured the touch interaction done by each user to bring keyboard focus to it.
- *Button*: The only button instances were recorded from the *Next* button shown in the bottom right corner of Figure 3.1.
- *Checkbox*: As shown in Figure 3.1, instances of both Checkbox were placed in two columns. In order to collect data from both placements of Checkbox instances, equal

number of correct answers were placed in both the columns.

- *Radiobutton*: Radiobutton instances would allow only have one possible selection in their group. Two instances of Radiobutton were included in each group and equal number of correct answers were placed in both the 1st and the 2nd Radiobutton instance in each group.

- *Spinner*: Placement similar to Radiobutton was used when designing the Activity containing instances of Spinner. Instances of Spinner were created to simulate the functionality provided by Spinner in order to collect data not only from the Spinner item selection but also from the initial button click to initiate the Spinner drop down menu.

- *Togglebutton*: Togglebuttons are different from other input controls because they can have only two possible values, one of which will be set as the default value in the Togglebutton instance. We needed to make users interact with Togglebutton instances in a natural way instead of simply changing the state of every Togglebutton instance. For this purpose, we created a set of 20 questions for which the answers were either *True* or *False*. 10 of these questions had a correct answer of *True* and the remaining 10 had a correct answer of *False*. We manipulated the Togglebutton instances such that 13 of the 20 Togglebutton instances required the users to change the state of the instance when attempting to answer correctly.

- *Switch*: We attempted to limit users to toggle the switch from left to right during the first interaction. Switches can be toggled by either tapping anywhere within the visible area of the Switch or by sliding the Switch to the intended state using horizontal fling gestures. While most users started by toggling switches using horizontal fling gestures, many of them resorted to using tap gestures with switches in the later app usage sessions.

- *Picker*: Every user was asked to pick a value from a given set of values by doing vertical flings but the set of picker values would circle back to the 1st value after the last value was flinged through. However, every picker would have an initial value set before the user's interaction. The target value was intentionally kept equidistant from the initial selected value in the picker.

4.2.2 Inter-session Consistency of User Interface

Among the five applications used for data collection, questions were designed for the only the first application. The order of the questions was changed in order to create the remaining four applications. This design decision was take primarily to keep the UI of all the applications consistent and to prevent users from memorizing answers to the questions they were being asked by the data collection applications. This design of the data collection app made users think at least momentarily before interacting with the questions's corresponding input control for providing an answer. Since most input controls were vertically aligned, changing the order of the questions changed the applications enough to prevent memorized interactions without changing the UI of the applications.

4.3 Data Collection Procedure

The questions were intentionally made simple to answer to keep the level of frustration low among volunteers but at the same time the questions were thought provoking enough to prevent users from memorizing an answer without reading the complete question. This procedure also helped make the data collection applications more realistic since during real Android application usage, users generally read and think before touching on any part of the screen. No private questions were asked as part of the data collection exercise. An example of a question asked: "Which letter comes after the letter a ?" Every user was identified by a subject number written in the consent form.

For every touch action performed on an input control instance, features from only

the user's first interaction with that input control instance were extracted and used, e.g. While answering a question, if a user selected the 1st checkbox on a page of an application and later changed their mind and deselected that checkbox, features from only the 1st interaction were extracted. The data collection procedure started with getting the user's written consent, then the users were asked to answer the questions in the data collection applications one after the other. Every touch interaction performed by the user with the data collection applications was logged. The only instruction given to each user before beginning the data collection procedure was to use the applications as normally as they could. This method of data collection helped capture variance in user behavior since some users changed sitting positions after the first two usage sessions, some users would stop switching the finger being used to interact with the application. The actual answers being given by each user were ignored and only touch interactions performed by each user with each input control were logged. After every app usage session, users were asked to hand over the mobile device back to us and the next app usage session would begin after a couple of minutes. Thus, users had a chance to reorient themselves to the mobile device at the beginning of every session which helped capture inter-session variance in user behavior.

4.4 Data Statistics

Table 4.1 shows some statistics on the total amount of data collected for this exercise.

The number of touch events for buttons is much higher than other input controls because button touch events were obtained from users in two ways

1. Users were asked to touch a button on the bottom right corner of every page of the application to move to the next page and each application usage session consisted of 11 such "next buttons".

2. Users had to touch a "spinner button" in order to cause every spinner drop down to show up and each application usage session consisted of 20 spinners.

Input Control	Number of Samples	Avg. samples per usage session
button	2306	11
checkbox	4849	23
radiobutton	4265	20
togglebutton	2880	14
switch	3416	16
edittext	3926	19
spinner	4357	21
spinner-button	4461	21
picker	3352	16
Total	33812	

Table 4.1: Data Set collected from 42 users

4.5 Data Collection Device

All data collection was done on a single Android device - the Google Nexus 7(2012) running Android 4.4.3.

Chapter 5

Classification Framework

5.1 Ensemble Classification

The presence of multiple instances of nine different types of interactions during each application usage session motivated the use of nine different classifiers. In addition, regardless of whether the problem of authentication or identification is being addressed, the final classification decision should be the same for the entire session given the assumption of the same user performing all the interactions seen during a session, thus further motivating the use of an ensemble classifier. These two decisions for taking an ensemble in order to classify every session for both the authentication and identification problems were done as follows.

- An interaction with every instance of an input control type was first classified by the corresponding input control classifier, e.g. interactions with instances of checkbox were classified by a checkbox classifier, interactions with instances of button were classified by a button classifier.

- All interactions belonging to the same session were combined by each input control classifier. In case of the authentication problem, a threshold value was used with every input control classifier to chose a final prediction. In case of the identification problem, the predicted class with the most number of votes for the entire session was chosen by every input control classifier.

- The most frequently predicted class among the set of predictions from the input control classifiers was chosen as the final prediction for both the authentication and identification problems

The design of the ensemble classifier can be summarized in Figure 5.1.

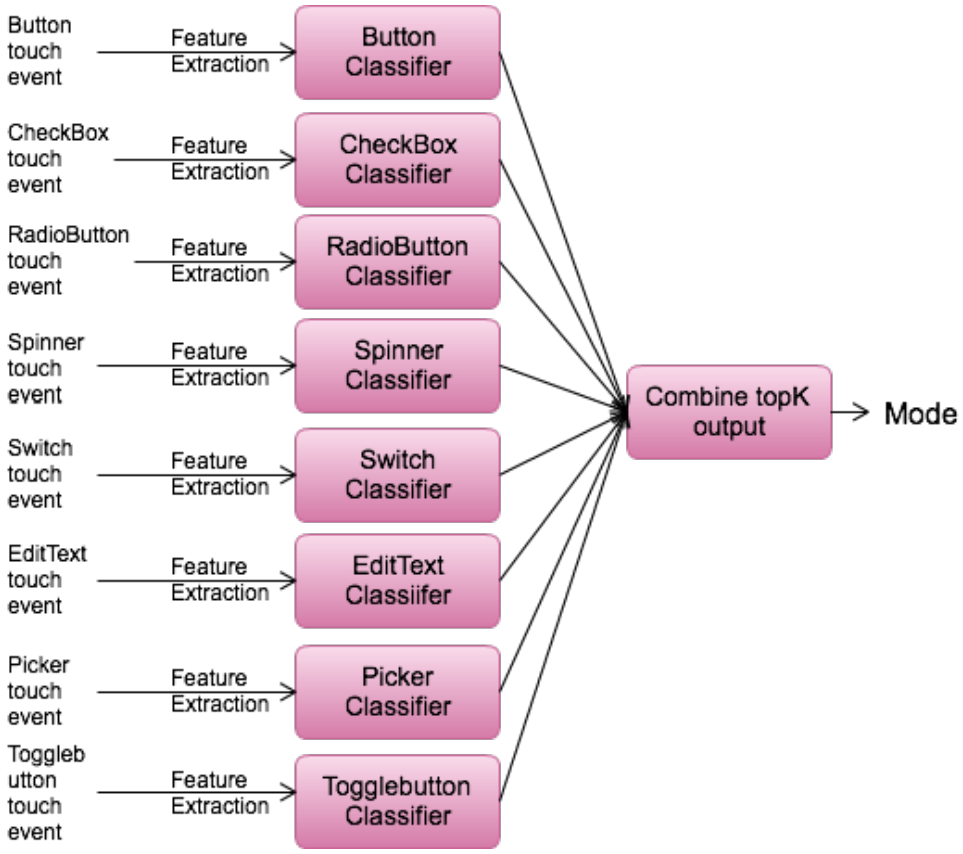

Figure 5.1: Ensemble Classification framework for user authentication and identification using Android input controls

The classification framework detailed below makes use of a key assumption: *All the interactions being performed during an app usage session are being performed by the same user.* This assumption leads to robustness in the ensemble classifier because all the interactions have to be ultimately classified as belonging to the same class. Hence, all classifications performed on all input control interactions have to be ultimately classified as belonging to the same class and hence can be combined into a single final decision. This ensemble classification was performed differently for authentication and identification.

5.1.1 Authentication

Every input control classifier classifies every interaction as belonging to either the legitimate user or an imposter. An assumption made during this evaluation is that there will be only one legitimate user for the device and all other users are assumed to be impostors. Since all test interactions coming from the same app usage session **must** belong to the same class, the number of classifications for all test interactions derived from the same type of input control and from the same session is calculated and compared against a *threshold* parameter. If this number falls below the threshold, then the input control classifier choses to predict the *legitimate* class for the entire session, else it choses to predict the *imposter* class for the entire session. Finally, the predicted class from every input control classifier from the same app usage session are combined to predict a final predicted class by finding the first most frequently occurring element(mode) for every app usage session.

5.1.2 Identification

Every input control classifier first classifies every interaction as belonging to one of 42 classes by creating 861((42*41)/2) classifiers, running all test interactions through the 561 classifiers and adding up votes for every test interaction. In addition, since all test interactions coming from the same app usage session **must** belong to these same class, the votes for every test interaction belonging to the same session are added up. At this stage, instead of choosing only a single class with the most votes, the top K classes are chosen as the most probable classes as predicted classes for all the test interactions coming from the same app usage session. This procedure is done for every set of test interactions performed on the same input control. Thus, with nine different types of input controls, we get the top K classes from each of the nine input control classifiers. Finally, the top K classes for every set of interactions from the same app usage session are combined to predict a final predicted class by finding the first most frequently occurring element(mode) for every app usage session. This classification framework is also illustrated in Figure 5.1.

5.2 Feature Set

The feature set explored for every touch interaction is as follows.

1. x coordinate of touch gesture start
2. y coordinate of touch gesture start
3. finger pressure at touch gesture start as reported by Android
4. finger size at touch gesture start as reported by Android
5. distance travelled by the finger along the x axis between touch gesture start and touch gesture end
6. distance travelled by the finger along the y axis between touch gesture start and touch gesture end
7. difference between finger pressure between touch gesture start and touch gesture end
8. difference between finger size between touch gesture start and touch gesture end
9. euclidean distance travelled between touch gesture start and touch gesture end
10. direction travelled in between touch gesture start and touch gesture end
11. time lapse between touch gesture start and touch gesture end

Table 5.1 captures the feature subset found to be most useful for classification. The columns indicate the classifier associated with each input control type, rows indicate if the feature was used by the classifier or not.

Feature selection was done by using Sequential Forward Selection. As can be concluded from Table 5.1 the difference in coordinates, distance, finger pressure, finger size and direction of touch were not found to be particularly useful. The x coordinate was particularly discriminant for all users and so was the amount of time each user spent in touching the

Feature	button	checkbox	spinner-button	radiobutton	togglebutton	edittext	spinner-item	switch	picker
x coordinate	✓	✓	✓	✓	✓	✓	✓	✓	✓
y coordinate	✓	✓	✓	✓	✓			✓	✓
finger pressure	✓	✓	✓	✓	✓	✓	✓	✓	✓
finger size	✓	✓	✓					✓	✓
delta-x									
delta-y								✓	
delta-pressure									
delta-size									
euclidean distance									
direction									✓
delta-time	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.1: Feature subset used for different input control classifiers

screen. All feature values were reported by the Android GestureListener except for time. Time was logged in milliseconds when a touch gesture start was reported and logged again when a touch gesture end was reported on the same input control. This difference was captured for every touch interaction and used. The y coordinate not being found to be useful in case of the edittext input control can be intuitively explained by the fact that when most users touch an instance of an edittext in order to bring keyboard focus to it, the width of the edittext instance was much more than the height of the instance thereby providing users a bigger range to target in the width than the height. In case of the switch input control, the delta-y feature being found to be useful can be explained by the fact that users were required to tap or fling the switch horizontally from left to right. These two different ways of interacting with the switch input control resulted in different users moving it in different ways along the height of the switch instance. Similar insight can be applied in case of the picker input control wherein the direction of the fling turns out to be useful.

While scrolls and keystrokes performed by users were also captured as part of this data collection process, classification based on scrolls and keystrokes on mobile devices are already explored techniques and hence were not used in classification by input controls.

5.3 Evaluation Methodology

The evaluation of the ensemble classifier for authentication was done in an entirely different manner than that for identification. We discuss both evaluations in the following two

sections. All Support Vector Machine[3] based classifiers were implemented using LibSVM[2].

5.3.1 Evaluation for Authentication

When authenticating a user to a device, we had data from 42 users, each user providing data from five application usage sessions. In order to correctly evaluate the performance of the ensemble classifier for authentication, we had to provide each of the 42 users a chance to be the legitimate user and allow the remaining users to attack the ensemble classifier. However, we also had to divide the imposter user set into training and test impostors. In addition to this, the *threshold* parameter value also affects the performance of the ensemble classifier. In order to accommodate all these factors into evaluating the ensemble classifier for authentication, we calculated the mean False Acceptance Rate(FAR) and False Rejection Rate(FRR) for every possible value of *threshold*, test usage session, legitimate user, training impostors and test impostors. Three different base one-class classifiers were tried for the ensemble classification task. We explain these in the following three sections. In all three base classification algorithms, any impostors who were used for training were not used for testing. This method of choosing impostors during the test phase closely resembles the real world scenario wherein unknown attackers on whom the ensemble classifier has not been previously trained may try to attack the application. The results from this evaluation are reported in Section 6.1.

5.3.1.1 Support Vector Data Descriptor

The Support Vector Data Descriptor(SVDD)[24] tries to fit a hypersphere in feature space around the training data provided for the legitimate user while also minimizing the volume of this hypersphere. Parameters for this classifier were found by doing 4-fold cross validation since training data was from four app usage sessions.

5.3.1.2 One Class Support Vector Machine

The One Class SVM-based classifier tries to create a hyperplane in feature space allowing all the training points to lie on one side of the hyperplane. Parameters for this classifier were found by doing 4-fold cross validation.

5.3.1.3 Two Class Support Vector Machine

The standard Support Vector Machine[3] based classifier was used wherein four usage sessions of the legitimate user were chosen to create training set for the *legitimate* class. Training data for the *imposter* class was created by choosing one usage session from four impostors. Thus equal number of samples for both the classes were used during training. For testing, the remaining usage session from the *legitimate* was used and all the remaining users who were not part of the *imposter* class during training were used for testing.

5.3.2 Evaluation for Identification

The performance of the ensemble classifier for multi-user identification can simply be considered to be the accuracy of the classifier when classifying each of the 42 test application usage sessions. Since data was collected from five application usage sessions, there are five possible test usage sessions which could be used. Considering each of the five application test usages one at a time, the remaining four app usage sessions were used for training and the mean accuracy of the classifier was calculated. three different base classification algorithms were used to test the performance of the ensemble classifier. In all three cases, the same base algorithm was used for all the nine input control classifiers and the *1-vs-1* multi class classification algorithm was used for all the three base classification algorithms. The results from this evaluation are reported in Section 6.2.

5.3.2.1 Support Vector Machine-based classifier

A Support Vector Machine-based classifier[3] was used to distinguish between samples of two classes. The Radial-Basis Function was used as the kernel function and parameters for the base classifier were found by doing 4-fold cross validation since training data was collected from four different usage sessions.

5.3.2.2 Gaussian Discriminant Analysis-based classifier

A Gaussian Discriminant Analysis-based classifier was created by estimating the parameters - mean and covariance - from the training data. No cross validation was required

for this base classifier. This classification technique had the advantage of not having to store the training data after the initial parameter estimation and was found to be faster than the SVM and 3NN based ensemble classifier.

5.3.2.3 3-Nearest Neighbor-based classifier

A 3-Nearest Neighbor-based classifier was tried as the base classifier for the ensemble. No training and cross validation phase was required but this technique has the disadvantage of having to store the training data at the time of classification. The ensemble classifier using only this technique was found to be the slowest of the three ensemble classification techniques.

5.4 Ensemble Parameters

An ensemble of the candidate input control classifiers was required for the authentication and identification problems. We discuss here the parameters introduced to achieve this ensemble. Parameters which were required for the base classification algorithms are discussed in Section 6. One parameter which influences the accuracy of the ensemble classifier for both authentication and identification is the number of interactions from the same session available per input control classifier.

5.4.1 Authentication

Every candidate input control classifier combines classifications for interactions from the same session into one predicted class which can be either *legitimate* or *imposter*. The total number of classifications for the *legitimate* class are summed up and compared against a *threshold* parameter. A plot showing the change in False Acceptance Rate(FAR) and True Acceptance Rate(TAR) is shown in Section 6.1. This is the only parameter which is required for the entire ensemble classifier.

5.4.2 Identification

The most important parameter for this classification framework is the value chosen for K. Choosing a large value of K causes too many incorrect predictions to be included in the

final ensemble classification while choosing too small a value requires the candidate input control classifiers to be very accurate in reporting their predictions. Another parameter which influences the accuracy of the test app usage session is the number of test samples available per input control in the test app usage session. We evaluate the influence of both these parameters in Section 6.2.

Chapter 6

Experimental Results

We evaluate the ensemble classifier for authentication and identification problems and present our results in the following sections.

6.1 Authentication

We evaluated the performance of the ensemble classifier using three different base classifiers - SVDD, One Class SVM and Two Class SVM. Every usage session of an assumed *legitimate* user was tested on once while the *impostor* class was tested on impostor data not seen during training. Figure 6.1 shows the mean True Acceptance Rate(TAR) plotted against the mean False Acceptance Rate(FAR) seen while varying the *threshold* parameter value from 0 to 40 for the ensemble classifier based on the three base classifiers. For every technique, all interactions available from the test usage session are used to reach a final predicted class. It can be seen that the Two Class SVM technique gives us the best overall performance while SVDD has the worst performance among the 3. However, all the three techniques perform better than a random guess at the final predicted class demonstrating that the level of discriminant information presented by this modality. Figure 6.1 can be used by application developers to strike the balance between the number of impostors that may successfully attack the ensemble classifier vs. the number of times a legitimate user will get rejected by the ensemble classifier. An example of such a balance can be seen at the value of FAR being 5%, a TAR of 76% is obtained. In other words, the authentication framework

incorrectly admits only 5% of all impostors while still admitting the legitimate user 76% of the time.

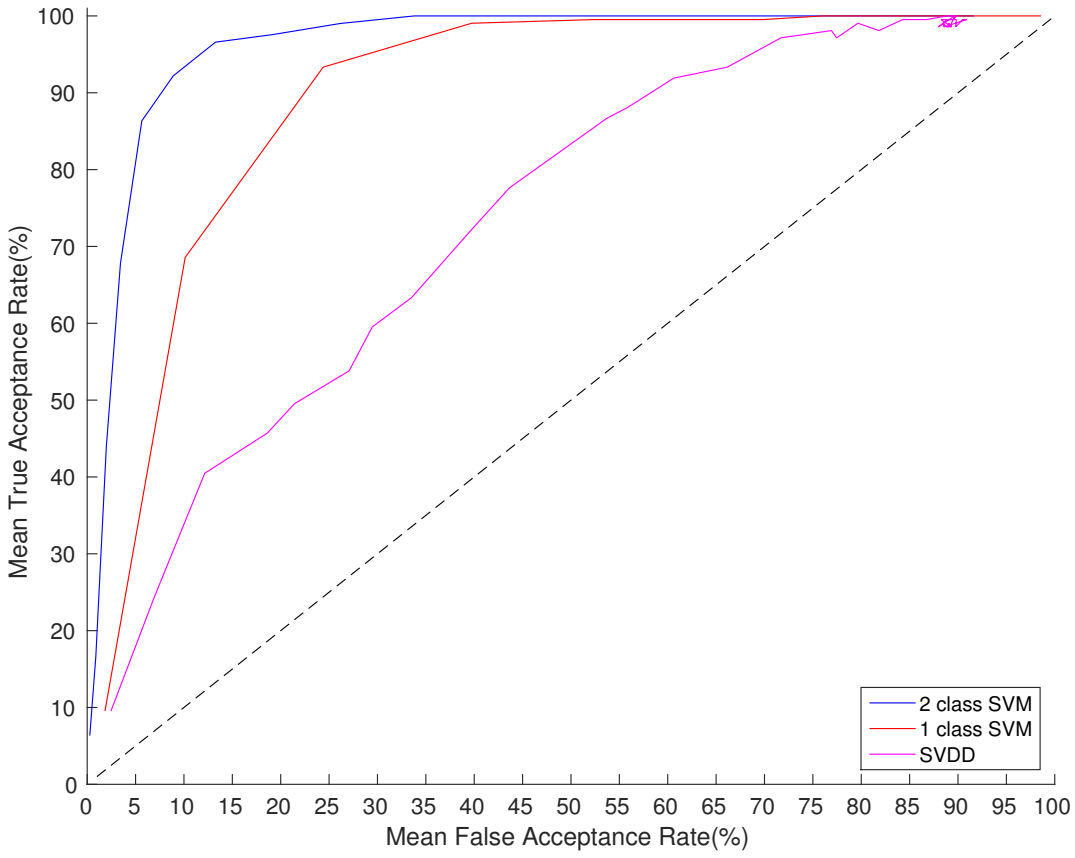

Figure 6.1: ROC curve comparing the ensemble classifier when using SVDD, One Class SVM and Two Class SVM as base classifiers

A metric which indicates true performance of the classification scheme is called the *Equal Error Rate(EER)*. This value for the accuracy of the classifier is the value at which the probability of a legitimate user being rejected *equals* the probability of an imposter being accepted by the ensemble classifier. Figure 6.2 shows the mean FAR and FRR seen at different values of the *threshold* parameter for the ensemble classifier based on the three base classification algorithms - SVDD, One Class SVM and Two Class SVM. It can be seen that the ensemble classifier using Two Class SVM as a base classifier achieves the lowest EER of 7%, with the ensemble classifier using One Class SVM coming 2nd at 16% and the SVDD-

based ensemble classifier achieving an EER of 30%. It can also be seen that the EER for both One Class SVM and Two Class SVM based approaches is reached at smaller *threshold* values of three and six respectively whereas the EER for the SVDD-based approach is reached at a *threshold* value of 14. The *threshold* parameter states how many interactions in the session for the input control are allowed to be classified as the impostor before the input control classifier changes its decision. Thus, Two Class SVM based ensemble classification works best among the 3 techniques correctly classifying both legitimate users and impostors 93% of the time.

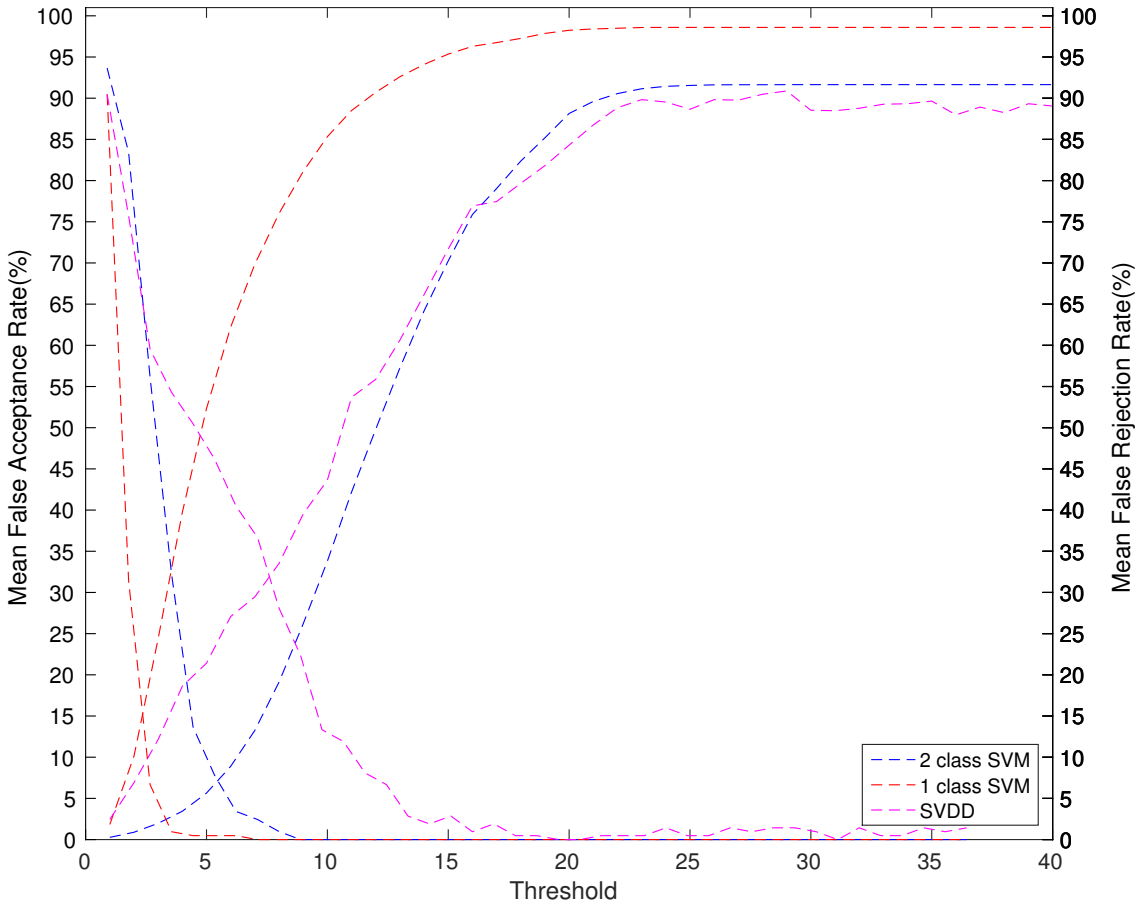

Figure 6.2: Equal Error Rate for the ensemble classifier when using SVDD, One Class SVM and Two Class SVM as base classifiers

6.2 Identification

We evaluated the accuracy of the ensemble classifier using three different base classifiers - K-Nearest Neighbor, Support Vector Machines(SVM) using the RBF kernel and Gaussian Discriminant Analysis(GDA) on the usage performed by 42 different users on these five applications. Figure 6.3 shows the median and variance observed in the test error for different kinds of classifiers when all five of the app usage sessions were used. It also shows the median and variance in the test error when only the last three of the five app usage sessions were used. When five app usage sessions were available, four of them were used for training and one was used for testing. The test app usage session could be switched around five times. When only three app usage sessions were available, the test app usage session could be switched around three times. For K-Nearest Neighbors, a value of three was used for the number of neighbors to be checked for every incoming test sample. In case of SVM, a Radial Basis Kernel was used where the parameter values for C and γ were selected by cross validating on one app usage session and training on the remaining app usage sessions, e.g. When all five app usage sessions were used for evaluation, one session was selected as the test session and of the remaining four sessions, three were used for training and one was used for cross validation. It can be observed from Figure 6.3 that the SVM classifier and GDA classifier reported the least error among the three classifiers. However, the accuracy given by the SVM classifier is marginally better than the accuracy reported by the GDA classifier. Further, Figure 6.3 also shows the variance in the test error for the GDA classifier is less than SVM classifier in both cases (last three app sessions and all five app sessions). The median test error reported for the 3-Nearest Neighbor(3NN) classifier is consistently higher than both the GDA and SVM classifiers but it has the distinct advantage of being simple to implement and use. Figure 6.3 also shows that the median error reported for the last three app usage sessions was consistently less than the median error observed for all five app usage sessions irrespective of the choice of the classifier. This observation shows that user behavior had more variance during the first two app usage sessions than the last

3. We can also observe the variance in test error for the 3NN classifier for five app usage sessions being much more than the variance in test error for the last three app usage sessions.

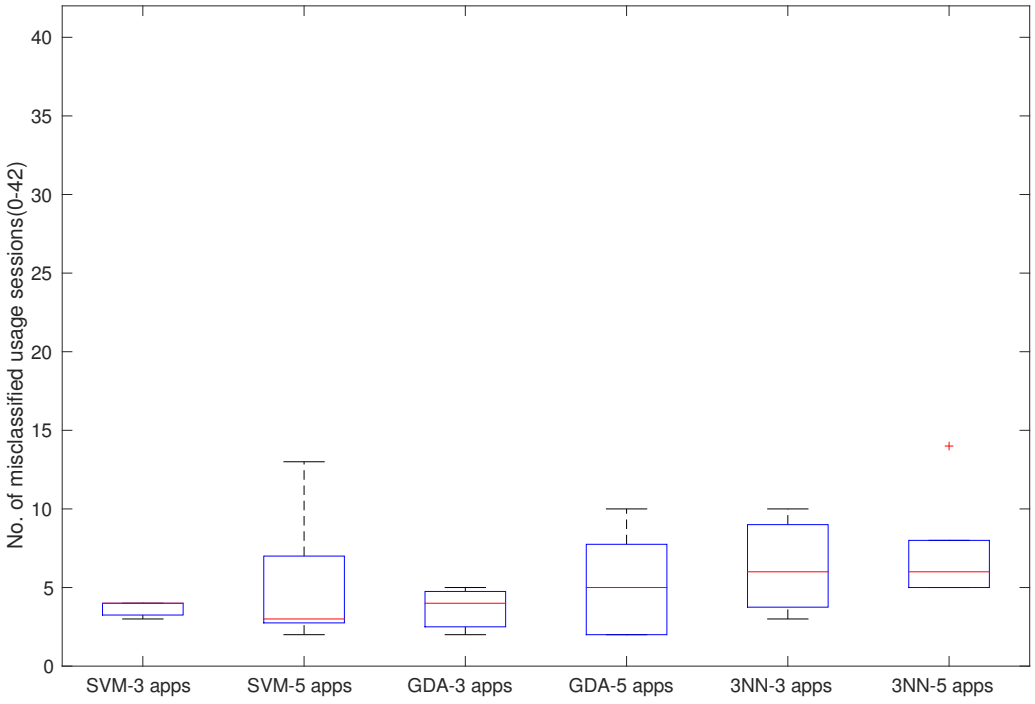

Figure 6.3: Box plot showing identification accuracy of the ensemble classifier based on Gaussian Discriminant Analysis, Support Vector Machine and 3 Nearest Neighbor classifiers

Figure 6.4 shows how accuracy changes as the number of enrolled users is increased. The minimum value of 11 users was used because the top seven predicted classes from each candidate input control classifier were combined into the ensemble classification in case of the 3NN classifier while the top three predicted classes from every candidate classifier were used in case of GDA. The accuracy of GDA and 3NN classifiers is shown in Figure 6.4 as the number of enrolled users changes from 11 to 42. It can be observed that the accuracy of the ensemble does not vary much as the number of users increases. Figure 6.4 demonstrates that the ensemble classifier is capable of correctly classifying a large number of users without suffering a degradation in classifier accuracy even with a weak candidate classifier such as

3NN.

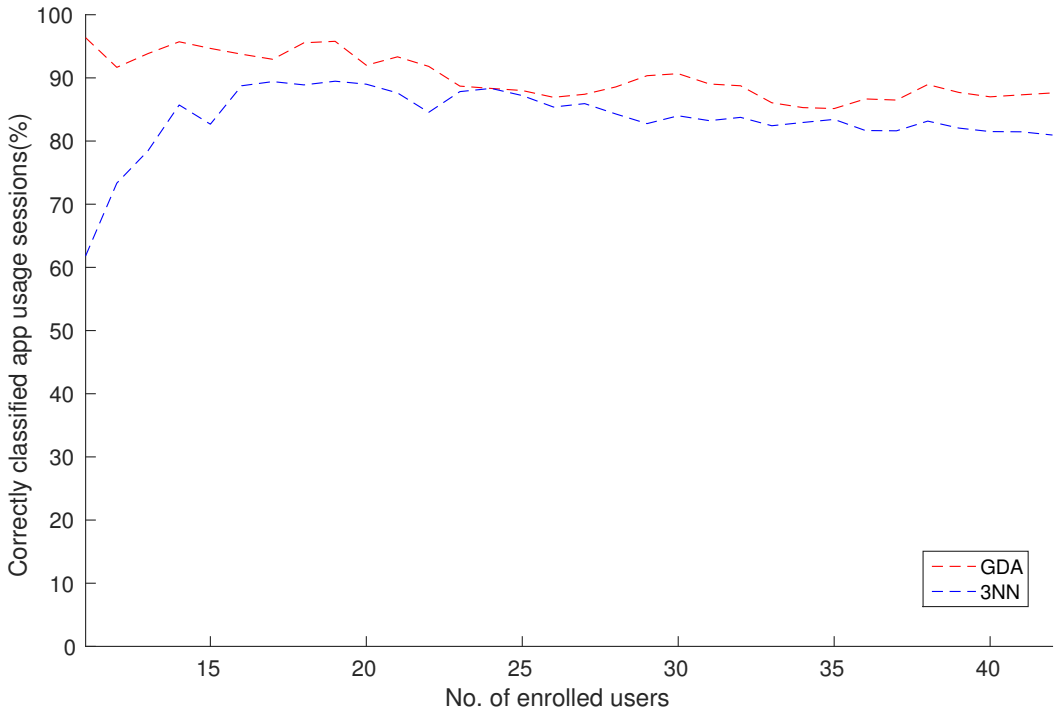

Figure 6.4: Change in Identification accuracy of ensemble classifier with increasing number of enrolled users. The ensemble classifier remains consistently accurate with change in user count.

Nine candidate input control classifiers were used as part of the ensemble. From each classifier the top K most probable classes for the app usage session were chosen. In order to examine the effect of this parameter on the overall accuracy of the classifier, we plot the number of misclassified app usage sessions vs. the chosen parameter value of K in Figure 6.5. This figure clearly shows that small values of K ranging from one to five are capable of giving a good ensemble classification. However, as the value of K starts to increase, more incorrect predictions from the candidate classifiers are used by the ensemble classifier in its final prediction causing its accuracy to degrade. The total number of available users was 42 and therefore, the value of K was adjusted from 4 to 42.

In order to investigate how heavily the ensemble classifier relies on the number of available test samples per input control, we increased the number of test samples in an app usage

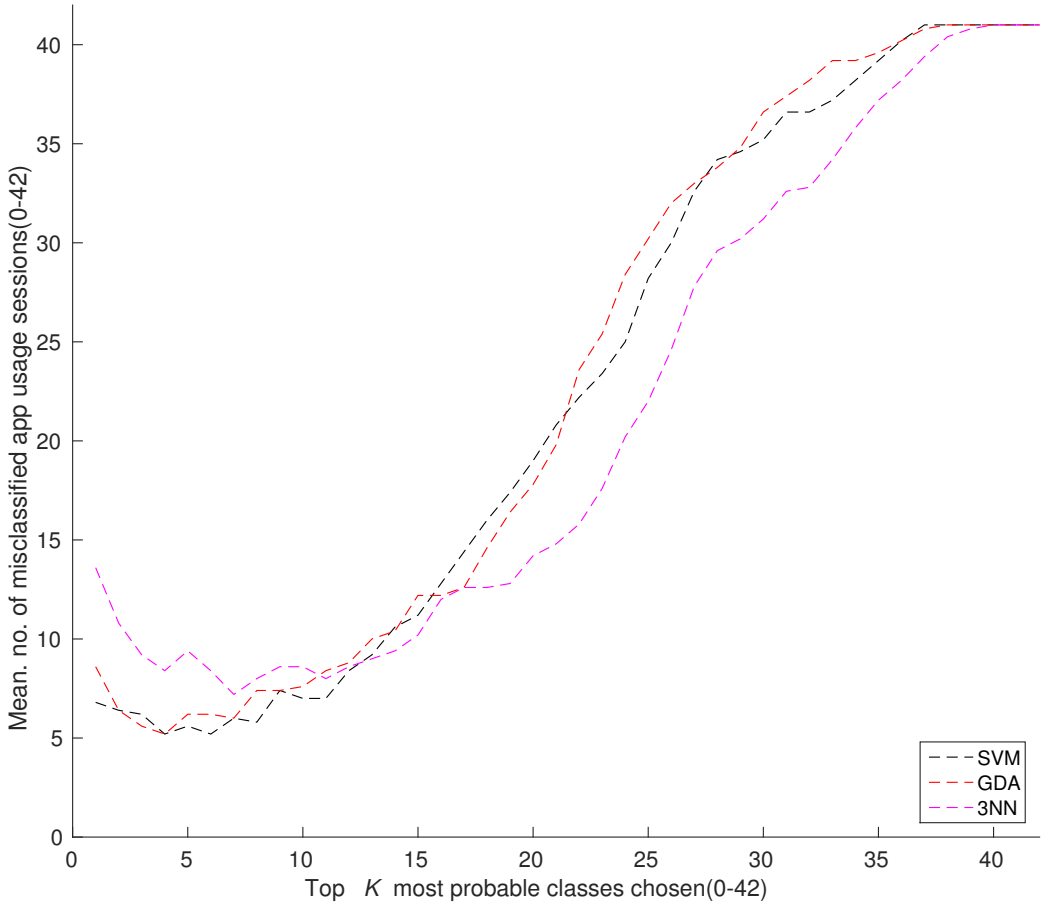

Figure 6.5: Number of misclassified app usage sessions vs. top K predicted classes from candidate classifiers

session from one up to 32 and measured the average test error reported by the ensemble classifier when the top three classes(K=3) were provided as output by each of the candidate classifiers. Figure 6.6 shows such a plot. It can be seen that even with a single interaction per input control, when using Gaussian Discriminant Analysis, around 26 of the 42 app usage sessions are still correctly classified.

6.3 Evaluation on Real-World Application

MSUPaths[25] is a mobile application available on both the iOS[13] and Android[10] operating systems. This application allows users to navigate between any two buildings

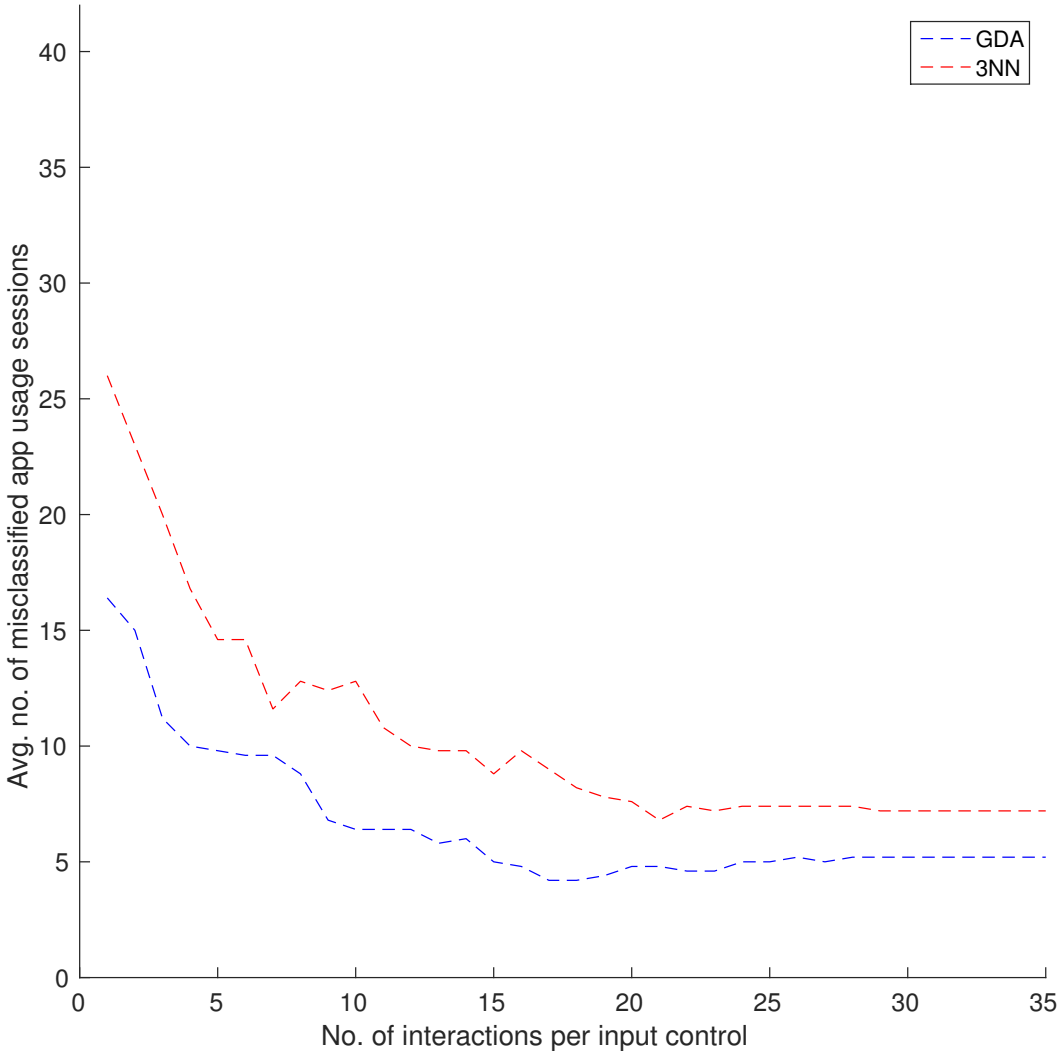

Figure 6.6: Input control interactions affect the number of misclassified sessions. Top three most probable classes were taken from candidate input control classifiers, GDA and 3NN classifiers were used, all five app usage sessions were used to get the average test error

on the Michigan State University campus. The user can select a building from a list of buildings loaded in the application and the application loads a set of directions from the user's current location to the building selected by the user. We modified the application to listen for gestures on four different input controls part of the MSUPaths user interface and asked users to navigate to 20 different buildings. Users were provided detailed instructions on

how MSUPaths is to be used for navigation and data was collected from 10 volunteers. The average data collection time was found to be 10 minutes. We used the first 19 navigations for training and the last navigation for testing and found the identification accuracy to be 100%.

Chapter 7

Discussion

Ethical Implications during Data Collection

An advantage of this modality is that it is impossible for any user to escape from their interactions being captured by the mobile application. Every user *must* interact with the User Interface of an application in order to use it. Thus, every application becomes fully capable of capturing user data at all times regardless of which device it is being used on. Application developers who chose to use this modality will be required to sufficiently disclose this data collection to their users either at the time of app installation or the first app usage session during which the data collection is begun.

Influence of Input Control Type

Nine different types of input control classifiers were used in this investigation for both training and testing purposes. This decision was taken primarily to know if the largest available set of input controls is useful for this modality or not. However, different subsets of these input control types are used by mobile application developers in the UI of their applications. It would be useful to know which input controls offer more discriminant information than others so that a usability vs. security tradeoff can be reached by application developers.

Influence of Parameters during Ensemble Classification

Both authentication and identification ensemble classifiers had their own set of parameters which required further fine tuning. Further investigation can be performed along the lines of finding which parameter values are more suited to different type of applications, e.g. a mobile banking application may prefer to allow only a small fraction of impostors which choosing to reject the legitimate user more often. This tradeoff depends on the functionality being provided by the mobile application.

Influence of Placement of Input Controls in the User Interface

During this investigation, we attempted to uniformly place input controls that were of the same type to ensure user interactions were not affected. For example, all Checkbox instances were vertically aligned to prevent the user's finger from traveling in the horizontal direction between interactions with different Checkbox instances. It should be further investigated if instances placed in a certain way provided more discriminant information when interacted with by the user.

Influence of Mobile Device

The same mobile device was used for the entire data collection exercise and the same instructor was involved in collecting data from all users. It needs to be further investigated if similar performance of the ensemble classifier for both authentication and identification is seen on other mobile devices as well. In addition, it also needs to be checked if any similarities in user behavior are seen in UI interactions across different devices.

Influence of Classification Algorithms

The same classification algorithm was used for all the nine input control classifiers when creating an ensemble classifier for both the authentication and identification classifiers. However, different combinations of base algorithms may improve the performance of the

ensemble classifier.

Chapter 8

Conclusion

A new modality for authenticating and identifying users from their usage of a mobile device was explored. Every available input control type on the Android OS was used to create base classifiers which were then used in an ensemble classifier. Data was collected from 42 users for this modality from five applications which had the same User Interface. Three different base classification algorithms were used for the ensemble classifier for both the authentication and identification problems. The best performance for authentication was found to have been achieved by a SVM-based ensemble classifier with a mean Equal Error Rate(EER) of 5%. Similarly, for identification, the best mean accuracy of 90% was reported by a SVM-based ensemble classifier. In case of identification, the SVM-based ensemble classifier was found to be robust when the number of enrolled users was increased and also when the number of available interactions per input control was varied.

In summary, this work provides a basic framework for authenticating and identifying users in a truly implicit and continuous way on mobile devices. While we find UI interactions to be useful for both authentication and identification, several avenues of further research for this modality still exist. This study does not make use of any UI interactions seen in other swipe-based and keystroke-based modalities so it would be interesting to integrate those techniques also into a bigger ensemble framework. Such a framework would truly integrate every possible interaction made by a user with a mobile application.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Input Controls — Android Developers. `http://developer.android.com/guide/topics/ui/controls.html`, 2015.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[4] Tao Feng, Jun Yang, Zhixian Yan, Emmanuel Munguia Tapia, and Weidong Shi. Tips: context-aware implicit user identification using touch screen in uncontrolled environments. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, page 9. ACM, 2014.

[5] Tao Feng, Xi Zhao, B. Carbunar, and Weidong Shi. Continuous mobile authentication using virtual key typing biometrics. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 1547–1552, July 2013.

[6] Tao Feng, Xi Zhao, and Weidong Shi. Investigating mobile device picking-up motion as a novel biometric modality. In *Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on*, pages 1–6. IEEE, 2013.

[7] Jordan Frank, Shie Mannor, and Doina Precup. Activity and gait recognition with time-delay embeddings. In *AAAI*, 2010.

[8] Mario Frank, Ralf Biedert, Eugene Ma, Ivan Martinovic, and Dawn Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. *Information Forensics and Security, IEEE Transactions on*, 8(1):136 –148, 1 2013.

[9] Google. Activity — Android Developers. `http://developer.android.com/reference/android/app/Activity.html`, 2015.

[10] Google. Android. `https://www.android.com/intl/en_us/`, 2015.

[11] Google. GestureDetector — Android Developers. `http://developer.android.com/reference/android/view/GestureDetector.html`, 2015.

[12] Eiji Hayashi, Oriana Riva, Karin Strauss, AJ Brush, and Stuart Schechter. Goldilocks and the two mobile devices: going beyond all-or-nothing access to a device's applications. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 2. ACM, 2012.

[13] Apple Inc. Apple - iOS 8. `https://www.apple.com/ios/`, 2015.

[14] Hilmi Günes Kayacık, Mike Just, Lynne Baillie, David Aspinall, and Nicholas Micallef. Data driven authentication: On the effectiveness of user behaviour modelling with mobile device sensors.

[15] Hilmi Gunes Kayacik, Mike Just, Lynne Baillie, David Aspinall, and Nicholas Micallef. Data driven authentication: On the effectiveness of user behaviour modelling with mobile device sensors. *arXiv preprint arXiv:1410.7743*, 2014.

[16] Hassan Khan, Aaron Atwater, and Urs Hengartner. A comparative evaluation of implicit authentication schemes. In *Research in Attacks, Intrusions and Defenses*, pages 255–275. Springer, 2014.

[17] Hassan Khan, Aaron Atwater, and Urs Hengartner. Itus: an implicit authentication framework for android. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 507–518. ACM, 2014.

[18] Hassan Khan and Urs Hengartner. Towards application-centric implicit authentication on smartphones. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, page 10. ACM, 2014.

[19] Oriana Riva, Chuan Qin, Karin Strauss, and Dimitrios Lymberopoulos. Progressive authentication: Deciding when to authenticate on mobile phones. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 15–15, Berkeley, CA, USA, 2012. USENIX Association.

[20] Premkumar Saravanan, Samuel Clarke, Duen Horng (Polo) Chau, and Hongyuan Zha. Latentgesture: Active user authentication through background touch analysis. In *Proceedings of the Second International Symposium of Chinese CHI*, Chinese CHI '14, pages 110–113, New York, NY, USA, 2014. ACM.

[21] Lookout Mobile Security. Phone Theft in America. `https://www.lookout.com/resources/reports/phone-theft-in-america`, 2014.

[22] Muhammad Shahzad, Alex X. Liu, and Arjmand Samuel. Secure unlocking of mobile touch screen devices by simple gestures: You can see it but you can not do it. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 39–50, New York, NY, USA, 2013. ACM.

[23] Weidong Shi, Jun Yang, Yifei Jiang, Feng Yang, and Yingen Xiong. Senguard: Passive user identification on smartphones using multiple sensors. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2011 IEEE 7th International Conference on*, pages 141–148, Oct 2011.

[24] David MJ Tax and Robert PW Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[25] Michigan State University. MSUPaths. `https://github.com/MSUPaths/MSUPaths_Android`, 2015.

[26] Jiang Zhu, Pang Wu, Xiao Wang, and Joy Zhang. Sensec: Mobile security through passive sensing. *2013 International Conference on Computing, Networking and Communications (ICNC)*, 0:1128–1133, 2013.