ABSTRACT

BRANSCOMB, ANDREW SCOTT. Behaviorally Identifying Smartphone Users. (Under the direction of Ting Yu and William Enck.)

As smartphones become increasingly capable and omnipresent, users will certainly continue to store highly sensitive personal information on them. A user has a strong desire to protect this information from access by a third party who picks up an unattended phone. To that end, many users opt to use a PIN or password to protect the device. These obtrusive authentication methods dramatically slow down the user who wishes to perform a quick task on the device. Smartphone locking systems should strive to protect the phone from unauthorized use while minimizing impact on the convenience of authorized use. The passwords entered on small virtual (or, less often, physical) keyboards are cumbersome and PINs or swipe patterns over a grid of points are easy to watch and duplicate. This thesis presents data supporting the hypothesis that user behavior on a smartphone can be used to identify that user. In this study we have collected application usage data from six users of Android phones. We have compared different users’ usage patterns in order to extract the identity of the user from his application usage. This ability is necessary to use behavior to authenticate a user. We hope that this could lead to a behavioral authentication mechanism in a smartphone system.
Behaviorally Identifying Smartphone Users

by

Andrew Scott Branscomb

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science

Computer Science

Raleigh, North Carolina

2013

APPROVED BY:

__________________________
Ting Yu
Co-chair of Advisory Committee

__________________________
William Enck
Co-chair of Advisory Committee

__________________________
Randy Avent
DEDICATION

To my family for supporting me and piquing my interest in computers.
BIOGRAPHY

The author was born in Manassas, VA, to two NCSU graduates. His interest in computers began at a young age. He attended Ravenscroft School from kindergarten through 12th grade. He did his undergraduate work in physics and computer science at the University of North Carolina at Chapel Hill and remains a Tar Heel fan. He then pursued this Masters degree from North Carolina State University. Hobbies include cycling and particle physics.
ACKNOWLEDGEMENTS

I would like to thank both of my co-chairs, Drs. Enck and Yu for their instrumental help in preparing this thesis. Their guidance has been invaluable in directing me in the proper ways to conduct this research.

I would also like to thank Dr. Avent for his help in preparation and him and his son Miles for the idea that led to this research.

I would like to thank Zach Jorgensen for his help with feature definition and machine learning.

I would like to thank the anonymous study volunteers who provided the data necessary for this research.

I would like to thank my grandfather for providing support throughout my educational career.

Finally, I would like to thank my parents for helping me in innumerable ways through my years of education.
# TABLE OF CONTENTS

LIST OF TABLES ............................................................................. vii

LIST OF FIGURES .......................................................................... viii

Chapter 1 Introduction ................................................................. 1

Chapter 2 Background ................................................................. 3
  2.1 Overview of Approach ......................................................... 3
  2.2 Android Background .......................................................... 4
    2.2.1 Application Components .............................................. 4
    2.2.2 Permissions ............................................................... 5
    2.2.3 Logs ........................................................................ 6
  2.3 Active Authentication ......................................................... 6
  2.4 Application Usage Behavior ............................................... 6
  2.5 Summary ........................................................................... 7

Chapter 3 Design ....................................................................... 8
  3.1 MouflonRecorder .................................................................. 8
    3.1.1 Activities .................................................................. 8
    3.1.2 Services .................................................................... 11
    3.1.3 BroadcastReceiver ................................................... 15
    3.1.4 Other ........................................................................ 15
  3.2 Server ................................................................................ 15
  3.3 Decryption ......................................................................... 16
  3.4 Data Preprocessing ............................................................. 16
    3.4.1 Cross-Validation ....................................................... 18
  3.5 Data Analysis Techniques .................................................... 18
  3.6 User Test .......................................................................... 19
  3.7 MouflonTester .................................................................... 20

Chapter 4 Results ..................................................................... 21
  4.1 Difficulties ......................................................................... 22
    4.1.1 Data Missing From Logs ............................................ 22
    4.1.2 Missing Power Events .............................................. 22
    4.1.3 Small Number of Users ............................................. 23
  4.2 IBk .................................................................................. 23
    4.2.1 User 5a5 .................................................................. 24
    4.2.2 User c59 ................................................................... 24
    4.2.3 User ca8 ................................................................... 25
    4.2.4 User cb8 ................................................................... 25
    4.2.5 User d48 ................................................................... 26
    4.2.6 User e35 ................................................................... 27
    4.2.7 Best and Worst Features ........................................... 27
Chapter 5 Related Work

5.1 Behavioral Profiling on Mobile Devices
  5.1.1 Symbian
  5.1.2 Windows Phone
  5.1.3 Ultra-Mobile PCs
  5.1.4 Android

5.2 Behavioral Profiling through Application Usage

5.3 Active Authentication Methods
  5.3.1 Actively Enhanced Authentication
  5.3.2 Active Biometric Authentication

5.4 Extensions
  5.4.1 Cloud Processing of Data
  5.4.2 Guest User Fallback

Chapter 6 Discussion

6.1 Simplifying Assumptions

6.2 Training Data
  6.2.1 Disjoint vs. Non-Disjoint intruder sets

Chapter 7 Conclusions

7.1 Extensions/Future Work
  7.1.1 Intra-application profiling
  7.1.2 Target profiling
  7.1.3 Application White- or Blacklisting
  7.1.4 Cloud processing
  7.1.5 Guest Mode
  7.1.6 Development as a System App

References

Appendices

Appendix A Data Formats
Appendix B Additional Data
  B.1 Non-Disjoint Intruder Datasets
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The categories applications were divided among</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>The amounts of data from different users</td>
<td>23</td>
</tr>
<tr>
<td>4.2</td>
<td>IBk on the test data with disjoint train and test intruder sets (see Section 3.4.1) and attribute preselection. To generate these data, each user was compared to others as described in Section 3.4.1.</td>
<td>24</td>
</tr>
<tr>
<td>4.3</td>
<td>Examining attribute selection on 5a5’s data</td>
<td>24</td>
</tr>
<tr>
<td>4.4</td>
<td>Examining attribute selection on c59’s data</td>
<td>25</td>
</tr>
<tr>
<td>4.5</td>
<td>Examining attribute selection on ca8’s data</td>
<td>25</td>
</tr>
<tr>
<td>4.6</td>
<td>Examining attribute selection on cb8’s data</td>
<td>26</td>
</tr>
<tr>
<td>4.7</td>
<td>Examining attribute selection on d48’s data</td>
<td>26</td>
</tr>
<tr>
<td>4.8</td>
<td>Examining attribute selection on e35’s data</td>
<td>27</td>
</tr>
<tr>
<td>4.9</td>
<td>NaiveBayes on the test data with disjoint train and test intruder sets (see Section 3.4.1) and attribute preselection.</td>
<td>28</td>
</tr>
<tr>
<td>B.1</td>
<td>IBk on the test data with non-disjoint train and test intruder sets (see Section 3.4.1) and attribute preselection.</td>
<td>46</td>
</tr>
<tr>
<td>B.2</td>
<td>Confusion matrices for the non-disjoint intruder data</td>
<td>47</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 3.1 Screenshots of MainScreen ............................................ 9
Figure 3.2 Screenshots of StatsActivity and SQLActivity .................. 11
Figure 3.3 The preferences menu .................................................. 12
Figure 3.4 The structure of the file uploaded to the server .................. 14

Figure A.1 Data Formats used in Mouflon ..................................... 45
Chapter 1

Introduction

As smartphones become increasingly capable and omnipresent, users will certainly continue to store highly sensitive personal information on them. A user has a strong desire to protect this information from access by a third party who picks up an unattended phone. To that end, many users opt to use a PIN or password to protect the device. These obtrusive authentication methods dramatically slow down the user who wishes to perform a quick task on the device. Smartphone locking systems should strive to protect the phone from unauthorized use while minimizing impact on the convenience of authorized use. The passwords entered on small virtual (or, less often, physical) keyboards are cumbersome and PINs or swipe patterns over a grid of points are easy to watch and duplicate.

We envision a system that utilizes application usage behavior to attempt to ascertain whether the current user is the device owner or not. Before any such system can be designed and implemented, it must first be shown that it is possible to identify users from this behavior. We use the name Mouflon to refer to the process of identifying or authenticating users from their application usage behavior. Thus, the Mouflon-based system presented in this paper uses the length of time that elapses in each category of app (including the home screen), along with the number of each category in a session and the total time in each category per session, to make a decision on whether the set of interactions under consideration is from the owner or another user. These categories were defined based on the overall use of the portion of the application in question. For example, the messaging part of Google Voice was placed in the messaging category and the phone part was placed in the phone category. Mouflon can also recognize the time per app for all apps (overall device usage speed) and whether or not the interaction takes place on a weekend (because users' habits may be quite different on weekends). These behaviors do not capture the full gamut of a user’s interaction with her phone. Behaviors not captured include (for example) touch patterns on the screen and the angle at which the user holds the device. However, the behaviors that we do capture can be observed without invasively
modifying the phone or consuming large amounts of system resources such as processing time and energy. Therefore, in this thesis we will attempt to show that

*Because different people use their phones in different ways, it is possible to determine the identity of a user of a smartphone from among a set of other smartphone users based solely on these metrics of his or her usage of applications on that phone.*

If it is possible to identify the phone owner from other users based on these behaviors, it should be possible to use the same behaviors to authenticate a user. A device secured with a system derived from Mouflon would allow a user identified as the owner to use the device normally, but if the user is identified as a foreign user, he will be presented with a prompt for a password. Clearly, this is not as secure as using a strong password that is required every time the device wakes; however, it provides a middle ground for users who can accept the trade-off of security for convenience.

This project builds off of existing research in the area of transparent unlocking for mobile devices. A team of researchers has designed a system which recognizes the motion a user makes when bringing the phone to his ear to answer a call. They found their system was able to recognize the user upwards of 80% of the time while accepting the input of an impostor less than 10% of the time if enough training data was acquired [16]. Another group has developed a method for behaviorally profiling a user’s actions and raising an alarm (i.e. locking the phone) if an unauthorized user is detected. This group looked at telephone, SMS, and application usage behaviors separately and attained error rates under 15% on data collected by other researchers years before the study [23]. A third group uses an approach similar to Mouflon, but with different specific methods. They consider the phone calls, SMSes, web browsing behavior, and location data when analyzing user behavior. The model also takes into account time of day and day of week to more fully capture usage patterns. In their model known phone numbers and websites are considered “good” and unknown ones “bad.” The model is based primarily on the time since the last “good” interaction. There are many ways to interact with an Android phone that do not involve any of these actions, so Mouflon hopes to provide a more holistic picture [30].

This thesis presents data supporting the hypothesis that user behavior on a smartphone can be used to identify that user. In this study we have collected application usage data from six users of Android phones. We have compared different users’ usage patterns in order to extract the identity of the user from his application usage. This ability is necessary to use behavior to authenticate a user. We hope that this could lead to a behavioral authentication mechanism in a smartphone system.
Chapter 2

Background

2.1 Overview of Approach

We seek to provide a path to a method for a user to protect his smartphone from an intruder without sacrificing the convenience of a passwordless unlocking interaction. A password can take a few seconds to enter, and sometimes a user may want to perform a quick task, such as checking a sports score, that may take less time than putting in his password. This inconvenience can inspire users to eschew the password option altogether, opting for a much weaker PIN or swipe pattern or even no authentication at all. A PIN or swipe pattern can be easily observed and duplicated, as evidenced by the myriad of techniques reported in the popular press for obtaining ATM PINs such as those given in [22].

If a Mouflon-based system can provide a method with a level of security approaching that of a password combined with a level of convenience approaching that of no password, it would likely encourage a larger proportion of smartphone users to enable security measures on their phones. A naive solution could be to leave the phone unlocked for quick tasks, delaying the password prompt for a few seconds. However, if the phone can profile the user’s behavior during this several second delay and determine whether she should be locked out, the authorized user need not worry about a password no matter how long she uses the phone. This is where the Mouflon-based system comes in. To an authorized user, it appears the phone has no password, and to the unauthorized user, it appears the phone has a password delay. There are, of course, the standard problems with all intrusion detection systems that apply in this situation. There will be false negatives, when a legitimate user is locked out, and false positives, when an intruder slips past. The goal is to minimize both of these rates, but there is often a trade-off where reducing one increases the other. Further details are given in the following sections: an overview of Android in Section 2.2, an introduction to active authentication in Section 2.3, and a more detailed description of how Mouflon could be used in Section 2.5.
2.2 Android Background

Android is an open-source smartphone operating system developed and maintained by Google. The operating system is based on the Linux kernel, but there are substantial differences between the Android OS and a standard GNU/Linux OS. The major unique feature of Android relevant for Mouflon is the Android API that is exposed to applications running on the phone. This is an interface through which applications written in Java\(^1\) but compiled to non-standard bytecode can interface with the hardware, other apps, or the Internet. This API provides some useful abstractions that make developing an application easier, discussed in Section 2.2.1, a permission system for controlling the capabilities of apps, discussed in Section 2.2.2, and a logging system, discussed in Section 2.2.3. Of course, it also provides additional functionality, but these are the pieces most relevant to Mouflon.

2.2.1 Application Components

The Android API defines four basic component types that each provide specific functionalities [8]. These components can all interact with each other within an app to enable a broad range of functionality.

- **Activity**: These are components that are visible to the user and provide the user interface to every Android application. When a user launches an app, its main Activity is created by the Android framework and brought to the foreground of the display. These are the only components that a user will interact with directly, so they are the only ones we need to keep track of when analyzing user behavior for Mouflon. Subclasses of Activity are used by Mouflon to allow the user to interact with the collected data (e.g. by viewing, uploading, or deleting it) or modify settings. When it launches a new Activity, the Android OS component ActivityManager handles the launching and creates a log entry, which is where Mouflon gets its data. With the ability to modify the OS, ActivityManager could be instrumented to provide this data directly.

- **Service**: These are components that run in the background and have no user interface. They are commonly used to preform background computation when an app is not in the foreground of the display. Mouflon uses Services to do the bulk of its work: collecting and uploading data. A user does not interact with a service directly, so Mouflon does not include them in its analysis of user behavior.

- **BroadcastReceiver**: These are components that are started by the Android framework when another app or the system sends out a broadcast message that the BroadcastReceiver

\(^1\)Applications can also contain native code written in C/C++
in question is designated to receive. This allows an app to be started by a general broadcast event. An example of this is the android.intent.action.BOOT_COMPLETED broadcast that the system sends out whenever it finished booting. Any BroadcastReceiver that registers to receive this event (and has the requisite permission) will be started by the system whenever the system finished booting. Mouflon uses this mechanism to set up the scheduling of the data collection and upload components.

- **ContentProvider**: These are components that provide data to other applications. They are not used or tracked by Mouflon.

For more on the specific components of MouflonRecorder, see Section 3.1.

### 2.2.2 Permissions

The Android framework restricts access to certain actions or data only to applications that have declared the appropriate permissions at install time. Permissions serve to warn a user about what capabilities the application has. Some of these permissions serve to restrict access to sensitive pieces of data such as the GPS, contacts, or SMS messages. Other permissions restrict what the application can do such as communicate over the Internet or write to external storage (formerly referred to as the SD card). There are also permissions deemed unsafe for any third-party app that are only available to apps signed by the signer of the OS. The important permission for Mouflon is the permission that allows it to read the logs created by the OS and other apps.

**Android 4.1 Changes**

There was an undocumented change in Android 4.1 that made the **READ_LOGS** permission accessible only to system apps or apps signed by the signer of the OS [19]. Therefore, root permissions are required for Mouflon to run on devices running Android 4.1 and later. Users of these devices were excluded from the study because they would be unable to return useful data. Google representatives have stated that this decision was made to improve the security of the platform because the privacy-sensitive data various applications would record in the logs could have been used by malicious parties to spy on users [19]. This is exactly what Mouflon does, albeit not maliciously, which is why users were informed of its behavior and signed consent forms before participating in the study. This new restriction can be mitigated by developing Mouflon as a system app; this is discussed in Section 7.1.6.
2.2.3 Logs

The Android system logs a large amount of data about what the system is doing. Of particular interest to Mouflon are the log entries that detail when an Activity is started. These log files are accessed through the logcat utility rather than through an API because the Android developers never wanted the interface to be used extensively [19]. Accessing the log files of an Android device requires the READ_LOGS permission since they contain sensitive data. Requesting this permission is uncommon but not unheard of for apps in the Play Store.

2.3 Active Authentication

Active authentication refers to any of a wide range of techniques where the device performing the authentication (in this case a smartphone) actively analyzes information about the user in order to determine if the user is authorized to use the device. In this setup the user performs no special actions, and, ideally, the authorized user will not know the authentication is being performed. Hence, it is sometimes referred to as transparent authentication. In a smartphone environment, this equates to allowing the user to unlock the device without a password and to use the device until and unless the user fails an authentication check. In this scenario, a false negative (false alarm) only means that the system falls back to the existing authentication method. A false positive is much more damaging as it allows an intruder to access the device and is the main disadvantage of using this method of authentication.

2.4 Application Usage Behavior

A very broad and flexible active authentication system is the analysis of a user’s application usage. The defining feature of smartphones is their ability to run powerful third-party applications, and the vast majority of users utilize this functionality. Analyzing application usage on Android primarily consists of examining the component names of Activities the user commonly accesses and the order in which he does so. This is augmented by the timing of the whole sequence from the time the screen is turned on. All of the information needed to determine this information is recorded in the phone’s logs by the operating system. Therefore, to profile user behavior, one needs only periodic access to the system logs. The log buffer is circular, so older entries are eventually pushed out. When enforcing the Mouflon policy, the app can receive a broadcast of the screen wakeup event and store the time that this event occurred. The only way to know when an app has been launched (without modifying the firmware) is to monitor the logs, whether in real-time when authenticating a user or periodically when learning user behavior. Mouflon uses pre-existing machine learning algorithms to analyze and classify the
interaction patterns.

2.5 Summary

In a production version of Mouflon, the user would dismiss the keyguard by the standard method of dragging the lock icon to the right. Users would also be able to utilize the Android 4.0 option to not have a lockscreen at all. In this scenario, Mouflon would be able to keep “pocket-dialing” and other inadvertent input methods to a minimum because the user’s customary inputs would differ from random inputs. The user would then be permitted to use the full functionality of the phone until and unless Mouflon flags him as a foreign user. The phone would note the time the screen is turned on and store it for later use. As the user uses the phone, Mouflon would examine the logs waiting for app to be launched. Until Mouflon has made a decision about whether the user is foreign or not, the behavior extracted from the logs will be noted and compared with the stored profile. If the user is determined to be the owner, Mouflon would exit. If the user is determined to be a foreign user, Mouflon would lock the phone with a method of the user’s choosing (e.g. a password or PIN).
Chapter 3

Design

This chapter covers the design of the various components of the system that was constructed to evaluate the idea that a user of an Android device can be classified as either the device's owner or another user. For the purposes of this work, the other user is considered the intruder regardless of her intent. In Section 3.1, we describe the application that collects and transmits test user data. In Section 3.2 we describe the web server that is used to collect the uploaded data from the users. In Section 3.3, Section 3.4, and Section 3.5 we cover the tools used to process and analyze the collected data.

3.1 MouflonRecorder

In order to collect large amounts of usage data, an app called MouflonRecorder has been developed. This app was uploaded to the Google Play Store for distribution to study participants. Its purpose is to allow study participants to collect and upload data about their usage habits to the authors. From a participant’s perspective data collection is passive once MouflonRecorder is installed, and uploading is either active or passive depending on participant preference. If the participant chooses to allow automatic uploading, their usage data is uploaded weekly. This app consists of eleven components, each of which is implemented as a Java class. This section is organized according to which of the main Android component classes the class extends (see Section 2.2.1 for an explanation of these).

3.1.1 Activities

MainScreen This is an Activity that is the central hub for user interaction with the device. On the first launch of the app, the user is presented with a screen prompting her for an activation code as shown in Figure 3.1a. This code is in place to prevent curious members of the public who are not participating in the study from inadvertently submitting their application usage
data and contaminating the study. The user is also presented with a choice of whether to upload data automatically or be notified to upload. On subsequent launches this Activity is the main menu. A screenshot is presented in Figure 3.1b. This screen provides several choices to the user. The user can upload his logs to the server manually, which some users may choose to do so that they may review the logs before submitting them. Viewing the statistics will allow the user to see a brief summary of the data collected on the device since the previous upload. This screen is further described below. The next option allows the user to clear the log data stored on the device. The user may want to do this if he uses an app he doesn’t want logged and reported to the authors. The fourth option allows the user to search through the stored data for a specific string, allowing the user to determine if a specific piece of sensitive information has been recorded in the logged data. This option is discussed further below. The final option takes the user to the settings menu where he can modify how the app behaves. The settings are explained below.

(a) The code request shown on the first run.  
(b) The main screen of MouflonRecorder

Figure 3.1: Screenshots of MainScreen
StatsActivity  This Activity seeks to present a user-friendly view of the data collected by the app. A technical or non-technical user can quickly scan the data presented on this screen to confirm the app is working and to look for anything he does not wish to report to the authors. A screenshot is presented in Figure 3.2a. This screen contains the following data:

- The total number of events. This measure consists of the number of application launches and the number of screen on and screen off events. These three numbers are also presented individually.

- A list of Activities that have been launched on the device since the last time the data was cleared. The strings listed here are the fully qualified class names of the Activities rather than the more user-friendly names users generally associate with apps. Fortunately there is often a strong correlation between the two different types of application names. For example the main UI Activity of the Pocket Casts app is au.com.shiftyjelly.pocketcasts/.ui.MainActivity, which is fairly easy to interpret. Sometimes the correlation is more subtle such as the Google Play Store which is com.android.vending/com.google.android.finsky.activities.MainActivity. There is no guarantee that an application developer will give her app a common name that is similar to the fully qualified name. A user may therefore struggle to recognize the name of a sensitive app. However, if a user is unable to identify a sensitive app due to this issue, it is unlikely that those the user wishes to keep the sensitive app from would notice it.

SQLActivity  This Activity allows the user to search through the data that MouflonRecorder has recorded. The interface (see Figure 3.2b) allows the user to enter a string and search for it. After the user performs a search the results in the form of the raw data format that Mouflon uses internally are printed to the screen as in the example in Figure A.1b. The user is likely to mainly be concerned with the presence or absence of results. The use of the raw data format allows the user to ensure that if the data sent to the server contains the data they search for, it will be found.

MouflonPreferences  The preferences screen, as shown in Figure 3.3, allows the user to modify the behavior of MouflonRecorder. The user can set his e-mail address, which (if set) will be reported to the server along with his data. He can also determine whether he wants the app to upload automatically (weekly), notify him to upload (weekly), and whether to delete the data on upload.
3.1.2 Services

As covered in Section 2.2.1 an Android Service runs in the background to perform tasks without requiring or affecting user interaction. MouflonRecorder uses several services to complete the required background tasks of recording the data and uploading it to the server. These services are set to run automatically by the RegisterAlarm BroadcastReceiver, which is described in Section 3.1.3.

RecorderService  This Service handles the core functionality of MouflonRecorder. It runs approximately every half hour to record the user’s behavior. As described in Section 2.4 the Mouflon system determines user behavior by looking at the Android system logs which record a very large amount of data. Mouflon operates by looking for two types of events in these logs: power events and Activity start events. These two types of events provide the most general picture of what the user does on the device regardless of what apps are installed, as they are generated by the stock Android OS\(^1\). This information is also accessible on Android

\(^1\)As discussed in Section 4.1.1 some vendor-modified versions of Android alter or eliminate these messages
4.0 and earlier without modifying the device or administrative access. All that is required is that the application declare the `READ_LOGS` permission in the manifest (see Section 2.2.2 for more information on permissions).

The Android logs are accessed through a program called `logcat` which can be executed from within an app with the required permissions. This program takes command line arguments that specify what data it should return. In Mouflon, we are interested in power and application start events so we ask for events from the `power` and `ActivityManager` tags. We also request a format that includes the date and time for each event. This information is then parsed to separate the plain text input into fields corresponding to the data we wish to store and later analyze. First, the date and time of the most recent event in the Mouflon database (from the last time this `Service` was run) is retrieved so that only subsequent events will be examined on this run to avoid any double counting. The following data are then stored in the database:

- Date and time
- Tag (`power` or `ActivityManager`)
- For `power` events, whether the screen is being turned on or off. For `ActivityManager`

![Figure 3.3: The preferences menu](image)
events, the string **START**

- **Component name.** *ActivityManager* events only. This is the fully qualified class name of the *Activity* as described in Section 3.1.1.

- **Action String.** *ActivityManager* events only. This is an optional instruction to the app being started.

- **Category.** *ActivityManager* events only. This is more optional data. Most often used to indicate that an app was launched from the home screen (as opposed to by another app) or that the home button was pressed.

Before storing into the database, any URLs beginning with http(s) are redacted to help preserve user privacy. Some of the log entries examined during early development included URLs, but only in fields that were not stored in the database, so this redaction is purely as a precaution against the case that an unexamined app leaks URLs in one of the fields we do store.

**UploadFile**  This **Service** is responsible for returning the data that is collected by **Mouflon Recorder** to the researchers for analysis. It is set to run every week (if the study participant chooses to do so) or launched manually from the main **MouflonRecorder** menu (see Section 3.1.1). In order to preserve user privacy the uploaded data are encrypted during transmission and storage on the server. Rather than separately encrypt the data during transmission (using, for example, https) and storage on the server, the data are encrypted once on the user’s device and left in that format until they are decrypted for analysis off line. Thus if the transmission line or the server is compromised, the attacker could not read the user data. This decision also obviated the need for an SSL certificate and the associated complexity. A public-key encryption system is used to encrypt the data. The private key is in the sole control of the researchers and is not stored on the server.

When this **Service** starts, the database containing the data recorded by **RecorderService** is converted to a CSV [29] format and dumped into a **StringBuffer** [12]. This data must then be prepared for upload. As is common practice [21], the actual data is encrypted with a symmetric cipher, the key for which is encrypted with the public key. The cryptographic operations in this program are performed using the Bouncy Castle [4] API provided by Android. The symmetric key is generated using an instance of **KeyGenerator** [10]. A random IV is also generated using an instance of **SecureRandom** [11]. The IV is written as the first 16 bytes of a file. The remainder of that file is a zip file encrypted with AES in CBC mode [21]. This is accomplished using the ability of Java **OutputStream**s to be chained together. The contents of this encrypted zip are:

- The device model (e.g. Nexus S) and Android version number
The participant’s e-mail address if he chose to enter it on the settings page

The CSV file previously generated from the database

This encrypted zip (encrypted with the secret key) and the secret key (encrypted with the public key) are put into a second zip file to be uploaded to the server. This final file is represented graphically in Figure 3.4. At this point, an HttpURLConnection is initiated with the server, and a manually constructed POST request is sent. This request contains the second zip file encoded in Base64. Upon a successful upload (one that did not generate any Java Exceptions) the intermediate files are deleted and the database containing the data that was just uploaded is cleared. If the upload is not successful, the data used by WeeklyChecker is modified so that the upload will be retried the next day instead of the next week. If the upload was manually initiated, a notification is put in the status bar informing the user that it is complete.

![Figure 3.4: The structure of the file uploaded to the server](image)

NotificationService This Service simply puts a notification in the status bar reminding the study participant to upload his files. It is run by WeeklyChecker if the user opts to upload manually.
WeeklyChecker  This Service is responsible for determining if it has been a week since the previous upload. MouflonRecorder uses the inexact alarm interface provided by Android to allow the device some flexibility when scheduling background tasks to avoid waking the device if possible. The longest interval that one of these alarms can be set for is daily. Therefore this service is started every day by the alarm and it increments a counter from 0 to 6, at which point it launches a file upload (or presents the user with a notification to do so) and resets the counter.

3.1.3 BroadcastReceiver

There is a single BroadcastReceiver class, called RegisterAlarm, in MouflonRecorder. This class serves the purpose of setting up the alarms that run the scheduled components of this app, RecorderService and WeeklyChecker. It uses the AlarmManager[7] interface provided by Android to set a twice-hourly alarm for RecorderService and a daily alarm for WeeklyChecker. It uses an inexact-repeating alarm, which Android will try to schedule with other alarms or when the phone is already awake to avoid waking the phone up if possible. This has the potential to decrease power usage.

3.1.4 Other

A class called DbAdapter, derived from an example in the Android documentation [2], provides a simple interface to the SQLite database that stores the recorded data.

3.2 Server

A virtual machine running Ubuntu 11.04 was set up as a publicly accessible web server. The purpose of this server is to receive uploads from devices running MouflonRecorder and store them for later retrieval by the researchers. It uses Apache as the http server software, and a custom Perl script was written to handle the upload. This Perl script takes the base64-encoded upload, decodes it, and writes it to disk. It stores the file in a folder without any read permission, so that even if the server is misconfigured, none of the encrypted uploads will be leaked. The server also notes the time that a file is uploaded and keeps a list of the most recent 10 times. If the current time is less than 20 seconds from the first time on the list, the server denies the upload and sends an e-mail alerting the researchers. This flooding precaution was never activated in practice.
3.3 Decryption

As discussed in the sections above, when the participants in the study submit their data to the server it is encrypted to protect their privacy. In order to perform analysis on this data, it must first be decrypted. A decryption program called DecryptLogs was written in Java using the Bouncy Castle [4] API to perform the reverse of the operations described above in Section 3.1.2 (see Figure 3.4 for a visual representation of the uploaded file). First the decryption program reads the private key from the disk and uses it to decrypt the encrypted symmetric key from the participant. This symmetric key is then used to decrypt the other zip file and extract the data the user uploaded. This process was made somewhat more complicated by the existence of a bug in some versions of Android that creates invalid zip files [1]. Fortunately, there is a workaround [27] that allows these files to be fixed at any time. This workaround and the decryption code were combined with existing Linux programs for unzipping and other file management tasks in scripts that automate the process of obtaining the desired data from the encrypted files.

3.4 Data Preprocessing

In order to minimize the amount of work performed by the smartphone running Mouflon Recorder, the data uploaded to the server is in a format very close to the raw Android logs. As shown in Appendix A, the data need to be transformed from this format, where each event is a single, timestamped line, to the arff format used by the Weka machine learning framework. Each interaction sequence from screen on to screen off between the study participant and the phone was treated as a discrete instance because the decision to accept or reject a potential intruder will be made for each time the screen is turned on. All of the applications launched by all users during the study were manually divided into 19 separate categories, which are listed in Table 3.1. For each interaction sequence, the following features were extracted from the raw data: number of app launches in each category, total time per category, the quotient of these two (time per app launch for each category), time per app launch averaged over all categories, and whether the sequence took place on a weekday or a weekend. For example, one user may send five e-mails, turning off the phone between each, and another may send five e-mails, one after another with no breaks. Another user may send text messages instead of emails. One user may bounce between multiple apps over the course of one session, whereas another may tend to spend a long time within one app or app category. These features were chosen because they were able to be straightforwardly derived from the data, required little processing time, and intuitively seemed like a good representation of behavior. Apps in the blacklist category include custom apps not appearing in the play store as well as MouflonRecorder itself. These
are ignored in the data analysis as it would unfairly bias the results by potentially making it easier to identify the developers of these custom applications.

Table 3.1: The categories applications were divided among

<table>
<thead>
<tr>
<th>Category</th>
<th>Subcategory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Browser</td>
</tr>
<tr>
<td></td>
<td>Calendar</td>
</tr>
<tr>
<td>Clock</td>
<td>Contacts</td>
</tr>
<tr>
<td></td>
<td>Email</td>
</tr>
<tr>
<td>Game</td>
<td>Launcher</td>
</tr>
<tr>
<td>Messaging</td>
<td>News &amp; Weather</td>
</tr>
<tr>
<td></td>
<td>Other</td>
</tr>
<tr>
<td>Phone</td>
<td>Photos</td>
</tr>
<tr>
<td>Productivity</td>
<td>Settings</td>
</tr>
<tr>
<td></td>
<td>Social Networking</td>
</tr>
<tr>
<td></td>
<td>Blacklist</td>
</tr>
</tbody>
</table>

Before converting the CSV files into arff files, the CSV files from all participants’ weekly uploads must be combined into a single CSV file for each participant. In order to avoid unfair fine-tuning of the classification algorithm, the first 20% of this data for each participant was then split off and used to determine which classifier, parameters, and categories to use. The remaining 80% was held in reserve until a procedure had been established. This 80% was then used to generate the data presented in this thesis. A Perl script was written to translate the files from the line-per-app CSV format to the instance-per-interaction arff format. It reads in the data from all of the study participants and creates a list of all interaction sequences, each labeled with the identifier of the participant who generated it. Each of these interaction sequences has the 56 attributes described above, three for each of the 18 categories plus average time per app and whether it’s a weekend.

An interaction sequence was defined as starting with a power on event and ending with a power off event. However, for reasons that are unclear without physical access to the device in question, one participant’s device was frequently missing screen power on events despite reporting plenty of power off and application start events. Therefore, it was decided to allow any application start event not already in a sequence to mark the start of a new sequence as it isn’t possible for the user to (for example) check her e-mail without turning on the screen. Manual examination of some users’ tuning data also revealed that some power off events were missing as well. It was decided that if no new app was launched for an hour, the next launch would constitute a new interaction sequence. This is justified because if the user is doing something like watching a movie, he may leave the device unattended at some point during the long period of single app usage. The timestamps provided in the Android logs only contain day and month information, so the year 2012 is assumed for all events so that existing date math libraries (in this case Perl’s DateTime::Format::Strptime library) can be used. Since only
durations of time on the order of seconds or minutes are used, the two dates are immediately subtracted from each other and adding a constant offset (the year) to each does not impact the calculation unless the sequence starts and ends in different years. This did not end up being a problem in practice because no sequences reported by participants spanned a new year.

3.4.1 Cross-Validation

It was determined that it might unfairly bias the classifier to provide it with the same user as an intruder for both training and testing, so for each participant the other participants’ data is used as a simulation of intruder data. Disjoint sets of other participants were selected to provide the training and testing data for the classifier. The Weka framework does not provide a facility to divide the training and test instances in this manner, so the 10-fold cross-validation algorithm was manually re-created in Perl scripts. These scripts take a list of all interaction sequences from all users and divide them up into 10 cross-validation folds with 90% of the data used for training and 10% for testing in each fold. Each data instance is used in exactly one test set. For each fold, the number of instances of authorized user data is equal to the number of instances of intruder data to avoid biasing the classifier one way or the other by providing too many instances of one type.

3.5 Data Analysis Techniques

This project uses the data mining/machine learning framework Weka [32] version 3.6.6, which is Java-based and open-source, released under the terms of the GNU General Public License (GPL). Since it is Java-based and provides a Java API, Weka can be included in and used by an Android application. It also includes a GUI and CLI for use on a traditional operating system. The data items to be analyzed are the interaction sequences in the arff files constructed as described in Section 3.4 above. Each interaction sequence is represented by 55 numbers, a boolean, and a class name. In order to find the most valuable attributes for differentiating instances, the 20% tuning data was used to select the 20 attributes that provided the most information gain for each participant using the InfoGain attribute selection algorithm. These attribute selections were then applied to the 80% test dataset before processing. Since the attribute selection algorithm is fully supervised, using it directly on the 80% test data would provide an unfair advantage. The IBk classifier was used to classify the data in this study. This classifier is a lazy classifier, which means it builds a model by storing the training instances until classification. When classifying test instances, it finds the closest training instance using the Euclidean distance from the test instance [33]. The test instance is then given the same class as the closest training instance. A KD-Tree is used to find the nearest neighbor more quickly.
by dividing the problem space up into regions in a tree structure which allows the algorithm to quickly find an approximation for the nearest neighbor, and, more importantly, prune large sections of the tree [33]. Some other classifiers that were examined include NaiveBayes, Logistic, RBFNetwork, KStar, HyperPipes, and DecisionTable. The results of this analysis are presented in Chapter 4.

3.6 User Test

In order to gather more data on the effectiveness of Mouflon at detecting non-owner device usage in a broader range of environments, a user test was performed. The test was designed to collect data using the MouflonRecorder app described in Section 3.1. An Institutional Review Board (IRB) proposal was written that described what study participants would be asked to do, justified those requests by explaining the expected knowledge to be gained, and demonstrated that participants were adequately protected from any potential harm that could come from participation. The only potential harm identified was user privacy. Because the app seeks to record all and report all of the applications that a participant uses as well as when he or she uses them, a participant may (or may not) consider the information that the application collects and reports to be privileged information. These potential IRB and user concerns were assuaged using the anonymization and encryption techniques described in Section 3.1.2. Participants’ data reported to the server (see Section 3.2) is encrypted with a public key on the participant’s device before being uploaded over a standard http connection. After receiving IRB approval, users were recruited to install MouflonRecorder on their phones via e-mail and personal communication. Participation was hampered by the change made to Android in version 4.1 (see Section 2.2.2) making the requisite log data inaccessible to apps without system permissions. Many potential participants had devices that ran the latest version of Android. After signing a consent form (as required by IRB procedure), users were directed to the Google Play Store to download a copy of MouflonRecorder. After enrolling in the study, participants were given a code that activates MouflonRecorder so that curious non-participants who happened to find the app on the Play Store would not inadvertently become participants. Data was collected from the main user study participants for several weeks from January 31, 2013 to March 9, 2013. Not all users began collecting data at the same time, however. Additionally data was also collected from lab members for beta testing purposes for several months prior to and concurrent with the main user study. All of this data was aggregated and analyzed using the procedures described in Section 3.4 and Section 3.5; the results of this analysis are presented in Section 4.
3.7 MouflonTester

A proof-of-concept app for testing whether Weka can be run in an Android application called MouflonTester was also developed. This app also consists of three main components:

- An Activity that registers the BroadcastReceiver because an Intent Filter for the ACTION_SCREEN_ON Intent cannot be registered in the manifest file [20].
- A BroadcastReceiver that gets started by the operating system when the screen is turned on. It starts the EnforcerService
- A Service, EnforcerService, that examines the system log to find the relevant data and processes it with Weka to determine whether or not the user is authorized.

In its research implementation, MouflonTester will just pop up a toast notification informing the user of its decision about her identity. However, it does not have a robust training dataset, so its results are not reliable. In a production implementation the Mouflon app would lock the device until a password or other authentication method was entered. A production version would also be able to add new user behavior to its training set and come with a well-tested intruder dataset for training purposes.
Chapter 4

Results

The data collected by MouflonRecorder from nine users were analyzed using the Weka machine learning framework, version 3.6.6, as obtained from the Ubuntu repositories. The data were processed according to the procedure outlined in Section 3.4 to generate the files for analysis in Weka. Then the data were analyzed according to the procedure in Section 3.5. The users participating in the study were assigned UUIDs by MouflonRecorder. These users are identified in this paper by the first three hexadecimal digits of their UUID. Some users in the beta phase either did not have UUIDs because that feature had not been added yet or multiple UUIDs stemming from uninstalling and reinstalling the app. These users are identified by the first three hexadecimal digits of a hash of the filename their data was stored in.

As described in Chapter 3, a data file is created for each user that marks her interaction sequences with her identifier and all other interaction sequences in the dataset with “other”. This approximates the situation where a user installs Mouflon on his device and it comes with the other users’ data in order to train the classifier on what to reject. If a Mouflon-based system were deployed in the wild, it would require training data from a good approximation of potential intruders as well as training data from the intended user. Supplying a fresh install with other users’ data is one way to provide this training data. In this evaluation, this data was then used by Weka to train and test a classifier. Weka was instructed to perform 10-fold cross validation on the data. When classifying, the classifier pretends that it does not know the user who generated the interaction sequence and places it in a class based on the training data. This placement is then checked for accuracy against the known result. The whole dataset contains approximately 5000 recorded sequences of user interaction with the device. Around 1000 of these were used to tune the classification procedure, and the remaining 4000 were used to generate the results in this chapter.

Two different classifiers were used in this analysis. A selection of the classifiers available in Weka were tested, and IBk was found to be very good at distinguishing the data in this
experiment. NaiveBayes is a very popular classifier used in papers seeking to do similar things to this one, such as [25], and is included for comparison.

4.1 Difficulties

There were some difficulties encountered in analyzing the data.

4.1.1 Data Missing From Logs

Not all users’ phones reported the same data. Two of the most common modified Android implementations are Samsung’s TouchWiz and HTC’s Sense. None of the users in the beta phase of data gathering used a phone with either of these pieces of software. Once the full user study came online, it was found that three of the recruited users used phones with either Sense or TouchWiz. Two phones only reported power on and off data\(^1\). These were excluded from all analysis as no user behavior beyond the length of time the screen is on can be derived from the reported data. The third phone’s logs contained all of the events expected (power and application start), but the application start events had all of the information about application identity removed. The raw logs were examined, and it was observed that the problem was that the information was not present rather than a bug in \texttt{MouflonRecorder}’s parsing algorithms. It is possible this information was scrubbed by the device vendor to protect user privacy.

4.1.2 Missing Power Events

Upon manual examination of some of the returned logs, it became evident that some power on and off events were missing. Sometimes applications would be started without a preceding power on event. In these cases, the first application launch is considered time zero instead of the power on event being time zero. In other cases, power on and off events were missing entirely, sometimes for weeks at a time. It is inconceivable that a user would have used the phone for that long without a break, so such long sequences were split whenever a gap of an hour or more existed between events. It is possible that these events were missed because the recording interval of 30 minutes was too long, and the power events had already been cleared from the circular buffer by the time \texttt{MouflonRecorder} recorded the data.

\(^1\)The start times of a particular Google Analytics process that isn’t manually started by the user were also recorded on most phones, even if no other applications’ start events are recorded in the logs. However, no application name is associated with it, just an action, so it is not included in this analysis.
4.1.3 Small Number of Users

Only five users responded to the call for volunteers, and only one of these returned useful data for the reasons outlined in Section 4.1.1. Therefore we fell back onto data collected from the beta phase of this study, and used that for this analysis. We therefore have six users with various amounts of data, as shown in Table 4.1. As can be seen from the table, different users produced instances of interactions with the phone at quite different rates, ranging from twenty per week to two hundred per week. The number of instances in Table 4.1 is different from the number of instances in Table 4.2 because the first 20% of the data was sequestered for tuning the classification methods and, for some users, we weren’t able to provide enough “other” instances to achieve a 50/50 user/“other” balance, so the size of the dataset was restricted. Users were also demographically similar, with all being students or faculty in the Computer Science department at North Carolina State University.

<table>
<thead>
<tr>
<th>Instances</th>
<th>5a5</th>
<th>c59</th>
<th>ca8</th>
<th>cb8</th>
<th>d48</th>
<th>e35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weeks</td>
<td>414</td>
<td>3266</td>
<td>98</td>
<td>187</td>
<td>644</td>
<td>992</td>
</tr>
<tr>
<td>Instances/week</td>
<td>207</td>
<td>109</td>
<td>20</td>
<td>94</td>
<td>59</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 4.1: The amounts of data from different users

4.2 IBk

IBk was found to perform the best on the tuning data, so it is the primary classifier used in this analysis. IBk is a lazy classifier, which means that no computation takes place in the training phase, all instances are merely stored. When it comes time to classify an instance, IBk finds the closest training instance by Euclidean distance in the multi-dimensional attribute space and returns the class of that instance [33]. IBk was chosen from amongst the other classifiers for its strong performance on the tuning data as discussed in Section 3.5. In this section, we present a general overview of its performance as well as more specific details about how well it classifies each user. As other papers [17] do, we define a positive classification as classifying the user as the owner and allowing him to pass. On a negative classification the user is locked out. Thus a false positive is an intruder who slips through, and a false negative is a legitimate user who is locked out. In general, the false positive rate is more critical than the false negative rate as a false positive is a breach, but a false negative is an inconvenience. A true positive is a correctly-identified legitimate user, and a true negative is a correctly-identified intruder.
The results of running IBk on all users' data are presented in Table 4.2 to facilitate comparison between users. For all users except c59, the classifier is able to correctly identify both authorized users and intruders at least 50% of the time. Were this not the case for at least one group, the classification would be useless and no better than a coin flip. This ability for five of the six study participants leads to the conclusion that this Mouflon system can identify users from their application usage data. The use of the procedure in Section 3.4.1 to ensure that the test instances are drawn from users who the classification model has never seen before makes this conclusion particularly robust. See Section 3.4.1 and Section 6.2 for further discussion.

### 4.2.1 User 5a5

Referencing Table 4.2, user 5a5’s interactions are accepted 84% of the time and the unknown intruder’s instances are accepted only 40% of the time. Selecting the most useful attributes using the method from Section 3.5 considerably improves this algorithm’s ability to find 5a5’s instances, finding about 50 more in Table 4.3a than in Table 4.3b.

### 4.2.2 User c59

Referencing Table 4.2, user c59’s interactions are accepted 94% of the time and the unknown intruder’s instances are accepted a full 61% of the time. User c59 would not find Mouflon very...
Table 4.4: Examining attribute selection on c59’s data

<table>
<thead>
<tr>
<th></th>
<th>c59</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>c59</td>
<td>889</td>
<td>57</td>
</tr>
<tr>
<td>other</td>
<td>577</td>
<td>369</td>
</tr>
</tbody>
</table>

(a) Selected Attributes

<table>
<thead>
<tr>
<th></th>
<th>c59</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>c59</td>
<td>827</td>
<td>119</td>
</tr>
<tr>
<td>other</td>
<td>583</td>
<td>363</td>
</tr>
</tbody>
</table>

(b) All Attributes

Useful as more attackers would be let in than locked out. A possible reason for this is discussed with user d48’s data in Section 4.2.5. User c59 generated considerably more data than any other user, but not all of that data was able to be used because the ratio of positive data to negative data was constrained to 1:1. This meant that when dividing the data into test and training sets, a significant (randomly-selected) proportion of c59’s instances were dropped. Running the same algorithm on the same data without the Section 3.4.1 restriction requiring training intruders and test intruders to be disjoint produces much better results. However, it is not realistic for the reasons discussed therein. See Section 6.2.1 for more discussion and Table B.2b for the raw data. User c59 is also well-served by attribute selection, lowering the number of false negatives by about half comparing Table 4.4b to Table 4.4a.

4.2.3 User ca8

Table 4.5: Examining attribute selection on ca8’s data

<table>
<thead>
<tr>
<th></th>
<th>ca8</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca8</td>
<td>52</td>
<td>32</td>
</tr>
<tr>
<td>other</td>
<td>35</td>
<td>49</td>
</tr>
</tbody>
</table>

(a) Selected Attributes

<table>
<thead>
<tr>
<th></th>
<th>ca8</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca8</td>
<td>53</td>
<td>31</td>
</tr>
<tr>
<td>other</td>
<td>35</td>
<td>49</td>
</tr>
</tbody>
</table>

(b) All Attributes

Referencing Table 4.2, user ca8’s interactions are accepted only 61% of the time; however the unknown intruder’s instances are accepted only 41% of the time. Ideally, the false positive rate would be lower than 41%, but user ca8 only produced 84 instances that were able to be tested. Considering how small this dataset is, the performance of the classifier is impressive. Attribute selection makes an insignificant difference for user ca8.

4.2.4 User cb8

Referencing Table 4.2, user cb8’s interactions are accepted 73% of the time and the unknown intruder’s instances are accepted only 43% of the time. User cb8 produced about twice as many
instances as user ca8, and this seems to be reflected in the rise in true positive rate; however, the false positive rate is almost unchanged. Attribute selection produces considerable improvement for user cb8 as shown in Table 4.6.

4.2.5 User d48

Referencing Table 4.2, user d48’s interactions are accepted 84% of the time; however, the unknown intruder’s instances are accepted 49% of the time. Similarly to user c59, user d48 sees much better results without the restrictions on the data in the training and test sets as shown in Table B.2e. It is possible that this is because these two users are quite similar to each other. Both generated a larger amount of data than most of the others, so each was necessarily included in the other’s test set because that allowed the smallest number of instances to be thrown away in order to meet the 1:1 user to intruder ratio. Comparing each to all other users instead of a subset (albeit with the caveats in Section 6.2.1) results in much better performance of the classifier. Attribute selection actually hurts the performance of the classifier for user d48 considerably. Since attribute selection is done on the first 20% of each user’s data (to avoid unfairly selecting attributes because we know the answers) then applied on the remaining 80%, this seems to indicate that user d48’s usage habits changed after the first 20%. This is not seen in other users. In the real world, a Mouflon-based system would need to constantly evaluate itself and decide between using attribute selection for a particular user or not.
Table 4.8: Examining attribute selection on e35’s data

<table>
<thead>
<tr>
<th></th>
<th>e35</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>e35</td>
<td>666</td>
<td>96</td>
</tr>
<tr>
<td>other</td>
<td>236</td>
<td>526</td>
</tr>
</tbody>
</table>

(a) Selected Attributes

<table>
<thead>
<tr>
<th></th>
<th>e35</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>e35</td>
<td>621</td>
<td>141</td>
</tr>
<tr>
<td>other</td>
<td>270</td>
<td>492</td>
</tr>
</tbody>
</table>

(b) All Attributes

4.2.6 User e35

Referencing Table 4.2, user e35’s interactions are accepted 87% of the time and the unknown intruder’s instances are accepted only 31% of the time. Attribute selection marginally improves the performance for user e35. The performance of the algorithm for user e35 is not as heavily impacted by switching to disjoint train and test intruders. This provides further evidence that users d48 and c59 are similar to each other because, due to their dataset sizes, one will be in the training dataset and the other will be in the test dataset. If this weren’t the case, most of user e35’s data would have to be thrown away. If d48 and c59 are so similar that the classifier confuses them, perhaps when classifying e35, the classifier matches up d48 instances in the training set with c59 instances in the test set (or vice versa) Both are labeled “other”, so the classifier rejects the test instance. Alternative interpretations are of course possible, but this one seems to make sense.

4.2.7 Best and Worst Features

The correctly and incorrectly classified instances were examined to determine if any information about which categories of apps contributed to correct and incorrect classifications could be gleaned. The categories whose features appeared disproportionately in correctly classified instances were audio, news, and e-mail. These occurred 52%, 30%, and 21% more often in correctly classified instances than in incorrectly classified instances, respectively. The categories whose features appeared disproportionately in incorrectly classified instances were settings, messaging, and calendar. These appeared 65%, 33%, and 33% more often in incorrectly classified instances than in correctly classified ones, respectively. It seems that the way users listen to music or podcasts is quite different from the other users, whereas the way they change settings on their phones is largely similar. This makes intuitive sense because when changing a setting, a user would tend to go to that setting and change it directly, without much of a chance to do anything unique. It seems unlikely that a user would sit down and tinker with phone settings for a period of several minutes on a regular basis. Messaging is quite similar in that one tends to send one message at a time and wait for a reply. Counterintuitively, the way that people check their e-mail seems to be quite a good indicator of identity. Of course, these data do not
indicate that these features caused the correct or incorrect classification, just that there was a correlation.

### 4.3 NaiveBayes

For comparison, the same analysis was performed with NaiveBayes as the classifier with less compelling results. These results are presented in Table 4.9. The results in this table do not compare favorably with Table 4.2. For every user either the false positive rate is unacceptably high, indicating that this algorithm provides little security, or the false negative rate is unacceptably high, providing little improvement in convenience. For user 5a5, for example, the intruder is locked out only slightly more often than the authorized user, and both are locked out less than one time in twenty. This is essentially equivalent to no security. On the other hand, user cb8 will be presented with a password upon unlocking the phone more than 4 times out of every 5. This is nearly equivalent to setting a traditional password. Mouflon should not be implemented using NaiveBayes for this reason.

#### Table 4.9: NaiveBayes on the test data with disjoint train and test intruder sets (see Section 3.4.1) and attribute preselection

<table>
<thead>
<tr>
<th></th>
<th>5a5</th>
<th>c59</th>
<th>ca8</th>
<th>cb8</th>
<th>d48</th>
<th>e35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instances</td>
<td>382</td>
<td>946</td>
<td>84</td>
<td>178</td>
<td>564</td>
<td>762</td>
</tr>
<tr>
<td>True Positive Rate</td>
<td>96.6%</td>
<td>96.0%</td>
<td>33.3%</td>
<td>16.9%</td>
<td>97.5%</td>
<td>26.6%</td>
</tr>
<tr>
<td>True Negative Rate</td>
<td>4.7%</td>
<td>20.0%</td>
<td>98.8%</td>
<td>93.8%</td>
<td>11.0%</td>
<td>76.8%</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>95.3%</td>
<td>80.0%</td>
<td>1.2%</td>
<td>6.2%</td>
<td>89.0%</td>
<td>23.2%</td>
</tr>
<tr>
<td>False Negative Rate</td>
<td>3.4%</td>
<td>4.0%</td>
<td>66.7%</td>
<td>83.1%</td>
<td>2.5%</td>
<td>73.4%</td>
</tr>
</tbody>
</table>

### 4.4 Performance

The classifying algorithms were run on a 1.7GHz laptop with 4GB of memory. Some runs during tuning ran out of memory and crashed. These were rerun with a command line option allowing Java to allocate more memory. Runtime of the IBk algorithm to generate all of the data in Table 4.2 was approximately ninety seconds. When implemented on the device, it should not be necessary to build a model every time Mouflon needs to make a judgment about a user and only one instance will be classified at a time, rather than thousands. Weka can save and load models [3], so the model would be generated once, or perhaps periodically as more data was acquired, and used repeatedly to classify new interaction sequences.
Chapter 5

Related Work

5.1 Behavioral Profiling on Mobile Devices

5.1.1 Symbian

Li et al. [23] have already profiled users by examining which applications they launch and on what date. However, they did not consider any timing features relative to other apps, just the date. In addition, the data they used was collected in 2004-2005 by users of Nokia 6600 phones running Symbian. The smartphone world has changed so significantly since then that it is very likely current smartphone users use their phones in very different ways to those of 8 years ago. Application markets now contain hundreds of thousands of apps and device sales grew from tens of millions per year in 2004-05 [5] to nearly one billion per year for 2012-2013 [6]. Given the much broader user base and the much larger variety of applications available, it is difficult to compare today’s smartphone user behavior to that of 8 years ago.

5.1.2 Windows Phone

Riva et al. [28], a team led from Microsoft Research, have done research into progressive authentication, which is a system with very similar goals and high-level implementation. They attempt to reduce the number of PIN/password prompts by authenticating or de-authenticating a user based the environment around the phone. The authors stress the continuous nature of the authentication, and the main way the system seems to reduce password prompts is to not deauthenticate the user when the algorithms indicate that the identity of the user has not changed since the last authentication. The phone listens for the user’s voice; senses whether it is in a pocket, in hand, or on a table; recognizes the user’s face; and detects the presence of a nearby PC to which the user is logged in. These factors, as well as the time since each was last observed, are fed into a machine learning algorithm to determine how confident the phone can
be that the person using it is authorized to do so. They have tuned their algorithm so that (in their experiments) it allows no intruders to access the phone. However, the number of times the authorized user is recognized is only reduced by 42%.

Polling the sensors and doing the processing they need in order to make a classification can be quite power consuming at 325 mW\(^1\). Recent Windows phones tend to have batteries with a capacity of 6-7 Whr [31], which would be drained in less than 24 hours at that rate. This seems to be the power draw just during computation, so it would not be continuous. However, with all processing offloaded to the computer, the system still draws 307 mW just by powering the sensors, and, given that it seems like the phone must be constantly polling the sensors in order to establish continuous authentication, this still is a large drain on the battery. If the sensors are polled intermittently when the phone’s screen is off, the total energy consumption will depend on the duty cycle of these polling events. Perhaps it can be made quite infrequent. Exact power number are not available from MouflonRecorder for comparison, but anecdotally, no users reported negative effects on battery life. Granted, MouflonRecorder only activates every 30 minutes, and in a real system, the authentication would need to activate whenever the phone screen turns on. This could negatively impact battery life. Assuming a system derived from Mouflon that detects behavior by applications performs similarly to the results in this paper, it seems like a system like progressive authentication and a Mouflon-based system have a power/security/convenience trade-off, as the data presented in this paper show that the current Mouflon algorithms grant access to both authorized users and intruders at a higher rate, but at what is likely to be a lower power draw.

These authors also introduce the idea of having three lists for applications based on how private their content is. They allow users of their system to define an application as public (not requiring any authentication), private (requiring moderate confidence that the current user is the owner), and confidential (requiring high confidence). This idea is similar to guest mode frameworks such as those in Section 5.4.2. This three-level model is possible in this approach because, under progressive authentication, the system has a more granular view of whether the current user is authorized or not, rather than the binary view provided by password authentication.

5.1.3 Ultra-Mobile PCs

In [15], Clarke et al. performed an extensive user study to determine the effectiveness of and user attitudes toward a system that seeks to transparently authenticate a user of a mobile device using various biometrics. These biometrics were facial recognition, voice recognition, and keystroke analysis. The system maintains a current state that moves on a scale from -5 to

\(^1\)Power drawn varies depending on which computations are offloaded to a remote computer. This is the figure the authors give in the abstract.
5.1.4 Android

Shi et al. [30] used the combination of several different measurements to actively authenticate a user. The system trains itself on user behavior such as where the user is on the planet on Monday at 10 A.M. (for example) and how long it has been since he has sent a text message to a known number. When the system is activated, these statistics are collected again and compared to the training data to determine whether the user is the correct user. Mouflon analyzes the user’s phone call and text message behavior to the extent that these behaviors require starting Activity. However, the identity of the other party of the phone call or text message is not analyzed. In addition, Mouflon also analyzes the usage of all other apps. Some of these ideas could be adapted into Mouflon by reading the call and text history and comparing it to the contact list, for example. This was not included in this research out of concern for the privacy of the study participants. Normalizing user activity to a specific point in the weekly cycle could be a useful addition to Mouflon.

5.2 Behavioral Profiling through Application Usage

Similar schemes that seek to authenticate computer users by their behavior have also been attempted on other platforms, such as desktop computers. A paper by McKinney and Reeves [25] did exactly this for some computers running Windows XP. They sought to profile Windows users based solely on what applications they use and how they use them. They use the handle counts of processes as a window on how the user is using the computer. They were able to identify their seven users with high accuracy. The amount of data that can be gathered from the Android logs is much smaller than the amount these authors were able to gather from a Windows XP computer. These results suggest that with more intrusive monitoring that looks at lower level details of application behavior, the accuracy of Mouflon might be improved. However,
one of the advantages of Mouflon is that it is not always-on; a production version would likely be configured to stop monitoring behavior after a few application launches that are consistent with the device owner.

5.3 Active Authentication Methods

5.3.1 Actively Enhanced Authentication

De Luca et al. [17] have sought to improve the swipe pattern method of authentication since it is so easy to observe and duplicate. They examine how the user performs the swipe pattern beyond just which points he swipes his finger over. This biometric information from the touchscreen (e.g., speed, pressure) allows them to differentiate between users even when the users are inputting the same swipe pattern into the phone. This is an alternative approach to the one presented in this paper: they seek to make an existing method more effective; we seek to replace it entirely. Their method made an incorrect decision about 20% of the time, which would be a high but possibly attainable goal for Mouflon.

5.3.2 Active Biometric Authentication

**Lifting Phone to Ear**  Gesture-based authentication was accomplished with an admirable success rate by Conti et al. [16], who developed a system that can recognize the motion a user makes when lifting the phone to the ear. This system works well for a scenario where the user is answering a phone call, but it seems inconvenient to use as a general authentication method as that would require the user to make this gesture every time she wants to use her phone. However, they demonstrate the feasibility of using the behavior that the user would perform anyway to authenticate the user. Since a Mouflon-based system cannot make a decision about the user when the only action the user takes is to answer the phone, these two approaches complement each other nicely. This approach can be used when the phone is ringing, and the Mouflon-based system can be used at all other times.

**Gait**  Derawi et al. [18] used the accelerometers on a smartphone to measure the user’s gait as he walked to determine if the phone owner is walking with the phone. These authors were able to achieve an error rate of only 20% when identifying users. However, since the phone was attached to the participants at the hip when the test was conducted, the usefulness of this measurement for authentication is not entirely clear. There are many cases when a user may wish to use the smartphone while not walking around. Additionally, since an intruder is most likely to gain access to a device when it is left unattended, she could use the device before
walking away, giving the system no time to reject her. However, it does further demonstrate the power of using a smartphone’s sensory capabilities to identify people.

5.4 Extensions

5.4.1 Cloud Processing of Data

A paper by Chow et al. [14] presented a system that offloads the heavy processing work of behavioral profiling into the cloud to avoid draining the resources of the phone. These authors sought to solve a different problem than Mouflon; they wish to detect theft of a mobile device, theft of a credit card, or theft of a password by combining behavioral data across all of these sources. Since not all of this data is available on the mobile device, the processing must be done in the cloud. This approach could be used by Mouflon to solve some of its drawbacks such as battery usage and speed. The data could likely be scrubbed sufficiently to defuse any privacy concerns without compromising the ability to analyze it by using the data processing techniques discussed in Section 3.4 remove the identities of the individual applications and absolute times, replacing them with categories and times spent in those categories. Since Mouflon depends on categorizing apps, one of the major maintenance tasks will be to keep these categories up to date. If the processing is done in the cloud, the lists of apps in each category can be stored off the device, which would remove the need for Mouflon to receive constant updates with updated categories. This would, unfortunately, remove any privacy benefit from the categorization.

5.4.2 Guest User Fallback

Mouflon could also potentially be combined with the existing frameworks [24, 26] that are designed to permit a guest user limited access. As they exist now, these guest modes are invoked only upon user request. However, rather than locking the device when a user is classified as foreign, the device could fall into a reduced-permissions mode that only allows the user to perform certain non-invasive actions. If a user wishes even greater security at the expense of convenience, she could potentially set the device to be in the reduced-functionality mode until a positive identification is made. However, a positive identification may be more difficult to accomplish since only a limited set of functionality is available to analyze.
Chapter 6

Discussion

In this study we show that profiling user behavior on a smartphone can be an effective way to differentiate between Alice and Carol, each using her own phone. This is not the problem that would be faced in the real world, however. In the real world problem, we would need to differentiate between Alice and Bob, both using Alice’s phone. There are two effects that could make this problem easier or more difficult than the one studied in this paper. This problem could be made easier because Alice and Bob will have different levels of familiarity with Alice’s phone. Alice will likely tend to find applications and bounce between them more quickly than Bob. This problem could be made harder because while Alice and Carol may have different sets of applications installed, Alice and Bob are both constrained to the applications installed on Alice’s phone. It is possible that this second effect will not be strong because the study in this paper contains data from several months of usage, during which time the study users are likely to have installed and removed apps. Thus, it can be concluded that, while this data is not directly indicative of the ability of Mouflon to be an effective user authentication system, it is highly suggestive of that ability.

6.1 Simplifying Assumptions

This paper requires several simplifying assumptions in order to justify its claims. First, we assume that the user’s habits are fairly consistent over the short and intermediate term. If a user’s habits change, a Mouflon-based system can be retrained, but retraining the classifier costs time and energy. Any behavioral authentication system necessarily depends on consistent behavior. However, if users were incapable of consistently entering passwords, password-based authentication would fail too, so user consistency requirements are not limited to behavioral authentication.

Secondly, Mouflon-based systems assume that each device is the personal device of a single
user. A Mouflon-based system would likely not work well on an Android tablet shared by multiple members of a family. Each one would have different behaviors, and a Mouflon-based system would have a difficult time determining if an intruder’s behavior did not fit with any of the several authorized users.

We must also assume that a foreign user interacts with the device differently than its owner. Intruder tests could potentially show that some foreign users interact with the device in the exact same way as the authorized user. In this case a Mouflon-based system would be unable to tell the difference and the false positive and false negative rates would grow.

In this study we also lack a true intruder model. We choose instead to use other users using their own phones as intruders. This simplifies data collection as we do not need to find any phone thieves or get users (who may have an inaccurate picture of what a phone thief does) to pretend to be phone thieves. The intruder could also be an information thief, who seeks not to use or sell a phone but to get sensitive information off of a phone without the owner’s knowledge. These two types of intruders may behave quite differently. The former may use the phone like his own, whereas the latter may use the phone quite differently than a phone owner.

### 6.2 Training Data

Because Mouflon uses machine learning techniques to build a classifier that makes the key determination, it requires training data to prepare the classifier. In order for a new user to use a Mouflon-based system, he would have to generate some data to train the classifier before it could be used to identify him. This could be accomplished by observing his usage for a period of time until enough data has been collected. At this point further training data could be collected whenever the user indicates that there was a false positive. When there is a false positive, the legitimate user is presented with a password prompt and could indicate after inputting his password if he is unlocking the device because of a false positive or to allow a friend to use it. This information could be used to add the data from that session to the training sets labeled as either owner or intruder depending on what the user indicates. A deployment of a Mouflon-based system would also need to come with some example intruder data as users cannot be expected to find or pretend to be an intruder to train the classifier. Thus the generation of a quality intruder dataset should be a top priority. In this study we used normal user behavior on foreign devices as an intruder model. This was found to be quite effective for many of the participants on the study, and would make a fine starting point for the first version of a Mouflon-based system. If possible, it would also help to find out what phone thieves tend to do with phones when they steal them.

It is also necessary to keep the ratio of user instances to intruder instances in the training data close to 1:1. Since the device owner will continue to generate interaction sequences that
could be used to train Mouflon as she uses the device, there must be some mechanism that permits updating the classification model as the user exhibits new behavior (e.g. a newly installed application) while keeping the overall size of the training dataset relatively constant. Perhaps some algorithm could be found or created that can take a set of interaction sequences and create something analogous to a minimum spanning tree, i.e., the minimum set of instances needed to sufficiently characterize user behavior.

6.2.1 Disjoint vs. Non-Disjoint intruder sets

Appendix B.1 contains data from running the same IBk algorithm with the same 10-fold cross validation except that there were no restrictions placed on which users could be in which “other” sets (see Section 3.4.1). These restrictions are necessary so that the classifier does not “recognize” any training instances as being from the same source as the test instances. In a real scenario the intruder will have never been seen before. This need would be obviated by obtaining real intruder data, but it is not entirely clear how one could go about this. The best way would be to leave a phone running MouflonRecorder somewhere in public and see what an intruder does. This would be prohibitively expensive though, as each dataset would come at the cost of a phone.
Chapter 7

Conclusions

The data presented in this paper show that we can gain some information about the identity of a user based on his behavior. However, the accuracy with which this determination can be made varies greatly from user to user. The analysis was also impeded by the small number of users from whom data was available and the restriction that each foreign user’s data could be used for training or testing, but not both. Without this latter restriction, which meant that only two foreign users’ data was available for training and three foreign users’ data was available for testing, the accuracy with which users are recognized was much higher (see Appendix B). However, in the most realistic model, the false positive rate is approximately 40% for most users. This would not be acceptable for an authentication system because an intruder is allowed in far too often.

There could be several reasons for the observed incorrect classifications. The first could be that users are simply behaving the same. It stands to reason that the way one person checks and replies to a text message may appear the same as another. As discussed in Section 4.2.7, it seems that for some categories of apps, the users in the trial presented here exhibited quite similar behavior, while exhibiting markedly different behavior while using others. Apps that are considered settings or messaging apps were the most likely to have been used in an incorrectly classified instance (relative to the rate they appear in correctly classified instances). Anecdotally, the interactions with these categories of apps will tend to be short. Since the classifier uses Euclidean distance as the metric for determining similarity between two instances, apps with a longer time of usage per instance would contribute more to this metric because it is an absolute metric rather than a relative one. The distance between 1 second and 2 seconds is smaller than the distance between 30 seconds and 35 seconds even though the latter pair is separated by a smaller relative gap, so when interaction times are short they would all tend to blur together. Perhaps a method for normalizing the values of the features would be useful, but this would require some data about the distribution of interaction lengths for each category.
This could be gleaned from the data gathered in this study, but it is a rather small sample as it draws from only six users.

A second possibility is that the features we extracted from the data were not the best features to extract. Some other potential features include

- Sequences of apps: this would allow us to differentiate between a user who habitually checks e-mail first then SMS messages from a user who does the reverse.

- Time of day (perhaps binned into two to three hour bins): if a user is always driving and not using his phone at a certain time, an intruder who uses the phone during that commute would be easily flagged.

- More categories: some categories (e.g. weather) were dropped because only one user in the sample used an app that fell into that category. More categories would allow for fewer apps to be lumped into the “other” category.

- More specific categories: the data allow us to distinguish between sending an e-mail and checking e-mail (because they are different Activities). This could be leveraged.

These additional features could potentially add to the ability to discriminate between users.

A deployment of an authentication system based off of this work is not directly doable for several reasons. First, the classifier is used in a supervised way. In a real scenario, the system would have no way to differentiate between a false and real positive, even after the fact. Therefore, it would be difficult to train on new behavior without compromising the ability to detect an intruder. A semi-supervised classifier would need to be employed. As previously discussed in Section 6.2, we require training data from a better simulation of an intruder. However, the observations in this paper suggest the conclusion that this method of distinguishing users based on their application usage behavior on a smartphone is possible and that further research into an authentication system based on this method is warranted.

7.1 Extensions/Future Work

7.1.1 Intra-application profiling

An extension that would greatly increase the power of a Mouflon-based system would be the addition of the ability to profile the way a user behaves within an app in ways specific to that app. For example, when Alice launches her SMS app, she will likely read and respond to unread messages. An intruder is more likely to search through her message history for potentially sensitive content or send text messages to premium numbers to steal money from Alice. Using the data recorded in the system logs, it is impossible to tell the difference between
these actions. In order to get the information necessary to form a profile of the user’s behavior within the SMS app, it would be necessary to modify the SMS app to record this information. The modifications should be localized to a small part of the application code related to handling user input. It would be easier to create a modified SMS app than it would be for many apps, as the default SMS app is open source [13]. The default Gmail app, on the other hand, is not.

7.1.2 Target profiling

Another extension along a similar vein is to pay attention to whom the user texts or calls as well as what websites he visits. This is referred to as target profiling because it is based on the targets the user performs actions on in addition to the actions themselves. This is some of a user’s most private information, but it need not leave the phone in a production version of Mouflon. It would be compared with that user’s history of sending text messages, making calls, or browsing the web, also all stored on-device. The user’s contact list would make a good starting point for a whitelist of contact information. All of this information should be accessible from an Android app with the proper permissions without modifying any other applications. This modification was not included in MouflonRecorder for the user study because it would be too invasive of the study participants’ privacy, but MouflonRecorder could be modified to collect this information as well. Shi et al. [30] use this method to create a behavioral profile of users as discussed in Section 5.1.4.

7.1.3 Application White- or Blacklisting

A possible extension of Mouflon would be to allow the user to mark certain Activitys as requiring protection regardless of the identity determination Mouflon has made. This allows the user to protect sensitive information that may be accessible by accessing certain apps such as e-mail or SMS messages. This would be useful if the device owner’s most common tasks are also sensitive. Without intra-application profiling as discussed in Section 7.1.1, it is impossible to tell if a user is checking her most recent e-mails or an intruder is scrolling through her entire e-mail history looking for sensitive information because both use the same Activity. The dual of this would be to always allow access to certain other non-sensitive applications regardless of Mouflon’s determination of identity. This would allow an intruder to, for example, play a specific game without the device locking. It could also allow a benign but unknown user to find and call one of the contacts on the phone so that a good Samaritan could try to return the phone. This is similar to the three levels of application protection proposed in [28]. This would be easy to implement in Mouflon since it already examines all Activitys soon after launch in order to create a profile of the user.
7.1.4 Cloud processing

In order to alleviate some of the computational strain of running the machine learning algorithms in Mouflon and collect data from a large number of users to form the intruder dataset, the data processing pieces of Mouflon could be relocated to the cloud as in [14]. All of the infrastructure for the processing of data in the cloud was created for the user study with MouflonRecorder except for the automation of the analysis. However, enough of the analysis is automated through scripts that only a small amount of glue code would be necessary to tie it all together and report the result back to the user’s device.

7.1.5 Guest Mode

The DiffUser [26] and xShare [24] systems allow a device owner to put the device into a guest mode to allow a guest user only limited access to the device. Mouflon could fall into the guest mode provided by one of these systems when a user fails the behavioral authentication check. This would allow any user, even the owner, to continue using the device if only limited functionality is needed. The reverse transition could also be implemented, with the device exiting guest mode into full-functionality mode once the behavioral authentication check is passed.

7.1.6 Development as a System App

The changes to the `READ_LOGS` permission discussed in Section 2.2.2 require any implementation of Mouflon that will work on new Android devices to be a system app. The proportion of devices running 4.1 and later versions of Android is at about 15% and rising at the time of writing [9]. It is likely possible to make a Mouflon-based system work entirely as an app in the Play Store for versions of Android prior to 4.1, but even before the recent changes, developing a Mouflon-based system app always provided distinct advantages. Developing a Mouflon-based system app allows the developers to build in additional functionality, such as interfacing with the internals of other system apps to profile the usage of that app as in Section 7.1.1 or perhaps gathering data on how the user touches the screen as in [17]. A hook could also be built into ActivityManager to pass application launch data to Mouflon directly rather than requiring it to read from the logs. Mouflon would always have benefited from being tightly integrated into the system; with these changes to the `READ_LOGS` permission, this just becomes necessary.
REFERENCES


APPENDICES
Appendix A

Data Formats

03-11 11:06:02.368: I/ActivityManager(1110): Starting: Intent
act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
flg=0x10200000 cmp=edu.ncsu.asbransc.mouflon.recorder/.MainScreen from pid
1110

(a) Android Log Format

03-11 11:06:02.368:::ActivityManager:::START:::cmp=edu.ncsu.
asbransc.mouflon.recorder/.MainScreen:::act=android.intent.action.MAIN:::
cat=[android.intent.category.LAUNCHER]

(b) MouflonRecorder Search Result Format. The colons are used as a highly visible separator.

03-11,11:06:02.368,ActivityManager,START,cmp=edu.ncsu.
asbransc.mouflon.recorder/.MainScreen,act=android.intent.action.MAIN,
cat=[android.intent.category.LAUNCHER]

(c) CSV Format

155,"...13.808,53...",e35

(d) One line of arff Format. The ellipses indicate the other attributes in the instance.

Figure A.1: Data Formats used in Mouflon
Appendix B

Additional Data

B.1 Non-Disjoint Intruder Datasets

The data was also analyzed without separating the other users into disjoint datasets for training and testing. These data may include bias derived from additional data not available in a real-world scenario. Section 3.4.1 and Section 6.2.1 discuss this in detail.

Table B.1: IBk on the test data with non-disjoint train and test intruder sets (see Section 3.4.1) and attribute preselection

<table>
<thead>
<tr>
<th>Instances</th>
<th>5a5</th>
<th>c59</th>
<th>ca8</th>
<th>cb8</th>
<th>d48</th>
<th>e35</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive Rate</td>
<td>383</td>
<td>90.9%</td>
<td>67.9%</td>
<td>61.8%</td>
<td>84.2%</td>
<td>75.5%</td>
</tr>
<tr>
<td>True Negative Rate</td>
<td>83.8%</td>
<td>68.3%</td>
<td>72.5%</td>
<td>83.8%</td>
<td>86.9%</td>
<td></td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>16.2%</td>
<td>31.7%</td>
<td>27.5%</td>
<td>16.2%</td>
<td>13.1%</td>
<td></td>
</tr>
<tr>
<td>False Negative Rate</td>
<td>25.3%</td>
<td>9.1%</td>
<td>32.1%</td>
<td>38.2%</td>
<td>15.8%</td>
<td>24.5%</td>
</tr>
</tbody>
</table>
Table B.2: Confusion matrices for the non-disjoint intruder data

<table>
<thead>
<tr>
<th></th>
<th>5a5</th>
<th>other</th>
<th></th>
<th>c59</th>
<th>other</th>
<th></th>
<th>ca8</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>5a5</td>
<td>286</td>
<td>97</td>
<td>other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td>62</td>
<td>321</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(a) User 5a5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c59</td>
<td></td>
<td></td>
<td></td>
<td>1792</td>
<td>180</td>
<td>other</td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
<td>429</td>
<td>1542</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(b) User c59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ca8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
<td>27</td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>26</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(c) User ca8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cb8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d) User cb8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(e) User d48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>other</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(f) User e35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>