Detecting relay attacks against Bluetooth communications on Android

A thesis

submitted in partial fulfillment

of the requirements for the degree

of

Master of Cyber Security

at

The University of Waikato

by

Jeremy Symon

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2018

Abstract

The widespread usage of mobile devices has lead to mobile devices being used to replace existing solutions, including smart cards for authentication in access control systems. While this can be convenient, there are security concerns which must be addressed. One such concern for access control authentication is relay attacks, which are more difficult to prevent when using Bluetooth on a mobile device rather than smart cards. This thesis investigates an alternative method of detecting relay attacks, using sensors present on typical mobile devices to fingerprint a location. As opposed to other research, this approach uses only the sensors available on a single mobile device, in order to be compatible with existing access control systems that do not have specialised sensors. The proof of concept shows several sensor types are strong indicators of location, particularly observable WiFi signals.

Acknowledgements

I would like to thank my supervisor Dr Ryan Ko from the Cyber Security Researchers of Waikato (CROW) lab, and my supervisor Orion Edwards from the Gallagher Group, for their advice and support throughout my research. I am also grateful to Gallagher Group for the Sir William Gallagher Cyber Security Scholarship, which provided financial assistance to help support my research. Lastly I would like to thank my friends and family for their support over the course of this research.

Contents

Glossary

ARFF Attribute-Relation File Format. 33

BLE Bluetooth Low Energy. 5–7, 12, 13, 22

IoT Internet of Things. 17

MITM Man In The Middle. 1, 8

MOA Massive Online Analysis. 21

NFC Near Field Communication. 5–7, 17

RFID Radio-Frequency Identification. 5, 6, 17

RSSI Received Signal Strength Indicator. 33–35, 38

WEKA Waikato Environment for Knowledge Analysis. 20, 21, 29–33

List of Figures

List of Tables

Listings

1 Introduction

The usage of mobile devices has increased significantly over the past decade. With most people now possessing and carrying a mobile device everywhere with them, mobile devices are being used for many purposes, including interacting with existing technology, supplementing or replacing existing interaction mechanisms. These changes come with many benefits, but also create or exacerbate security concerns.

Relay attacks are a class of attack in which the attacker forwards communication between two legitimate endpoints, without their knowledge that messages are being forwarded [3, 9]. The attacker does not compromise the communication's confidentiality by reading or integrity by modifying the messages as in a Man In The Middle (MITM) attack, but instead merely compromises the endpoints' assumptions about their respective location. This broken assumption compromises the integrity of authentication systems relying on proximity of devices, enabling the attacker to pose as someone with legitimate access to a system by forwarding authentication requests to a device with legitimate access, and forwarding the response back to the access control system.

As mobile devices are being used for proximity-based authentication, relay attacks against mobile devices are becoming a serious concern. There are a number of mechanisms used for handling relay attacks in traditional access control systems, but mobile devices pose new challenges and limitations for dealing with such attacks. Typically relay attacks are detected using a timing-

based distance bounding protocol [13], which times the latency of messages to determine the distance between the communicating devices. Limitations in the hardware and software stacks of mobile devices make timing-based distance bounding unreliable, therefore a new mechanism is required for detecting relay attacks on mobile devices, while working within the hardware constraints of a typical mobile device.

1.1 Objectives

The goal of this thesis is to investigate the feasibility of using the sensors and radio signals available to a typical consumer smartphone device for the purpose of detecting relay attacks. This research will investigate using the sensor inputs available on a typical mobile device to detect relay attacks, by using machine learning to fingerprint the location of the access attempt.

Timing-based distance bounding can be unreliable at detecting relay attacks due to limitations of the software and hardware stacks on consumer smartphone devices. These limitations make it possible for attackers to reliably defeat timing-based distance bounding. The objective of this research is to determine whether a sensor and radio signal based approach can detect relay attacks with a comparable accuracy to that of timing-based distance bounding. Such techniques need not replace timing-based distance bounding; by combining multiple alternative methods of detecting relay attacks, higher accuracy and reliability may be achieved while limiting the attacker's ability to defeat the system.

Consumer smartphone devices commonly contain multiple wireless radios and a range of environmental sensors. Available radios typically include WiFi, Bluetooth, Cell, and GPS. Available sensors commonly include environmental

information such as magnetic fields, acceleration, and orientation. The full list of sensors available on the test device is enumerated in listing A.1.

1.1.1 Scope

The concept is generic enough to be applied to any common mobile device (Android, iOS, Windows Phone). However it is not feasible to target all of them for a proof-of-concept, so only Android will be specifically covered by this research. The scope of this research is limited to testing the feasibility of using sensors and radio signals on an Android device for fingerprinting the locations of static (non-moving) access control points.

1.1.2 Hypothesis

By using the sensors and radio signals available on a common Android smartphone to fingerprint the location of access control authentication attempts and comparing with the fingerprints for known valid prior access, relay attacks can be detected with accuracy comparable to that of traditional timing-based distance bounding.

1.2 Requirements

There are some key requirements which must be met in order to produce a viable system for detecting relay attacks, given the hardware constraints of the target system and the timing constraints of real-time access control authentication.

1. the machine learning model must be trained on the device rather than distributing a pre-trained model, as the data is specific to each access

controlled location

2. models cannot even be shared between user devices for the same location, as the data for the models is recorded by the user devices, which have different sets of sensors with differing calibrations

3. models must perform well on a wide range of low-power mobile devices

4. there is only one class (successful access), as unsuccessful access covers everything else, and is unlikely to occur often enough to train even if it were possible to train for

5. the model needs to be updated with new instances over time as the environment can change

1.3 Thesis Structure

Chapter 2 will discuss background information relevant to this research. This covers relevant wireless protocols, relay attacks and mitigation techniques, and data analysis techniques.
Chapter 3 will discuss the different sensors available on a typical Android device, their value for detecting relay attacks, and the design of our classifier for detecting relay attacks.
Chapter 4 will evaluate the classifier's effectiveness at detecting relay attacks using different sets of sensors.
Chapter 5 will give recommendations on implementation of sensor-based relay attack detection.
Chapter 6 will conclude the research, and give suggestions for future work.

2 Background

2.1 Wireless Access Control

Historically, access control systems have used Radio-Frequency Identification (RFID) smart cards as an authentication mechanism [22, 31]. However, with the recent ubiquity of smartphones, these systems have sought to develop comparable authentication mechanisms using smartphones in place of smart cards. Near Field Communication (NFC) was a logical choice for such a mechanism, being based on RFID standards and compatible with existing RFID technology. However, NFC was not available for use by apps on iOS until the recent release of iOS 11 [6]. This led multiple security systems companies to instead develop authentication functionality using Bluetooth Low Energy (BLE) [21, 31].

2.1.1 Smart Cards

Contactless smart cards are the technology traditionally used for authentication by access control systems [21, 31]. There are a number of standards used in access control. Older Wiegand smart cards operating on the 125 kHz frequency band behave as a simple value store, which typically allows encoding and retrieving 26-bits of data [30]. This causes serious problems for secure usage, as an attacker could listen to the communications and learn the "secret" value, or simply brute-force the id (typically 16 bits). These have been superseded by the newer 13.56 MHz smart cards, which have an internal chip

capable of cryptographically generating a single-use code which can be verified without revealing the data stored on the card [1]. These devices typically have a read range of less than 10 cm and a low, predictable packet latency (except when running cryptographic operations), which is useful for detecting and preventing attempted relay attacks.

2.1.2 NFC

NFC is a set of wireless communication protocols that enables low-bandwidth communication between electronic devices (typically a smartphone) over a distance of less than 4 cm [2]. Contactless RFID smart cards and NFC devices both operate in the 13.56 MHz band, and are inter-compatible between reader devices [2]. This enables a smartphone to fill the role of a smart card, however restrictions by certain smartphone platforms have limited the uptake of smartphones replacing smart cards. iOS did not allow apps to use NFC until recently with iOS 11 (2017) [6]. iOS still does not allow listening for NFC tags in the background, which has been supported by Android since API 9 (2010) [5].

2.1.3 Bluetooth

Bluetooth is a wireless communications technology that supports wireless communications over distances between 1 and 100 metres, depending on the mode and class of the devices being used [12]. Bluetooth has evolved through several iterations with different features and properties. Of these, BLE is most relevant to this project's domain, as it is being used for authentication in access control systems, in place of traditional short-range authentication mechanisms such as contactless smart cards [21, 31]. Bluetooth behaves significantly differently from the contactless smart cards and NFC, which enables some useful new functionality (such as authenticating from a great enough distance that a

truck driver would not need to leave their vehicle), but also introduces some new problems.

2.1.3.1 BLE

BLE is a wireless communications technology based on Bluetooth, that was designed as a low-power solution for control and monitoring applications. Low power consumption is achieved by maximising the proportion of time that the Bluetooth hardware is in a low-power sleep state, at the expense of latency and data throughput [23, 41]. BLE on smartphones (especially Android) has unpredictable latency, which makes it easier for an attacker to perform a relay attack compared to alternative technologies such as smart cards. The connection interval when the device actually transmits can vary from 7.5 ms to 4 seconds, and is negotiated between the two devices [12]. The possible range of values varies between different devices, and may not be possible to determine from application code on the device. A sent or received packet may spend a significant length of time in the system Bluetooth stack before being actually sent or passed to the application code. These factors differ between hardware, different software versions or configurations, and between different battery levels or "power saving" modes on the same device. The diversity of the Android platform makes this a much more significant problem than on other platforms such as iOS which have a comparatively small set of devices to consider.

2.2 Relay Attacks

A relay attack consists of an attacker forwarding a communication between two legitimate endpoints, without their knowledge. This is distinct from a MITM

attack [14, 25] as the attacker does not read or modify the communication (and in fast may not be capable of doing so), but instead gains an advantage by merely forwarding the communication. This advantage typically consists of breaking the endpoints' assumptions about who they are communicating with.

A traditional description of a relay attack is the Chess Grandmaster Problem: a story in which a little girl wins a game of chess against a grandmaster [17]. In the story, the girl plays two simultaneous games against two grandmasters via mail, one as black and one as white. Once her white opponent makes a move, she copies that move to her game against the black opponent, then copies their response to her game against the white opponent, and so on. In this way, she is guaranteed to either win one game or draw two games against chess grandmasters, without needing any skill at the game.

In the scenario of an access control system, a relay attack can be used to gain access by talking to both the access control system (verifier) and the user's mobile device (prover). By forwarding an authentication request from the access control system to a user's device which has legitimate access to that system, and forwarding the authentication response from the device back to the access control system, the attacker can gain entry. The access control system grants access because it assumes a device with legitimate access is present, however that device may be a long distance away and being forwarded by the attacker.

There are two common types of relay attack against authentication systems: mafia fraud; and terrorist fraud [9]. In a mafia fraud, there are two cooperating attackers taking the place of the little girl in the chess grandmaster problem, one talking to the prover and one talking to the verifier, and forwarding messages between each other. In a terrorist fraud, the prover collaborates with an attacker to fool the verifier into granting access to the attacker. This re-

search is only concerned with the "mafia fraud" attack, as the mitigation will be performed in the prover (application on a mobile device), which is already compromised in the "terrorist fraud" attack.

2.2.1 Detection / Prevention

Wireless authentication systems commonly rely on distance as a form of security: the wireless connection requires the prover to be in close proximity to the verifier, which means the authorised party must be physically present. Relay attacks break this assumption by arbitrarily extending the distance between the prover and verifier by placing an attacker between them. Therefore an attempt is made to prevent this attack by some type of "distance bounding": proving the distance between prover and verifier is within an acceptable range.

2.2.1.1 Signal Strength

Rudimentary distance bounding can be achieved by measuring the strength of the received signal [7, 15]. If the transmit power is known, the distance can be calculated based on the difference between the received signal strength, and the strength of the transmitted signal. This has some serious problems that make it impractical to use in reality however. Interference from other signals, physical barriers, and relative movement of the devices can all affect the received signal strength, which will reduce the accuracy of the distance calculation [7]. An attacker can relay and re-transmit the signal from a closer location, resulting in a high received signal strength and short perceived distance. An attacker can also transmit with a higher than expected power or focus the transmission on the receiver with a directional antenna, resulting in the received signal strength being higher than expected even over a large distance.

2.2.1.2 Timing based

Typically "distance bounding" refers to timing-based distance bounding. The idea behind this is that a relay attack will take some amount of additional time to receive, relay, and re-transmit the data. If this additional delay is detectable, then the relay attack can be detected.

Timing based distance bounding works by sending a series of packets and timing the round trip time to establish a lower-bound on the distance between the devices. There are a large number of approaches for achieving this with varying levels of reliability and efficiency in different network environments. Inevitably they rely on predictable packet processing delay in order to minimise unknown timing and calculate an accurate distance. A naive approach sends "ping" requests which are immediately responded to by the receiver. This approach can produce very high resolution distance bounding as no digital processing is required before responding, so the response delay can be as low as a few nanoseconds [38, 13]. In practise such a protocol could be trivially intercepted by an attacker near the sender, who responds immediately and fools the sender into measuring the distance between the attacker and the sender, rather than the longer distance between the sender and intended receiver. In order to prevent this preemptive response attack, timing packets need to contain some information that the attacker cannot predict, so only the intended receiver can produce a valid response which is verified by the sender. Some approaches to preventing this attempt to perform some cryptographic operation such as signing a value at the receiver before responding [7]. This is not ideal, as it takes a significant amount of time compared with the actual latency, which complicates timing, but may still produce an accurate result as long as the cryptographic operation takes a predictable time. Other approaches use a shared secret, and perform distance bounding by sending single bits from that

secret value [13, 33, 36]. The verifier sends random challenge bits, and the prover responds with the result of a function of the challenge bit and the next sequential bit from the secret value—for instance, $XOR(challenge, secret)$. This simplifies the necessary data processing so the response delay can be very small, on the order of 100 ns [13]. Assuming the attacker doesn't know the shared secret, they have a $(\frac{1}{2})^n$ of guessing the correct response sequence in order to respond faster than expected and break the distance bounding calculation. There are multiple extensions to this distance bounding protocol. [27] achieves much faster operation over a high-bandwidth lossy channel, at the cost of the attacker gaining higher chance of guessing the correct response at $(\frac{3}{4})^n$, meaning more challenge-response cycles must be completed to achieve the same accuracy. [36] extends [27], introducing an third "empty" challenge bit value, to which there must be no response. This complicates processing, but improves the accuracy back to $(\frac{1}{2})^n$.

In a wireless communications medium such as Bluetooth where signals propagate at approximately the speed of light, proving co-presence within a 100 metre radius requires packets to be processed and replied to in less than a microsecond (because it takes less than a microsecond for light to travel 100 metres). The most efficient distance bounding protocols use analog processing to process and reply to a packet within a nanosecond, two orders of magnitude faster than the most efficient digital implementation [38].

Some distance bounding protocols use a slower-than-light medium such as ultrasound to perform distance bounding [15, 26]. Distance can be measured more precisely in a slower-than-light medium, as it takes longer for the signal to propagate the same distance. Sound for instance is six orders of magnitude slower than light [15], which enables even cheap simple hardware to accurately measure distances based on signal latency. By comparison, where measuring a

100 metre distance requires sub-microsecond processing at light-speed, it can be measured with ultrasound with timing accuracy on the order of 100 milliseconds, because sound only travels at approximately $340ms^{-1}$ in air. However this comes with a serious flaw which can be exploited by an attacker: as the communications medium is slower than light, an attacker can relay the communications over an out-of-band communications link which propagates faster than the intended channel (e.g. a radio or wired electrical link), thereby delivering the data to the recipient earlier than expected and beating the distance bounding algorithm.

The Bluetooth Low Energy (BLE) stack introduces a significant amount of fixed delay in communications, which increases the minimum round-trip-time to several milliseconds (six orders of magnitude slower than an analog RF implementation). Due to the band jumping nature of the BLE protocol, an attacker cannot significantly reduce this delay. In fact, a naive relay attack requires two Bluetooth connections rather than one, which introduces additional delay which can be trivially measured. Using this method, Gallagher's [21] internal testing achieved high relay-attack detection rates (90–100% depending on the device for a basic proof-of-concept relay attack). However, this method of relay attack detection has some flaws which can be exploited by an attacker to achieve a relay attack with much higher rate of success:

- There is variable delay in the Bluetooth stack caused by packet processing and retransmission, which cannot be controlled or observed by an application running on a mobile phone. This means the detection accuracy will vary between devices, and resulting in lower accuracy or even false-positives on some devices.

- A sophisticated attacker could record and relay the raw RF signals used for Bluetooth communication, either by recording all of the RF chan-

nels (79 for Bluetooth Classic; 40 for BLE [12]), re-transmitting them at the other side of the relay, or by detecting the active channels or intercepting the data used for picking the channel hopping sequence. This would result in a single Bluetooth connection transparently bridged over an arbitrary distance (so long as the distance is small enough for the transmit windows to still line up). There does exist some research into using channel hopping as an anti-relay attack system, based on the assumption that an attacker cannot relay all channels of a channel hopping communication system in parallel [3]. This assumption does not appear to hold against a sufficiently equipped attacker: at best channel hopping provides a linear increase to the complexity of a relay attack. Even if an attacker cannot detect the active channel and is forced to relay all of the channels, they can relay each channel individually by simply adding more hardware to handle each channel as required.

One possible attack against this kind of timing is to overclock the networking hardware to send at a slightly higher frequency [15]. Typically, communications systems include a clock in the transmission to which the receiver synchronises, to avoid the sender and receiver getting out of sync due to clock drift. If the attacker slightly increases the rate of the signal's clock while remaining within the tolerance of the receiver's equipment, the communication will proceed at a higher speed, resulting in the packet being received slightly faster than expected. This clearly cannot decrease the time taken for a bit of information to actually travel from the sender to the receiver, but may decrease the time spent in a send queue, resulting in a full packet being completely received faster than it otherwise would be.

2.2.1.3 Environment based

Another method of detecting relay attacks is to test for co-presence in the same environment based on sensor readings the devices can gather about the environment [40, 43, 24]. When two devices want to prove they are present at the same location (not communicating over a relay attack), they use built-in sensors to read information about their environment, and compare that information to determine whether they are in the same location. Devices that are co-located should read similar values on all of their sensors, whereas they are unlikely to do so when in completely different locations during a relay attack. This requires the two devices both have matching sets of sensors which are calibrated to read values similar enough to detect co-presence, and precisely enough to detect when the devices are not co-present. The sensors also need to provide accurate readings within a reasonable time-frame in order to provide a smooth user experience. A common upper limit for the sensor window is 500 ms [24]. Not all of the sensors commonly available on Android devices return a usable value within this 500 ms window.

[24] enumerated the sensors on an Android device, and evaluated them based on time required to produce a usable value. The sensors that could produce a usable reading in the 500 ms window were:

- Accelerometer
- Gravity
- Gyroscope
- Light
- Linear Acceleration

- Magnetic Field
- Rotation Vector

These sensors were evaluated individually, and were barely better than random guessing (the best sensor having a 38.9% equal error rate, i.e. a best case of 38.9% false positives and 38.9% false negatives). The low accuracy of these sensors is unsurprising, as they measure more about the exact position and orientation of the device than any proximity to another nearby device. They suggested for future work to combine multiple sensors to provide a more accurate combine reading.

[40] evaluated a number of sensors provided by a third-party peripheral device "sensordrone". This device had a different set of sensors to those commonly available in an Android device:

- Temperature
- Humidity
- Precision Gas (Carbon Monoxide level)
- Atmospheric Pressure

Individually these sensors provided 70–80% accuracy by comparing readings between two sensordrone devices. When combining multiple sensors, accuracy was improved to 90–95%. The sensors were also able to instantaneously return a reading when polled, however this is because they were always on and constantly monitoring. Any mobile device could achieve the same by constantly monitoring its sensors, at the cost of battery life. This is impractical as a mobile solution, because both devices require an external sensor device with its own power source.

[43] evaluated a different set of sensors on an Android device, and evaluated them individually as well as when combined:

- Audio (Microphone)
- WiFi (RSSI of nearby devices)
- Bluetooth (RSSI of nearby devices)
- GPS Raw Data

The GPS data is unlikely to be practical within a reasonable timeframe, as it is normally turned off to save battery and takes too long to turn on and detect the necessary satellites. For this reason the paper [24] excluded GPS from their testing. This paper used a window ranging from 5–15 seconds for recording data, which is why GPS can be used here. WiFi and Bluetooth were also excluded by [24], likely because devices do not typically send announcements more frequently than 500 ms in order to get a new RSSI value within that timeframe. However this does not matter as much as for GPS, because WiFi is typically always on, and Bluetooth will always be on when running an app that is authenticating over Bluetooth. As WiFi and Bluetooth will be turned on, they will have received the most recent RSSI readings, which will be reasonably accurate (unless the device is moving extremely fast, in which case a relay attack may be falsely detected and the user would need to stop and try again). It was found that WiFi was the best predictor, and combining multiple predictors created improved resilience against attacks while retaining a low false-positive rate.

2.2.2 Attacks

There are a number of relevant examples of relay attacks against wireless authentication systems, including research into defeating existing distance bounding protocols, and research specific to attacks against Bluetooth.

Smart locks are a recent Internet of Things (IoT) development that provide an access control solution targeted at individual consumers. Among a myriad of other critical vulnerabilities commonly found in IoT devices, they have also been found to be vulnerable to relay attacks against their Bluetooth communications [39, 32]. Bluetooth in particular has been shown to be especially vulnerable to relay attacks [34], in part due to its high and unpredictable latency reducing the effectiveness of latency-based distance-bounding [15]. Some possible mitigations such as geo-fencing are discussed in [32]. Geo-fencing requires a recorded location for the access control reader, which may not be practical to add to existing systems which can have hundreds of existing readers without a location record. It also requires an accurate location reading on the user's mobile device, which will be inaccurate without either leaving the GPS on (draining battery) or waiting for a signal after turning it on (taking too long for a smooth user experience).

Smart card and NFC technology has also been shown to be similarly vulnerable to relay attacks. Attacks have been demonstrated against card-based financial transactions [18], Mifare access control cards [26], and mobile NFC financial transactions [19]. In all of these instances distance bounding is suggested as a mitigation to protect against relay attacks, however it is noted that distance bounding is not always practical or reliable due to the tight timing constraints. This is less of an issue for RFID technologies than with Bluetooth, however remains an issue with low-cost passive RFID tokens due

to their resource and cost constraints [26]. Some additional mechanisms for preventing and detecting relay attacks are discussed in [19]. These include detection mechanisms such as checking location and cell tower signals, which will be further investigated in this research; and modifying the communications protocol to make attacks more difficult (which is out of scope, as this research focuses on consumer smartphone devices using Bluetooth).

2.3 Data Analysis

In order to detect relay attacks using smartphone sensors, the data gathered from the smartphone's sensors will be analysed using a machine learning algorithm. Due to the requirements described in section 1.2, this classification problem requires a classifier capable of streaming and single-class classification.

2.3.1 Single-class Classification

Single-class classification is a relatively uncommon type of classification, in which the algorithm tries to distinguish instances of a specific class from all other instances, by training on only instances of the single class [35]. This differs from the typical multi-class classification problem, in which training data is available for all the classes, and the algorithm finds differences which separate each class. Single-class classification may be used when there is only training data available for a single class, or when there are an infinite (or very large) number of other "classes". The algorithm has to use the information it has about the one class it knows, to determine whether an instance looks like it fits with the training data, or appears to be different.

Fingerprinting locations using sensor data is a single-class classification problem. There are nearly infinite possible locations, so the classifier cannot

possibly be trained on each of them. Instead a single-class classifier must be used, and trained on the single location for which there is data. When the device is present in the expected location, the classifier will determine the instance is in the same class as the training set, and when it is in another location the classifier will determine the instance is an outlier.

A common technique for building a multi-class classifier uses a density function [35, 42, 44]. The probability density of the target class is calculated from the training data, and a region enclosing the target class is computed, with a threshold at a chosen cutoff probability density. When classifying a new instance, instances outside of this threshold will be classified as unknown (not of the target class). The threshold value affects the false-positive and false-negative rate, and must be chosen as such depending on the nature of the data and the application. In effect, this threshold can be represented as a region in N-dimensional space, where each dimension is an attribute normalised by the probability density function for that attribute.

Single-class classification can also be built on top of a multi-class classifier by generating artificial data to produce a second class which the multi-class classifier can use [28]. This enables using any multi-class classifier that can produce class probability estimates, and can be combined with the traditional density function to yield improved performance over using the density function or multi-class classification model alone.

2.3.2 Streaming Classification

A streaming machine learning classifier always has a usable model, and continually updates the model whenever it receives a new training instance [8]. This differs from traditional batch machine learning, which has access to all of its training instances when training, and can only be used for classification

once training is completed. In order to handle an arbitrary amount of samples gathered (streamed) over time, a streaming classifier has different requirements from a traditional batch classifier [11]:

1. Process one example at a time, and inspect it only once (at most)
2. Use a limited amount of memory
3. Work in a limited amount of time
4. Be ready to predict at any time

This streaming behaviour is useful when working under resource constraints such that the entire training set cannot be analysed at once, or when dealing with a continual source of new training data, where it is impractical to retain all previous training instances to recompute the model entirely. Both of these situations apply to this project—the most important requirements are 1 & 4 as new instances are received at a very slow rate, but the classification is time-sensitive, and the entire set of instances may become large enough to encounter resource constraints.

2.3.3 Weka

Waikato Environment for Knowledge Analysis (WEKA) is a collection of machine learning algorithms, data pre-processing tools, and analysis tools which supports easily exploring datasets and evaluating machine learning approaches [20].

WEKA contains a large set of common machine learning algorithms well suited to classifying data into different classes, such as the J48 decision tree algorithm and the Naive Bayes classification algorithm [37, 10, 16]. These can achieve high accuracy on many datasets, but lack features required for

this domain. Namely, a suitable classification algorithm must be capable of single-class classification; and incremental updating / streaming learning.

There are approaches for enabling streaming learning in many machine learning algorithms [8, 11], some of which implemented in WEKA, and more supported in Massive Online Analysis (MOA), WEKA's streaming-focused machine learning counterpart [11]. There are also approaches for single-class classification, with one such algorithm implemented in WEKA [29]. However, none of the classifiers supported by WEKA support both streaming and single-class classification.

Therefore a custom classifier will have to be produced to meet the requirements of this project. WEKA's analysis tools will be used for parsing input data, and for testing and validating the custom classifier.

2.3.4 Predictors

There are a range of potential predictors available on a smart phone device which can be used for detecting relay attacks. For the purposes of this research only the sensors and wireless signals available to a typical Android smartphone are considered, however other possible predictors have been enumerated for reference.

2.3.4.1 Timing

Timing-based distance bounding works by calculating an estimate of the distance between two devices, based on the latency in communications between the devices. This relies on the communications medium between the devices having a predictable latency which can be used to determine the distance between the devices. In the case of distance bounding with a light-speed communications medium, it can be functionally impossible to achieve a relay

attack as any added distance between the devices will add latency in accord with the speed of the communications medium. In sub light-speed communications media, it is possible for an attacker to forward communications over a faster medium, tricking the distance bounding and allowing communication over a greater distance. In the case of Bluetooth, this problem is even worse: Bluetooth communications are technically transmitted over a light-speed medium (radio frequency), however a number of optimisations for power usage and reliability result in high latency which can be variable and unpredictable. As shown in Figure 2.1, existing timing-based distance bounding implementations for Bluetooth (such as those used by devices providing authentication over Bluetooth) rely on the doubling of software stack latency caused by waiting for the transmit window of an additional Bluetooth connection, rather than the inherent latency of the communications medium. This itself is unreliable due to the unpredictable latency of introduced by the Bluetooth stack on different devices and software versions. This can also be defeated entirely by an attacker capable of recording and re-transmitting all of the Bluetooth wireless channels (79 for Bluetooth Classic; 40 for BLE [12]), in effect bridging a single Bluetooth connection over another link. As shown in Figure 2.2, this can significantly reduce the latency introduced by the relay attack, making packet timing an unreliable predictor of relay attacks.

2.3.4.2 Historical Access

In a typical access control situation, a user's access will be predictable to some extent by two main factors: time; and path. Both of these predictors would optimally be implemented on the access control server which has all of the necessary historical data for each user. This requires integration with a specific access control server, which is out of scope for this project.

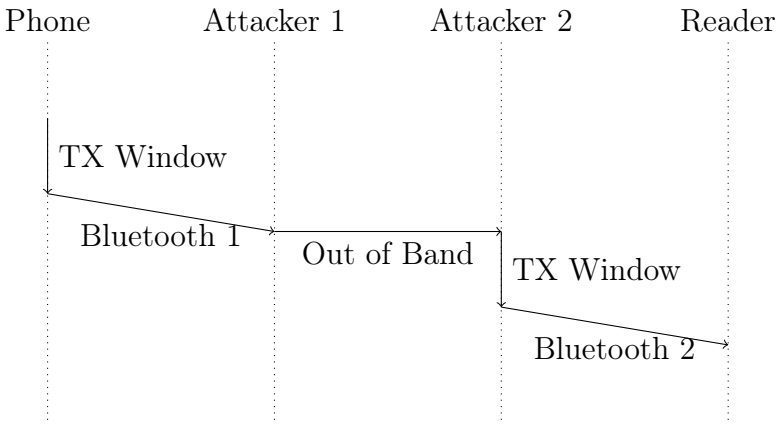


Figure 2.1: Relay attack on Bluetooth authentication using two connections

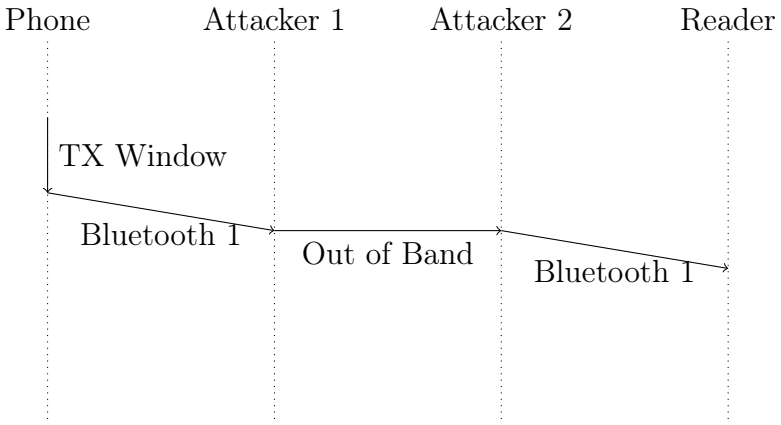


Figure 2.2: Relay attack on Bluetooth authentication using single bridged connection

Time Based

If the user follows a schedule for accessing zones (such as arriving at their office at the same time every day), that schedule can be learned and used to detect and mark anomalous access attempts as a probable attack.

Path Based

Areas in an access control system can be represented as nodes in a directed graph. When authenticating for access to an area, the system checks that the user's most recent known location is in an area that contains the door being used, and is connected to the new area. An access attempt from an invalid source location would be marked as a possible relay attack. This system requires support from the access control system in order to track users through zones. A lesser offline approach could be achieved by recording on the mobile device which access points have been entered, and building a model of how the user normally gets into a zone. Either of these approaches could of course be defeated by an attacker, but they are an additional barrier and potentially require the attacker to make multiple access attempts, raising the chance of detection.

2.3.4.3 Wireless Signals

By recording detected wireless signals during previous instances of successful access, a model can be built for what signals are expected in a future access. There are four common wireless signals available on mobile phones: Bluetooth; WiFi; Cell; and GPS. Using wireless signal strengths for distance bounding is a similar concept to the basic signal strength based distance bounding discussed in Section **??**. However this approach differs by using all available signals to recognise a location, thereby placing less reliance on the signal strength of a

single signal, and also considering the presence of other signals in the area. As such it is more complicated to attack, but still suffers the same general weaknesses with signal reliability and ability to spoof signals.

Bluetooth

Bluetooth is likely to have a high number of signals and a short range which can be helpful for building a precise model, however there are a number of significant problems here: Due to the short range and common usage of Bluetooth for personal devices, it is likely that a significant proportion of detected devices will frequently move or disappear entirely, making them useless and potentially detrimental to the model. As Bluetooth is also the protocol being protected for access control purposes, a number of the measured signals will belong to access control devices which must exist in order for access control to work (which makes the use of such signals in the model redundant). Any attacker is also already going to be spoofing Bluetooth addresses, and could spoof additional addresses with relatively low effort.

WiFi

WiFi covers a larger area than Bluetooth and typically has a more consistent signal strength and doesn't move around. WiFi access points in particular typically stay in a fixed location until they are (infrequently) added, removed, or replaced. It is likely the strongest WiFi signals in the vicinity of the access control point are controlled by the operator of the access control system, however WiFi access points can still change or appear/disappear without notice, or be temporarily created by mobile devices sharing their cellular connection. An attacker can also spoof WiFi signals without much effort, just as they can with Bluetooth.

Cell

Cell towers cover a much larger area than Bluetooth or WiFi, to the point they are likely not to be useful for distance bounding over short distances. With signals from multiple cell towers it may be possible to triangulate a location, however signal strength readings from unused cell towers may not be frequently updated or directly exposed by cellphone operating systems, which prevents this from working reliably. As with all wireless signals, cell towers can also change addresses or appear/disappear unexpectedly, and can be spoofed by an attacker with the right equipment.

GPS

GPS is unlikely to be useful, as the GPS radio typically won't be powered on, and the time it takes to power on and retrieve usable data is too long for an access control authentication scenario [24]. Due to this limitation, reading GPS coordinates would most likely fallback to computing a location based on cell towers and nearby WiFi access points—which are already being used directly, or use a cached location—which can be extremely inaccurate. As the absolute location is unimportant (only the distance relative to the access control device matters), this computed location estimate doesn't provide any additional useful data, and instead would be redundant and potentially unreliable.

2.3.4.4 Other Sensors

Android devices have a number of other sensors such as orientation, magnetic field, light, and proximity sensors. None of these appear to have any strong correlation with location or proximity to an access control reader, but they should be considered for completeness. Listing A.1 shows the complete list of sensors available on the Android device used for testing. Many of these

sensors have high correlation, being different measures of the same property, or different processed forms of the same measurement. This would result in low efficiency when needlessly combining multiple sensors with high correlation, and may negatively impact prediction accuracy depending on the classification algorithm.

Accelerometer, Gyroscope

There are a range of acceleration/rotation sensors available on Android devices. These sensors are closely related and are used to calculate a range of software sensors.

- Accelerometer—provides the acceleration force applied to the device
- Gyroscope—provides the rate of change of rotation of the device, which is used along with the accelerometer and magnetometer to determine the rotation vector of the device.
- Gravity and Linear Acceleration—calculated based on the accelerometer and the rotation vector (the acceleration applied by gravity, and the acceleration excluding gravity, respectively).

About half of the sensors available on a typical Android device are actually based on some combination of the accelerometer and gyroscope. This makes some of them redundant as predictors for classification, however others may be improved by their software pre-processing which removes potentially confounding information (e.g. linear acceleration is the accelerometer with gravity removed).

These sensors are not expected to be useful for distance bounding, as the phone will most likely be in a person's hand or pocket, rendering its orientation and acceleration unreliable. The best possible scenario would be if the phone

was placed on a flat surface, then it could perhaps detect any slope of the floor or swaying of the building.

Magnetism

Magnetism is the most promising of the sensors available on a typical Android device. It detects not only magnetic north, but also local magnetic fields which may be caused by electronics and machinery. This is unlikely to be a strong indicator in general, but may be useful in some extreme situations. One issue is that the sensor is directional, but the orientation of the phone is not consistent. This could be dealt with by taking multiple measurements and filtering out the magnetic north component of the reading, or by processing the magnetic field to only consider the strength rather than the relative orientation of the field. Reducing the reading to a magnitude scalar would greatly reduce the information available for prediction. Instead of detecting a magnetic field in a specific direction, only the presence of an abnormal magnetic field would be detected, which is far less unique and therefore less useful as a predictor for classification.

Light, Proximity

The light and proximity sensors are completely useless as predictors for distance bounding. The light sensor is often used to implement a proximity sensor in software by detecting reduced light, and is not designed for measuring ambient light which could potentially useful information about the environment. They provide information about whether the phone is in a pocket/bag/closed case/against someone's face, rather than any useful information about the environment.

3 Design

This project focuses on protecting Bluetooth-based Mobile phone authentication systems from relay attacks. Timing-based distance bounding is not reliable on Bluetooth, and therefore is insufficient for securing an authentication system that communicates over Bluetooth. To solve this problem, timing-based distance bounding can combined with a number of other relay attack indicators to form a statistical model that provides more accurate detection of relay attacks. To use these other relay attack indicators, a custom machine learning classifier will be created. This classifier fingerprints locations based on sensor data, and predicts whether a given reading is at the expected location (not an attack), or elsewhere (a relay attack).

For the purposes of testing, the classifier is split into two applications: an Android app gathers sensor data, and a WEKA classifier performs the actual classification from input datasets. WEKA is the University of Waikato's machine learning suite, that includes a collection of tools for processing data, running machine learning algorithms, testing the performance of classifiers. Implementing the classifier for WEKA is convenient, as it enables tweaking and analysis of the classifier's predictions using pre-existing tools built into the suite. As WEKA is implemented in Java, the classifier can also be directly used in an Android app without any changes to the implementation. WEKA also provides meta classifiers which can be used to combine multiple classifiers into a single classifier that uses the predictions of all of its combined classifiers

to provide a more accurate prediction. This feature could be used to combine classifiers for multiple relay attack predictors, including a timing-based distance bounding approach.

3.1 Classifier

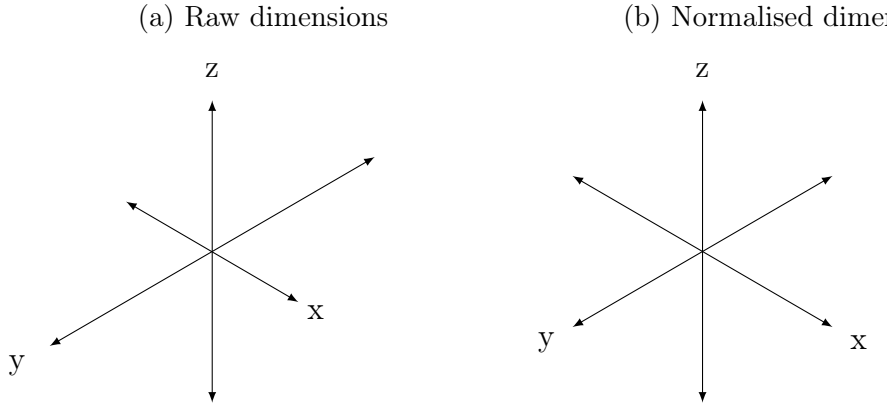
WEKA contains a large set of machine learning algorithms, none of which meet all of the requirements for this problem listed in section 1.2. Some of the algorithms could theoretically be modified to support more of the requirements, however the necessary modifications would require extensive changes and still not be ideal. For instance: only a small subset of WEKA's available algorithms support updating a model with new instances (requirement 5). These algorithms don't support classifying instances for a single class (requirement 4). On the other hand there are single-class classifier algorithms, which support classifying instances as either in a single class, or unknown. The intersection of streaming and single-class classifier algorithms is smaller again, and is not implemented in WEKA. Some could potentially be modified to handle both, for instance using a wrapper algorithm that wraps a multi-class classifier to produce a single-class classifier, or modifying an algorithm to support single-class classification. For example, K-nearest-neighbours algorithm can support incremental updating as well as handling single-class classification by checking the distance to the neighbours (as all the neighbours will be in the same class, so distance is necessary to determine whether an instance is in or out of the class). This approach becomes very complex, and raises performance problems due to high storage usage for the model itself, and high processing power demands for classifying new instances (requirement 3). A common way for dealing with high requirements for processing power on mobile devices is

to offload the processing to a server that pre-processes the data, so the mobile device can perform minimal processing on its end. Requirements 1 & 2 prevent this from being practical, as each (mobile-device, access-control-point) pair needs its own model.

To meet these requirements while also being usable in WEKA for performance analysis, a simple custom machine learning algorithm has been implemented. This algorithm builds a cluster from the known good instances. The algorithm takes a set of instances, each containing readings for a number of different wireless signals. The data is processed by considering each unique wireless signal or sensor as a dimension, where each instance is represented by a position in N-dimensional space, N being the number of attributes. A cluster of instances is formed based on the trends in the attribute (sensor / signal) values. Instances are then classified by calculating the distance between the instance's position, and the centre of the cluster of known good instances.

Some bias can be produced by the magnitude of different attributes, shown in Figure 3.1a. Attributes with a greater range in values will affect the distance calculation more than an attribute with a smaller range. This is particularly common between different types of sensors representing different physical properties with different scales. Another cause can be a sensor/signal with a lot of noise. This sensor should be ignored due to being unreliable, but will instead have a high weight due to its large range in readings. To prevent this, attributes are normalised and stored as a standard deviation and mean, as shown in Figure 3.1b. This makes attributes contribute more evenly to the distance calculation, and means unreliable sensors with a large range will have a high standard deviation and therefore a low normalised distance. Conversely, sensors with a tight group of values will have a low standard deviation, so any abnormal values will contribute more significantly.

Figure 3.1: Example cluster of three attributes



All calculations used in the algorithm are chosen to ensure they can be efficiently and accurately updated with minimal data, so updating the model with new training instances can be done quickly and without needing to store or re-process previous training instances. In order to allow updating the classifier, the only additional data to store is the number of instances containing each attribute. This means that each attribute is represented by its mean, standard deviation, and count. When storing each value as a double, this requires only 24 bytes per attribute, making the model very manageable on any hardware. If an access attempt (instance) is successful, it used to further train the model, incrementally updating the model over time as it is used. The count is trivially updated: $count \mathrel{+}= 1$. The mean is also trivially updated using the count as shown in Figure 3.2. The standard deviation can also be updated by using the variance, which is the square of the standard deviation, the mean and the count, and the equation in Figure 3.3. As the model has only been implemented for evaluation in WEKA and not in a live environment, incremental updating is not implemented beyond proving that it is supported.

The major remaining problem with this model is that it requires data which can only be generated by successful usage in the live environment. The first

Figure 3.2: Updating mean with a new value

$$mean = \frac{x + mean \times oldcount}{count}$$

Figure 3.3: Updating variance with a new value

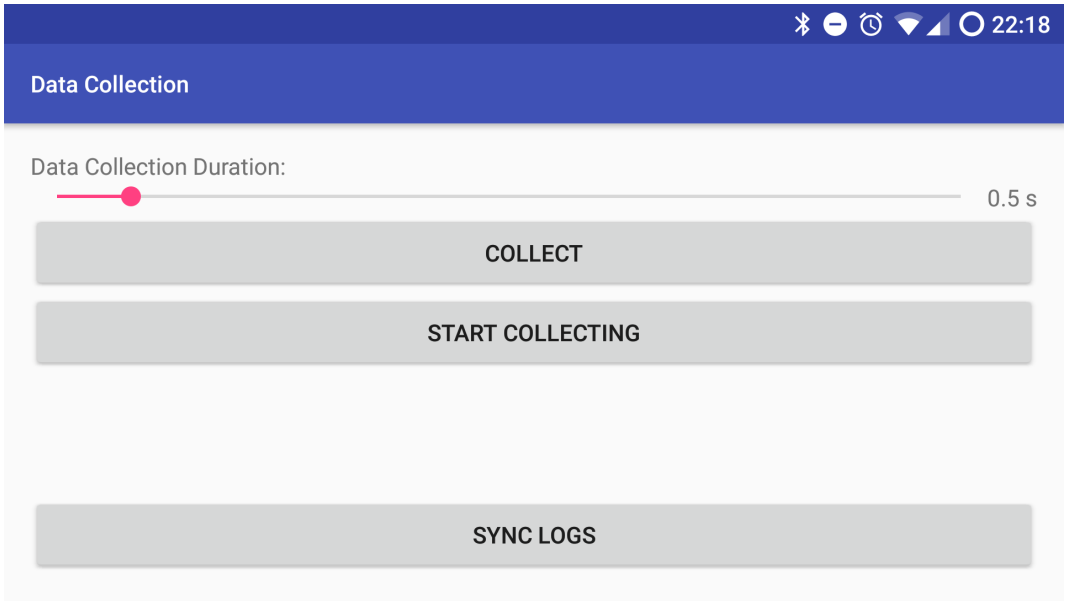
$$s_N^2 = \frac{(N-2)s_{N-1}^2 + (x_N - \bar{x}_N)(x_N - \bar{x}_{N-1})}{N-1}$$

few usages will have no protection against any form of relay attack. This can be handled by requiring interactive authentication during initial training of the model, or combining with another distance bounding technique which works without training such as timing, or accepting reduced accuracy (and security) during the first few access attempts.

3.2 Data Collection

Data is collected using a mobile application (shown in Figure 3.4) that outputs values to a log file over a short period (less than one second) representing the data that could be recorded in the short period of time available during an access control authentication event. Bluetooth, WiFi, and Cell Received Signal Strength Indicator (RSSI) readings are recorded using ad hoc classes which specially handle their respective wireless signal, as these signals must be accessed using specific API classes. All other sensors are handled in a generic manner by enumerating all available *Sensor* implementations using the *SensorManager* API. These sensors produce an arbitrary number of values, unlike the single RSSI value being considered for wireless signals. All values are stored in the log file in a CSV format which can be processed later. After data collection, the log files for a specific location are processed into a WEKA Attribute-Relation File Format (ARFF) data file. Sensors are translated directly to attributes

Figure 3.4: Data Collection Application



in the dataset. For sensors that output a single value (such as wireless signal RSSI values), that sensor is translated to a single attribute. For sensors that output multiple values—such as the accelerometer which outputs a vector of acceleration in X, Y, Z directions—each scalar value is translated to a single attribute, resulting in multiple attributes for that sensor. Each log file is converted to a single instance, with values for every attribute (sensor reading) which was recorded in the log file. Log files do not contain information about the location they were recorded in, instead logs from different locations are stored in different folders, so the processing script translates the parent folder of each log to the class for that instance. Missing values are left empty in the dataset, to be handled separately handled in the classifier. Some log files may contain multiple values for the same attribute, for sensors which report a continuous stream of data, and produce new readings frequently enough to output more than once in a single recording period. If multiple values exist for the same attribute, they are combined to a single value by recording their

Figure 3.5: RSSI Trend Graph

mean.

For radio signals, the app also supports some simple visualisation of the RSSI values for a particular signal, in the form of a graph (shown in Figure 3.5) that displays the RSSI over time and shows some calculations about the trend of the data. The calculations provided in the visualisation demonstrate how signal-strength based distance bounding can work, and show how unreliable it is, especially with a low-power signal such as Bluetooth.

4 Analysis

The classifier supports some configuration settings which can be changed to modify the behaviour of the classifier, resulting in different prediction performance.

The *threshold* setting controls the final decision for whether an instance is considered valid or not. This threshold value is the number of standard deviations from the mean distance to accept for instances in the cluster. A threshold of 1 would mean to accept values with a distance of at most one standard deviation from the mean distance for samples in the cluster. Changing the threshold value affects the ratio of false-positive to false-negative. A low threshold value minimises the false-positive rate, which is ideal for use in access control where it is better to deny entry (prompt the user to confirm the entry) than to allow an attacker entry. Higher thresholds increase

The classifier was tested on each predictor using 10-fold cross validation for different values of *threshold*. Figure B.3 shows a typical command for running the classifier on a dataset. The range of threshold values was from 0.5 to 2.2, in increments of 0.1.

Figure 4.1: Testing the classifier with 10-fold cross validation for threshold=0.5

```
java weka.classifiers.rules.ClusterDistanceClassifier \
    -t wifi-merged.arff -x 10 -C 0 -D 0.5
```

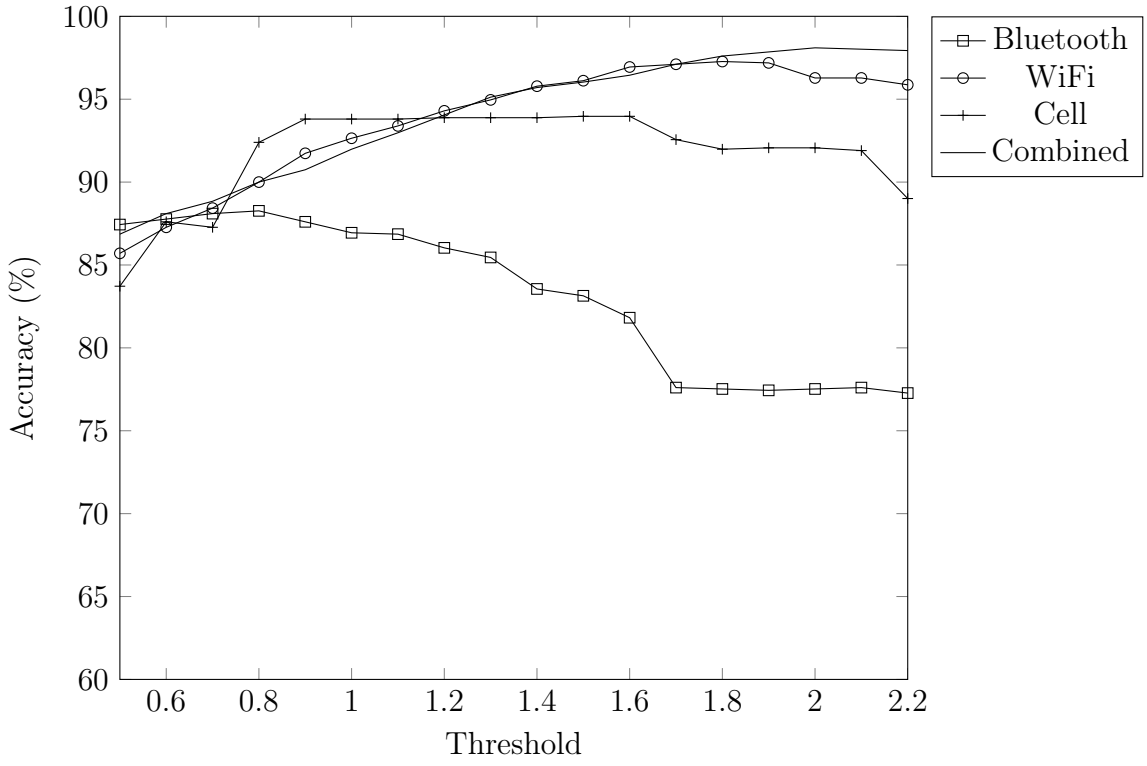
4.1 Wireless Signals

Models were tested with different thresholds for short-range and long-range data sets. The short-range data set was gathered at three locations in an office building, close enough for some Bluetooth signals to be shared between locations. The long-range data set was gathered approximately 10km across a city, far enough for WiFi and Bluetooth signals to be completely different, and for different cell towers to be used (even if the same cell towers are technically in range).

Figure 4.2 shows the classification accuracy achieved with cross validation for each of the tested thresholds for the short-range data set, for each gathered wireless signal type, plus one classifier that combines all the wireless signal types together. The results show that WiFi provides the best prediction accuracy for a single variable, followed by Cell and then Bluetooth. The combination of all three wireless signals closely matches the performance of WiFi alone, but does achieve a slightly higher prediction accuracy at high threshold values, resulting in the best overall prediction accuracy. All wireless signals achieved close to zero false-positive rate at low (0.5) threshold, increasing to 5% at high (2.2) threshold. The combination of all wireless signals achieved zero false-positive rate at low (0.5) threshold, increasing to 1% at high (2.2) threshold.

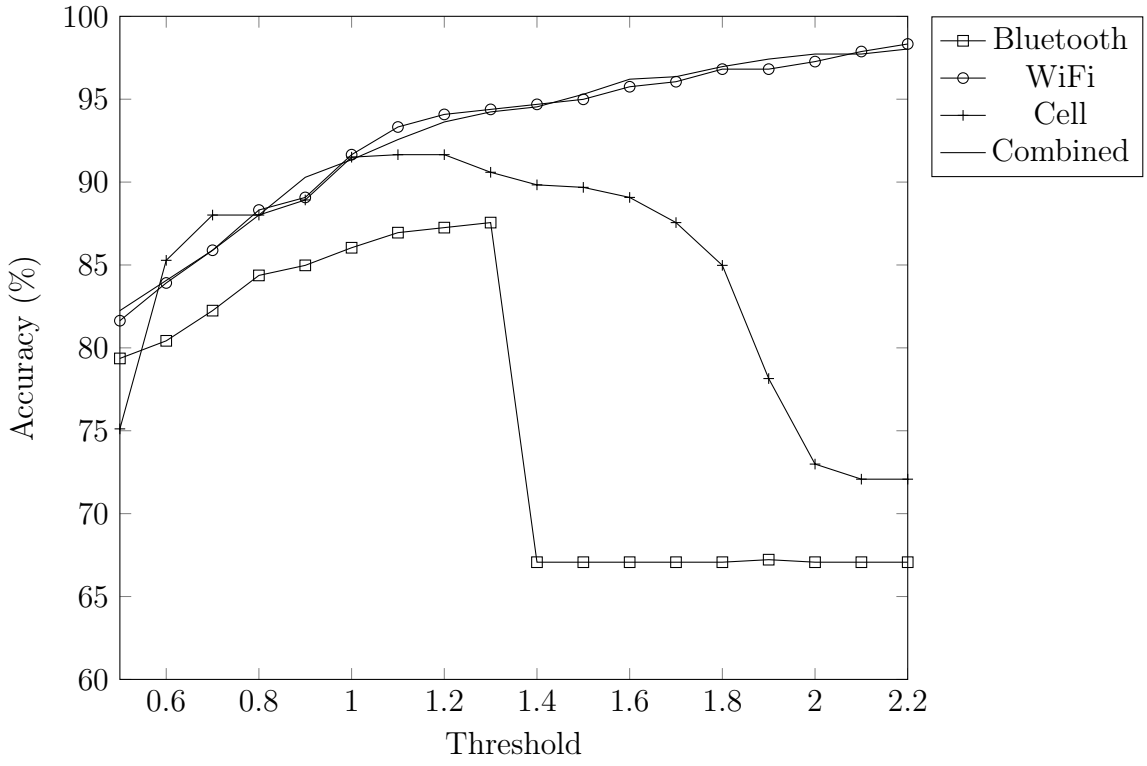
The long-range data set in figure 4.3 shows similar accuracy to the short-range data set, with slightly improved overall performance, and reduced performance for the Cell predictor at high threshold values. This is interesting, as the Bluetooth and WiFi predictors should trivially have perfect classification accuracy at such a long range based on the mere presence or lack of a signal. When a signal is missing from an instance, it is given a value of -100,

Figure 4.2: Wireless Signals over short range at different threshold values



as if it was present with a very low RSSI. This is done to handle legitimate readings where a signal is missing, which happens frequently when on the edge of a signal's viable range. From inspection of the input data sets, it appears that indeed a significant proportion of the available signals are not present in every valid reading, which results in the classifier learning that it is okay for signals to be missing. As such, changing the default value to something other than -100 does not result in improved classification accuracy, as a more distinct "missing" value would also be used when training the model, thereby biasing the model to still accept missing values for that signal. To handle this properly, the classifier needs to reduce the impact of unreliable predictors (predictors which accept missing values), without reducing the accuracy of the reliable predictors (predictors which do not accept missing values).

Figure 4.3: Wireless Signals over long range at different threshold values



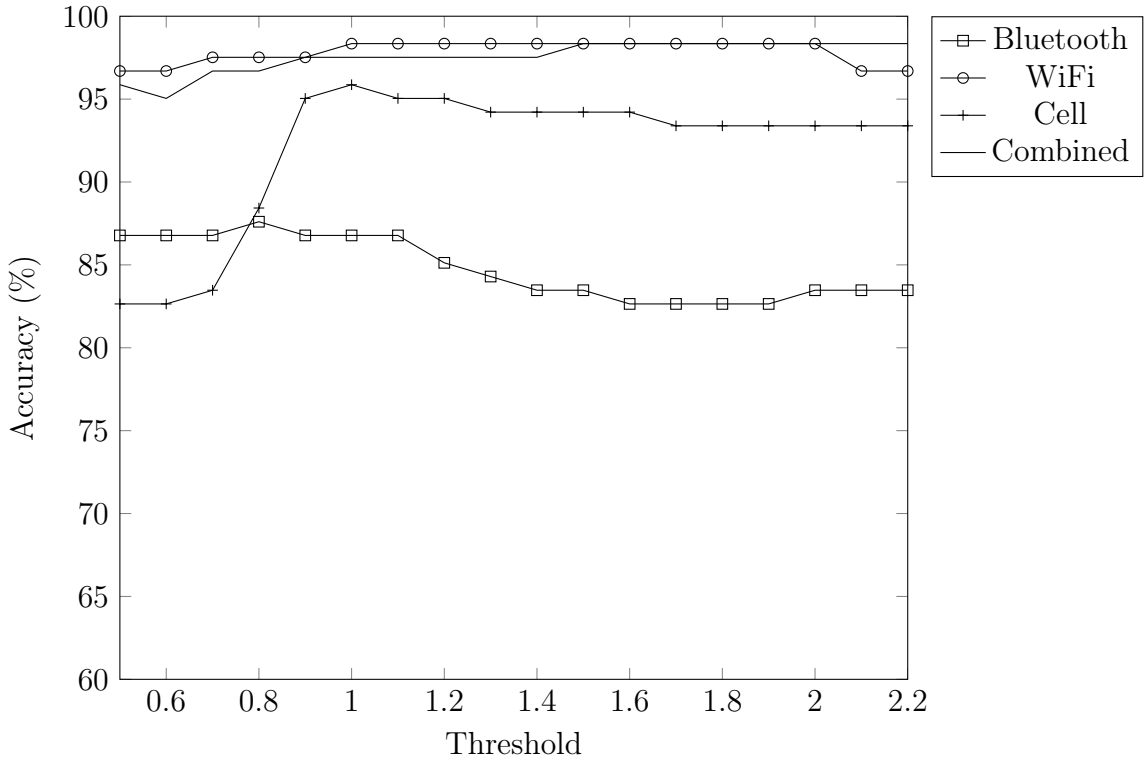
The input data set consisted of 1200 instances, which would represent a long period of time for a single user's access history events. In order to evaluate the classifier on smaller datasets, subsets were generated using Figure 4.4 with 10% and 1% of the instances, shown in figures 4.5 and 4.6. As expected there is a decrease in classification accuracy for small data set sizes, however the classifier does appear to behave consistently across data set sizes, and acceptably on small data sets which will necessarily be encountered when initially building a model.

Figure 4.4: Generate a random 10% subset of a dataset

```
java weka.filters.unsupervised.instance.Resample -S 1 -Z 10 \
    -i wifi-merged.arff -o wifi-merged-10.arff
```

Interestingly, Bluetooth consistently performs the worst of the wireless signals, despite having the shortest range (which should improve its accuracy at predicting proximity to a location). There are multiple reasons for this: ephemerality; and reliability. Because of Bluetooth's use in portable devices, signals are likely to appear or move unpredictably, meaning a signal may only briefly be a useful predictor. Bluetooth's short range also comes with the most unreliable signal, which means signal strengths may differ significantly between readings, even without moving the sender or receiver devices.

Figure 4.5: Classifier performance with reduced dataset (10% / 121 instances)



The different wireless signals appear to perform best at different threshold values. Bluetooth performs better at low thresholds, WiFi performs better at higher thresholds but doesn't change as much as Bluetooth, and Cell signals perform approximately the same across a wide range of thresholds. This cor-

relates with the properties of each signal: Bluetooth is a short-range signal which fluctuates significantly in strength, meaning the standard deviation of signal strength is already quite large. The threshold is the number of standard deviations from the mean allowed, so increasing the threshold for Bluetooth further increases the range of accepted values, including instances which should be rejected. Missing signals are treated as zero signal strength, but at a high threshold zero may be within the threshold range. WiFi is a medium-range signal with more reliable signal strength, which means the standard deviation is lower and there is a more distinct difference between accepted and rejected instances. This results in less extreme response to the threshold, and enables high threshold values to work better. Cell signals are a more extreme case of the same principles applied to WiFi—a long-range signal with significantly less fluctuation in signal strength (barring interference), results in low standard deviation. In this case the fluctuation in signal strength is low enough that most values are classified correctly over a wide range of threshold values.

4.2 Other Sensors

Each of the sensors provided by the test device (listing A.1) were tested across the range of threshold values. Most of the sensors were highly inaccurate, typically having accuracy of less than 50%. See tables A.5–A.7. Figure 4.7 shows the accuracy of the most accurate sensors (choosing the best sensor where multiple sensors represent the same base property), and some combinations.

Some sensors do appear to provide good classification accuracy. The most significant of these are the Acceleration and Magnetic sensors. The magnetic sensor could potentially be picking up a difference in magnetic field at the different locations due to nearby metal and electronics, however the accuracy of

Figure 4.6: Classifier performance with reduced dataset (1% / 12 instances)

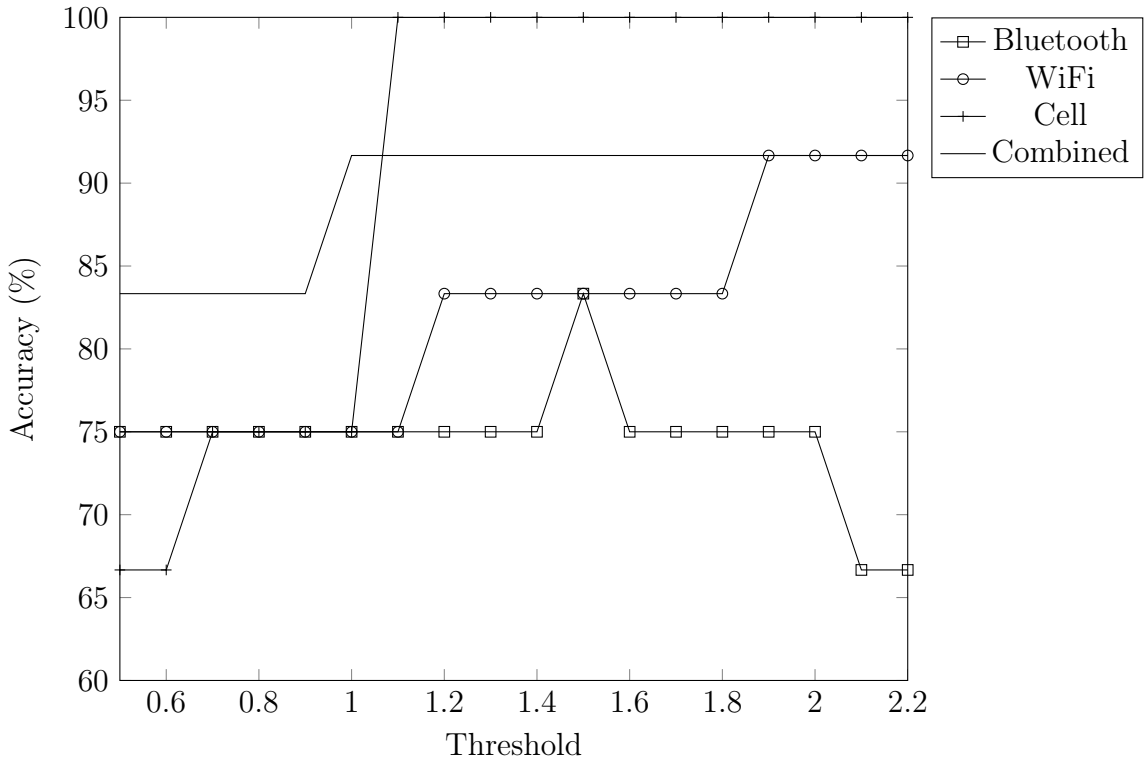
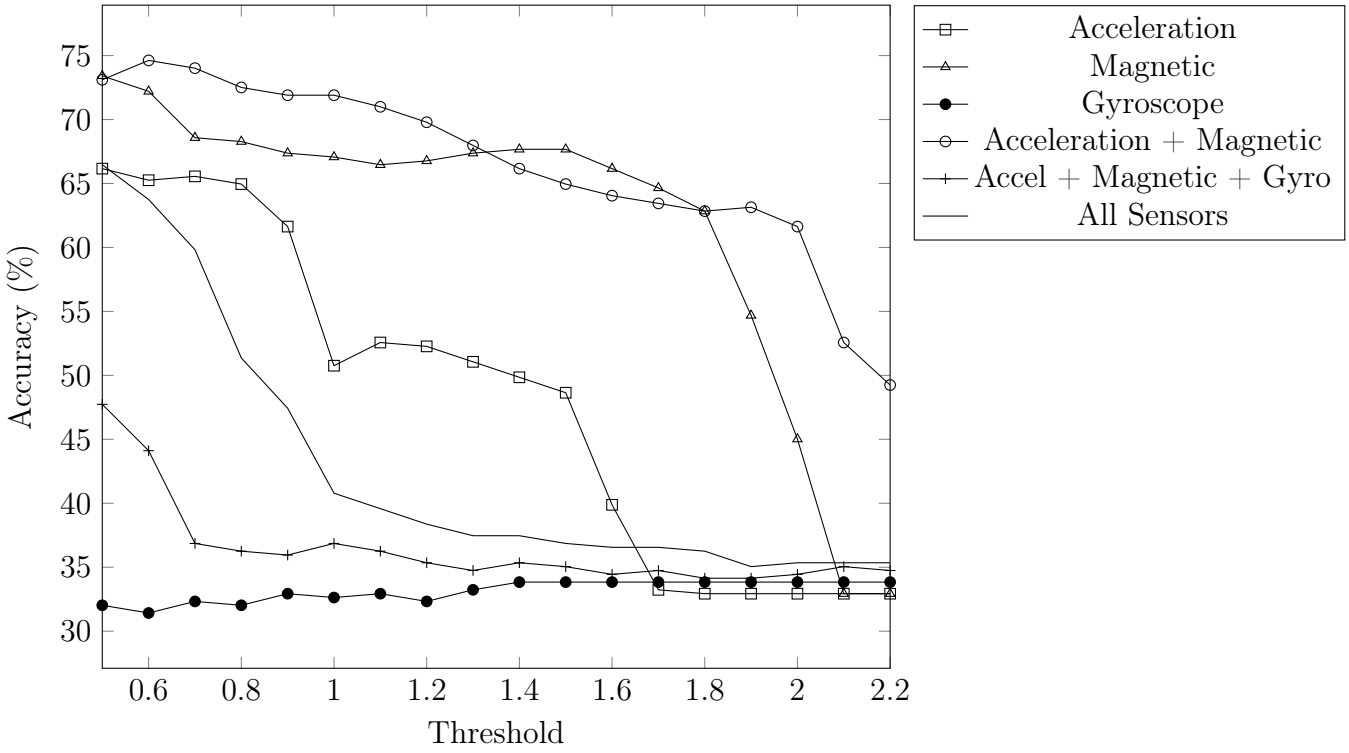


Figure 4.7: Sensors at different threshold values



the acceleration sensor at distinguishing locations seems dubious. Most likely it is distinguishing differences in user behaviour, or possibly a difference in the angle of the floor at that location. Regardless, these sensors appear to provide a classification accuracy significantly better than random guessing, and so could be combined with other predictors to provide a more accurate combined prediction. Combining the Acceleration and Magnetic sensors produces a slightly better classifier that drops off far less at higher thresholds.

Unlike the wireless signals, these sensors appear to drop off significantly in accuracy at higher threshold values. This makes sense, as the threshold is the number of standard deviations from the average euclidean distance to accept—how far from the typical reading before an instance is rejected. Increasing the threshold increases the range of accepted values, which for environmental

sensors can be a small range to begin with. Increasing the threshold too far can make it impossible to distinguish between valid and invalid authentication attempts. Conversely with wireless signals, increasing the threshold helps filter out fluctuation/noise in the wireless signal strength. This difference in requirement for threshold value means that when combining predictors, different threshold values may be necessary for different predictors in order to maximise classification accuracy.

The combination of all the sensors approaches the accuracy of the best single sensors for low threshold values, but quickly drops off to below 40% at higher thresholds. This is due to the classification algorithm lacking an attribute selection mechanism. Attributes are normalised to a mean and standard deviation, which evens out differences in the magnitude of different attributes. However, if an attribute provides no useful information, it is still used with the same priority as any other attribute. As the classifier classifies instances based on their euclidean distance from the centre of the known "good" instances, adding useless attributes reduces this distance, reducing the accuracy of the classifier. To avoid this, attribute selection can be performed to only include attributes which provide useful information. Attribute selection is problematic with the constraints mentioned in section 3.1. A common method of attribute selection is the Wrapper method—score sets of attributes by training and evaluating a model with that set of attributes. This method is not practical in this situation, as it requires significant processing time, and needs examples of unsuccessful access in order to score each model. Perhaps the application could collect "unsuccessful access" data in the background when no access has been attempted, however this will drain the battery and raises user privacy concerns. One attribute selection behaviour that can be automatically performed is detecting redundancy between multiple attributes. This can help avoid skewing

the model with multiple redundant attributes, but does not solve the problem with useless attributes. The best selection that can be achieved for useless attributes is to remove attributes which appear to have random values. This still may do more harm than good, as a "random" attribute at the access control point could be in a different range than when away from the access control point, or for wireless signals it could only exist at the access control point. Both of these types of attributes could be useful in classification, but could also be detrimental and are not easily distinguished in practice.

Interestingly, there is a significant difference in classification accuracy between different sensors that read the same property—in some cases, between different processed outputs from the same physical sensor. Figure 4.8 shows the differences between four orientation sensors present on the test device, and figure 4.9 shows the difference between the calibrated and uncalibrated magnetic field readings. The difference between processed outputs of a sensor is likely due to smoothing resulting in a loss of information, which explains why the uncalibrated magnetic sensor is slightly more accurate than the calibrated form. However it also appears than in some cases the processed form of the sensor is more accurate than the raw sensor—for example at a threshold of 1, the software-based orientation sensor [4] is more accurate than any other orientation sensor.

Figure 4.8: Orientation Sensors at different threshold values

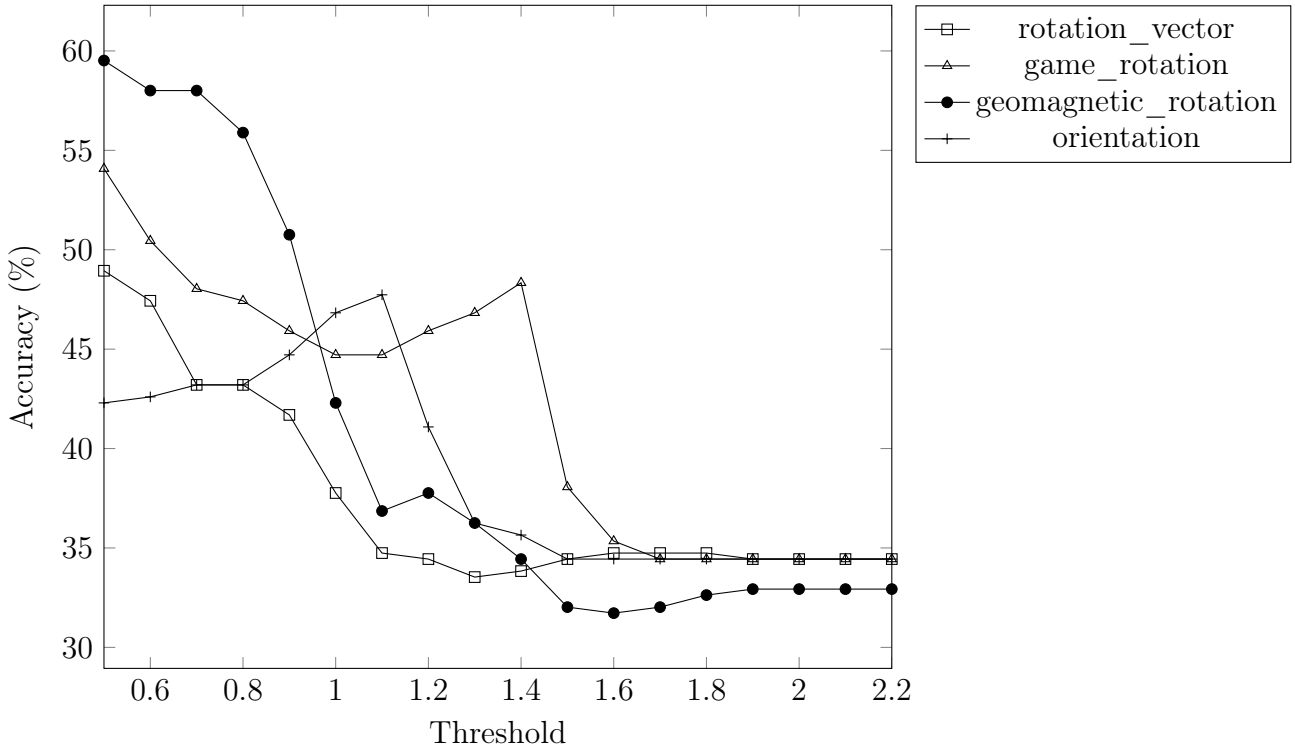
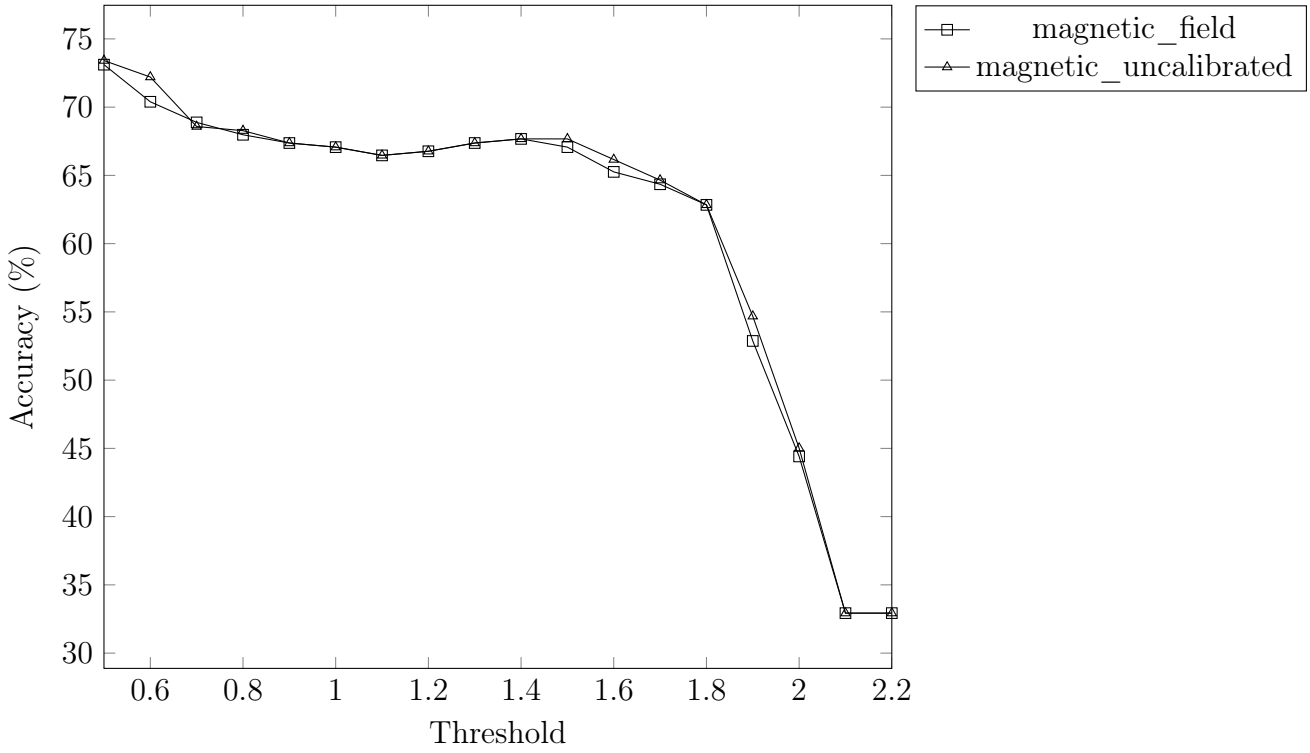


Figure 4.9: Magnetic Sensors at different threshold values



5 Recommendations

Predictor Selection

There are a large collection of sensors available on a typical Android device. Listing A.1 shows the full list from the test device. Many of these sensors represent the same underlying property, or are not useful as predictors of proximity to an access control device. Due to technical limitations that prevent accurate attribute selection on the mobile device, these predictors should be selected ahead of time using offline analysis of sample data. Different devices may have different sensors available, however standard sensors are always exposed under the same sensor identifier [4] if they are available, which makes it possible to use a pre-defined list of sensors. Based on analysis of collected data, the acceleration and magnetic field sensors appear to be the most useful sensors for use as predictors.

Combine Predictors

Evaluation of the classifier on sensor and wireless signal data available on the test device found that wireless signal strength and some sensors (acceleration and magnetic field) proved to be strong indicators of proximity to the access control device. Combining these together with a typical timing-based distance bounding algorithm could be used to achieve highly accurate relay attack detection, and resilience to an attacker attempting to specifically subvert the predictors. For example, an attacker could spoof wireless signals, but with sensors and timing also used in the classifier, the relay attack can still be

detected.

Predictor Thresholds

Different predictors perform better at different threshold values. This is easily demonstrated by comparing the wireless signal predictors (which when combined perform best at a threshold of approximately 2.0), with the sensor predictors (which when combined perform best at a threshold of approximately 0.6). When combining these predictors, they may need to use different thresholds to achieve maximum classification performance. The wireless predictors individually perform better at lower thresholds (0.8 for Bluetooth; 1.6 for Cell; and 1.8 for WiFi), so the combined wireless predictor may benefit further from using their best individual thresholds when combining. The algorithm could be modified to normalise attributes according to their threshold in order to handle different thresholds for each attribute, or multiple individual classifiers could be used and combined to produce a single final prediction.

It is also necessary to take into account the false-positive rate when choosing a threshold value. Lower thresholds result in lower false-positive rates, at the cost of potentially lower accuracy.

Avoid Long-Range Silent Authentication

The purpose of this project was to detect relay attacks against a wireless communications protocol (Bluetooth) with properties that make conventional distance bounding unreliable or impossible. This problem can be minimised by not performing silent (non-interactive) authentication over Bluetooth. Instead a wireless protocol with less range and more reliable timing (such as NFC) should be used, to minimise the possibility of a relay attack, and maximise the effectiveness of distance bounding. Authentication can still be performed over Bluetooth for use cases where it makes sense (longer distance), however

requiring the user to interact with their device (respond to a prompt) minimises the risk of a relay attack being successful.

Conclusion

When used individually, the predictors investigated in this project do not provide acceptable accuracy in detecting relay attacks. When combined however, they can achieve a very high classification accuracy with resilience against an attacker attempting to break individual predictors. Security can also be improved by using an appropriate wireless protocol that is less susceptible to relay attacks, where possible.

6 Conclusions

Traditional timing-based distance bounding is not reliable on mobile Bluetooth connections, due to significant delays which cannot be measured by the mobile application. This thesis investigated alternative methods for using sensors and observable signals to fingerprint a location, thereby detecting and preventing relay attacks when the application is used in an unexpected location.

The prototype created in this research used a custom classification algorithm designed to work with the specific requirements of the input data and the target mobile devices. Using this classification algorithm, the prototype was able to detect relay attacks at an accuracy of over 95% for several radio signals, and over 70% for several other sensor types, with the best performing single predictor being observed WiFi signals. This performance is comparable to the accuracy of timing-based distance bounding in internal tests at Gallagher, which varied significantly between devices. Combining multiple sensors was able to improve accuracy compared to using a single sensor, as well as theoretically providing resilience against an attacker spoofing the sensors.

6.1 Future Work

The scope of this research focused on investigating an alternative to traditional timing-based distance bounding for a wireless communications protocol where timing is unreliable. The basis of this came from the fact that attacks have been

demonstrated to be possible against weak timing-based distance bounding. If considering using any of the predictors investigated in this research, it would be worthwhile to investigate how an adversary would attack these predictors, and assessing the risk relative to a timing-based approach.

In theory combining multiple predictors makes it harder for an attacker to trick the system by fooling the predictors. An interesting area of research would be to quantify the accuracy which can be achieved by a classifier that combines multiple predictors, and how resilient such a system really is in practice, if one or more predictors are fooled and produce incorrect classifications.

This research identified the requirements for classification in an access control situation (i.e. time constraints, incremental update), and identified some predictors which should work under those requirements. Future work could look into the performance of predictors under those requirements in order to judge the practicality of implementing such a system in reality. In particular, some predictors may only be able to produce cached values within the time constraints, which may impact the performance of the predictor in a real-world use case.

References

[1] ISO/IEC JTC 1/SC 17. ISO/IEC 7816-15:2016 identification cards – integrated circuit cards – part 15: Cryptographic information application. 2016.

[2] ISO/IEC JTC 1/SC 31. ISO/IEC 18000-3:2010 information technology – radio frequency identification for item management – part 3: Parameters for air interface communications at 13,56 mhz. 2010.

[3] Ammar Alkassar and Christian Stuble. Towards secure iff: preventing mafia fraud attacks. In *MILCOM 2002. Proceedings*, volume 2, pages 1139–1144. IEEE, 2002.

[4] Android. Sensors overview. `https://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-intro`, accessed 2017-12-06.

[5] Android. Android nfc documentation. `https://developer.android.com/reference/android/nfc/NfcAdapter.html#ACTION_NDEF_DISCOVERED`, accessed 2018-01-02.

[6] Apple. Core nfc. `https://developer.apple.com/documentation/corenfc`, accessed 2018-01-02.

[7] Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardaş, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing rfid dis-

tance bounding protocols. *Journal of Computer Security*, 19(2):289–317, 2011.

[8] Yael Ben-Haim and Elad Tom-Tov. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research*, 11(Feb):849–872, 2010.

[9] Thomas Beth and Yvo Desmedt. Identification tokens—or: Solving the chess grandmaster problem. In *Conference on the Theory and Application of Cryptography*, pages 169–176. Springer, 1990.

[10] Neeraj Bhargava, Girja Sharma, Ritu Bhargava, and Manish Mathuria. Decision tree analysis on j48 algorithm for data mining. *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6), 2013.

[11] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.

[12] SIG Bluetooth. Bluetooth specification version 4.2. *Bluetooth SIG*, 2014.

[13] Stefan Brands and David Chaum. Distance-bounding protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 344–359. Springer, 1993.

[14] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[15] Jolyon Clulow, Gerhard P Hancke, Markus G Kuhn, and Tyler Moore. So near and yet so far: Distance-bounding attacks in wireless networks. In *European Workshop on Security in Ad-hoc and Sensor Networks*, pages 83–97. Springer, 2006.

[16] JCDM COE. Performance comparison of naïve bayes and j48 classification algorithms. *International Journal of Applied Engineering Research*, 7(11):2012, 2012.

[17] John Horton Conway. *On numbers and games*, volume 6. IMA, 1976.

[18] Saar Drimer, Steven J Murdoch, et al. Keep your enemies close: Distance bounding against smartcard relay attacks. In *USENIX Security*, volume 2007, 2007.

[19] Lishoy Francis, Gerhard Hancke, and Keith Mayes. A practical generic relay attack on contactless transactions by using nfc mobile phones. *International Journal of RFID Security and Cryprography (IJRFIDSC)*, 2(1-4):92–106, 2013.

[20] E Frank, MA Hall, and IH Witten. The weka workbench. *Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", 4th edn. Morgan Kaufman, Burlington*, 2016.

[21] Gallagher. Gallagher mobile connect. `https://play.google.com/store/apps/details?id=com.gallagher.security.commandcentrecardholderapp`, accessed 2017-12-23.

[22] Gallagher. T-series readers. `https://security.gallagher.com/assets/2523/T-Series_Reader_Datasheet.pdf`, accessed 2017-12-23.

[23] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[24] Iakovos Gurulian, Carlton Shepherd, Konstantinos Markantonakis, Raja Naeem Akram, and Keith Mayes. When theory and reality collide:

Demystifying the effectiveness of ambient sensing for nfc-based proximity detection by applying relay attack data. *arXiv preprint arXiv:1605.00425*, 2016.

[25] Keijo Haataja and Pekka Toivanen. Practical man-in-the-middle attacks against bluetooth secure simple pairing. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM'08. 4th International Conference on*, pages 1–5. IEEE, 2008.

[26] Gerhard P Hancke. A practical relay attack on iso 14443 proximity cards. *Technical report, University of Cambridge Computer Laboratory*, 59:382–385, 2005.

[27] Gerhard P Hancke and Markus G Kuhn. An rfid distance bounding protocol. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*, pages 67–73. IEEE, 2005.

[28] Kathryn Hempstalk, Eibe Frank, and Ian H Witten. One-class classification by combining density and class probability estimation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 505–519. Springer, 2008.

[29] Kathryn Hempstalk, Eibe Frank, and Ian H. Witten. One-class classification by combining density and class probability estimation. In *Proceedings of the 12th European Conference on Principles and Practice of Knowledge Discovery in Databases and 19th European Conference on Machine Learning, ECMLPKDD2008*, volume Vol. 5211 of *Lecture Notes in Computer Science*, pages 505–519, Berlin, September 2008. Springer.

[30] HID. Understanding card data formats. https://www.hidglobal.com/sites/default/files/hid-understanding_card_data_formats-wp-en.pdf, 2006.

[31] HID. HID Mobile Access® iCLASS SE® & multiCLASS SE® Mobile Enabled Readers. https://www.hidglobal.com/sites/default/files/resource_files/hid-mobile-access-readers-nfc-ds-en.pdf, accessed 2017-12-23.

[32] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 461–472. ACM, 2016.

[33] Chong Hee Kim and Gildas Avoine. Rfid distance bounding protocol with mixed challenges to prevent relay attacks. In *International Conference on Cryptology And Network Security*, pages 119–133. Springer, 2009.

[34] Albert Levi, Erhan Çetintaş, Murat Aydos, Cetin Kaya Koç, and M Ufuk Çağlayan. Relay attacks on bluetooth authentication and solutions. In *International Symposium on Computer and Information Sciences*, pages 278–288. Springer, 2004.

[35] TC Minter. Single-class classification. 1975.

[36] Jorge Munilla and Alberto Peinado. Distance bounding protocols for rfid enhanced by using void-challenges and analysis in noisy channels. *Wireless communications and mobile computing*, 8(9):1227–1232, 2008.

[37] Tina R Patil and SS Sherekar. Performance analysis of naive bayes and j48 classification algorithm for data classification. *International Journal of Computer Science and Applications*, 6(2):256–261, 2013.

[38] Aanjhan Ranganathan, Nils Ole Tippenhauer, Boris Škorić, Dave Singelée, and Srdjan Čapkun. Design and implementation of a terrorist fraud resilient distance bounding system. In *European Symposium on Research in Computer Security*, pages 415–432. Springer, 2012.

[39] Anthony Rose and Ben Ramsey. Picking bluetooth low energy locks from a quarter mile away. 24. DEF CON, 2016.

[40] Babins Shrestha, Nitesh Saxena, Hien Thi Thu Truong, and N Asokan. Drone to the rescue: Relay-resilient authentication using ambient multi-sensing. In *International Conference on Financial Cryptography and Data Security*, pages 349–364. Springer, 2014.

[41] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15. 4. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, pages 232–237. IEEE, 2012.

[42] David Martinus Johannes Tax. One-class classification. 2001.

[43] Hien Thi Thu Truong, Xiang Gao, Babins Shrestha, Nitesh Saxena, N Asokan, and Petteri Nurmi. Comparing and fusing different sensor modalities for relay attack resistance in zero-interaction authentication. In *Pervasive Computing and Communications (PerCom), 2014 IEEE International Conference on*, pages 163–171. IEEE, 2014.

[44] Hwanjo Yu. Single-class classification with mapping convergence. *Machine Learning*, 61(1):49–69, 2005.

Appendix A Data Listings

Listing A.1: Full list of supported sensors

```
1  android.sensor.accelerometer
2  android.sensor.game_rotation_vector
3  android.sensor.geomagnetic_rotation_vector
4  android.sensor.gravity
5  android.sensor.gyroscope
6  android.sensor.gyroscope_uncalibrated
7  android.sensor.light
8  android.sensor.linear_acceleration
9  android.sensor.magnetic_field
10 android.sensor.magnetic_field_uncalibrated
11 android.sensor.orientation
12 android.sensor.proximity
13 android.sensor.rotation_vector
14 android.sensor.step_counter
15 com.qti.sensor.amd
16 com.qti.sensor.motion_accel
```

Table A.1: Evaluation of clustering classifier accuracy on wireless signals over short range (all instances)

threshold	Bluetooth	WiFi	Cell	Combined
0.5	87.438	85.7025	83.719	86.8595
0.6	**87.7686**	87.2727	87.6033	88.0992
0.7	88.0992	88.4298	87.2727	88.843
0.8	88.2645	90	92.3967	90
0.9	87.6033	91.7355	93.8017	90.7438
1.0	86.9421	92.6446	93.8017	91.9835
1.1	86.8595	93.3884	93.8017	92.9752
1.2	86.0331	94.2975	**93.8843**	94.0496
1.3	85.4545	94.9587	93.8843	95.124
1.4	83.5537	95.7851	93.8843	95.7025
1.5	83.1405	96.1157	93.9669	96.0331
1.6	81.8182	96.9421	93.9669	96.4463
1.7	77.6033	97.1074	92.562	97.1074
1.8	77.5207	**97.2727**	91.9835	97.6033
1.9	77.438	97.1901	92.0661	97.8512
2.0	77.5207	96.281	92.0661	98.0992
2.1	77.6033	96.281	91.9008	**98.0165**
2.2	77.2727	95.8678	89.0083	97.9339

Table A.2: Evaluation of clustering classifier accuracy on wireless signals over short range (10% / 121 instances)

threshold	Bluetooth	WiFi	Cell	Combined
0.5	86.7769	96.6942	82.6446	95.8678
0.6	86.7769	96.6942	82.6446	95.0413
0.7	86.7769	97.5207	83.4711	96.6942
0.8	**87.6033**	97.5207	88.4298	96.6942
0.9	86.7769	97.5207	95.0413	97.5207
1.0	86.7769	**98.3471**	**95.8678**	97.5207
1.1	86.7769	98.3471	95.0413	97.5207
1.2	85.124	98.3471	95.0413	97.5207
1.3	84.2975	98.3471	94.2149	97.5207
1.4	83.4711	98.3471	94.2149	97.5207
1.5	83.4711	98.3471	94.2149	**98.3471**
1.6	82.6446	98.3471	94.2149	98.3471
1.7	82.6446	98.3471	93.3884	98.3471
1.8	82.6446	98.3471	93.3884	98.3471
1.9	82.6446	98.3471	93.3884	98.3471
2.0	83.4711	98.3471	93.3884	98.3471
2.1	83.4711	96.6942	93.3884	98.3471
2.2	83.4711	96.6942	93.3884	98.3471

Table A.3: Evaluation of clustering classifier accuracy on wireless signals over short range (1% / 12 instances)

threshold	Bluetooth	WiFi	Cell	Combined
0.5	75	75	66.6667	83.3333
0.6	75	75	66.6667	83.3333
0.7	75	75	75	83.3333
0.8	75	75	75	83.3333
0.9	75	75	75	83.3333
1.0	75	75	75	**91.6667**
1.1	75	75	**100**	91.6667
1.2	75	83.3333	100	91.6667
1.3	75	83.3333	100	91.6667
1.4	75	83.3333	100	91.6667
1.5	**83.3333**	83.3333	100	91.6667
1.6	75	83.3333	100	91.6667
1.7	75	83.3333	100	91.6667
1.8	75	83.3333	100	91.6667
1.9	75	**91.6667**	100	91.6667
2.0	75	91.6667	100	91.6667
2.1	66.6667	91.6667	100	91.6667
2.2	66.6667	91.6667	100	91.6667

Table A.4: Evaluation of clustering classifier accuracy on wireless signals over long range

threshold	Bluetooth	WiFi	Cell	Combined
0.5	79.3627	81.6388	75.1138	82.2458
0.6	80.4249	83.915	85.2807	84.0668
0.7	82.2458	85.8877	88.0121	85.8877
0.8	84.3703	88.3156	88.0121	88.0121
0.9	84.9772	89.0744	88.9226	90.2883
1.0	86.0395	91.654	91.5023	91.3505
1.1	86.9499	93.3232	91.654	92.5645
1.2	87.2534	94.0819	91.654	93.6267
1.3	87.5569	94.3854	90.5918	94.2337
1.4	67.0713	94.6889	89.8331	94.5372
1.5	67.0713	94.9924	89.6813	95.2959
1.6	67.0713	95.7511	89.0744	96.2064
1.7	67.0713	96.0546	87.5569	96.3581
1.8	67.0713	96.8134	84.9772	96.9651
1.9	67.2231	96.8134	78.1487	97.4203
2.0	67.0713	97.2686	72.9894	97.7238
2.1	67.0713	97.8756	72.0789	97.7238
2.2	67.0713	98.3308	72.0789	98.0273

Table A.5: Evaluation of clustering classifier accuracy on misc sensors

threshold	android.sensor.light	android.sensor.proximity	android.sensor.step_counter
0.5	65.5589	35.9517	67.0695
0.6	65.5589	35.9517	67.0695
0.7	65.5589	35.9517	67.0695
0.8	65.5589	35.9517	67.0695
0.9	65.5589	35.9517	67.0695
1.0	65.5589	35.9517	67.0695
1.1	65.5589	35.9517	67.0695
1.2	65.5589	35.9517	67.0695
1.3	65.5589	35.9517	67.0695
1.4	65.5589	35.9517	67.0695
1.5	65.5589	35.9517	67.0695
1.6	65.5589	35.9517	67.0695
1.7	65.5589	35.9517	67.0695
1.8	65.5589	35.9517	67.0695
1.9	65.5589	35.9517	67.0695
2.0	65.5589	35.9517	67.0695
2.1	65.5589	35.9517	67.0695
2.2	65.5589	35.9517	67.0695

Table A.6: Evaluation of clustering classifier accuracy on orientation sensors

threshold	android.sensor.game_rotation_vector	android.sensor.geomagnetic_rotation_vector	android.sensor.gyroscope	android.sensor.gyroscope_uncalibrated	android.sensor.magnetic_field	android.sensor.magnetic_field_uncalibrated	android.sensor.orientation	android.sensor.rotation_vector
0.5	54.0785	59.5166	32.0242	30.8157	73.1118	73.4139	42.2961	48.9426
0.6	50.4532	58.006	31.4199	31.4199	70.3927	72.2054	42.5982	47.432
0.7	48.0363	58.006	32.3263	32.3263	68.8822	68.5801	43.2024	43.2024
0.8	47.432	55.8912	32.0242	33.2326	67.9758	68.2779	43.2024	43.2024
0.9	45.9215	50.7553	32.9305	33.2326	67.3716	67.3716	44.713	41.6918
1.0	44.713	42.2961	32.6284	32.3263	67.0695	67.0695	46.8278	37.7644
1.1	44.713	36.858	32.9305	32.6284	66.4653	66.4653	47.7341	34.7432
1.2	45.9215	37.7644	32.3263	32.3263	66.7674	66.7674	41.0876	34.4411
1.3	46.8278	36.2538	33.2326	33.2326	67.3716	67.3716	36.2538	33.5347
1.4	48.3384	34.4411	33.8369	33.8369	67.6737	67.6737	35.6495	33.8369
1.5	38.0665	32.0242	33.8369	33.8369	67.0695	67.6737	34.4411	34.4411
1.6	35.3474	31.7221	33.8369	33.8369	65.2568	66.1631	34.4411	34.7432
1.7	34.4411	32.0242	33.8369	33.8369	64.3505	64.6526	34.4411	34.7432
1.8	34.4411	32.6284	33.8369	33.8369	62.8399	62.8399	34.4411	34.7432
1.9	34.4411	32.9305	33.8369	33.8369	52.8701	54.6828	34.4411	34.4411
2.0	34.4411	32.9305	33.8369	33.8369	44.4109	45.0151	34.4411	34.4411
2.1	34.4411	32.9305	33.8369	33.8369	32.9305	32.9305	34.4411	34.4411
2.2	34.4411	32.9305	33.8369	33.8369	32.9305	32.9305	34.4411	34.4411

Table A.7: Evaluation of clustering classifier accuracy on acceleration sensors

threshold	android.sensor.accelerometer	android.sensor.gravity	android.sensor.linear_acceleration	com.qti.sensor.amd	com.qti.sensor.motion_accel
0.5	66.1631	65.2568	38.6707	51.6616	65.861
0.6	65.2568	65.861	38.6707	51.6616	64.9547
0.7	65.5589	57.4018	36.858	51.6616	65.2568
0.8	64.9547	50.4532	36.2538	51.6616	62.5378
0.9	61.6314	51.3595	35.6495	51.6616	52.2659
1.0	50.7553	50.7553	35.6495	51.6616	49.8489
1.1	52.568	50.1511	33.8369	51.6616	51.3595
1.2	52.2659	47.1299	35.0453	51.6616	50.1511
1.3	51.0574	36.858	35.0453	51.6616	49.5468
1.4	49.8489	35.6495	35.0453	51.6616	47.7341
1.5	48.6405	34.4411	34.139	43.2024	39.2749
1.6	39.8792	34.4411	35.0453	36.2538	33.2326
1.7	33.2326	34.4411	34.7432	36.2538	32.9305
1.8	32.9305	34.4411	34.4411	32.9305	32.9305
1.9	32.9305	34.4411	34.4411	32.9305	32.9305
2.0	32.9305	34.4411	34.4411	32.9305	32.9305
2.1	32.9305	34.4411	34.4411	32.9305	32.9305
2.2	32.9305	34.4411	34.4411	32.9305	32.9305

Table A.8: Evaluation of clustering classifier accuracy on combined sensors

threshold	All	accelerometer + magnetic	accelerometer + magnetic + gyroscope
0.5	66.4653	73.1118	47.7341
0.6	63.7462	74.6224	44.1088
0.7	59.8187	74.0181	36.858
0.8	51.3595	72.5076	36.2538
0.9	47.432	71.9033	35.9517
1.0	40.7855	71.9033	36.858
1.1	39.577	70.997	36.2538
1.2	38.3686	69.7885	35.3474
1.3	37.4622	67.9758	34.7432
1.4	37.4622	66.1631	35.3474
1.5	36.858	64.9547	35.0453
1.6	36.5559	64.0483	34.4411
1.7	36.5559	63.4441	34.7432
1.8	36.2538	62.8399	34.139
1.9	35.0453	63.142	34.139
2.0	35.3474	61.6314	34.4411
2.1	35.3474	52.568	35.0453
2.2	35.3474	49.2447	34.7432

Appendix B Code Listings

Listing B.1: Processing RSSI input CSVs into ARFF dataset

```
1  #!/usr/bin/env ruby
2
3  CLASSES = %w(
4  CellDataCollector
5  WifiDataCollector
6  BluetoothDataCollector
7  )
8
9  class Reading
10   attr_reader :source
11   attr_accessor :value
12   def initialize(source, value)
13     @source = source
14     @value = value
15   end
16 end
17
18 class Sample
19   attr_reader :readings, :class_value
20   def initialize(class_value)
```

```
21      @class_value = class_value
22      @readings = {}
23    end
24  end
25
26  attrs = {}
27  samples = []
28
29  ARGV.each do |class_name|
30    Dir.entries(class_name).each do |file_name|
31      file_path = "#{class_name}/#{file_name}"
32      next unless File.file? file_path
33      sample = Sample.new(class_name)
34      File.foreach(file_path) do |line|
35        cols = line.split ','
36        next unless CLASSES.include? cols[0]
37        reading = Reading.new(cols[2], cols[-1].to_i)
38        attrs[reading.source] = 1
39        existing_reading = sample.readings[reading.source]
40        if existing_reading.nil?
41          sample.readings[reading.source] = reading
42        else
43          existing_reading.value = (existing_reading.value
              ↪  + reading.value) / 2
44        end
45      end
46      samples << sample
```

```
47    end
48  end
49
50  puts "@relation 'radio-signals'"
51
52  ordered_attrs = attrs.keys
53  ordered_attrs.each do |key|
54    puts "@attribute '#{key}' numeric"
55  end
56
57  class_names = ARGV.map { |c| "'#{c}'" }
58  puts "@attribute class {#{class_names.join ','}}"
59
60  puts "@data"
61  samples.each do |sample|
62    values = ordered_attrs.map { |attr| sample.readings[
        ↪ attr]&.value or '?' }
63    puts "#{values.join ','},#{sample.class_value}"
64  end
```

Listing B.2: Processing sensor input CSVs into ARFF dataset

```
1  #!/usr/bin/env ruby
2
3  CLASSES = %w(
4  android.sensor.accelerometer
5  android.sensor.magnetic_field_uncalibrated
6  android.sensor.gyroscope
```

```
7  )
8
9  class Reading
10   attr_reader :source
11   attr_accessor :value
12   def initialize(source, value)
13     @source = source
14     @value = value
15   end
16 end
17
18 class Sample
19   attr_reader :readings, :class_value
20   def initialize(class_value)
21     @class_value = class_value
22     @readings = {}
23   end
24 end
25
26 attrs = {}
27 samples = []
28
29 ARGV.each do |class_name|
30   Dir.entries(class_name).each do |file_name|
31     file_path = "#{class_name}/#{file_name}"
32     next unless File.file? file_path
33     sample = Sample.new(class_name)
```

```
34      File.foreach(file_path) do |line|
35        cols = line.split ','
36        next unless CLASSES.include? cols[0]
37        cols[2..-1].each_with_index do |value, index|
38          name = "#{cols[0]}-#{('A'.ord + index).chr}"
39          reading = Reading.new(name, value.to_f)
40          attrs[reading.source] = 1
41          existing_reading = sample.readings[reading.
              ↪ source]
42          if existing_reading.nil?
43            sample.readings[reading.source] = reading
44          else
45            existing_reading.value = (existing_reading.
                ↪ value + reading.value) / 2
46          end
47        end
48      end
49      samples << sample
50    end
51  end
52
53  puts "@relation 'radio-signals'"
54
55  ordered_attrs = attrs.keys
56  ordered_attrs.each do |key|
57    puts "@attribute '#{key}' numeric"
58  end
```

```
59
60 class_names = ARGV.map { |c| "'#{c}'" }
61 puts "@attribute␣class␣{#{class_names.join␣',␣'}}"
62
63 puts "@data"
64 samples.each do |sample|
65   values = ordered_attrs.map { |attr| sample.readings[
       ↪ attr]&.value or '?' }
66   puts "#{values.join␣','},#{sample.class_value}"
67 end
```

Listing B.3: Running the classifier for the WiFi dataset

```
1 #!/usr/bin/env sh
2
3 for i in 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6
    ↪  1.7 1.8 1.9 2.0 2.1 2.2
4 do
5     java weka.classifiers.rules.
        ↪ ClusterDistanceClassifier -t wifi-merged.arff
        ↪ -x 10 -C 0 -D $i | grep 'Correctly Classified'
        ↪  | tail -1 | grep -o '[^ ]\+ \+%'
6 done 2>/dev/null
```

Listing B.4: Bluetooth Data Collector

```
1 public class BluetoothDataCollector extends
    ↪ DataCollector {
2   private BluetoothScanner mBluetoothScanner;
```

```
3
4     @Override
5     public void collect(Context context) {
6       mBluetoothScanner = new BluetoothScanner(new
          ↪ ScanCallback() {
7         @Override
8         public void onScanResult(int callbackType,
            ↪ ScanResult result) {
9           output(timeFromNanos(result.getTimestampNanos())
              ↪ ,
10                result.getDevice().getAddress(),
11                result.getDevice().getName(),
12                result.getRssi());
13        }
14      });
15      mBluetoothScanner.startScan();
16    }
17
18    @Override
19    public void stop(Context context) {
20      mBluetoothScanner.stopScan();
21    }
22  }
```

Listing B.5: Cell Data Collector

```
1   public class CellDataCollector extends DataCollector {
2     @Override
```

```
3   public void collect(Context context) {
4     TelephonyManager telephonyManager = (
        ↪ TelephonyManager) context.getSystemService(
        ↪ Context.TELEPHONY_SERVICE);
5     for (CellInfo info : telephonyManager.getAllCellInfo
        ↪ ()) {
6       Date date = timeFromNanos(info.getTimeStamp());
7       if (info instanceof CellInfoCdma) {
8         CellInfoCdma infoCdma = (CellInfoCdma) info;
9         output(
10              date,
11              infoCdma.getCellIdentity().getBasestationId
                  ↪ (),
12              "CDMA",
13              infoCdma.getCellSignalStrength().getDbm()
14        );
15      } else if (info instanceof CellInfoGsm) {
16        CellInfoGsm infoGsm = (CellInfoGsm) info;
17        output(
18              date,
19              infoGsm.getCellIdentity().getCid(),
20              "GSM",
21              infoGsm.getCellSignalStrength().getDbm()
22        );
23      } else if (info instanceof CellInfoLte) {
24        CellInfoLte infoLte = (CellInfoLte) info;
25        output(
```

```
26                  date,
27                  infoLte.getCellIdentity().getCi(),
28                  "LTE",
29                  infoLte.getCellSignalStrength().getDbm()
30              );
31          } else if (info instanceof CellInfoWcdma) {
32              CellInfoWcdma infoWcdma = (CellInfoWcdma) info;
33              output(
34                  date,
35                  infoWcdma.getCellIdentity().getCid(),
36                  "WCDMA",
37                  infoWcdma.getCellSignalStrength().getDbm()
38              );
39          }
40      }
41  }
42 }
```

Listing B.6: WiFi Data Collector

```
1  public class WifiDataCollector extends DataCollector {
2    private WifiScanner mWifiScanner;
3    private boolean gotNewData = false;
4
5    private void logData() {
6      for (ScanResult result : mWifiScanner.getScanResults
         ↪ ()) {
7        output(timeFromNanos(result.timestamp * 1_000),
```

```
            ↪ result.BSSID, result.SSID, result.level);
      }
    }

    @Override
    public void collect(Context context) {
      mWifiScanner = new WifiScanner(context, new
          ↪ BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent
            ↪ intent) {
          gotNewData = true;
          logData();
        }
      });
      mWifiScanner.startScan(context);
    }

    @Override
    public void stop(Context context) {
      mWifiScanner.stopScan(context);
      if (!gotNewData) {
        logData();
      }
    }
}
```

Listing B.7: Sensors Data Collector

```
1  public class SensorCollector extends DataCollector {
2    private class SensorCollectorHelper extends
       ↪ TriggerEventListener implements
       ↪ SensorEventListener {
3      private final String mName;
4      private final Sensor mSensor;
5
6      SensorCollectorHelper(Sensor sensor, SensorManager
         ↪ sensorManager) {
7        mName = sensor.getStringType();
8        mSensor = sensor;
9        if (sensor.getReportingMode() == Sensor.
           ↪ REPORTING_MODE_ONE_SHOT) {
10         sensorManager.requestTriggerSensor(this, sensor)
             ↪ ;
11       } else {
12         sensorManager.registerListener(this, sensor,
             ↪ SensorManager.SENSOR_DELAY_NORMAL);
13       }
14     }
15
16     @Override
17     public void onSensorChanged(SensorEvent event) {
18       outputFloats(mName, timeFromNanos(event.timestamp)
           ↪ , event.values);
19     }
```

```
20
21      @Override
22      public void onAccuracyChanged(Sensor sensor, int
            ↪ accuracy) {
23      }
24
25      @Override
26      public void onTrigger(TriggerEvent event) {
27        outputFloats(mName, timeFromNanos(event.timestamp)
              ↪ , event.values);
28      }
29
30      void stop(SensorManager sensorManager) {
31        if (mSensor.getReportingMode() == Sensor.
              ↪ REPORTING_MODE_ONE_SHOT) {
32          sensorManager.cancelTriggerSensor(this, mSensor)
                ↪ ;
33        } else {
34          sensorManager.unregisterListener(this);
35        }
36      }
37    }
38
39    private SensorManager mSensorManager;
40    private final List<SensorCollectorHelper> mSensors =
          ↪ new ArrayList<>();
41
```

```
42    @Override
43    public void collect(Context context) {
44      mSensorManager = (SensorManager) context.
          ↪ getSystemService(Context.SENSOR_SERVICE);
45      for (Sensor sensor : mSensorManager.getSensorList(
          ↪ Sensor.TYPE_ALL)) {
46        mSensors.add(new SensorCollectorHelper(sensor,
            ↪ mSensorManager));
47      }
48    }
49
50    @Override
51    public void stop(Context context) {
52      for (SensorCollectorHelper sensor : mSensors) {
53        sensor.stop(mSensorManager);
54      }
55    }
56  }
```

Listing B.8: Weka Classifier Implementation

```
1  public class ClusterDistanceClassifier extends
     ↪ AbstractClassifier implements
     ↪ WeightedInstancesHandler, OptionHandler {
2    class Dimension implements Serializable {
3      public double mean, variance, count;
4    }
5
```

```
6   class Location implements Serializable {
7     public final Map<Attribute, Dimension> dimensions =
        ↪ new HashMap<Attribute, Dimension>();
8   }
9
10  private int m_ClassIndex = 0;
11  private double m_DistFactor = 2.0; // set in options
12  private Location m_Cluster;
13  private double m_DistCutoff;
14
15  @Override
16  public void buildClassifier(Instances instances)
      ↪ throws Exception {
17    getCapabilities().testWithFail(instances);
18
19    Attribute classAttr = instances.classAttribute();
20    m_Cluster = new Location();
21
22    Enumeration<Attribute> attrs = instances.
        ↪ enumerateAttributes();
23    while (attrs.hasMoreElements()) {
24      Attribute attr = attrs.nextElement();
25      Dimension dim = new Dimension();
26      double total = 0;
27      for (Instance instance : instances) {
28        if (instance.value(classAttr) != m_ClassIndex ||
            ↪ instance.isMissing(attr)) {
```

```
29            continue;
30          } else {
31            total += instance.value(attr);
32            dim.count += 1;
33          }
34        }
35        dim.mean = total / dim.count;
36        dim.variance = 0;
37        for (Instance instance : instances) {
38          if (instance.value(classAttr) != m_ClassIndex ||
              ↪ instance.isMissing(attr)) {
39            continue;
40          } else {
41            double xMinusMean = instance.value(attr) - dim
                ↪ .mean;
42            dim.variance += (xMinusMean * xMinusMean);
43          }
44        }
45        if (dim.count > 1) {
46          dim.variance /= (dim.count - 1);
47          m_Cluster.dimensions.put(attr, dim);
48        }
49      }
50
51      double totalDist = 0;
52      double count = 0;
53      for (Instance instance : instances) {
```

```
54        if (instance.value(classAttr) == m_ClassIndex) {
55          totalDist += dist(instance);
56          count += 1;
57        }
58      }
59      double meanDist = totalDist / count;
60      double varianceDist = 0;
61      for (Instance instance : instances) {
62        if (instance.value(classAttr) == m_ClassIndex) {
63          double xMinusMean = dist(instance) - meanDist;
64          varianceDist += (xMinusMean * xMinusMean);
65        }
66      }
67      double stdDevDist = Math.sqrt(varianceDist / (count
          ↪ - 1));
68      m_DistCutoff = meanDist + stdDevDist * m_DistFactor;
69    }
70
71    double err(double x, double mean, double stdDev) {
72      double z = Math.abs((x - mean) / stdDev);
73      double t = 1.0 / (1.0 + 0.5 * Math.abs(z));
74      // approximate erf
75      double erf = 1 - t * Math.exp(-z*z - 1.26551223 +
76                                    t * ( 1.00002368 +
77                                    t * ( 0.37409196 +
78                                    t * ( 0.09678418 +
79                                    t * (-0.18628806 +
```

```
80                                          t * ( 0.27886807 +
81                                          t * (-1.13520398 +
82                                          t * ( 1.48851587 +
83                                          t * (-0.82215223 +
84                                          t * ( 0.17087277)))))
                                              ↪ )))));
85      if (z >= 0) return  erf;
86      else     return -erf;
87    }
88
89    double dist(Instance instance) {
90      double dist = 0;
91      Enumeration<Attribute> attrs = instance.
          ↪ enumerateAttributes();
92      while (attrs.hasMoreElements()) {
93        Attribute attr = attrs.nextElement();
94        Dimension dim = m_Cluster.dimensions.get(attr);
95        double value;
96        if (instance.isMissing(attr)) {
97          value = -100;
98        } else {
99          value = instance.value(attr);
100       }
101       double delta = 0;
102       if (dim == null) {
103         delta = Math.sqrt(100.0 + value);
104       } else {
```

```
105           double attrErr = err(value, dim.mean, Math.sqrt(
                  ↪ dim.variance));
106           double attrDist = attrErr * Math.log(dim.count);
107           delta = Math.sqrt(attrDist);
108         }
109         if (!Double.isNaN(delta)) {
110           dist += delta;
111         }
112       }
113       return dist;
114     }
115
116     @Override
117     public double classifyInstance(Instance instance) {
118       if (dist(instance) < m_DistCutoff) {
119         return m_ClassIndex;
120       } else {
121         return Math.abs(m_ClassIndex - 1);
122       }
123     }
124
125     // Option parsing and metadata omitted for brevity
126 }
```