Android Pin Cracking

A dissertation report submitted in fulfilment of the requirements for the degree of Bachelor of Technology

by

Nidheesh Rajesh Panchal (2016UCP1008)
Siddhant Gupta (2016UCP1455)
Rahul Jangir (2016UCP1396)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY

APRIL 2020

Certificate

We,

NIDHEESH RAJESH PANCHAL (2016UCP1008)

SIDHHANT GUPTA (2016UCP1455)

RAHUL JANGIR (2016UCP1396)

Declare that this thesis titled, “Android Pin Cracking” and the work presented in it are our own. I confirm that:

- This project work was done wholly or mainly while in candidature for a B.Tech. degree in the department of computer science and engineering at Malaviya National Institute of Technology Jaipur (MNIT).

- Where any part of this thesis has previously been submitted for a degree or any other qualification at MNIT or any other institution, this has been clearly stated. Where we have consulted the published work of others, this is always clearly attributed. Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this dissertation is entirely our own work.

- We have acknowledged all main sources of help.

Signed: _____

Date: _____

Dr. Vijay Laxmi

Professor

April 2020

Department of Computer Science and Engineering

Malaviya National Institute of Technology Jaipur

Abstract

The project aims at predicting the PIN codes of an android phone that a user may unlock using numeric PIN code. Sensors pre-installed on every smartphone work without any acknowledgement to the user and can fetch data continuously without raising suspicion. Using the orientation and movement data from such sensors, a person can detect any touch screen activity by observing a spike in the data from sensors that logs motion of the phone. Exploiting the loophole that sensors like accelerometer, gyroscope and gravity do not stop logging even when the phone is locked, we aim to crack the PIN of a phone.

Using an app, data was collected from multiple environments such as putting a phone on a table, using it in one hand. Different sensors' data used and harvesting machine learning power, we aim to predict what part of the screen is being touched and in turn predict the pin.

Acknowledgement

I would like to express my deep gratitude to Dr Vijay Laxmi, our project supervisor, for her patient guidance, enthusiastic encouragement and useful critiques of this work. A constant push and constructive feedbacks made this possible and helped us to have hands-on experience with the new technologies.

Nidheesh Rajesh Panchal (2016UCP1008)

Siddhant Gupta (2016UCP1455)

Rahul Jangir (2016UCP1396)

Contents

List of figures

List of tables

Chapter 1

1. Introduction

In this rising age of smartphones, it is very essential to know how safe your phone is from all the types of cracks, how safe is your lock screen and what background processes are running in your phone.

Looking specifically in the direction of exploiting the motion and position sensors of the android smartphones in the background, we aim to record the sensor data from the phone and predict the numeric PIN used to lock the phone.

It is observed that whenever the screen is touched on specific portions, a spike is created in the motion and position sensors, extracting such information and using it to know which part of the screen was touched, we can crack the pin.

And to do so, we log the data from sensors such as accelerometer, gyroscope and gravity, to know the orientation and movement of the phone in 3D space. Using it as a dataset to feed into a supervised multiclass machine learning (ML) model, that classifies the data into 10 classes which represents the keys pressed, or in other words, tell which part of the screen was touched where that particular number key will be present when the pin is being entered by the user.

Chapter 2

2. Important Terms and Concepts

2.1 Android Sensors

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy and are useful if you want to monitor three-dimensional device movement or positioning.

The Android platform supports three broad categories of sensors:

Motion sensors:

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors:

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors:

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers. [1]

2.1.1 Motion Sensor

Motion sensors are useful for monitoring device movement, such as tilt, shake, rotation, or swing. The movement is usually a reflection of direct user input (for example, a user steering a car in a game or a user controlling a ball in a game), but it can also be a reflection of the physical environment in which the device is sitting (for example, moving with you while you drive your car). In the first case, you are monitoring motion relative to the device's frame of reference or your application's frame of reference; in the second case, you are monitoring motion relative to the world's frame of reference.

All of the motion sensors return multi-dimensional arrays of sensor values for each SensorEvent. For example, during a single sensor event, the accelerometer returns acceleration force data for the three coordinate axes, and the gyroscope returns rate of rotation

data for the three coordinate axes. [2]

2.1.2 Position Sensor

The Android platform provides two sensors that let you determine the position of a device: the geomagnetic field sensor and the accelerometer.

NOTE: The orientation sensor was deprecated in Android 2.2 (API level 8), and the orientation sensor type was deprecated in Android 4.4W (API level 20).

That is, instead of using raw data from an orientation sensor, we use a data combination from the accelerometer and geomagnetic sensor. [3]

2.1.3 Accelerometer Sensor

An accelerometer is a device that measures proper acceleration.[4] Proper acceleration, being the acceleration (or rate of change of velocity) of a body in its instantaneous rest frame,[5] is not the same as coordinate acceleration, being the acceleration in a fixed coordinate system. For example, an accelerometer at rest on the surface of the Earth will measure an acceleration due to Earth's gravity, straight upwards [6] (by definition) of $g \approx 9.81$ m/s2. By contrast, accelerometers in free fall (falling toward the centre of the Earth at a rate of about 9.81 m/s2) will measure zero.

2.1.4 Gyroscope Sensor

The gyroscope allows you to determine the angular velocity of an Android device at any given instant. It tells you how fast the device is rotating around its X, Y, and Z axes.

2.1.5 Gravity Sensor

The gravitational force has three vector components, in X, Y & Z directions. A gravity sensor senses these components of a smartphone and tells the alignment or orientation of the phone. It uses gravity acting on it. (Z=9.8 when at rest).

2.2 Background Process

When a SensorEventListener is registered and used on a website or in-app, even after locking the phone while keeping the app/website open in the background, it could still log the sensor data without giving any warning or message to the user of the phone.

2.3 Multiclass Classification Models

Classification task with more than two classes. Each sample can only be labelled as one class. For example, classification using features extracted from a set of images of fruit, where each image may either be of an orange, an apple, or a pear. Each image is one sample and is labelled as one of the 3 possible classes. Multiclass classification assumes that each sample is assigned to one and only one label. One sample cannot, for example, be both pear and an apple.

2.3.1 Random Forest

A random forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. [7]

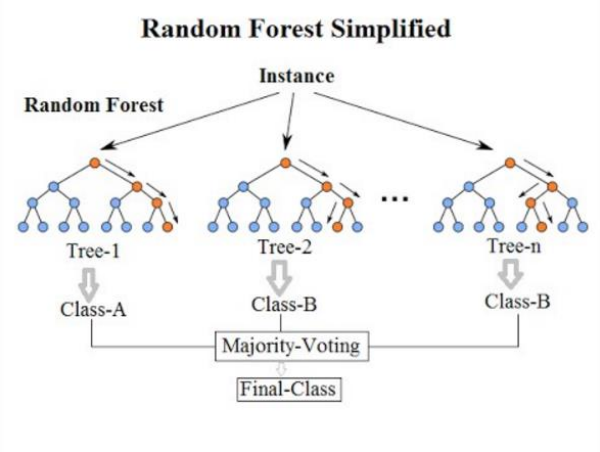

Figure 1: Random Forest

2.3.2 Logistic Model

The logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one. Although complex extension exists for multiclass classification. [8]

2.3.2 Naïve-Bayes Classifier

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. [9]

2.3.4 KNN Classifier

The basic logic behind KNN is to explore your neighbourhood, assume the test datapoint to be similar to them and derive the output. In KNN, we look for k neighbours and come up with the prediction.

In case of KNN classification, a majority voting is applied over the k nearest datapoints whereas, in KNN regression, mean of k nearest data points is calculated as the output. As a rule of thumb, we select odd numbers as k. KNN is a lazy learning model where the computations happen only at runtime. [10]

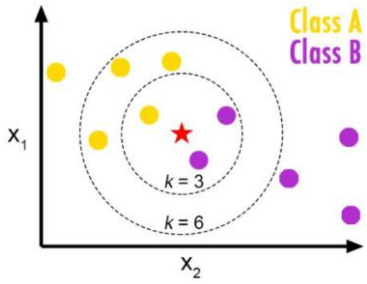

Figure 2: KNN Classifier

2.3.5 Decision Tree

A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification. [11]

2.3.6 Support Vector Machines

The objective of the support vector machine algorithm is to find a hyperplane in N-dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, many possible hyperplanes could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. [12]

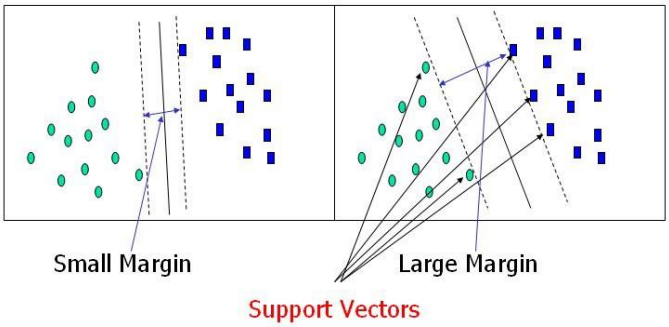

Figure 3: Support Vector Machine

2.4 Multioutput Classification

Classification task which labels each sample with a set of non-binary properties. Both the number of properties and the number of classes per property is greater than 2. A single estimator thus handles several joint classification tasks. This is both a generalization of the multilabel classification task, which only considers binary attributes, as well as a generalization of the multiclass classification task, where only one property is considered. [13]

Chapter 3

3. Literature Survey

3.1 0-permission sensors

Today we have various operating systems on our phones which includes the popular Android and iOS. According to the survey conducted by StatCounter, in March 2020, there are 72.26% Android users, 27.03% iOS users and 0.71% other mobile operating systems. We will be using Android phones that also shows that we will be targeting the larger group of people.

Creating an Android app (GyroAccData) for the phone, using OnePlus 5T and Redmi Note 8 Pro in every experiments and data collection. Learning from the Android Developers documentation [14] about the motion and position sensors and implementing it in the app, enabling the sensors, namely gyroscope, gravity and accelerometer sensor on the foreground of the app and display a live graph of the data being generated by them. Whenever a SensorListener is registered and an event is generated when there is motion detected for sensors. It is handled by onSensorEvent.

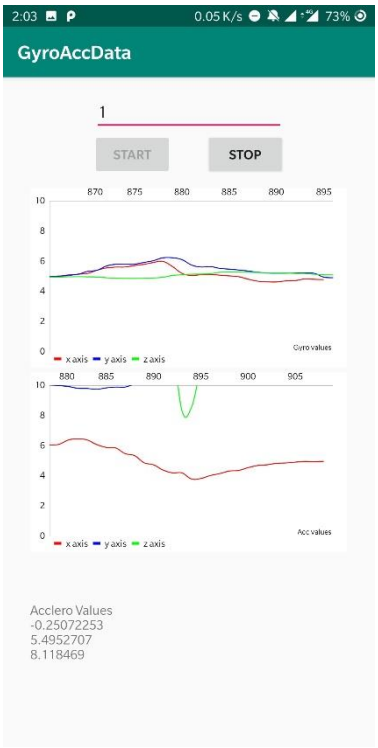

Figure 4: Sudden rise is seen in the graph when the screen is touched.

The top graph shows the gyroscope and bottom graph shows accelerometer events

All the sensors were kept at maximum frequency and it was observed from the graphs that on touch of the screen it creates a spike in the graph as it generates small movements when the screen is touched and released.

In the process of creating this app and checking the functions of the sensors, there was no single permission asked or any message shown on the screen that tells a user that these sensors are being used. Many such sensors are being used for activity tracking like step counting, augmented reality which also requires great computation power, but with advancements in processing power, the security measures are set aside for a while and not paid attention to with this detail.

These sensors are called zero-permission sensors which can leak sensitive information and, in our case, we can get the Personal Identification Number (PIN) from the user and his/her motion patterns.

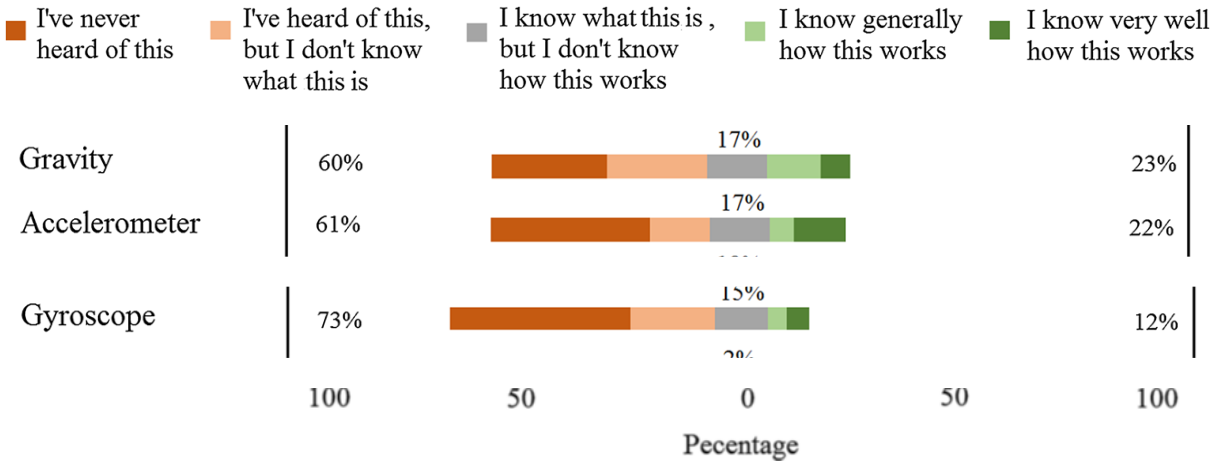

Figure 5: Level of self-declared knowledge about different mobile sensors

Figure 5., a survey conducted and published in [15] (Fig.5), we can see that majority of the people have never even heard about these sensors and hence the vulnerability increases.

3.2 Using sensors for predicting touch

From Figure 5, we know that it is possible to know when the screen is being touched. Using the high-frequency sensors it is also possible to know what portion of the screen is being touched by the user by looking at the specific pattern in the motion that is being recorded by these sensors.

Using the same app (GyroAccData) with modifications that store the data generated by the sensor events and storing them in (.csv) file format, we observe the following graph.

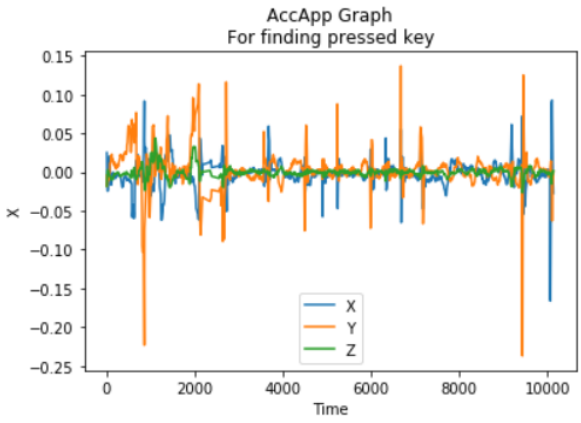

Figure 6: Unique spikes in the graph when different keys are pressed

It was observed that when the phone is held in one hand and kept stable and when numeric keypad (on-screen keyboard) keys are pressed, unique sensor data pattern is being produced which can be used to predict which part of the screen is being touched.

This experiment was tried in different environments and with multiple users. As the number of users increases, the way the phone is held changes. The person can be left-handed, right-handed, may keep the phone in both the hands, use it in one hand, touch with both thumbs or just one or use index finger. Keeping phone on a stationary material like a table, it was observed that due to the bulge of the camera module at the rear of the phone and also due to the protective cover (may or may not be present), the spikes in graph were observed, suggesting there might be a chance of predicting data even if it is kept on some stationary material. Different profiling can be done to the way the phone is being held and the prediction can be powered by machine learning classification models.

3.3 Ways to exploit leakage

Exploring how to exploit a leakage like this to steal PINs. There are two possible approaches. Either we can make a website that uses sensors in the background or we can build an Android application.

3.3.1 Site

As seen in the implementation of the research paper [15], they implemented a web page with

embedded JavaScript code to collect the data from voluntary users. The code registers two listeners on the window object to have access to orientation and motion data separately. The event handlers defined for these purposes are named DeviceOrientation-Event and DeviceMotionEvent, respectively. On the client-side, they had developed a GUI in HTML5 which shows random 4-digit PINs to the users and activates a Numpad for them to enter the PINs as shown in Figure 7.

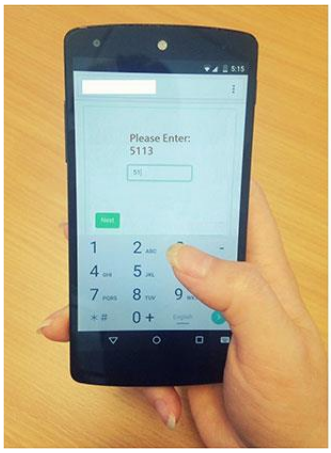

Figure 7: Site opened on a phone

All sensor sequences are sent to the database along with their associated labels which are the digits of the entered PINs. In the future implementation of the attack, they may focus on working with active web pages, which allows them to easily identify the start of a touch action through the JavaScript access to the onkeydown event. In an extended attack scenario, a more complex segmentation process would be needed to identify the start and end of a touch action. This could be achieved by measuring the peak amplitudes of a signal, as done in [16].

3.3.2 Android Application

An android application “Train” created by us, uses the Gyroscope, Accelerometer and Gravity sensors to collect data from the users. As soon as the person press the “start” button, the sensor listeners are registered and starts saving the data generated in (.csv) file format, with each file containing data for one key press only. As seen in Figure 8, the app also provides a numerical keypad, the layout and position of the numerical keypad is identical to the one provided by the Android OS.

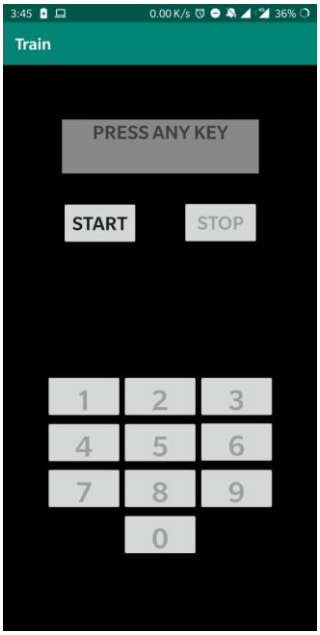

Figure 8: "Train" application that takes sensor data in background

In the training phase, a user is made to hold the phone in one hand and press the keys as displayed on the phone. Each key is to be pressed 4 times once the "start" button is pushed and all the 10 digits having 4 files each is stored on the external storage. A user press "stop" button and all sensors are unregistered and stop data collection. After each keypress, the corresponding file is saved with the label key press, key down and key up timestamp. Using the onKeyDown and onKeyUp events under android activity, we can get the required timestamps.

In the future implementation of an actual attack, we can deploy a service that will be running in the background that makes use of sensors and collect data from the user using similar approach from the section 3.3.1. The data can further be processed either locally and passing it to a trained machine learning model and then send the result back to the attacker or simply send the raw data in .csv (Comma-separated values) file format to attacker directly and then processing is done on the attacker end.

Chapter 4

4. Related Work

Two of the well-known published papers on the topic of exploiting smartphone sensor for cracking a PIN are [15][17]. They have shown remarkable results in this research topic and the method used by both Berend et al and Mehrnezhad et al is different.

Method used by Mehrnezhad et al:

Considering a set of 50 fixed PINs with uniformly distributed digits. They created these PINs in a way that all digits are repeated about the same time (around 20 times). Conducting user studies using Chrome on an Android device (Nexus 5). The experiments and results are based on the collected data from 10 users, each entering all the 50 4-digit PINs for 5 times.

Concerning the environmental setting for the data collection, the users remained seated in a chair while working with the phone. It did not require users to hold the phone in any particular mode (portrait or landscape) or work with it by using any specific input method (using one or two hands). They let them choose their most comfortable posture for holding the phone and working with it as they do in their usual manner.

To build the feature vector as the input to the classifier algorithm, they considered both time-domain and frequency-domain features. Different sequences obtained from the collected data include orientation (ori), acceleration (acc), acceleration including gravity (accG), and rotation rate (rotR) with three sequences (either x, y and z, or α, β and γ) for each sensor measurement. As a pre-processing step and to remove the effect of the initial position and orientation of the device, they subtracted the initial value in each sequence from subsequent values in the sequence. Used these pre-processed sequences for feature extraction in time domain directly. In the frequency domain, applying the fast Fourier transform (FFT) on the pre-processed sequences and used the transformed sequences for feature extraction. In order to build the feature vector, first obtained the maximum, minimum, and average values of each pre-processed and FFT sequences. These statistical measurements give $3 \times 12 = 36$ features in the time domain and the same number of features in the frequency domain. Considering the total energy of each sequence in both time and frequency domains calculated as the sum of the squared sequence values, i.e., $E = \sum v_i^2$ which gave 24 new features.

The next set of features were in the time domain and were based on the correlation between each pair of sequences in different axes. Having 4 different sequences; ori, acc, accG, and

rotR, each represented by 3 measurements. They calculated 6 different correlation values between the possible pairs; (ori, acc), (ori, accG), (ori, rotR), (acc, accG), (acc, rotR), and (accG, rotR), each presented in a vector with 3 elements. They used the Correlation coefficient function in order to calculate the similarity rate between the mentioned sequences. By adding these new 18 features, their feature vector consisted of a total of 114 features.

They applied a supervised machine learning algorithm by using an artificial neural network (ANN) to solve this classification problem. As input, ANN received a set of 114 features for each sample. Overall, out of 2500 records collected from 10 users, 12 of the PINs were entered wrongly. Hence, they ended up with 2488 samples for ANN.

The feature vectors were mapped to specific labels from a finite set: i.e., 50 fixed random 4-digit PINs. Training the network with 70% of data, validating it with 15% of the records, and testing it with the remaining 15% of their data set. They used a pattern recognition/classifying network in MATLAB with one hidden layer and 1000 nodes. Pattern recognition/classifying networks normally use a scaled conjugate gradient (SCG) back-propagation algorithm for updating weight and bias values in training.

Table 1: PIN identification rates in different attempts

Attempts	Multiple users (%)	Same user (%)
One	74	79
Two	86	93
Three	94	97

In this multiple users mode, the results are based on training, validating, and testing ANN using the collected data from all of the 10 participants. As the table shows, in the first attempt, PINlogger.js is able to infer the user's 4-digit PIN correctly with an accuracy of 74.43%, and it gets better in further attempts. By comparison, a random attack can guess a PIN from a set of 50 PINs with the probability of 2% in the first attempt and 6% in three attempts.

In single-user mode, the complete dataset was filled by one user alone. Classifier performed better when it was personalized.

Table 2: Average digit identification rates in different attempts

Attempts	Multiple users (%)	Same user (%)
One	70	79
Two	83	90
Three	92	96

The results in multiple-users mode indicate that they were able to infer the digits with a success probability of 70.75, 83.27, and 94.03% in the first, second, and third attempts, respectively. This means that for a 4-digit PIN and based on the obtained sensor data, the attacker can guess the PIN from a set of $3^4 = 81$ possible PINs with a probability of success of $0.9206^4 = 71.82\%$. [15]

Method used by D. Berend et al:

The second approach is to identify each digit in the sensor data individually. For this, first, they cut out the corresponding part from the relevant sensor data and then classify the digit presses individually. The benefit is, that only ten possible classes exist, each representing a key on the numerical PIN pad.

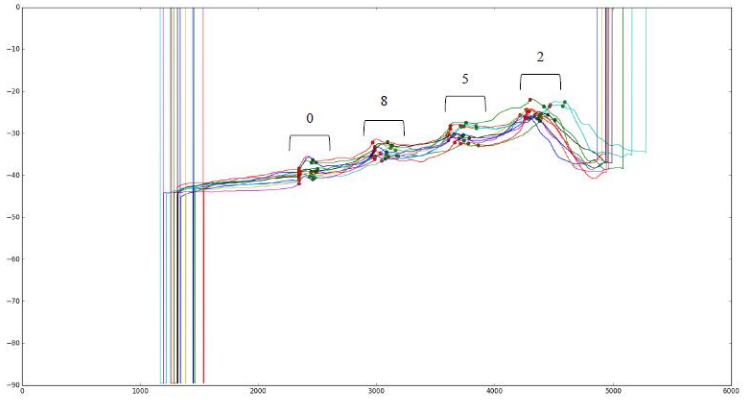

Figure 9: Rotation vector sensor showing spikes when a numeric key is pressed

The underlying data is identical to the data used in the first approach. However, they first split data into parts corresponding to the individual presses and then train and classify using only parts of the sensor data. This resulted in training and classification of only ten digits instead of 10,000 combinations, which is easier to achieve with a high success rate for individual

digits. Also, if they could identify individual digits of a PIN with a very high success rate, the overall success rate for a four-digit PIN will be high too.

This approach is more flexible, achieves a high success rate for any PIN length and also reduces the amount of training data significantly at the same time. Since they train only individual digits, they can apply their attack to any length of the PIN, with higher (shorter PIN) or lower success rate (longer PIN), while the other classification schemes working on complete PINs would not be able to classify any PINs shorter or longer than four digits.

For the classification of more PINs and also of different PIN lengths, they did not have to change the amount of training data, nor the nature of it. In contrast, the full-PIN classification schemes from previous work usually had to be trained with each PIN.

They chose an LG Nexus 5 for the common technology standard. For measuring the sensors during a PIN entry, they developed an android application, which runs in the foreground and stores the records in external storage. The sampling of each zero-permission sensor could be turned on or off individually. The app provided a numerical keypad, which appears once the measurement process starts. The layout and position of the numerical keypad are identical to the one provided by the Android OS. Figure 10.

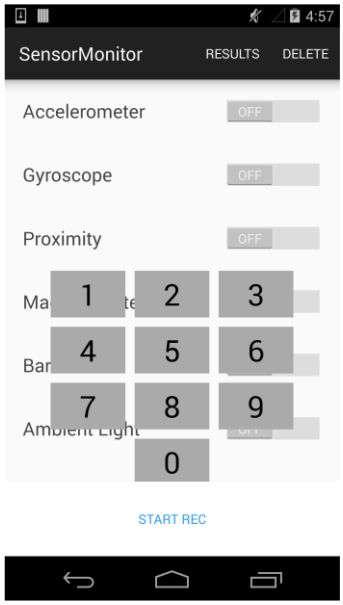

Figure 10: Layout of the application

A candidate was asked to interact with the application in a consistent natural position. Five times 70 chosen four-digit PINs were entered for training. They stated that the combination of the four digits plays a crucial role, to recreate a training base of data, which qualifies to be practically relevant, each key must be entered from all different positions. This was achieved by entering all possible ten keys before and after each key. In addition to the training data, 50

randomly generated pins were entered for validating the algorithm.

For this experiment the user sat on a chair, holding the phone in the right hand, typing the passwords with the hand's thumb at a consistent speed. They tested the contribution of all available zero-permission sensors for PIN recovery, excluded the barometer and proximity sensor, due to the low sampling frequency (0,5 - 4 Hz), which caused high inaccuracy and hence, were not providing any significant signal.

Each sensor had a list of (time; value) pairs for each dimension. They transformed the (time; value) pairs into an array, which holds a value for each millisecond. Dropping the beginning four data pairs, to eliminate initial sensor inaccuracy. The final step, to set the first value (denoted by V_{init}) of all-time arrays to zero and readjust all other values relative to the adjustment of the first one.

The primarily used classification algorithm was a multi-layer perceptron (MLP) neural network. The network possesses two middle layers. The first includes as many nodes as input features exist. The second as much nodes as classes. Through this architecture, the features become weighted more accurate and enter a pooling layer afterwards, to granulate the information. Backpropagation optimizes with a limited memory Broyden Fletcher Goldfarb Shanno (L-BFGS) algorithm, which can handle relatively small amounts of data well.

500 random training samples of the training data of three candidates were added to a training set. Then randomly generated testing data (30-50 four-digit combinations entered by each candidate) was classified by the fitted algorithm. Afterwards, another 500 training samples were added to the training set and fitted onto the algorithm again. The same testing took place. This process was repeated until the training set reached a total size of 2500 samples. For the classification, four algorithms were tested, namely MLP neural network, Random Forest, K - Nearest Neighbour and Gaussian naive Bayes classifier. The comparison is as follows, which showed the success of classification algorithms fitted with 500 to 2500 training samples and tested with 116 test samples in each iteration.

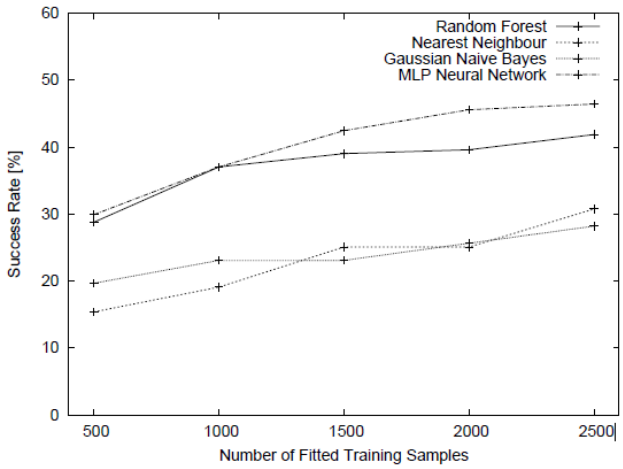

Figure 11: Success of classification algorithms

As a general conclusion, they reported a PIN retrieval success of 83:7% after combining data from relevant sensors and exploiting the information of thumb movement. This, to their knowledge, was the best-achieved figures in the context of PIN classification. The achieved success was reported when applied to complete the PIN search space of 10,000 PINs. [17]

Chapter 5

5. Proposed Method

After observing the results from the experiments performed, we propose a method to identify individual digits of a PIN.

5.1 Data collection

To start with training a Machine Learning model, first, we start with a basic step of data collection from a single user. Using the application "Train" as mentioned in section 3.3.2, a user is made to either sit or stand and hold the phone in right hand in one identical position every time. The app collects 4 types of sensor data

- Accelerometer with gravity: It records the device motion in 3D space and shows the rate of change in each of the 3 axes merged with gravity. (TYPE_ACCELEROMETER). When the phone is stationary (z=9.8).
- Accelerometer without gravity: It records the device motion in 3D space and shows the rate of change in each of the 3 axes without merging gravity. (TYPE_LINEAR_ACCELERATION). When the phone is stationary (z=0).
- Gyroscope: Rate of rotation in all 3 axes.
- Gravity: Force of activity in all 3 axes.

When the "Start" button is pressed, the sensors are registered and starts collecting data until a number key is pressed by the user as shown on the screen. As soon as the key is pressed, a gap of 2 seconds is kept to allow the keypress data to be well distributed. After the gap, a file is saved in (.csv) file format from all 4 sensor types to their corresponding folders. A new random digit is then displayed on the screen and the process repeats for 40 times and then the user is asked to press the "stop" button. So, in a single session, a user enters 40 digits i.e., a digit (0-9) is pressed 4 times each. Multiple sessions are taken on different days to allow a little variation in initial orientation of the phone as we are presently not removing the initial values from the dataset.

560 files are recorded using this method of keeping the phone in the right hand and using only the right-hand thumb. To know if there is a possibility that the digits can be identified if there was minimal phone movement, we collected data by keeping the phone on the table (with protective case) and another 400 files were collected.

5.2 Preprocessing

The raw data collected from the application contains X, Y, Z coordinate values and time. The file name is coded in such a way that it contains the timestamp of onKeyDown, onKeyUp and contains the number key that was pressed. We have used 3 ways of preprocessing the data and compared the same with different classification models.

5.2.1 Windowing

Taking a window of 25 values so that the initial and final values that are noisy and not helpful for classification and will simply confuse a model, such data is removed. So now the window of 25 values contains some initial values, the middle index is this window of (onKeyDown to onKeyUp) data, and the rest of the data is appended according to the space left in this window of 25 values. After stripping the "time" from the data, these values are converted to a single row as feature vectors in a dataset, so that there are (25*3) feature vectors.

5.2.2 Three values

Taking the same window frame of 25 values as windowing, but now we have taken only 3 values i.e., first value, mid-value of the key down and key up timestamp and the final value of the window and then it is converted to a single row and put in the dataset as feature vectors after stripping the "time". So, this pre-processing method gives (3*3) feature vectors.

5.2.3 Three values from uniform data

In this, first, we make the data uniform as the data collected by onSensorEvent is not uniform. It will not have data at uniform intervals. So, we take the data and make it uniform in intervals of 5 milliseconds where we fill the gap by taking the nearest timestamp to the interval and copying the corresponding data. After making the data uniform, the same processing is done as in the previous section.

All these 3 processing methods are used for the window of 25 as specified and also for a window of 30 and applied to batches of files. There are 3 batches of files.

- On-table data (400 files)
- In-hand data (560 files)
- Hybrid (merged both the files) (960 files)

These batches of files are created from all 4 sensor data files and comparison is done for each classification model.

5.3 Classification model

We are currently in the process of choosing which classification model will fit best and will give the best result even when this project will be scaled up. The following are the models that we have tried and used for classification. Taking all types of pre-processed dataset and all of them are run with the same configurations so as to do justice to the comparison.

- Random Forest: The configuration of random forest is such that, the maximum depth of the tree is set as 10 and the number of trees in the forest is also set as 10.
- Decision Tree: The configuration is set as: maximum depth of tree as 15.
- Logistic Regression: It is set to multiclass classification with solver as (lbfgs).
- Naïve Bayes: A simple Gaussian Naïve Bayes model is used.
- KNN: The number of neighbours is kept as 7.
- SVM: Keeping the decision function as one-vs-rest.
- Multioutput: It uses Random Forest in the multi-output classification where the input labels are set such that the keys that are adjacent to the actual key press are given as the input label.

Using the sklearn library for python, the listed classifiers are implemented and the results are compared.

Chapter 6

6 Experimental Results

It was observed that motion detected by the gravity sensor, it did not show remarkable changes in sensor data values, hence did not contribute to give accurate prediction (16%). The sensor data from the sensor type accelerometer without gravity also did not give accurate results (12.3%). So we can observe that gyroscope and accelerometer with gravity are the winners among all the other type of sensors. It can be seen in Table 3. It shows maximum accuracy among all types of processing methods.

Table 3: Comparison of classification models

Type of data	Random Forest		Decision Tree		Logistic		Naïve Bayes		KNN		SVM	
	Acc	Gyro	Acc	Gyro	Acc	Gyro	Acc	Gyro	Acc	Gyro	Acc	Gyro
On-table	37.5	42.5	20	34	19	19	22.5	23.75	35	28.75	17.5	35
In-hand	38.93	40.7	27.65	31.2	36.87	41.13	26.54	33.62	32.74	42.47	33.62	42.47
Hybrid	34.71	31.6	29.87	29.46	19.91	26.14	12.43	22.27	36.78	39.89	26.94	27.46

Table 4: Multioutput classification results

	Multioutput			
Type of data	Acc (%)		Gyro (%)	
	First 2	List of 4	First 2	List of 4
On-table	41	62	44	61
In-hand	49.64	72.34	59.57	73.75
Hybrid	43.56	61.82	51.45	67.63

In Table 3, it is seen that gyroscope is giving best results up to 42.47% accuracy of a single-digit prediction when the phone is in hand showing that gyroscope is a sensor that should be given more weight if it is to be merged with another sensor type.

In Table 4, we can see that from the list of 4 output, the actual key matching either first or the second output can be maximum of 59.57% accurate by keeping the phone in hand and using only gyroscope sensor. From the list of 4 output, the actual key present in that list, the maximum accuracy achieved is 73.75 using phone in-hand and gyroscope is the sensor being used.

Chapter 7

7 Conclusion

Exploring the sensors used on smartphones and exploiting the vulnerability, it is possible to develop a single-digit classification methodology to recover PIN from maliciously captured sensor data. Using different types of preprocessing methods and different classification models to classify a single digit, the comparison of all such models used is shown in Table 3 and Table 4 where maximum accuracy achieved for on-hand data collection method is 42.5 with random forest, in-hand data collection method is 42.47 with SVM and merged data is 39.89 with KNN.

The next steps to be followed are finding new preprocessing method, classification model and then selecting the best classification model out of the lot. Training the selected model with the collected data after using the best preprocessing method and creating either an Android background app service or host a website to exploit sensor data.

References

[1] https://developer.android.com/guide/topics/sensors/sensors_overview

[2] https://developer.android.com/guide/topics/sensors/sensors_motion

[3] https://developer.android.com/guide/topics/sensors/sensors_position

[4] Tinder, Richard F. (2007). Relativistic Flight Mechanics and Space Travel: A Primer for Students, Engineers and Scientists. Morgan & Claypool Publishers. p. 33. ISBN 978-1-59829-130-8. Extract of page 33

[5] Rindler, W. (2013). Essential Relativity: Special, General, and Cosmological (illustrated ed.). Springer. p. 61. ISBN 978-1-4757-1135-6. Extract of page 61

[6] Corke, Peter (2017). Robotics, Vision and Control: Fundamental Algorithms In MATLAB (second, completely revised, extended and updated ed.). Springer. p. 83. ISBN 978-3-319-54413-7. Extract of page 83

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier

[8] https://en.wikipedia.org/wiki/Logistic_regression

[9] https://www.geeksforgeeks.org/naive-bayes-classifiers/

[10] https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222

[11] https://www.geeksforgeeks.org/decision-tree/

[12] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[13] https://scikit-learn.org/stable/modules/multiclass.html

[14] https://developer.android.com/docs

[15] Mehrnezhad, M., Toreini, E., Shahandashti, S.F. et al. Stealing PINs via mobile sensors: actual risk versus user perception. Int. J. Inf. Secur. 17, 291–313 (2018). https://doi.org/10.1007/s10207-017-0369-x

[16] Narain, S., Sanatinia, A., Noubir, G.: Single-stroke languageagnostic keylogging using stereo-microphones and domain specific machine learning. In: Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless and Mobile Networks, WiSec'14, pp. 201–

212. ACM, New York (2014)

[17] Berend, David & Bhasin, Shivam & Jungk, Bernhard. (2018). There Goes Your PIN: Exploiting Smartphone Sensor Fusion Under Single and Cross User Setting. ARES 2018: Proceedings of the 13th International Conference on Availability, Reliability and Security. 1-10. 10.1145/3230833.3232826.