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 of the main sources of help.

Signed: ______________________________

Date: ______________________________

Dr Vijay Laxmi

Professor

May 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 a 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 the motion of the phone. Exploiting the loophole that sensors like accelerometer, gyroscope and gravity do not stop recording even when the phone is locked, we aim to crack the PIN of a mobile phone.

The data was collected using an app 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

We would like to express our sincere 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	Siddhant Gupta	Rahul Jangir
(2016UCP1008)	(2016UCP1455)	(2016UCP1396)

Contents

List of figures

List of tables

Chapter 1

1. Introduction

In this advancing age of smartphones, it is 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 ten classes which represent 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 user is entering the PIN.

Chapter 2

2. Important Terms and Concepts

2.1 Android Sensors

Almost all Android devices have built-in sensors that can measure motion, orientation and various environmental conditions. They can be used to monitor three-dimensional movement or position of a device.

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

It monitors movements such as tilt, shake, rotation, or swing. The movement can be caused by direct user input or by the physical environment in which the device is sitting. You can monitor motion relative to the device's frame of reference and movement relative to the world's frame of reference. [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. 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). [3]

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

2.1.3 Accelerometer Sensor

An accelerometer is a sensor that measures the linear acceleration of a device in its instantaneous rest frame,[5] An accelerometer at rest will measure an acceleration due to Earth’s gravity, straight upwards [6] g ≈ 9.81 m/s2. By contrast, accelerometers in free fall will measure zero.

2.1.4 Gyroscope Sensor

Gyroscope fetches the data that tells us about the angular velocity of the device on its axes, i.e. speed of rotation around its 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

It is a classification model that labels the input and categorizes it under one class from the multiple classes that are defined. For example, there can be many weather conditions possible according to the data input, and each sample is labelled under one class.

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]

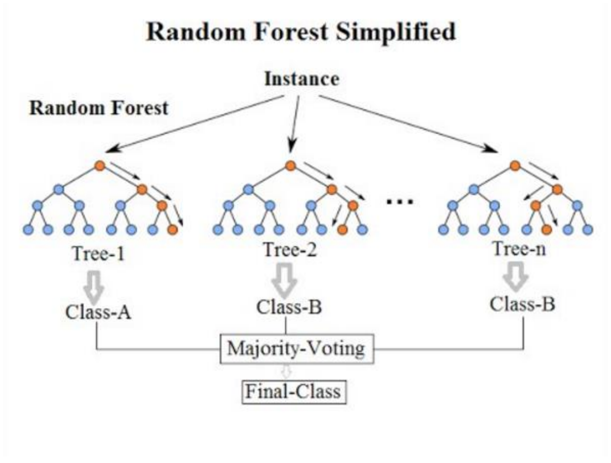


Figure 1: Random Forest

2.3.2 Logistic Model

The logistic model is used to model the probability of a specific class or event that has binary results which can either true or false. This can be extended to model with multiple classes similar to the example of weather forecasting. This is a complex extension 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

In KNN, we look for k number of 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]

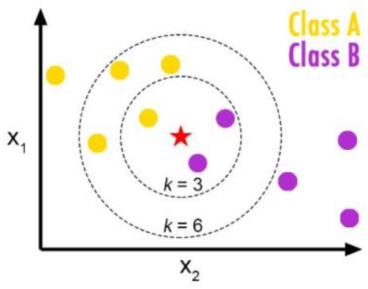


Figure 2: KNN Classifier

2.3.5 Decision Tree

Decision tree is like a flowchart tree structure. Each internal node is like an attribute, and the outcomes are the possible labels or predictions. If we follow the branches down to a leaf node, we can get all the attributes leading up to the predictions. The branches are created when a new value is discovered for an attribute and continues until it adds no value to predictions. They are very useful for exploratory knowledge discovery. It can also handle high dimensional data. They can give good accuracies in general. [11]

2.3.6 Support Vector Machines

A hyperplane is generated between all the data points such that it is at maximum distance from both sets of points that it separates (chosen support vectors). It helps is classifying the set of points in future with more confidence. We can have multi-dimensional hyperplane according to the features in the input. It can also be used for multiclass classification. [12]

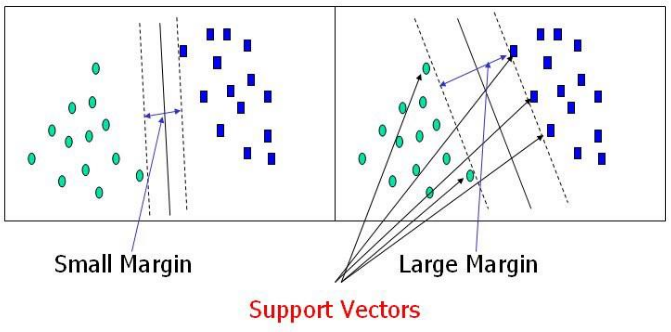


Figure 3: Support Vector Machine

2.3.7 Multi-Layer Perceptron model (MLP)

The basic perceptron algorithm says that in the perceptron, we just multiply with weights and add bias and do this in one layer only. In MLP, there can be more than one linear layer (combinations of neurons), i.e. having hidden layers in the network of nodes. Here we will be using MLP from sklearn library and Keras library naming the models as MLP and Deep Learning model respectively. These two different libraries provide a wide range of activation functions, optimizers and the different modifiable parameters, making them both important.

2.4 Multioutput Classification

As the name suggests, it used to get multiple outputs for a single input sample. Each output has its own set of properties, and each property may have multiple classes. It is like a combination of multilabel and multiclass classification. [13]

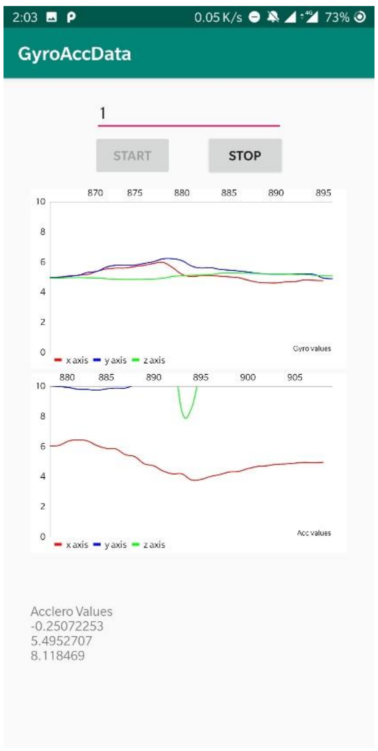
Chapter 3

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 devices.

By creating an Android app (GyroAccData) for the phone and using OnePlus 5T and Redmi Note 5 Pro in every experiment of 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.



Acclero Values
-0.25072253
5.4952707
8.118469

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

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

All the sensors were kept at maximum frequency, and it was observed from the graphs that on a 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. Still, 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.

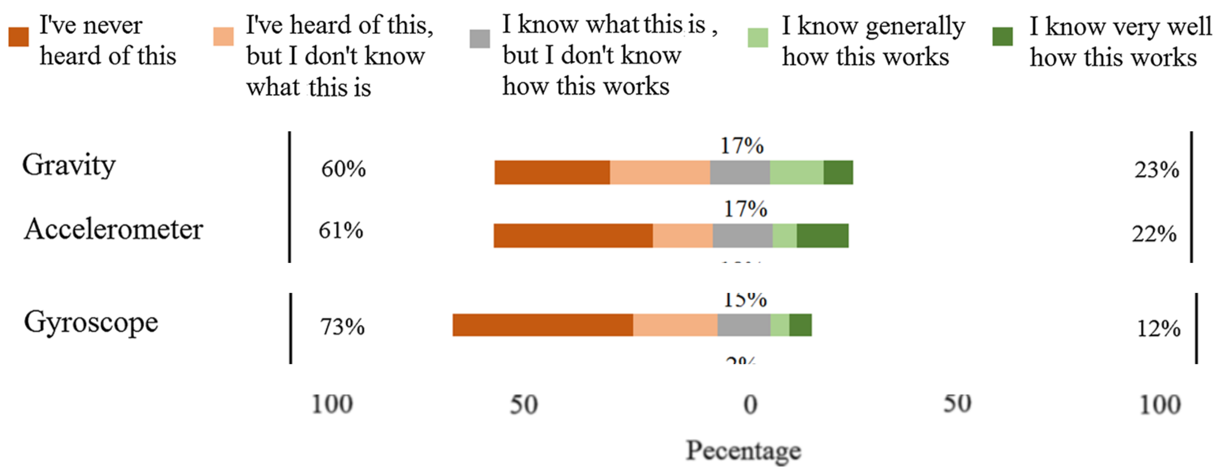


Figure 5: Level of self-declared knowledge about different mobile sensors

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 see 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.

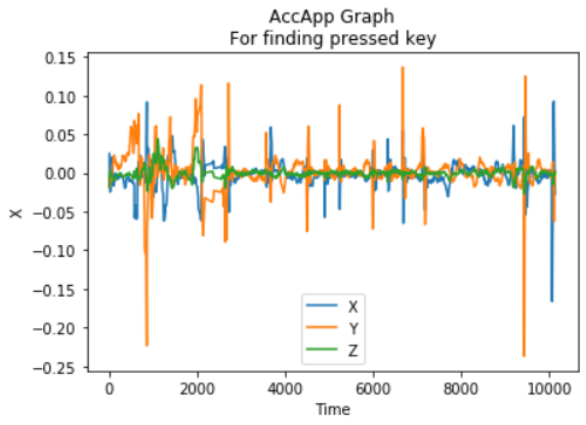


Figure 6: Unique spikes in the graph when different keys are pressed

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, a unique sensor data pattern is 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. By keeping the 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 the 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 machine learning classification models can power the prediction.

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 paper [15], they implemented a web page with embedded JavaScript code to collect the data from voluntary users. The code registers two and gets access to orientation and motion data. 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 brings up the virtual keyboard for users and then randomly generated 4-digit PINs are to be entered by the users. [15]

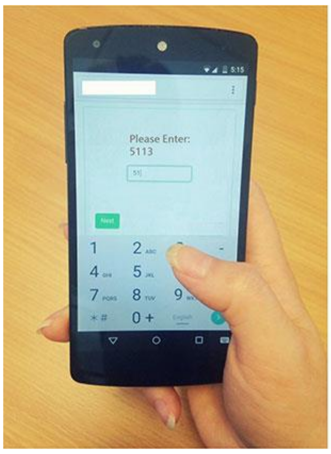


Figure 7: Site opened on a phone

The sensor data is sent to the database along with the PIN for which the data is held by the file. 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 a real-world attack scenario, we need to identify the keypress timestamp from the peak amplitudes in the data as the actual timestamp will not be recorded. We can see such implementation 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 start 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 are identical to the one provided by the Android OS.

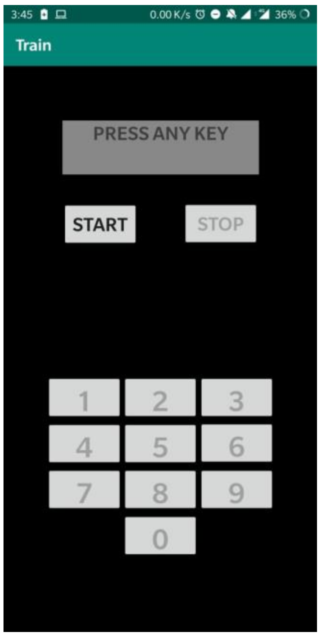


Figure 8: "Train" application that takes sensor data in background

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 four times once the "start" button is pushed and all the ten digits having four 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. It is discussed in details further in the report.

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 methods used by both Berend et al. and Mehrnezhad et al. are different.

Method used by Mehrnezhad et al:

They are 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). They used a Chrome browser on Nexus 5 device for their studies. They collected data from 10 users entering 50 4-digit PINs for five times, and the experimental results are based on this data

Concerning the environmental setting for the data collection, the users remained seated in a chair while working with the phone. They let the user choose their most comfortable posture for holding the phone and work with it as they do in their usual manner (portrait or landscape) (using one or two hands for input).

They considered both time-domain and frequency-domain features to get the feature vectors for input. There are four different types of sensor data: 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, they subtracted the initial value in each sequence from subsequent values in the sequence. They 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. To build the feature vector, first, it is required to obtain 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., $E = \sum v_i^2$, 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 four different sequences; ori, acc, accG, and rotR, each represented by three 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). As input, ANN received a set of 114 features for each sample. They had 2488 input samples for ANN.

They trained 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 typically 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. The table shows that in first attempt it can infer the user’s 4-digit PIN correctly with an accuracy of 74.43%, and gets better as the number of attempts increases. 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 tries.

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% in the first attempt and increasing with the number of attempts. This means that for a 4-digit PIN, the attacker can guess the PIN from a set of 81 possible PINs with 71.82% probability of success. [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 independently. The benefit is, that only ten possible classes exist, each representing a key on the numerical PIN pad.

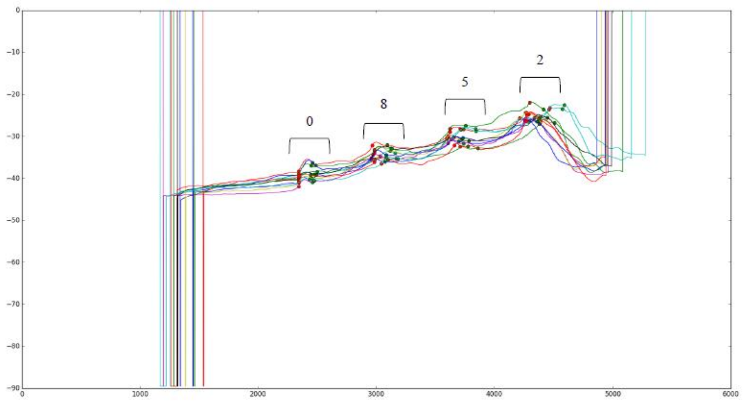


Figure 9: Rotation vector sensor showing spikes when a numeric key is pressed

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). In contrast, 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 conventional 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—figure 10.

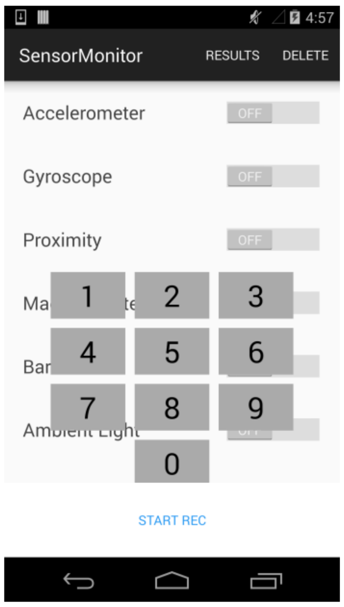


Figure 10: Layout of the application

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 right-hand thumb at a consistent speed.

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. Dropped the beginning four data pairs, to eliminate initial sensor inaccuracy. The final step was 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 and second includes as many nodes as classes. Back-propagation optimizes with a limited memory Broyden Fletcher Goldfarb Shanno (L-BFGS) algorithm, which can handle relatively small amounts of data well.

Five hundred random training samples of the training data of three candidates were added to a training set. Then randomly generated the trained model classified testing data (30-50 four-digit combinations entered by each candidate). 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.

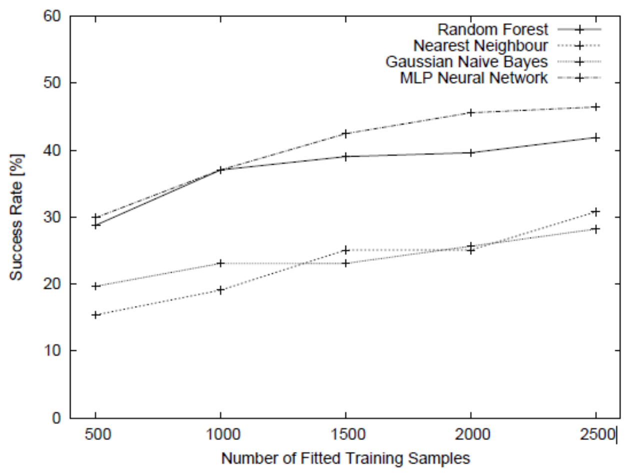


Figure 11: Success of classification algorithms

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 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.

We have decided to separate all the results in the order that it was evolved namely Method 1 Method 2.

5.1 Data collection

To start with training a Machine Learning model, first, we begin with a fundamental 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 four 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 three 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 three axes without merging gravity. (TYPE_LINEAR_ACCELERATION). When the phone is stationary (z=0).
- Gyroscope: Rate of rotation in all three axes.
- Gravity: Force of activity in all three 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 1 second is kept to allow the keypress data to be well distributed. After the gap, a file is saved in (.csv) file format from all four sensor types to their corresponding folders.

- Method 1: A new random digit is displayed on the screen. There are 40 such digits in a single session, i.e. a number (0-9) is pressed four times each.
- Method 2: A new digit is then displayed on the screen such that every digit appears before and after each digit at least once similar to [17]. There are 90 such digits in a single session, i.e. a number (0-9) is pressed nine times each.

After completion, the user is asked to press the “stop” button. Multiple sessions are taken on different days or a different time of day to allow a little variation in initial orientation of the phone as we are presently not removing the initial values from the dataset.

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 Pre-processing

The raw data collected from the application contains Time and X, Y, Z coordinate values. The file name is coded in such a way that it includes the timestamp of onKeyDown, onKeyUp and contains the number key that was pressed. We have used different methods of pre-processing the data and compared the same with varying models of classification.

For Method 1, we use:

- Windowing
- Three values
- Three values from uniform data

For method 2, we use:

- Windowing 2.0
- Window with uniform time
- Window with uniform time and timestamp (TS) at the end
- Window with uniform time and removing initial values
- Window with uniform time, removing initial values, timestamp (TS) at the end

In method 2, we combine two sensor type data, i.e. accelerometer without gravity combined with gyroscope and accelerometer with gravity combined with gyroscope and also the individual sensor data is used for comparison. When the two sensor type data is combined, different combinations of window size is used (200 or 400) for both accelerometer and gyroscope. The Accelerometer data is converted to a row first, and then the gyroscope data is appended at the end in the same row (more feature vectors).

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 considered only three 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 three 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 three 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 four sensor data files, and comparison is made for each classification model.

5.2.4 Windowing 2.0

Learning from the results of method 1, it was observed that the window of 25 is too small as the simple key up and key down time stamp itself has a gap of approx. 100ms. So, the window size is now increased (200 or 400 is used). Then the same steps are followed as in windowing. After stripping the "Time" column from the data, these values are converted to a single row as feature vectors in a dataset, so that there are (window size*3) feature vectors.

5.2.5 Window with uniform time

As the data recorded by the sensor is entirely dependent on the "sensor event" and is not continuous, the timestamp is first converted to uniform continuous time stamp such that the missing timestamp has the X, Y, Z coordinate value from the previous row same as used in [17]. Every millisecond has got an X, Y, Z value now, and then the windowing 2.0 is used to get the processed data.

5.2.6 Windowing with uniform time and TS at the end

It is similar to the previous section. The only difference is that instead of keeping the window of key up and key down as the centre of the total window size of 200 or 400. It is kept at the end of the 200 or 400 window size.

5.2.7 Windowing with uniform time and removing initial values

It is similar to the 5.2.5, but before windowing (200 or 400) values the starting 200 values are discarded to remove noise and then from the remaining data the first 300 values of the data file are averaged, and the absolute value from subtraction replaces the original value. It is used with a logic that the initial orientation of the phone should not affect the data arrangement of the same digit file if the phone is held differently. First, we used the method of merely removing the initial first value from the complete file without using absolute, then referring to [17] and looking at their success with such processing; we continued with this method.

5.2.8 Windowing with uniform time, removing initial values, TS at the end

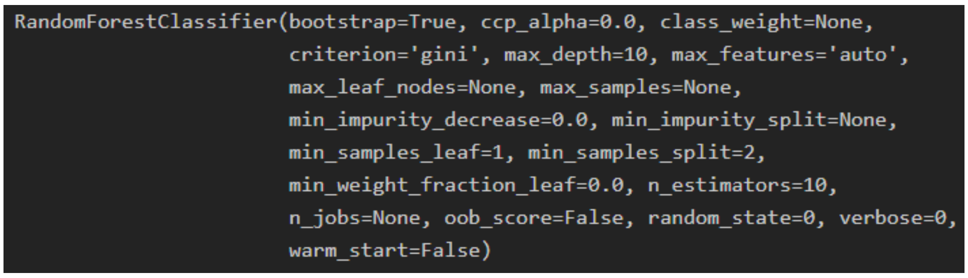
Taking the processed intermediate file from 5.2.7, and keeping the window of key up and key down at the end of the total window size of 200 or 400. It is similar to [17] (differs in window size) used for comparison of our results.

5.3 Classification model

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 to do justice to the comparison. The parameters set are decided by trying out different combinations of the various settings in each model, and then the figures below show the parameters for the best model that gave the highest accuracy for the train-test split.

Method 1:

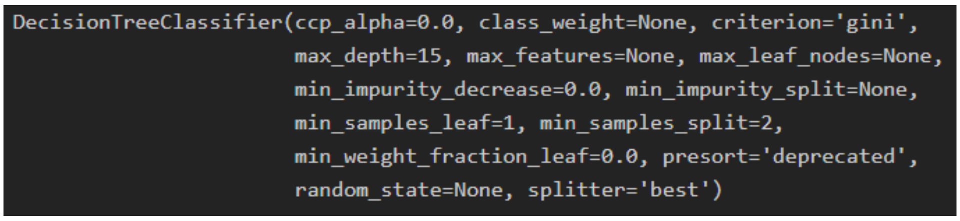
- Random Forest:



```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

Figure 12: Random Forest Classifier parameters

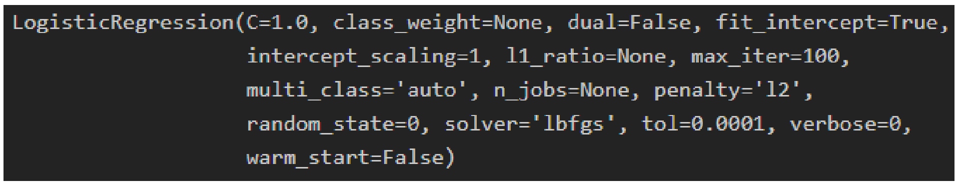
- Decision Tree:



```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=15, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Figure 13: Decision Tree Classifier parameters

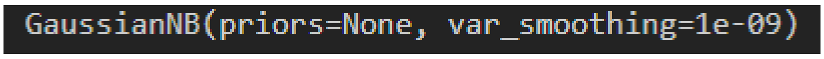
- Logistic Regression:



```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Figure 14: Logistic Regression parameters

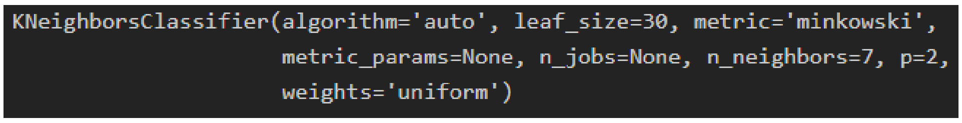
- Naïve Bayes:



```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Figure 15: Naive Baye's parameters

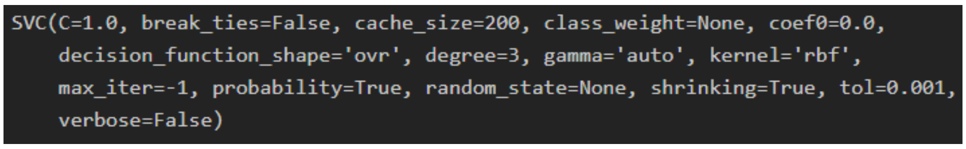
- KNN:



```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

Figure 16:K-Neighbours Classifier parameters

- SVM:



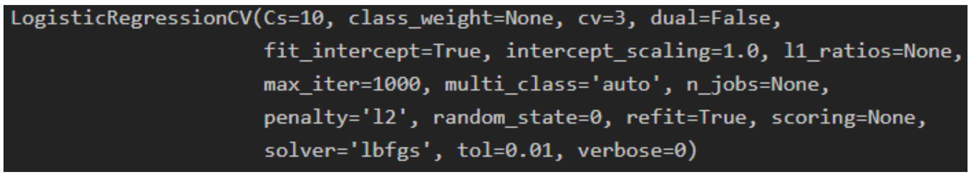
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
    max_iter=-1, probability=True, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

Figure 17: Support Vector Machine Classifier parameters

- 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.

Method 2:

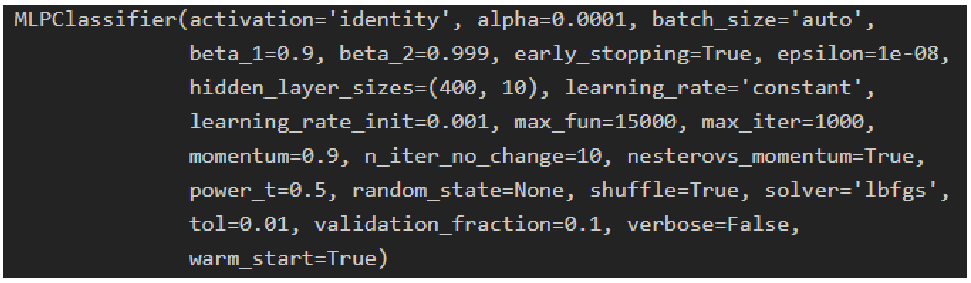
- Logistic regression CV: Increased the max_iter value to allow Convergence of the data.



```
LogisticRegressionCV(Cs=10, class_weight=None, cv=3, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=1000, multi_class='auto', n_jobs=None,
                     penalty='l2', random_state=0, refit=True, scoring=None,
                     solver='lbfgs', tol=0.01, verbose=0)
```

Figure 18: Logistic Regression parameters

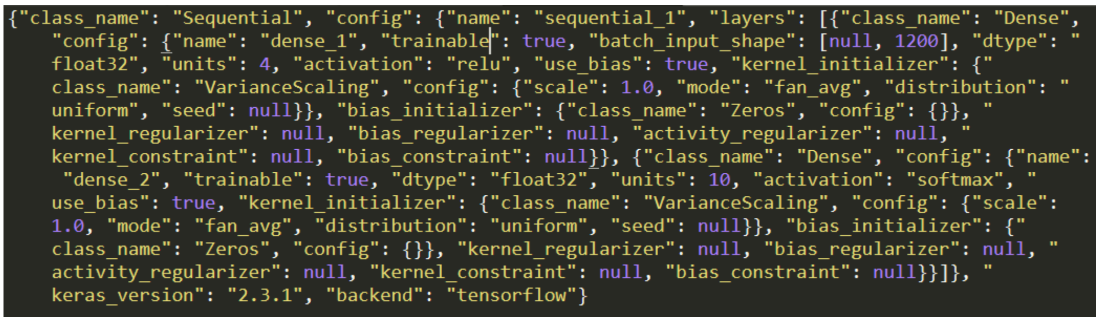
- SVM: Same parameters as before
- MLP: The hidden layer size is the same as to the window size



```
MLPClassifier(activation='identity', alpha=0.0001, batch_size='auto',
              beta_1=0.9, beta_2=0.999, early_stopping=True, epsilon=1e-08,
              hidden_layer_sizes=(400, 10), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=1000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=None, shuffle=True, solver='lbfgs',
              tol=0.01, validation_fraction=0.1, verbose=False,
              warm_start=True)
```

Figure 19: Multi-Layer Perceptron parameters

- Deep learning model using Keras library



```
{"class_name": "Sequential", "config": {"name": "sequential_1", "layers": [{"class_name": "Dense",
    "config": {"name": "dense_1", "trainable": true, "batch_input_shape": [null, 1200], "dtype": "
    float32", "units": 4, "activation": "relu", "use_bias": true, "kernel_initializer": {"
    class_name": "VarianceScaling", "config": {"scale": 1.0, "mode": "fan_avg", "distribution": "
    uniform", "seed": null}}, "bias_initializer": {"class_name": "Zeros", "config": {}}, "
    kernel_regularizer": null, "bias_regularizer": null, "activity_regularizer": null, "
    kernel_constraint": null, "bias_constraint": null}}, {"class_name": "Dense", "config": {"name":
     "dense_2", "trainable": true, "dtype": "float32", "units": 10, "activation": "softmax", "
    use_bias": true, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"scale":
    1.0, "mode": "fan_avg", "distribution": "uniform", "seed": null}}, "bias_initializer": {"
    class_name": "Zeros", "config": {}}, "kernel_regularizer": null, "bias_regularizer": null, "
    activity_regularizer": null, "kernel_constraint": null, "bias_constraint": null}}]}, "
    keras_version": "2.3.1", "backend": "tensorflow"}
```

Figure 20: Deep learning model parameters

Using the sklearn library and Keras library for python, the listed classifiers are used.

5.4 Exploitation App

After observing the results from all the above experiments to predict the individual PIN digit, it was time to test it on the field. There are two significant ways by which this can be done, either by using a site or an app. It was observed that after losing focus from the tab which runs the exploitation service, it stops recording the sensor data, so it is not usable. For the app, too, there is this limitation that the user should install the app first.

The functionality of the app must include that it should run a service in the background such that it will not alert the user of its existence. The app “Pin the PIN” was developed keeping this in mind. Due to the updates after the Nougat Android version, it is not possible to keep the service running in the background without keeping a notification in the notification panel informing the user about its existence. So, all the phones having Nougat Android version or below are vulnerable to this exploit once the app is installed on the phone.

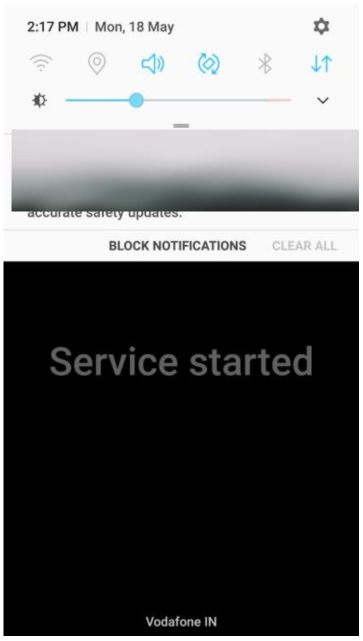
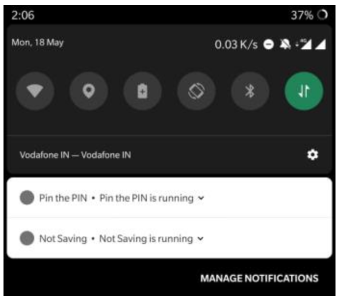


Figure 21: No notification for Android Nougat

The above screenshot is of a Samsung Galaxy A7 running Android version Nougat, and thus the service running does not get displayed in the notification panel and will not be killed.

With Android 10 in the market, there are 39.2% of Android phones that use Nougat or below (till April 2020). To overcome this low percentage of target phones what can be done is, for the Android versions higher than Nougat the app service will simply sit there in the notification panel saying “Service running in the background” and this notification is given a top priority. If the notification is not displayed, then the service is killed after some time to save power. Considering that the service must run all the time even if the app is opened once, the notification is to be created by the service. The app can be modified a bit to resemble any app that runs in the background and shows different functionalities from what it announces to be doing. Taking an example of Notification logger app which has to work 24/7 on the phone to log the user notifications, the user is okay with it running in the background as he/she trusts the app can be one possible modification of the app.



2:06 37%

Mon, 18 May 0.03 K/s

Vodafone IN — Vodafone IN

Pin the PIN • Pin the PIN is running

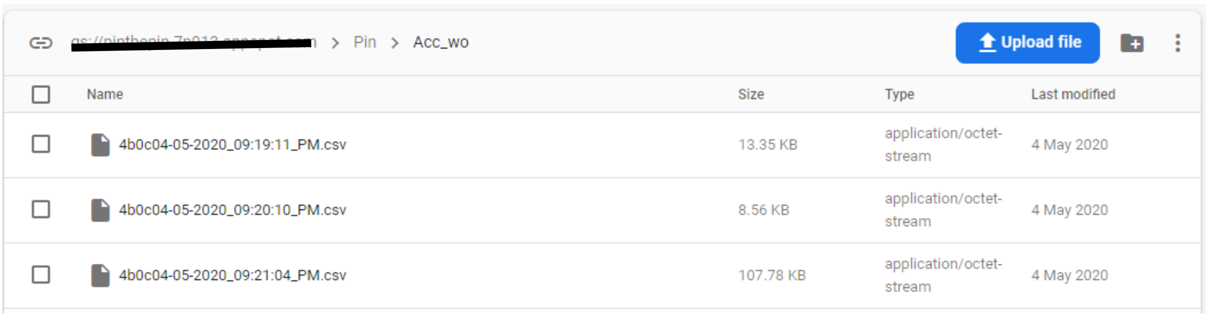
Not Saving • Not Saving is running

MANAGE NOTIFICATIONS

Figure 22: "Service running in background" notification on Android Pie

The above screenshot is from OnePlus 5T Using Android version Pie (9.0). As it can be seen that the notification is the same as the notification logger app “Not saving”.

What the app does is, when the phone is locked, and the screen is off, the data of the sensor is not recorded. As soon as the phone screen is turned on by the user, it starts logging the sensor values. After the phone is unlocked, the recording stops and saves the file in the internal app storage so that it is not available as a media file on the storage and along with that the files are uploaded to the Google Firebase storage so that it is accessible from anywhere. It requires an internet connection to upload the data files, but in today’s world of increased usage of the internet, there are hardly any phones that do not use the internet on the phone regularly. The files which are uploaded to the Google Firebase storage are then downloaded for further processing.



[redacted] > Pin > Acc_wo

Upload file

Name	Size	Type	Last modified
4b0c04-05-2020_09:19:11_PM.csv	13.35 KB	application/octet-stream	4 May 2020
4b0c04-05-2020_09:20:10_PM.csv	8.56 KB	application/octet-stream	4 May 2020
4b0c04-05-2020_09:21:04_PM.csv	107.78 KB	application/octet-stream	4 May 2020

Figure 23: Google Firebase storage where data is uploaded

First, a graph is plotted for the file which contains the accelerometer without gravity sensor data as it shows a proper graph of the linear motion and hence interpretable. It could be seen that the PIN of length four was entered by observing the five spikes in the graph. The first spike is the one when the user picks up a phone and swipes up on screen to get the PIN keypad and the other four spikes being the corresponding keys that were pressed. There is a little gap between them all, and are split in that way. Now we need to get the timestamp of the keypress by using only the graph of the file. All the four split data files having accelerometer without gravity sensor data is subjected to local minima search, and the lowest value of the Z coordinate is chosen as the timestamp of the key press and is appended in the file name.

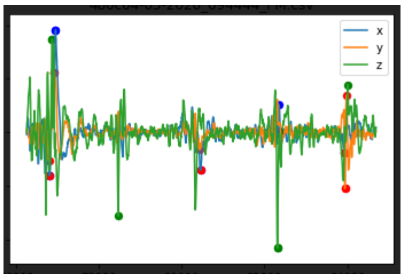


Figure 24: 5 spikes seen in the data for one unlock

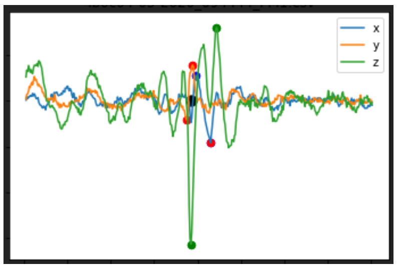


Figure 25: Split and get timestamp for processing (black dot shows timestamp)

The file is then further processed according to the model which we are using to predict the keys, using the same processing method, we get a file with four rows (4-digit PIN). Each key press gives 2-guess data for the key that might have been pressed. All the results are described in Experimental Results.

Chapter 6

6 Experimental Results

6.1 Method 1

It was observed that motion detected by the gravity sensor 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 provide accurate results (12.3%). 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 kinds of processing methods.

Table 3: Comparison of classification models in method 1

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.

6.2 Method 2

In the second method, we have used different Classifiers and discarded the previously used models that were not giving good results. The following table shows the maximum accuracies after comparison (the complete data is too big to be displayed). Using 2700 files in total and splitting it to get 25% of test data for validation.

Table 5: Method 2 test accuracies

Model	Type of processing			Accuracy (%)
	Acc w/o grav (size)	Gyro (size)	Type	
Logistic Regression	200	200	5.2.4	64.29
SVM	400	400	5.2.5	58.55
MLP	200	400	5.2.4	66.38
Deep learning model	200	200	5.2.4	61.57

Here, table 5 shows the window sizes used and the type of processing that shows the best accuracies from the complete comparison. All the other processing file types were eliminated. Then the single file that was giving good accuracies for 2 of the models was taken up for exploitation, i.e. accelerometer without gravity and Gyroscope sensor data with 200 window size of each and combined in a row to form feature vectors.

6.3 Pin the PIN

Taking 50 random 4-digit PINs to see how well the prediction works for the actual exploitation. The models are trained with the complete dataset of 2700 files and used for predictions. The following table shows the accuracy of predicting a single digit from all the keys pressed for those 50 PINs (50x4=200 digits) and the count of the correct digits predicted from a 4-digit PIN. The second guess of the digits surely boosts up the accuracy by at least 10% for each model.

(G1 → Guess 1) (G2 → Guess 2).

Table 6: Count of correct digits and accuracy

Model	1 digit		2 digits		3 digits		4 digits		Accuracies	
	G1	G2	G1	G2	G1	G2	G1	G2	G1	G2
Logistic	20	18	15	17	3	8	0	2	28.36	40.38
SVM	21	19	11	16	1	9	0	0	22.11	37.5
MLP	24	18	11	21	1	6	0	1	23.55	39.42
Deep Learning	13	21	1	7	1	3	0	0	8.65	21.15

After extending the dataset to 6450 files, the count of the correct digits predicted is as follows along with the accuracies.

Table 7: Effect of extension of dataset

Model	1 digit		2 digits		3 digits		4 digits		Accuracies	
	G1	G2	G1	G2	G1	G2	G1	G2	G1	G2
Logistic	26	12	13	29	1	6	1	1	28.36	44.23
SVM	19	17	9	16	7	7	0	4	27.88	41.34
MLP	19	19	13	20	2	8	0	0	24.51	39.9
Deep Learning	20	21	3	8	0	4	0	0	12.5	23.55

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 pre-processing 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.

Method 2 includes new pre-processing methods and classifiers. We can observe that the combination of both sensor type, accelerometer without gravity and gyroscope with a window of 200 each gives a good accuracy for Logistic Regression (64.29%) and Deep learning model (61.57%). Getting a window of 400ms for a digit in the real-world scenario is not possible every time as the speed of typing may be high for a PIN as it is entered very frequently. So the processing of file with larger window size is eliminated.

It will be safe to say that the PIN lock of a phone is crackable using these motion sensors. The accuracy of predicting a single digit of a PIN is 52% which is better than the brute force accuracy (10%). The prediction accuracy of 28.36% for 200 digits is significantly higher than the primary cracking method of trying out every combination. Comparing our results with [15] and [17] the model accuracies are not coming up to the same mark because of the different methods of data collection. Actual exploitation of data and their results are absent in [15] and [17], so the comparison cannot be made for that part. The further exploitation of PIN from the phone and processing the file using only the graph with no proper time stamp reduces the accuracy of the already trained model of ours. More amount of data may help increase the prediction accuracy.

References

[1] Sensors Overview | Android Developers. https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/sensors/sensors_overview.html

[2] Motion sensors | Android Developers. https://developer.android.com/guide/topics/sensors/sensors_motion

[3] Position sensors | Android Developers. 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] 3.2.4.3.2. sklearn.ensemble.RandomForestRegressor — scikit .... https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

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

[9] Biopython - Machine Learning - Tutorialspoint. https://www.tutorialspoint.com/biopython/biopython_machine_learning.htm

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

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

[12] Our objective is to find a plane that has the maximum https://www.coursehero.com/file/p27f3qtq/Our-objective-is-to-find-a-plane-that-has-the-maximum-margin-ie-the-maximum/

[13] 1.12. Multiclass and multilabel algorithms — scikit-learn .... 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 language-agnostic 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.