PERMISSION BASED MALWARE APP DETECTION IN ANDROID USING DEEP LEARNING

ABSTRACT

The world is contracting with the growth of mobile phone technology. The usage of smartphones and mobile tablets is continuously increasing due to the powerful Android Operating System which supports a large number of applications that make life more comfortable and advanced for the users. Android have developed a lot in last 15 years. However, being a leading and the most popular operating system for smart phones and tablets, it has also become a prime target for the attackers due to its growing users and it being an open source platform it has resulted to the countless number of malware hidden in the large number benign apps in Android marketplace. Thus there is a rising danger associated with the malware targeted at mobile devices and android security. Mobile malware has become a serious threat in the network security and privacy.

We, in this paper, tried to tackle the problem of detecting hidden malware in android applications and categorize them as malicious app, thus strengthening the android security. We describe our work done in detecting malware in the Android platform by performing static and dynamic analysis on the permission based framework in Android platform. In our work, we have extracted a number of permission based features, sensitive API calls and dynamic behaviours by disassembling the Android application (apk) files from an available dataset of android applications which can be found on android marketplace. We used deep learning techniques to implement a model that can automatically detect whether an app is malware or not based on the features extracted. Deep learning is a part of a broader family machine learning research that attempt to model high-level abstractions in data by using multiple processing layers. We are confident that the deep learning techniques are more suitable for characterizing Android malware and are highly effective with the availability of more training data. The models are evaluated with real malware samples and the results of experiments are presented to demonstrate the effectiveness of the proposed approach. We conclude our study with a number of implications for research and practice in future in the same field.

1. INTRODUCTION

There has been a very rapid increase in the Android applications in the past few years. Android has been the most popular OS on tablets and mobile phones since 2013, and is dominant by any metric. It has remained No.1 mobile operating system since 2013 according to the report from Gartner. In 2014, the company revealed that there were over one billion active monthly Android users, up from 520 million users in 2013. Google play store is a digital distribution platform which is operated by Google and serves as the official marketplace for android applications, thus allowing users to browse and download various android applications. There are also other third party marketplaces from where we can download android apps which tend to become popular with the popularity of apps provided there. These markets play an important role in the popularity of android OS.

However, being a leading and the most popular operating system for smart phones and tablets, it has also become a prime target for the attackers due to its growing users. Due to the openness of android market, it has resulted to the countless number of malware hidden in the large number benign apps in Android marketplace thus threatening the privacy and security among users. According to a survey conducted by Kaspersky Lab and Interpol between August 2013 and July 2014, one out of five Android devices protected by Kaspersky Lab products was attacked by malware at least once during the reporting period. Moreover, according to a report from McAfee Labs it was revealed that 3.73 million pieces of mobile malware were identified in 2014, increasing an astounding 197% from 2013. This vulnerability helps malware attacks to affect millions of user in the android ecosystem at once and thereby putting them at major security risk through their android applications. This has become very a challenging task to mitigate the risk of malware in an ecosystem where millions of users are present. And hence, there is an urgent need to develop powerful and efficient solutions to detect these malware and characterize the applications as benign and malware. We, in this paper, therefore attempted to detect and characterize the malware in android apps thereby strengthening the android privacy and security.

This problem is becoming more challenging as there is no as such solution available to detect these malwares. Today, we only have risk communication mechanism available to tackle this problem. It warns users about the permissions required by each app before installing it. However, this mechanism is ineffective as it fails to distinguish the permissions between benign and malware apps. They involve tricking users into downloading apps that may appear to be fine, but are actually malicious in nature and designed to conduct illicit activities. Thus we implemented a model that can effectively detect malware in android apps and characterize the apps as benign or malware. In the view of these situations, deep learning algorithms are being proposed to characterize the android malware that extract features by static and dynamic analysis of android apps and learn to distinguish between the features of benign and malware apps automatically.

Deep learning is a part of a broader family machine learning research that attempt to model high-level abstractions in data by using multiple processing layers. Deep learning algorithms are more suitable for characterizing Android malware and are highly effective with the availability of more training data. In this study, we first extracted a total of 153 features from manual analysis of app and then applied the deep learning technique to train a supervised learning model that can distinguish malware apps from benign apps. Experiments and research on large datasets have proved that deep learning algorithms can are especially suitable for correctly detecting characterising android malware with higher accuracy in the recent years. In this study we try to explore the concepts of deep learning algorithms by implementing deep network with three layers and recurrent neural networks (RNN). We compared our results with those of the models implemented using traditional machine learning techniques for supervised learning to classify like logistic classifier and Naïve Bayes classifier.

In upcoming sections, we will explore more about the project. In section 2 we will discuss about the background of the work. Section 3 depicts the methods and fundamentals employed in the work. Whereas section 4 depicts the proposed work. It is followed by the experiments and results in section 5. Finally, we conclude with conclusion and future work in section 6. Let us therefore discuss each of the sections in depth as follows.

2. Background and Related Work

This section explores the general related work covering a few sub-areas of the work presented in this section. All related work precisely relevant to the work is described in each of the following sub-sections. It is necessary to consider more information on the Android malware detection techniques employed by various researchers in the past. Our primary is to look for only the machine learning methods from the past that detect the malware in android applications and classify them as malware or benign based on the learning model used in them.

In their study for creating a tool for scaling Android App to real-world app stores [5], Siegfried Rasthofer and their colleagues conducted a study and in their work they thus present DROIDSEARCH, a search engine that aids a multi-staged analysis in which fast pre-filtering techniques allow security experts to quickly retrieve candidate applications that should be subjected to further automated and/or manual analysis. DROIDSEARCH identified 40 known malware applications in Google Play and detects over 35,000 applications that use both http and https connections for accessing the same resources, effectively rendering the https protection ineffective. It also reveals 11,995 applications providing access to potentially sensitive data through unprotected content providers. This is a part of our motivation to conduct more detailed analysis of android apps available to users for potential malwares in them.

Nicolas Viennot, Edward Garcia and Jason Nieh built PlayDrone [6], a scalable Google Play store crawler, and used it to index and analyse over 1,100,000 applications in the Google Play store on a daily basis, the largest such index of Android applications. PlayDrone leverages various hacking techniques to circumvent Google's roadblocks for indexing Google Play store content, and makes proprietary application sources available, including source code for over 880,000 free applications. They demonstrate the usefulness of PlayDrone in decompiling and analyzing application content by exploring four previously unaddressed issues: the characterization of Google Play application content at large scale and its evolution over time, library usage in applications and its impact on application portability, duplicative application content in Google Play, and the ineffectiveness of OAuth and related service authentication mechanisms resulting in malicious users being able to easily gain unauthorized access to user data and resources on Amazon Web Services and Facebook. This encouraged us to pursue the task of finding hidden malware apps in Android app eco-system.

Justin Sahs and Latifur Khan [7] presented a machine learning based system for the detection of malware on Android devices. The system extracts a number of features and trains a One Class Support Vector Machine in an offline (off-device) manner, in order to leverage the higher computing power of a server or cluster of servers.

Naser Peiravian, Xingquan Zhu [8] used permissions and API calls as features to characterize each Apps, and trained a model to learn a classifier to identify whether an App is potentially malicious or not. An inherent advantage of their method is that it does not need to involve any dynamical tracing of the system calls but only uses simple static analysis to find system functions involved in each App. In addition, because permission settings and APIs are always available for each App, our method can be generalized to all mobile applications. Experiments on real-world Apps with more than 1200 malware and 1200 benign samples validated the algorithm performance of their learning model.

3. Methods and Fundamentals

Taking forward the idea and concepts of implementing a model for analysis and characterization of malware from the android applications and learning from the short comings encountered in basic traditional approach, we proposed the concepts of Deep Learning Algorithms which include 3 layer network and RNN (Recurrent Neural Networks) in our paper. They are proposed in this section of Methods and Fundamentals as given.

Neural networks have hidden layers. Normally, the state of a hidden layer is based ONLY on your input data. So, normally a neural network's information flow would look like this:

input -> hidden -> output

This is straightforward. Certain types of input create certain types of hidden layers. Certain types of hidden layers create certain types of output layers. It's a closed system. Memory changes this. Memory means that the hidden layer is a combination of your input data at the current time-step and the hidden layer of the previous time-step. Why the hidden layer? Well, we could technically do this.

(input + prev_hidden) -> hidden -> output

(input + prev_input) -> hidden -> output

Here, we have 4 time-steps of a recurrent neural network pulling information from the previous hidden layer.

(input + empty_hidden) -> hidden -> output

(input + prev_hidden) -> hidden -> output

(input + prev_hidden) -> hidden -> output

(input + prev_hidden) -> hidden -> output

Focus on the last hidden layer (4th line). In the hidden layer recurrence, we see a presence of every input seen so far. In the input layer recurrence, it's exclusively defined by the current and previous inputs. This is why we model hidden recurrence. Hidden recurrence learns what to remember whereas input recurrence is hard wired to just remember the immediately previous data point. The hidden layer is constantly changing as it gets more inputs. Furthermore, the only way that we could reach these hidden states is with the correct sequence of inputs. Now the money statement, the output is deterministic given the hidden layer, and the hidden layer is only reachable with the right sequence of inputs.

We focus on the detection of Malware using static and dynamic analysis techniques. Static analysis refers to analysing the source code for malicious patterns without actually running the code. All the features fall under one of three types: required permissions, sensitive APIs, and dynamic behaviours. Among them, required permissions and sensitive APIs are extracted through the static analysis, whereas dynamic behaviours are extracted through dynamic analysis. An important static analysis technique focuses on analysing the manifest file (AndroidManifest.XML) included in every application for the set of permissions used and other components like Services, Broadcast receivers and Intents. Dynamic analysis includes extracting the dynamic behaviour of app that are included when the app is running. All we need is the installation file i.e. .apk file for each of the app that is collected from dataset. We them uncompressed the .apk file. As a first step each application is disassembled using Apktool to generate the manifest file and the source code. Then the permissions from the manifest file are extracted using an XML parser written in Python. We parsed these two files AndroidManifest.xml and classes.dex file.

We thus obtain the permissions required by the app. For example, android.permission.call phone is the permission required for an app to make a phone call and android.permission.camera is the permission required for an app to access the camera. Similarly chmod is a sensitive API that might be used for changing users' permissions on files and ContentResolver;->delete is a sensitive API that might be used for deleting users' messages or contacts. Also action send net is the action that sends data over the network, action phonecalls is the action that makes a phone call, and action sendsms is the action that sends SMS messages. In this way, we obtained a total of 192 features for each app through static and dynamic analyses. Here each feature is in the binary form which indicates that when a feature occurs in an app, its feature value is 1; otherwise, its feature value is 0. Thus we can easily classify the app by learning the pattern of the features present in benign and malicious apps independently.

A feature vector for each application is created based on a total of 192 permissions which is considered as the feature set. The accuracy of detecting the malware samples based on the features extracted is evaluated by applying deep learning classification algorithms viz., RNN.

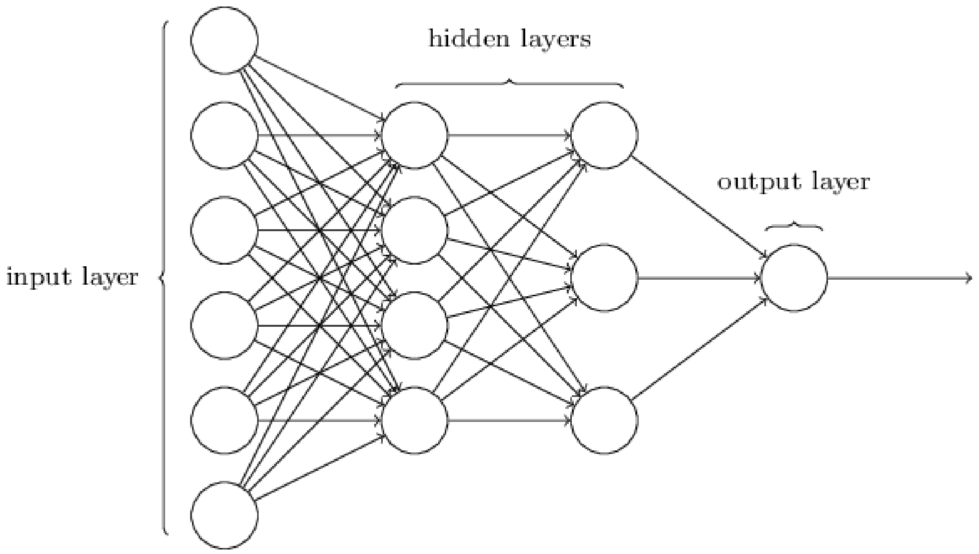


Fig 1. A 3-layer neural network with fully connected visible and hidden units. Note there are no hidden-hidden or visible-visible connections.

In this study we try to explore the concepts of deep learning algorithms by implementing deep network with three layers and recurrent neural networks (RNN). Recurrent nets are a powerful set of artificial neural network algorithms especially useful for processing sequential data. Recurrent nets and feed forward nets both "remember" something about the world, in a loose sense, by modelling the data they are exposed to. But they remember in very different ways. After training, feed forward net produces a static model of the data it has been shown, and that model can then accept new examples and accurately classify or cluster them. We compared our results with those of the models implemented using logistic classifier and Naïve Bayes classifier.

To summarize, we can say that solely relying on the risk communication mechanism that is the permission that are asked at the time of installation of each app lacks the ability to learn and distinguish the features of the app and detect the hidden malware present in the app, if any. Thus, efforts have been made to strictly extract the static and dynamic behaviours of each app, thereby learning the pattern among benign and malicious apps and training the dataset to efficiently detect the hidden malwares. Thus we collected installation file (.apk file) for various number of benign and malware apps, we uncompressed it and extracted the features through them. Later, the model was built using deep learning and deep neural networks to detect the hidden malwares. The model was implemented in Python.

4. Proposed Work

In this section, we illustrate the proposed work for our model to detect and analyse the hidden malware in the android applications effectively using Deep Learning algorithms. The experiments have been carried on the actual android applications dataset. We proposed our work in following subsections.

In section 4.1 we discuss the dataset collection and feature extraction of the applications. In section 4.2 we discuss the system architecture of the model proposed in our work. We explain all the components involved in the model and also discuss their functions respectively. Section 4.2 is followed by section 4.3 where we will propose the pseudo code for our algorithm used in implementing our deep learning model. The section 4.4 discusses about how we implemented Recurrent Neural Networks and 3 Layer Neural Network - Deep Learning algorithm to design our model. We conclude this by summarizing the proposed work. Let us therefore discuss each of the sections as described below.

4.1 Feature Extraction

We collected total of 500 malicious applications and 20,000 benign applications from Contagio community. We performed static and dynamic analysis on the applications collected in order to extract the features for each of the application. All the features fall under one of three types: required permissions, sensitive APIs, and dynamic behaviours. Among them, required permissions and sensitive APIs are extracted through the static analysis, whereas dynamic behaviours are extracted through dynamic analysis. An important static analysis technique focuses on analysing the manifest file (AndroidManifest.XML) included in every application for the set of permissions used and other components like Services, Broadcast receivers and Intents. Dynamic analysis includes extracting the dynamic behaviour of app that are included when the app is running. All we need is the installation file i.e.,.apk file for each of the app that is collected from dataset. We them uncompressed the .apk file.

As a first step each application is disassembled using Apktool to generate the manifest file and the source code. Then the permissions from the manifest file are extracted using an XML parser written in Python. We parsed these two files AndroidManifest.xml and classes.dex file. We thus obtain the permissions required by the app using Androguard decompiling tool. Androguard is a full python tool to analyse Android files; dex, odex, APK, Android's binary xml, Android resources, disassemble dex byte codes and decompiler for dex files. To extract features from any android app, we just need to use some quick command line commands that will print app permissions as output on the screen. For example, we can use the get_permissions() command like follows:

Fig 2: get_permissions() command as used in Androguard for extracting permissions

```
Command: get_permissions()
In [19]: a.get_permissions()

Out[19]:['android.permission.RECEIVE_BOOT_COMPLETED','android.permission.INT
ERNET','android.permission.READ_PHONE_STATE','android.permission.WRITE_EXTER
NAL_STORAGE','android.permission.ACCESS_NETWORK_STATE','android.permission.S
END_SMS','android.permission.RECEIVE_SMS']
```

For example, android.permission.call phone is the permission required for an app to make a phone call and android.permission.camera is the permission required for an app to access the camera. Similarly chmod is a sensitive API that might be used for changing users' permissions on files and ContentResolver;->delete is a sensitive API that might be used for deleting users' messages or contacts. Also action sendnet is the action that sends data over the network, action phonecalls is the action that makes a phone call, and action sendsms is the action that sends SMS messages.

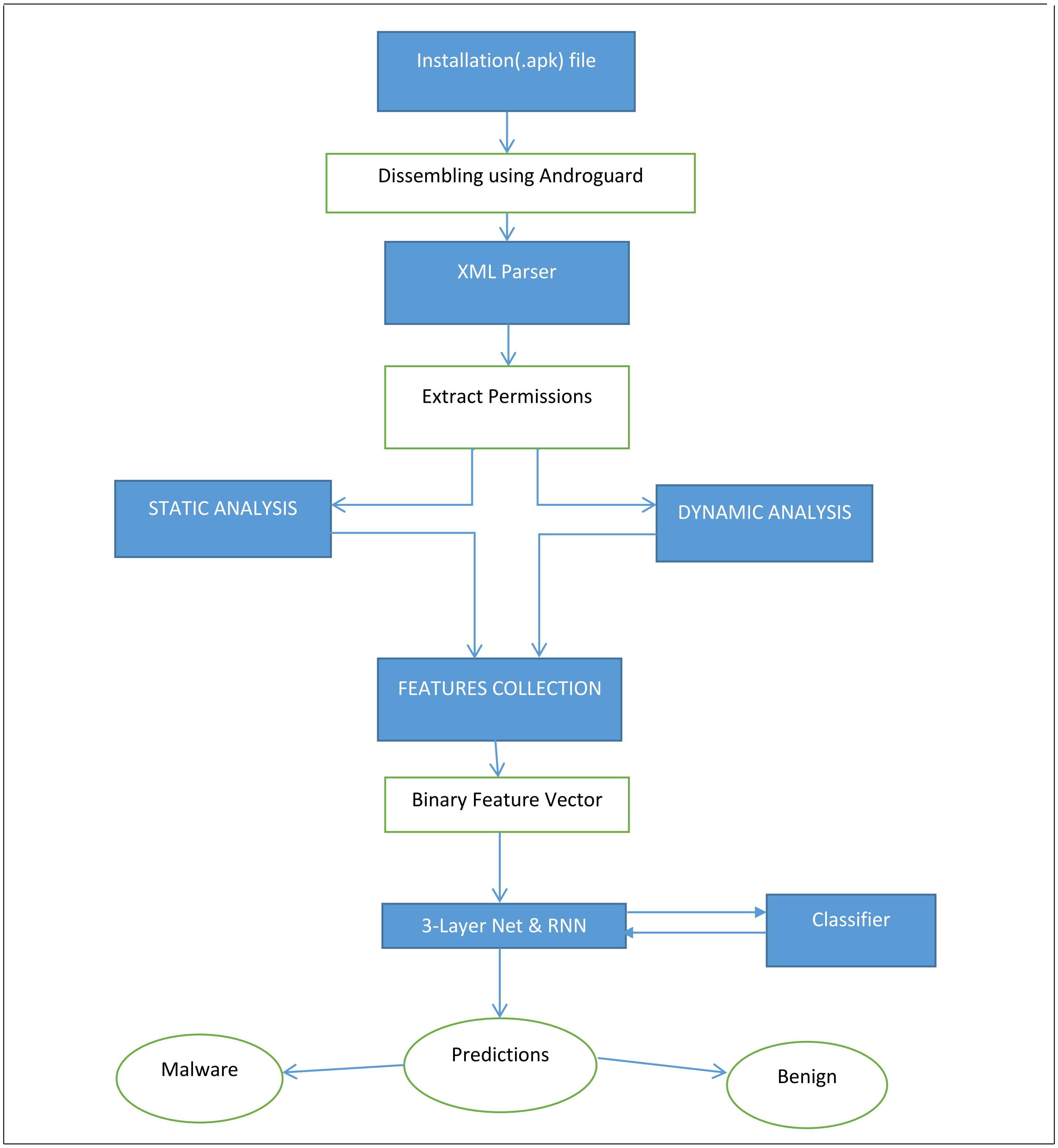
In this way, we obtained a total of 153 features for each app through static and dynamic analyses. Here each feature is in the binary form which indicates that when a feature occurs in an app, its feature value is 1; otherwise, its feature value is 0. Thus we can easily classify the app by learning the pattern of the features present in benign and malicious apps independently.

We obtained the dataset from Contagio community. Then for each installation file we uncompressed the file and extracted its static and dynamic features in order to get a input matrix for our model. We extracted total of 192 permissions, both static and dynamic. A feature vector for each application is created based on a total of 192 permissions which is considered as the feature set. These features are fed to the training model. Then it is fed to our deep learning model- 3 layer network and recurrent neural network which learns the pattern of the features and correctly detects the hidden malware. The figure below depicts the system architecture of the model which includes all these components.

4.2 System Architecture

Figure 1 (please see below) shows the system architecture of the model proposed in our work. The model comprises of following components – Installation file, APK Tool, XML Parser, Extracted Permissions, Static and Dynamic analysis, Feature extraction, binary feature vector, 3 layer net and RNN.

Figure 3. System Architecture of Android Malware Detection Model



4.3 Pseudocode

<u>Feature Extraction from apk</u>

```
Input : Set of apks for extracting permissions
Output : List of permissions used by each apk
1. Download/Obtain apk for a set of app.
2. Set up Androguard apk analysis tool.
3. Input apk in Androguard get_permissions() function call using comman-line:
```

```
In [19]:a.get_permissions()

Out[19]:['android.permission.RECEIVE_BOOT_COMPLETED','android.permission.INT
ERNET','android.permission.READ_PHONE_STATE','android.permission.WRITE_EXTER
NAL_STORAGE','android.permission.ACCESS_NETWORK_STATE','android.permission.S
END_SMS','android.permission.RECEIVE_SMS']
```

<u>Model</u>

```
Input : Extracted permissions of a set of apks.
Output : Predictions indicating whether an app is malware (1) or benign (0).
1. Seed dataset of extracted permissions in sigmoid function
2. Allocate weight to each (n-1) synapse from the output of sigmoid function
3. Sigmoid() //maps any numeric value to a value between 0 and 1.
4. Allocate binary (0 and 1) values to each neuron in network from the dataset
5. for number_of_iterations :
```

```
6.        Matrix_Mutliplication()
7.        Perform matrix multiplication of each neuron and synapse
8.        Forward_Propagation()
9.        Backward_Propagation()
9. Propagate values obtained from previous multiplication to next layer of neurons.
10. Print prediction values from output layer.
```

4.3 Approach

There are like a million and one machine learning models out there but neural nets in particular have gotten really popular recently because of two things. Faster computers and more data. There are really just 3 steps involved. We first build our model, train it and test it. Once we build our model, we trained it against our input and output data to make it better and better at pattern recognition. So we have built our model using a Layered neural network and Recurrent Neural Network algorithms which are actually the Deep learning algorithms. We created a function that maps any value to a value between 0 and 1. This is called sigmoid. This function will be run on every neuron in our network when data hits it. It's useful for creating probabilities out of numbers. Once we've created that, we initialize our dataset as a matrix. Each row is a different training example. Each column represents a different neuron. So we have app as layer and each permission as a neuron. Then we created our output dataset. For example, 1 output neuron each which corresponds to 0 or 1 based on target features whether app is benign or malware.

Next we created our synapse matrices. Synapses are the connections between each neuron in one layer to every neuron in next layer. Since we have n layers in our network, we'll need (n-1) synapses. Each synapse has a random weight assigned to it. After that we begin the training code. We created a for loop that iterates over the training code and optimizes the network for the given dataset. We start off by creating our first layer. It's just our input data. Later comes the prediction step. We performed matrix multiplication between each layer and synapse. Then we run our sigmoid function on all the values in a matrix to create next layer. The next layer contains a prediction of output data. Then we do the same thing on that layer to get our next layer. Which is a more refined prediction. So we will get a prediction at the output value in (n-1)th layer. Then we compared it with a output data using subtraction to get error rate. We also find out the average error rate at a set interval to make sure it goes down every time. Next we multiply our error rate by the result of our sigmoid function. The function is used to get the derivative of our output prediction from layer (n-1). This will give us a delta which will reduce the error rate of prediction when we update our synapse every iteration. Then we see how much layer 1 contributed to layer 2 and so on. This is called back propagation. Finally, we predicted the output. Error rate decreases every iteration and predicted output is very close to the actual output.

In the next section, we present the experiments conducted in our study along with the results of our implemented work.

5. Result

In this section, we discuss an experimental study which is conducted on real-world datasets to demonstrate the effectiveness of our approach.

5.1 Dataset

We obtained the dataset of malware apps from Contagio community. For extracting permissions dataset for benign apps, we used freely available dataset of benign apps from the web which contains hundreds of thousands of benign apps arranged along with their specific permissions in binary values where 0 indicates app is non-malware/benign and 1 indicates that app is a malware. Then for each installation file we uncompressed the file and extracted its static and dynamic features in order to get an input matrix for our model. We extracted total of 192 permissions, both static and dynamic. A feature vector for each application is created based on a total of 192 permissions which is considered as the feature set. These features are fed to the training model. Then it is fed to our deep learning model- 3 layer network and recurrent neural network which learns the pattern of the features and correctly detects the hidden malware.

5.2 Experimental Set-up

On last.fm dataset, we first carry out necessary pre-experimental processing to avoid possible errors when we actually use the dataset as an input in the established algorithms used in the later stages. We removed error causing null characters in the dataset other than 1s and 0s. Thus, we get clean dataset retrieved essentially from thousands of apk permissions. Next task is to feed this dataset into our deep learning models that is made of two popular deep learning techniques: - 3-layers Neural Network and Recurrent Neural Network.

We conducted our experimental study on a laptop with 8 gigabytes of RAM and four-core 2.3 gigahertz processor. For additional computational support for the experiments, we also employed an NVidia GPU that helps in speeding up the whole model training process. Subsequently, we mixed together an equal number of benign apps permissions dataset from benign app dataset with the malicious apps dataset obtained from malware apps retrieved from Contagio community. In doing so, we are able to obtain a training set and a test set. Training set and Test includes 500 benign apps and 500 malicious apps selected randomly. All the experiments were then performed on these two datasets.

We are able to set several parameters when we train a deep learning model like number of layers in the network, alpha parameter, and number of neurons in each later, proper approximation of synapses. Synapses are the connection between two neurons in a neural network in deep learning. We can also set the number of iterations before which we obtain predictions on the output layer. Moreover, we compared the predictions results of our model with that of traditional machine learning model namely, Naïve Bayes Classifier, Logistic Regression etc.

Table 1. The comparison between deep learning and machine learning models.

Model	**Benign Apps**		**Malicious Apps**	
	Precision (%)	Recall (%)	Precision (%)	Recall (%)
SVM	93.63	91.93	92.08	93.75
C4.5	98.01	67.27	75.09	98.64
Naïve Bayes	90.27	75.91	79.22	91.82
Logistic Regression	91.91	46.48	64.81	95.91
Perceptron	97.88	78.75	82.22	98.30
DBN	97.79	95.68	95.77	97.84
3layerNet	99.49	96.15	99.41	92.32
RNN	96.72	91.17	96.78	97.61

In table 1, we see that our deep learning model produces better results in comparison to traditional machine learning models by significant amount. Next we present the experimental results of our model by adjusting its parameters and we'll see which combinations of values of parameters can give us better results in terms of precision and recall.

Table 2. Precision and recall with different deep learning model parameters.

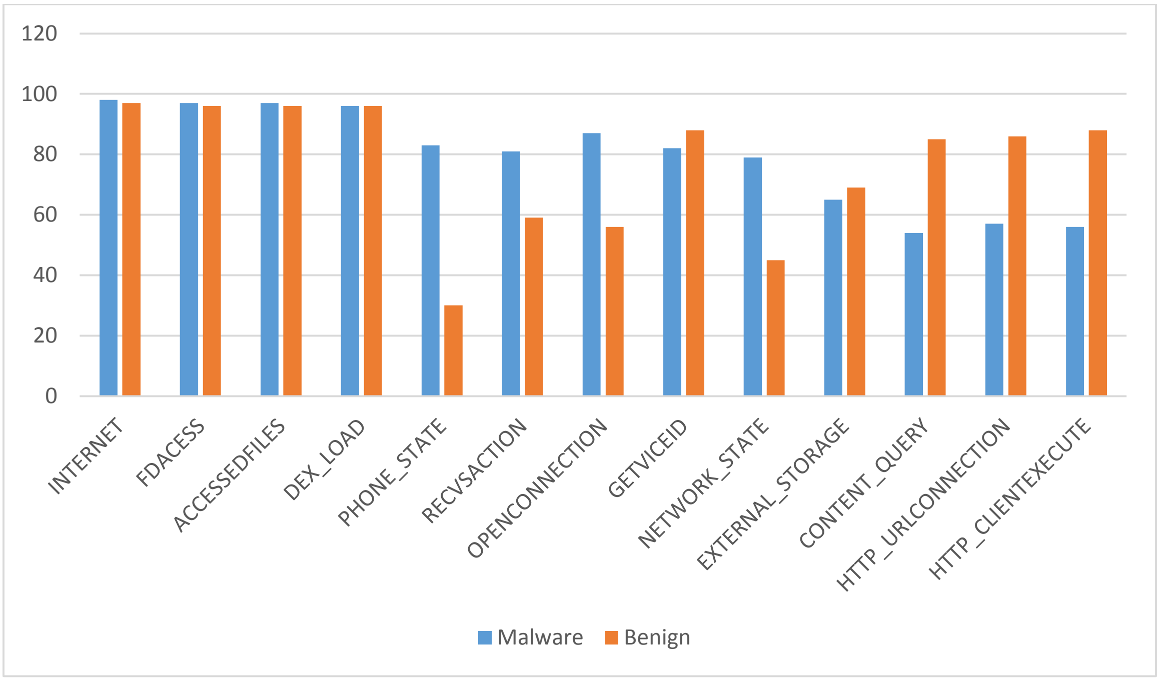
Number of Layers	**Alpha parameter**	**Benign Apps**		**Malicious Apps**	
		Precision (%)	Recall (%)	Precision (%)	Recall (%)
6	0.001	97.45	95.68	95.76	97.50
5	0.001	97.93	91.36	91.91	98.07
4	0.01	93.25	95.68	95.58	93.30
3	10	97.22	97.68	95.17	97.63
2	100	97.23	95.68	95.55	97.27
2	**100**	97.96	95.65	95.77	97.84
1	100	97.85	92.68	95.77	97.84

In table 2, we see that our deep learning model has the best performance when we set alpha parameter to 100 and when the number of layers are equal to 2. From the table, we can clearly deduce that the deep learning model has much better performance than any machine learning models such as C4.5, Naïve Bayes, Logistic Regression, SVM and Perceptron.

Moreover, it is worth noticing that machine learning models are noise tolerant during training models and we may conclude that the few instances of noise in the training data of hidden malware in benign app dataset must have had some influence on the performance of our models.

Next in Figure 4, we present the top-ranked features in our dataset. It is worth noticing that our in-depth analysis of the app permissions dataset can give us a clear picture of the impact of various permissions in deciding whether an app is malware or not can be more than that of certain other permissions in the dataset. So, we the ratio of top-ranked features in the dataset.

Figure 4. Top-ranked permissions in the app permissions dataset.



6. Conclusion

Deep learning as it seems, is a new area of machine learning. It has gained massive popularity in the recent years due to its unmatchable accuracy of all the models trained using deep learning. It can also be employed to build powerful learning models to solve problems in almost every sphere of real-world life. It has been used in various areas already like recommendation, computer vision, text classification, medical problems to help develop new vaccines of incurable diseases and many more. We used deep learning techniques for characterisation and detection of android malware in Google play ecosystem.

In this paper we have proposed a deep learning which we used to characterize and detect android malware apps and as we've already seen it outperforms all the existing traditional models which are traditional machine learning models. The advantage, as we've already seen are pretty clear as our deep learning model performs better than all other machine learning models that have been used to characterize and detect android malware apps in the past.

In this study, we extracted a total of 153 app permissions from the app dataset extracted from various sources that were made available openly for the research purposes. We used static and dynamic app permissions of android apps and characterized malware using 3-layer Neural Network and Recurrent neural network (RNN). We evaluated our model with 20 000 benign apps dataset and 500 Contagio community malware dataset. The results have shown that with a deep learning model we are able to achieve classification accuracy and make pretty good prediction about whether an app is malware or benign. Also, it is important to note that recent study by ten popular antivirus software indicated that it is a matter of sheer urgency to make appropriate viable changes in Android malware detection.

There were lot of challenges that we go through while implementing our approach. Manipulating the dataset, changing the training set and testing set and analysing the result, were few challenges and obstacles that we faced. Dealing with this huge dataset was the toughest job. But eventually, we overcome all those challenges and successfully concluded our study of android characterization and detection using popular deep learning techniques.

Future Work

There is always a room for improvement and deep learning is such a vast topic for research that still many initiatives need to be taken to make it better. Due to the time constraints we were only able to implement the proposed algorithms and techniques as described before. We hope that we get opportunities in future to implement other deep learning techniques in combinations with recent machine learning algorithms to our implemented work in order to produce more accurate and precise results. We are hopeful that we can produce better results.

The future work includes decrease the runtime and processing of such a huge dataset within memory on faster GPU machines as deep learning techniques are highly computational intensive in particular. More fine grained features for app datasets can be extracted and used appropriately to train a deep learning model for better detection rate of that model. We may also add the semantic-based features which carry significant knowledge about each apk. This will improve overall richness of the app features dataset. In addition to that, we may also introduce discrete values for each feature rather than binary values for increase the richness of the dataset.

References

[1] DroidDetector: A deep learning based Android malware detection engine, http://analysis.droid-sec.com, 2015.

[2] Contagio mobile malware dump, http://contagiodump.blogspot.com, 2015.

[3] Bouncer: Android and security, https://googlemobile.blogspot.com/2012/02/android-and-security.html, 2015.

[4] J. H. Friedman and N. I. Fisher, Bump hunting in high dimensional data, Statistics and Computing.

[5] Siegfried Rasthofer, Steven Arzt, Max Kolhagen, Brian Pfretzschner. DroidSearch: A Tool for Scaling Android App Triage to Real-World App Stores

[6] Nicolas Viennot, Edward Garcia and Jason Nieh. A Measurement Study of Google Play

[7] N. Jones, The learning machines, Nature.

[8] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, Profiledroid: Multi-layer profiling of Android applications.

[9] Justin Sahs and Latifur Khan University of Texas. A Machine Learning Approach to Android Malware Detection

[10] Naser Peiravian, Xingquan Zhu. Machine Learning for Android Malware Detection Using Permission and API Calls

[11] Android malware genome project, http://www.malgenomeproject.org, 2015.

[12] DroidBox: An Android application sandbox for dynamic analysis, http://www.honeynet.org/gsoc2011/slot5, 2015.