A Machine Learning approach to detect Malicious Payloads in Android APKs

Mr. Prashanth Ch[1], Dr. Gnanaprakasam Thangavel[2], Dr. Ajaya Kumar Akasapu[3]

[1,2,3]*Gayatri Vidya Parishad College of Engineering (Autonomous), Vishakhapatnam*

prashanth.deva.777@gmail.com[1], gnanagvp@gvpce.ac.in[2], ajaykumar@gvpce.ac.in[3]

Abstract

Smart Devices are become mandatory in human's day to day life. According to statisica.com, in India around 91% mobile users are Android users in 2019. There are three security mechanisms are maintained by the company and the customer. 1.Device Protection, 2. Data Protection and 3. Application Management. In which, Application Management is the difficult process by the company and the naïve users of the Android. In this research paper we are proposing a hybrid approach to detect the malicious payloads. This will be highly useful for the Android users. Based on the proposed approach, first we use the SVM (Support Vector Machine) to classify the payload types. Secondly, we use the LSTM(Long-Short Term Memory) method to detect the malicious payloads. The experimental results proved the hybrid approach solves the payload detection in a easy way.

Keywords: *Android, Application Management, Malicious, Machine Learning Approach.*

1. Introduction

In the cyber era, smart-phones are every important thing in human life. They store a lot of personal information. Android operating system has reached 15 years of rapid development and most popular operating system in mobile devices. There are millions of application on android market in this market users can easily download and install their devices but this application some are secure , some are malicious developers are upload malicious apps in android market try to steal sensitive data or damage the mobile device. The attack vectors of malicious payload are malware such as viruses, worms, Trojan horses, programming flaws etc., Some other types are intentional and accidental coding flaws, logic bomb and backdoor. Every malware have a malicious payload which is quietly occupies running in background and send information to workspace. In this research paper we focus on APK (Android Package Kit) files which are malicious or not classified by the machine learning algorithms. Machine learning and Machine Intelligence are taking cyber security and other tech fields by storm, and you can easily find a great deal of information on the use of ML. To predict the malicious payloads in APK files first we need to decompile the APK by using 'apktool' and extract the features from AndroidManifest.xml files to get permissions of apk file and disassembled the classes.dex codes to get string API calls.

2. Related Work

The analysis and detection of malicious payloads the research community deals with many approaches. The approaches are static model (API Calls, CFG, and Opcode),

dynamic model (Function Call, Function Parameters, and Instruction Traces) and hybrid model analyses. The following Figure 2.1 shows the models of malware analysis.

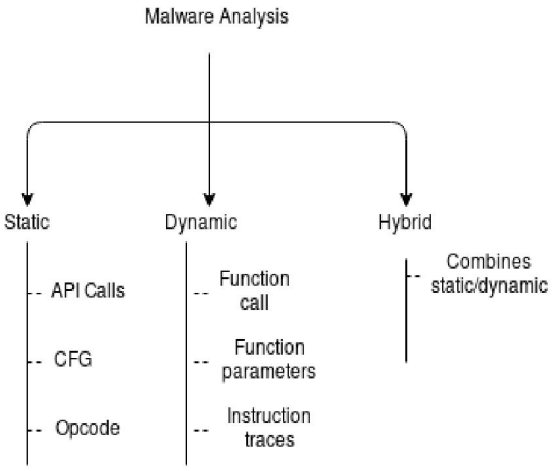

Figure 2.1Malware Analysis

2.1 Static Analysis

Static analysis follows the method of reverse engineering; it examines the code without executed the android application. The static analysis provided an understood of the code structure detecting the malware. The extraction of static analysis androidmainfest.xml and classes.dex byte code in APK files. AndroidManifest.xml is an important file in APK which deals with service, activities, version type, metadata, and permissions are used to coordinate the application. When the binary classes.dex byte code is disassembled into the source code, a suspicious string API calls CFG (Control Flow Graph). The string API calls are coordinating with all system kernels which communicate with all departments

2.2 Dynamic Analysis

Dynamic analysis adopts the opposite approach and is executed while a program is in operation and it is a popular technique in computer programs [2]. Dynamic analysis is also a method called behavioral analysis which traces the malicious behavior and system calls of android application during execution in the virtual environments. The dynamic analysis also traces software applications at runtime and captures data that can be used to analyze and identify malicious payloads. In dynamic analysis, we need to debug the functionality of the malicious executable files and analyze the classes, methods and etc. Dynamic analysis deals with function call, function parameters, intrusion trace and flow vetDroid construct and rebuild the malicious and sensitive behaviors of android applications.

2.3 Hybrid Analysis

Hybrid analysis is a combination of static and dynamic analysis used together with the suspicious behavior at runtime of the application and the static byte code tools are used for the analysis. The static analysis extraction of APK in permission and classes, methods, etc., The dynamic analysis analyze the function calls and function traces the hybrid analysis activities of monitoring via runtime of the applications.

3. Methodology

As we know that machine learning is sub branch of AI and aimed to learn the new behaviors of empirical data. At present, computerized world machines are battling machines and propelled assailants and criminal gatherings are creating complex better approaches to propagate their missions. The risk today isn't only the great situations of information burglary or a hacked site, yet the quiet danger hiding underneath the surface. These assailants hush up, sneaking in unannounced and clandestinely changing information voluntarily, or introducing off buttons fit to be enacted.

Using custom code, only crossing the perimeter boundary once and never sending information outside, such threats are almost impossible to find. In this classic scenario attacks to detected as fast as in machine learning algorithms and in this research focuses on apk files to extracting parameters Androidmainfest.xml and classes.dex byte-code disassembled in to the corresponding source code suspicious string API calls, android activities, classes and methods, CFG (Control Flow Graph).

3.1 Process Flow

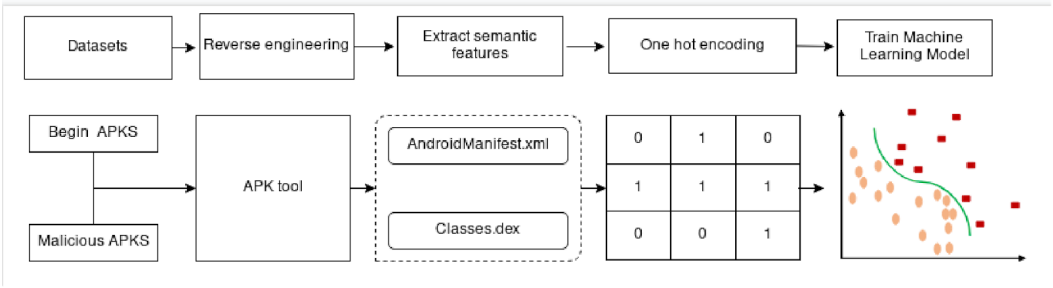

Figure. 3.1 Process Flow

3.2 Data Sets

For machine learning approach, the data sets are collected from contagion mobiles and android malware samples. If duplicate data set appears, we need to remove the redundant data set and keep the original data sets.

source1 U source2 U . . . . . source N

The begin dataset are from apkpure, apkmirror, google play store collecting and construct the dataset. This datasets are going to be use in this research workspace.

3.2.1 Details of Malware APKs

Table 3.1 Malware APKs

Class	N Samples	Class	N Samples
Adware_beauty	13	Triada	12
Candy_corn	18	Motion_dectation	53
Benews	23	Xbot	33
Debdroid	9	Fraud_apps	43
Malbus	29	Desencrypt	4
Farseer	20	Fake_updates	12
Rouge_skype	29	Plankton	6

Aples	6	Erop	10

3.2.2 Details of begin APKs

Table 3.2 Begin APKs

Class	N Samples	Class	N Samples
Music player	23	Puzzle games	38
System tools	47	Books	25
Photos	17	Travelling apps	55
Wallpapers	5	Social media	10
Shopping	33	Entertainment	13
Food apps	19	Health and fitness	49
Communication	12	Arcade	11
Video Calling apps	15	Productivity	3

3.3 Feature extraction and embeddings

The exploration work plays out the figuring out of malware tests and start tests. The semantic highlights are parsing from the dismantled paired classes.dex codes and the Androidmainfest.xml records. Each android application has exercises, consents, capacities, classes, techniques that all contain fundamental data of the application. we are extricating this data to portray each malware test and start tests. Android applications are formed into the java programming language and afterward aggregated into classes.dex byte-code for its execution in the Dalvik virtual machine and Androidmainfest.xml document have consents and exercises. The .xml documents are allowed to get to the information from different classes and techniques. The classes.dex byte-code contains the complete semantic information about the string API calls and information access inside the application. In order to train a machine learning model, we extract all features like string apis and permission by using the androguard module. Androguard is used to analyze the apk file and extract the information like methods, classes, string api calls from Androidmainfest.xml, and classes.dex files And extracting the permission and string APIcalls embedded into one-hot encoding. We apply the one hot encoding to each malware test and start a sample,then we map every one of these highlights to a joint component vector space. For each element fi, on the off chance that it is introduced in x, at that point the I-th measurement highlight esteem is set to 1, in any case, the relating measurement including esteem is 0. An enormous number of API calls, consents, and different highlights the one-hot encoding inserting strategy will create high dimensional vector the item highlights can be put away into csv, csc, and pickle format.

4. Experimental analysis

The research problem is done to build a classification model, to evaluate how good prediction the model of begin and malware apks. They are some metric which helps to improve our models

-TP: the size of begin apks samples being correctly predicted as begin apks.

-TN: the size of malware apks samples being correctly predicted as malware apks

-FP: the size of begin apks samples being incorrectly predicted as malware apks.

-FN: the size of malware apks samples being incorrectly predicted as begin apks.

-Precision: the size of correctly identified malware apks over the size of correctly identified malware apks samples and incorrectly identified malware apks.

$$P = TP/(TP+FP)$$

-Recall: the size of correctly identified malware apks over the size of correctly identified malware apks samples and incorrectly identified begin apks samples

$$R = TP/(TP+FP)$$

-F1-Score: Accuracy can be used class distribution is similar f1 score is better metric when they are imbalanced the classes f1 score is a combination of TN.TP,FN and FP it can reflect the prediction effectiveness of the classifiers

$$F = 2\ P * R / P + R$$

4.1 Training Model and Prediction

An approach to find the malicious payloads the collection of begin and malware apks samples datasets is class-imbalanced however the imbalanced rate is high. At present traditional approach machine learning classifiers can work well in this scenario. In this research we employ the classical kernel support vector machine deep neural network LSTM to train the prediction models

The SVM kernel has been used in many cyber-security related classification tasks and features in predicting the malicious payloads. The SVM kernel have higher dimension to mapping the kernel methods data as higher dimensional space so that it become linear separable. The kernel SVM has tricks. In this trick we generalized the polynomial kernel. The polynomial kernel is a kernel function commonly used with support vector machines; the polynomial kernel looks not only at the given features of input samples . The training dataset is based on beginning apks and malware apks. The training dataset had so much overlap we were unable to find a satisfying support vector classifier to separate the malicious payloads . However the given each point y-axis coordinates by squaring the original dataset measurement we used sum with a polynomial kernel to and the found a good support vector classifier based on the high dimensional the polynomial kernel uses look like

$$K(x, y) = (x^{\mathsf{T}} y + c)^d$$

Where x and y are the vector in the input space ,i.e. Vector of features completed from training or test samples c>0, ‘c’ determines the coefficient of the polynomial and ‘d’ sets the degree of the polynomial . The polynomial kernel to classifier the AndroiMainfest.xml files to detect the malicious permission in APK files we employ the open source python machine learning package Scikit-learn to train the model and predicted the results

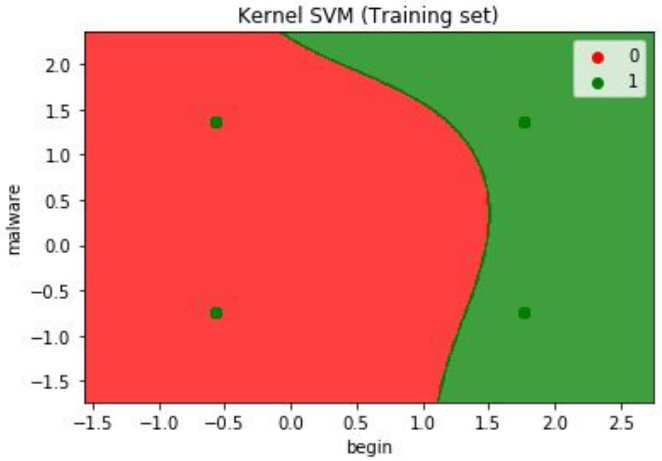

Figure 4.1 Visualization result of the training Androidmanifest.xml

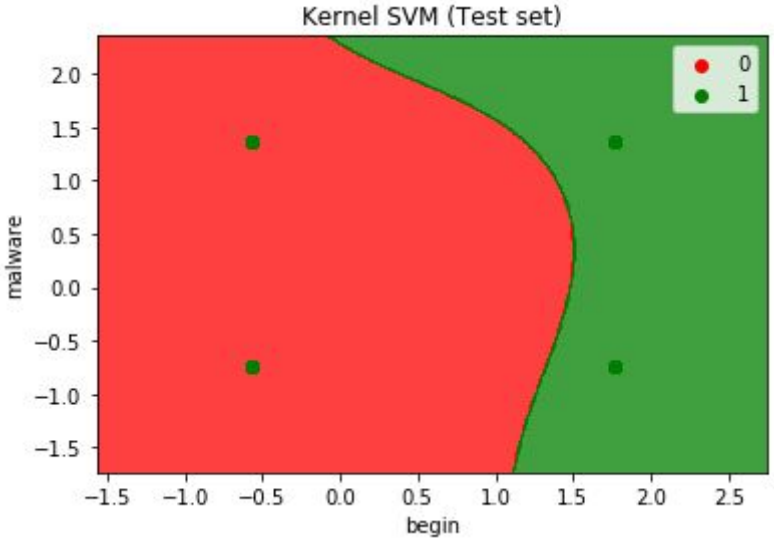

Figure 4.2 Visualization result of the testing Androidmanifest.xml

The LSTM (Long Short-Term Memory) has been used to achieve many great successes in various applications, stack prediction, machine translation, weather prediction and cyber security related areas LSTM is a part of deep neural network and sub set of RNN. Due to the vanishing gradient problem RNN'S effectiveness is limited when it needs to go back deep in to the context there is no finer control over which part of the context needs to be carried forward and how net of the past need to be forgotten.

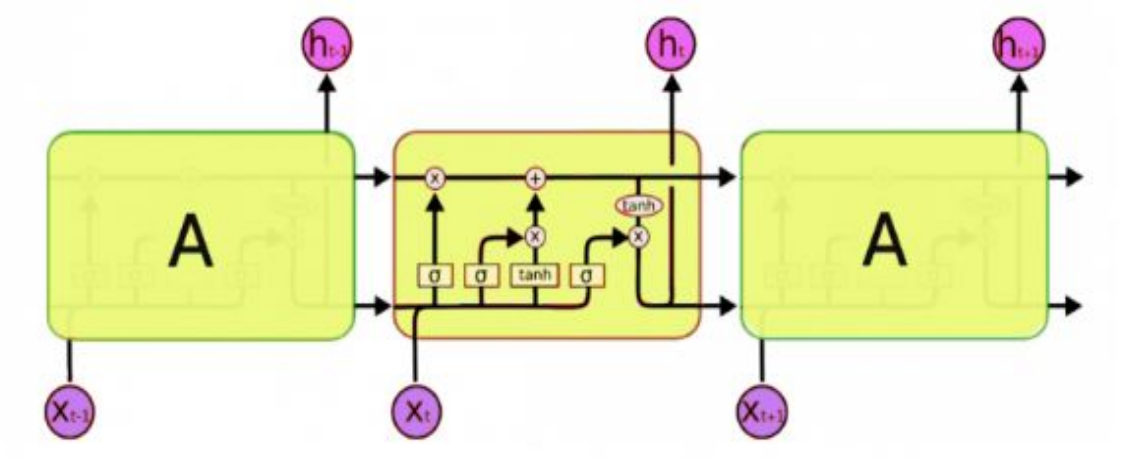

Figure 4.3 LSTM Architecture

The LSTM has input, output and hidden layers. The hidden layer replacing the gradient unit Atypical LSTM network consists of different memory blocks called cells. The memory blocks are responsible for remembering things and manipulation to this memory is done through major mechanisms called working gates. A forget gate is responsible for removing information from the cell state, the information that is no longer required for the LSTM. This gate takes two inputs h_t-1 and x_t. H_t-1 is the hidden state from the previous cell as the output and x_t is the input. The sigmoid function output vector output from the sigmoid function is multiplied the cell state.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Input gate adding the information to the cell state is responsible of input gate cell state by involving a sigmoid function this is very similarly to forget gate acts filter as h_t-1 and x_t the vector contain all possible values added in cell state and used tanh function output value from -1 to +1

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

The out-gate is useful the information from the current cell state before applying the tanh function to the cell satte create a vector and regulate the values to range -1 to +1 and filter the values h_t-1 and x_t and again filter the sigmoid function multiplying the values of this regulatory filter to the vector created in step1, and sending it out as a output and also to the hidden state of next cell and both the update gate as well as forget gate have a 0 and 1 values

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

5. Result Analysis

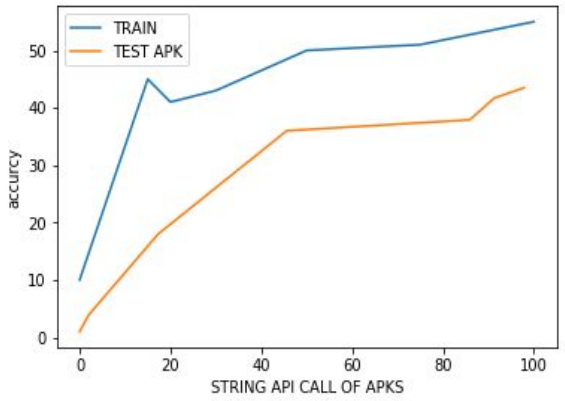

Figure 5.1 Visualization result of the training and testing classes.dex

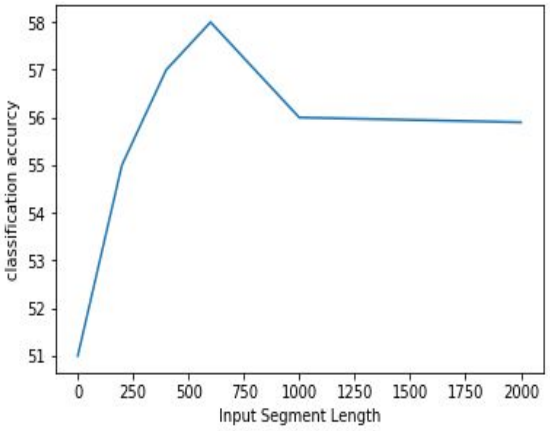

Figure 5.2 Visualization of Input Segment length(API calls) classes.dex

Table 5.1 Prediction results of SVM Kernel and LSTM

Name	Precision	Recall	F1-Score
SVM Kernel	0.91	0.89	0.89
LSTM	0.96	0.97	0.97

The above Table 5.1 shows the prediction performance of SVM Kernel and LSTM using features of both classes.dex and AndroidMAinfest.xml. Aggregating more features should produce better prediction performance. SVM also shows some better results to identify malicious payloads. However, employing more features also implies it costs more expensive resources time, consumption, memory consumption in reverse engineering. In reality, the security practitioner should balance the tradeoff between accuracy and efficiency.

6. Conclusion

Android malware prediction is going to be easy hereafter. The result of the precision, recall and F1-Score the malware applications are identified easily. The application management task is now hassling free. The proposed model used the machine learning algorithms like support vector machine and LSTM given major support to predict the malware applications. Future researchers can do this exercise for Apple (ios) applications and other mobile platforms.

References

[1] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, Djedjiga Mouheb, "MalDozer: Automatic framework for android malware detection using deep learning", Proceedings of the Fifth Annual DFRWS Europe, S48-S59.

[2] Fabio Martinelli, Fiammetta Marulli, Francesco Mercaldo, "Evaluating Convolutional Neural Network for Effective Mobile Malware Detection", International Conference on Knowledge Based and Intelligent Information and Engineering Systems, (2017), pp. 2372-2381.

[3] Long Nguyen-Vu, Jinung Ahn, Souhwan Jung, "Android Fragmentation in Malware Detection", Journal of Computers & Security, (2019), pp. 1-10.

[4] Karim O. Elish, Xiaokui Shu, Danfeng (Daphne) Yao, Barbara G. Ryder, Xuxian Jiang, "Profiling user-trigger dependence for Android malware detection", Journal of Computers & Security, (2019), pp. 255-273.

[5] Md. Shohel Rana, Andrew H. Sung, " Malware Analysis on Android Using Supervised Machine Learning Techniques", International Journal of Computer and Communication Engineering, vol. 7, no. 4, **(2018)**, pp. 178-188.

[6] Vinod P, Akka Zemmari, , Mauro Cont, "A machine learning based approach to detect malicious android apps using discriminant system calls", Future Generation Computer Systems, **(2019)**, 333-350.

[7] Wang, Wei; Zhao, Meichen; Gao, Zhenzhen; Xu, Guangquan; Xian, Hequn; Li, Yuanyuan; Zhang, Xiangliang, " Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions", Special Section on Security and Privacy for Cloud and IoT, vol. 7, no. 1, **(2020)**, pp. 67602-67631.

[8] Yerima, S. Y., Sezer, S., Muttik, I. " Android Malware Detection Using Parallel Machine Learning Classifiers", 8th International Conference on Next Generation Mobile Apps, Services and Technologies, **(2014)**, pp. 1-6.