Indian Institute of Technology, Kanpur
Department of Computer Science and Engineering

CS658A: Topics in Malware Analysis and Intrusion Detection

Project Report

Android Malware Detection

Supervised By: Prof. Sandeep Shukla
Academic Year 2021 - 2022

Team Name: CYBER_KNIGHTS

Abhishek Dnyaneshwar Revskar (21111003)
(abhishekdr21@iitk.ac.in)

Akash Gajanan Panzade (21111006)
(akashp21@iitk.ac.in)

Deepak Kumar (21111023)
(dkumar21@iitk.ac.in)

Mayuresh Diwakar Shandilya (21111041)
(mayuresh21@iitk.ac.in)

Prajwal Pradiprao Thakare (21111047)
(prajwalt21@iitk.ac.in)

Mandar Kishor Dhake (21111405)
(mkdhake21@iitk.ac.in)

Keywords: Android Malware, Malware Classification, Static Analysis,Machine Learning, Neural Network, Android Security, APK Reverse Engineering

Contents

1 Abstract

Malware targeting mobile devices is a pervasive problem in modern life. The detection of malware is essentially a software classification problem based on information gathered from program analysis. We are focusing on classifying the Android applications using various properties like Android Permissions, API calls and OPcode sequences used by the application. These properties are captured using some open source APK reverse engineering tools like Androguard ,APKtool etc.

Identifying malware is essentially a software classification problem. There are two main steps in the process of analyzing and classifying an application. First, through program analysis, various descriptions of a program are generated and features are extracted from the program descriptions. Then some machine learning models can be constructed to perform classification based on these features. Due to the sheer number of apps that need to be processed, automated software analysis and classification is necessary.

Our aim was to use some of the major properties of an APK like Android Permissions, API calls and OPcode sequences as features to train several machine learning and deep learning models. We have analysed the accuracy of these models for the test data and it gave us some promising results. The models were performing very well on the new and unseen APKs.

2 Data Sources and Data Extraction

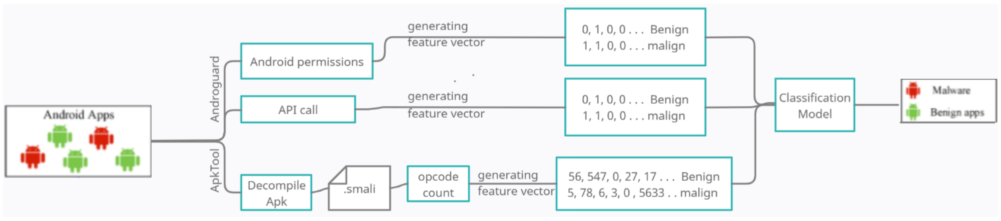
We have gathered the samples of the malicious as well as the benign APKs from Virustotal and Androzoo. After getting the sample of the APKs, we have extracted the features of these APKs which we will be using to train the machine learning and deep learning models. We mainly focused on extracting Android Permissions, API calls and OPcode sequences as features to train the models. All of our models will be the classification models since we are classifying the APKs into either malicious or benign.

All of the necessary files for an Android application are bundled into an APK file archive. Each APK file typically contains the following items: classes.dex file, AndroidManifest.xml file, lib folder etc. The most relevant components for our work are the classes.dex and Androidmanifest.xml files. The Androidmanifest.xml file contains the android permissions.The classes.dex file contains the bytecode for the application and is the main source of executable code.We use the open source tool Androguard to obtain a smali code representation of the classes.dex as well as the Android Permissions from Androidmanifest.xml for our analysis.

3 Methodology

There are two types of analysis that can be done on the android applications namely static and dynamic. The static methods are based on the binary analysis of the source code, a fairly widespread anti-malware approach due to its lightness, efficiency, and the ability to proactively detect malware before it gets installed.The dynamic analysis on the other hand, is about analyzing the behavior of running applications, in other words, collecting the characteristics of malicious activity by running the applications in a sandbox environment.

In our system we have used static analysis of android applications by examining the essential features like permissions, API calls and opcode sequences. We have extracted these features by using some open source apk analyzer tools like androguard and apktool. Androguard helped us to gather the permissions and the API calls where as using apktool we have gathered opcode sequences for the APKs.



3.1 OPcodes

In our system, the preprocessing of an application consists of disassembling the application and extracting opcode sequences for static malware analysis. An Android application is an apk file, which is a compressed file containing the code files, the AndroidManifest.xml file, and the application resource files. A code file is a dex file that can be transformed into smali files, where each smali file represents a single class and contains the methods of such a class. Each method contains instructions and each instruction. consists of a single opcode and multiple operands.After extracting the opcode sequence from each method, discarding the operands we obtain all the opcode from all the classes of the application. The opcodes are then counted and their respective counts are stored along with them.

3.2 API calls

System-API calls describe how an app interacts with the underlying Android OS. Such interactions are essential for an app to carry out its tasks, hence providing fundamental information on an app's behaviors.

The API calls are present in classes.dex file as follows: android. accessibilityservice.AccessibilityServiceInfo.getCanRetrieveWindowContent. As the first prefix 'android.accessibilityservice' is same for all the API calls we can remove it and yet also distinguish between them. So after chopping the common part we get API call as follows:
AccessibilityServiceInfo.getCanRetrieveWindowContent.
DalvikVMFormat() method is used to get classes.dex file. app_dex.get_classes() method is used to get classes from the classes.dex file. app_dex.get_methods() method is used to get API calls of each class. In this way, we will collect all the API calls from the smali files and stored their count respective count to get the final dataset for the API calls.

3.3 Android Permissions

This malware detection system targets identification of insecure permissions existing in mobile applications. In that respect, our project implements a static analysis-based approach by focusing on the manifest file (AndroidManifest.xml). The manifest file consists of specific permissions customized towards Android application development. Once, development is done, these permissions cannot be changed.

An optimisation approach is utilised to select the most suitable permissions to detect the malware. Around a total of 597 permissions are considered here. The manifest file features are extracted for a particular application and one-hot vector for that application is designed. Then, the classification process is worked out on several applications. Our project attempts to evaluate the efficacy of the static analysis technique by using the machine learning approach and using the application's permission noted in the manifest file. To get permissions from AndroidManifest.xml file we use get_permissions() method.

3.4 Model Training and Evaluation

The next step after extracting the required features is to train the machine learning models for the classification task. For each of the above three features we have trained the SVM, Random forest and ANN classification models.

For each APK sample, we have represented it using a boolean vector where columns represents the permissions that the application requires for training the model based on permissions. The model based on API calls uses the same representation where columns are the API calls that the application invokes while running. For the opcode setting each APK is represented using the opcode count. All these representations are saved as a CSV(comma separated values) files which are used to train the models.

After the models are trained on the given dataset, we measured the precision, recall, f1-score and accuracy. It came out as following:-

3.4.1 Permissions Based Models

Model	Precision	Recall	F1-Score	Accuracy
SVM	0.891	0.890	0.890	89.2%
Random Forest	0.910	0.912	0.910	91.3%
ANN	0.883	0.881	0.881	88.1%

3.4.2 API-Calls Based Models

Model	Precision	Recall	F1-Score	Accuracy
SVM	0.94	0.93	0.93	94%
Random Forest	0.92	0.91	0.91	91%
ANN	0.92	0.92	0.92	92.5%

3.4.3 OPcodes Based Models

Model	Precision	Recall	F1-Score	Accuracy
SVM	0.894	0.892	0.893	89.3%
Random Forest	0.902	0.900	0.899	90%
ANN	0.904	0.902	0.902	90.25%

We can see from the above figures that all the models based on all the three features have performed pretty well on the test dataset.

4 Analysis

All these trained models are tested on an unknown dataset which is a collection of benign and malign APKs of year 2018 taken from the C3i Centre, IIT Kanpur. For this dataset we are getting pretty good accuracies which are as follows:-

Model	Permissions Based	OPcode Based	API-calls based
SVM	78.0%	85.0%	88.0%
Random Forest	80.5%	87.2%	86.0%
ANN	77.2%	88.9%	83.9%

From this, we can say that the systems can work considerably better on an alien data.

5 Limitations

- As any learning-based approach requires samples and the machine learning models are data hungry, the more data we have,the more precise and robust our classifiers will be and the better the model can learn to generalize on the new and unseen data.
- Also, some evasive attacks are possible which protects the malicious APKs from being detected by the basic anomaly detectors.
- If the APKs contain the APIs which are in emerging phases, they can bypass the malware detection task.

6 Conclusion

To design our Android Malware Detection system, we analysed different features of the Android applications and trained a few machine learning and deep learning models to classify the given APK as malicious or benign. The first step was to extract the features from the available APK file using some open source reverse engineering tools for APK. Using these features, we trained a few classifiers to classify the APK. Finally, these trained classifiers were tested on a test dataset to determine whether or not they can perform well on the new and unseen APKs. The results showed that these models can be used to detect the malicious APK in most of the cases. The system is also showing considerably better accuracies on the unknown dataset which makes it effective for unknown malware detection task.

7 Future Scope

- This system can be made to use in android mobiles to keep the malicious applications away from the android devices.
- More balanced and diverse dataset can be used to improve the performance of the system.

8 References

1. https://ieeexplore.ieee.org/abstract/document/9065765
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7374318/
3. https://www.researchgate.net/profile/Win-Zaw-2/publication/288267309_Permission-Based_Android_Malware_Detection/links/59aac8850f7e9bdd114fb7b2/Permission-Based-Android-Malware-Detection.pdf
4. https://www.ripublication.com/ijaer20/ijaerv15n1_02.pdf
5. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=7966078
6. https://dora.dmu.ac.uk/bitstream/handle/2086/16947/Deep-Android-Malware-Detection.pdf?sequence=1isAllowed=y
7. https://core.ac.uk/download/pdf/160471635.pdf
8. https://ieeexplore.ieee.org/abstract/document/8871171
9. https://ieeexplore.ieee.org/document/8902627
10. https://ibotpeaches.github.io/Apktool/
11. https://github.com/androguard/androguard