A SEMINAR REPORT ON

ANDROID MALWARE DETECTION USING MACHINE LEARNING

SUBMITTED BY

Rishikesh A. Ksheersagar
Roll No : 303050

UNDER THE GUIDANCE OF
Prof. Mrs. S. Panicker

Department Of Computer Engineering
MAEERâ€™s MAHARASHTRA INSTITUTE OF TECHNOLOGY
Kothrud, Pune 411 038

2017-2018

MAHARASHTRA ACADEMY OF ENGINEERING AND EDUCATIONAL RESEARCHâ€™S

MAHARASHTRA INSTITUTE OF TECHNOLOGY PUNE

DEPARTMENT OF COMPUTER ENGINEERING

C E R T I F I C A T E

This is to certify that

Rishikesh A. Ksheersagar (303050)

of T. E. Computer successfully completed seminar in

ANDROID MALWARE DETECTION USING MACHINE LEARNING

To my satisfaction and submitted the same during the academic year 2017-2018 towards the partial fulfillment of degree of Bachelor of Engineering in Computer Engineering of Pune University under the Department of Computer Engineering , Maharashtra Institute of Technology, Pune.

Prof.Mrs. S.Panicker
(Seminar Guide)

Dr. Prof. Mrs.V.Y.Kulkarni
(Head of Computer Engineering Department)

Place: Pune

Date:

ACKNOWLEDGEMENT

I take this opportunity to express my sincere appreciation for the cooperation given by Dr. Prof. Mrs. V. Y. Kulkarni, HOD (Department of Computer Engineering) and need a special mention for all the motivation and support.

I am deeply indebted to my guide Prof. Mrs. S.Panicker for completion of this seminar for which she has guided and helped me going out of the way.

For all efforts behind the Seminar report, I would also like to express my sincere appreciation to staff of department of Computer Engineering, Maharashtra Institute of Technology Pune, for their extended help and suggestions at every stage.

Rishikesh A. Ksheersagar
(Roll no. 303050)

Contents

List of Figures

List of Tables

Abstract

Android OS experiences a blazing popularity since the last few years. This predominant platform has established itself not only in the mobile world but also in the Internet of Things (IoT) devices. This popularity, however, comes at the expense of security, as it has become a tempting target of malicious apps. Hence, there is an increasing need for sophisticated, automatic, and portable malware detection solutions. Reverse engineering of the Android Apps is done to extract manifest files, and binaries, and employ state-of-the-art machine learning algorithms to efficiently detect malwares. Existing Android malware analysis techniques can be broadly categorized into static and dynamic analysis.

Keywords:
Android Malware,Reverse Engineering, Machine learning, Security, Research.

Chapter 1

INTRODUCTION

In this report, some of the commonly used ways to detect malwares or malicious applications in android platform with the help of Machine Learning are stated. Further, one of the classification algorithm used in the process: Logistic Regression Classifier. This report starts with explaining the terms necessary to understand the concept and then covers 2 ways of malware detection which is followed by a description of the LR algorithm and the references in the end.

1.1 MALWARE

Malware, or malicious software, is any program or file that is harmful to a computer user. Malware includes computer viruses, worms, Trojan horses and spyware. These malicious programs can perform a variety of functions, including stealing, encrypting or deleting sensitive data, altering or hijacking core computing functions and monitoring users' computer activity without their permission. The point of nearly all malwares is to make money. [11]

1.1.1 Types of android malware

- **Ransomware**
  Type of malware that holds your device to "ransom" by locking it down so it cant be used until you pay the hostage-takers. Hit android in 2014. Svpeng is one type which combined ransomware and payment-card theft. For Russians (whom Svpeng was originally created to target) Svpeng would present a screen to input credit card details every time a user went to Google Play, which it would then send to the cybercriminal gang that created it. For people in the US and UK it would present itself as the FBI, locking down the infected device for supposedly having child pornography on it. The user would then have to pay a 'fine' in order to have the device released. Svpeng also checked to see if a banking app was

installed, though it is unclear what it did with that information. Russian Police arrested Svpeng's 25 year old creator earlier in April, after having stolen over 50 million rubles (930,000 dollars) and having infected over 350,000 Android devices.

- **Installation of apps without your consent**
  Some people have been opening links inside their apps without ever going to their browser app. This is done with a component called Webview. If you happen to be running Android 4.3 'Jellybean' or lower, there is something you need to know about.

  Apparently, there is a vulnerability that allows users to click on malicious links as they're browsing in Webview. The vulnerability is referred to as the Universal Cross-Site Scripting (UXSS) attack. It is letting users click on malicious links where the attackers are able to execute malicious codes using JavaScript.

  They are somehow able to get around the security that's there to protect the users. Once the attackers get this far, they can use the vulnerability and install any app they want onto your device. It has been said that Google does not plan on patching the vulnerability found in Android 4.3 and lower.

  So, if you are interested in not being a target for these attackers, you should upgrade and get the latest version of Android. This should be done as soon as possible. Alternatively, you could simply choose not to use the Webview to browse, and instead you can open your links in secure browsers, such as Dolphin, Firefox, and Chrome among many others

- **PowerOffHijack**
  It does this when you are shutting it down, letting you think your device shut down when it's actually being hijacked without you knowing. At this point, it is able to do things like taking pictures, making phone calls, and whatever else the hijackers want to, since no one is aware that anything wrong is going on.

  This is different from the first malware (ransomware) discussed earlier in this article. Android/PowerOffHijack only affects Android 5.0 and above, requiring root access to enable it to work.

  The total number of devices that have been infected, as of February 18, was approximately 10,000. However, unless you are in the habit of shopping in Chinese app stores, you shouldn't worry about this kind of threats.

- **Innocent apps hiding dormant malware**
  In February we learned that certain Android apps were giving their users more than they bargained for. A patience/solitaire game, an IQ test, and a history

app all sound innocent enough, don't they? And you would never expect they had a problem if they behaved as intended for a month before doing anything dubious, wouldn't you? However, each of these apps, which were downloaded more than five million times, had code in them that would trigger popups that, if clicked on, would lead to fake webpages, run illicit processes, or start unwanted app installs and downloads.

Filip Chytry of Avast Antivirus sheds light on the clue that tells you if you have this kind of malware:

Each time you unlock your device an ad is presented to you, warning you about a problem, e.g. that your device is infected, out of date or full of porn. This, of course, is a complete lie.

Google has suspended these apps from the Google Play Store, so as long as you don't download them from another source, you'll be okay.

- **Malware for Sextorsion**
  Cybercriminals in South Korea have created fake social media profiles of attractive women to lure people into cybersex, whom they then blackmail by threatening to release the video on YouTube.

  Here's where the malware comes in. The perpetrators are now pretending that they experience audio problems with the chosen software (like Skype) and persuade their victim to download an chat app of their preference. In truth, the chat app steals the victims contacts to send to the blackmailer. The criminal uses the contact information to extort money more effectively by threatening to share the video with the victim's close friends and family.

- **Android Installer Hijacking Vulnerability**
  Nearly 50percent of all Android devices are at risk of a vulnerability called "Android Installer Hijacking". Put simply, when you go to download a legitimate app, the installer can be hijacked allowing an app you didn't want to be installed in its place. This happens in the background while you are reviewing the permissions of the app you want to install, either by setting up the benign app to install malware later, or by masking the true permissions it requires.

  This vulnerability affects third party app stores, such as the Amazon App Store. Android devices 4.4 and higher are safe from this. [11]

1.2 MACHINE LEARNING

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explic-

itly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.[12]

1.2.1 Types of Machine Learning Algorithms

- **Supervised ML Algorithms**
  Can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

- **Unsupervised ML Algorithms**
  Are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

- **Semi supervised ML Algorithms**
  Fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.

- **Reinforcement ML Algorithms**
  Is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior

within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal. [12]

1.3 MALWARE DETECTION TECHNIQUES

Early detection is the most important thing to mitigate the harmful effects of malware. Throughout the years, a number of malware detection methods has been proposed. These can be broadly categorized into: signature-based, change-based, and anomaly-based methods.

- **Signature- based methods**
  A signature based intrusion method detects a malware based on its signature. It first gathers data and analyzes it, and when a program or file has a similar signature to an already existing malware (which it compares to from a database) it detects it. This method is often used for detecting popular malware signatures, but it can be quite slow since it should compare the signatures from a large database, meaning it cannot be instant.

- **Change-based methods**
  Change based detection is a method that identifies when changes occurred in the system. It relies on probability distribution to detect the changes. These techniques include, online and offline change detection techniques.

- **Anomaly-based methods**
  In the anomaly based system, a system administrator defines the baseline, or normal state of the network's traffic load, breakdown, protocol, and typical packet size. The anomaly detector monitors network segments to compare their state to the normal baseline and look for anomalies

Up until now, virtually almost all real-world deployments of malware detection (like virus scanners) are signature-based and change-based methods. Though efficient, these methods are not able to identify new types of malware (such as those carrying out zero-day attacks). Some research has been done on anomaly-based malware detection, which are good at identifying new malware. However, so far the anomaly-based methods are not widely deployed yet because of practical issues such as efficiency/scalability, high false positive rate, difficulty to use, etc.[3]

1.4 REVERSE ENGINEERING

Reverse engineering, also called back engineering, is the process of where a man-made object is de-constructed to reveal its designs, architecture, or to extract knowledge from the object. This process is similar to scientific research but the only difference is that scientific research is about a natural phenomenon. Reverse engineering is applicable in the fields of mechanical engineering, electronic engineering, software engineering, chemical engineering, and systems biology.[13]

1.4.1 Reverse Engineering for feature-extraction

To build up an efficient Android malware detection model, it is highly desirable to collect robust and most representative features such as user permissions, providers and receiver's information, intent filters, process name and binaries from under analysis applications. In the proposed framework, all the mentioned features are extracted via reverse engineering, breaking down its application APK in to simple java code and after modification it is again converted in to an APK file. In this context, Easy APK Disassembler is adopted to reverse engineering the applications. After reverse engineering the Android application, features are extracted that are constant strings from binaries and permissions, providers, receivers, intent filters, process name from Android manifest.xml files. Constant strings are actually binaries of Android applications that exist in folder after reverse engineering of an application. Attackers can make change in constant strings to attack a device by reverse engineering an application. As we get source code of an application after reverse engineering process, hackers make changes in constant strings, repack that application and upload on play stores. If an application has constant string [const-string v1, "Rooted Device"], malware attackers can change it to [const-string v1,"Device Not Rooted"]. Similarly, they can attempt multiple attacks by making changes in constant strings of Android applications. While every Android application has one manifest.xml file in which all permissions from users are requested. Users can't install application without accepting all requested permissions. Along with constant strings, keywords (manifest feature) are also extracted from applications.[6]

1.5 DATASET

A data set is a collection of related, discrete items of related data that may be accessed individually or in combination or managed as a whole entity. For Malware Detection using Machine Learning, we need to create a dataset of benign softwares/apps and malicious softwares/apps in order to make the machine learning classifiers learn

and deduce by itself wether or not a software/app is a malware or not. A dataset can be created individually by our own by putting some benign and some malicious apps. Or there are various ready-made datasets available for direct usage: eg. M0Droid,DREBIN, AndroMalShare etc.

- **M0Droid**

  M0Droid basically is android application behavioral pattern recognition tool which is used to identify android malwares and categorize them according to their behavior. It utilized a kernel level hook to capture all system call requests of the application and then generate a signature for the behavior of the application.[5]

- **DREBIN**

  The dataset contains 5,560 applications from 179 different malware families. The samples have been collected in the period of August 2010 to October 2012 and were made available by the MobileSandbox project.

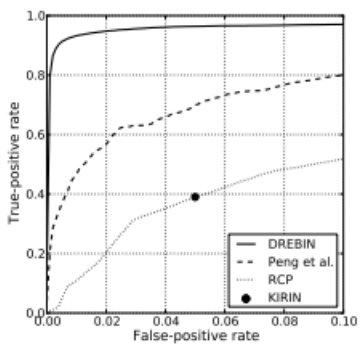
(a) The Malware Genome dataset.

Total apps	1260	100.00%
No. of unique certificates	134	10.63%
Representative apps	879	69.76%
Unique apps	379	29.61%
Unique apps with unique certificates	86	6.72%
Twins	290	23.02%
Siblings	91	7.22%
False Siblings	2	0.16%
Step-siblings	584	45.63%
Cousins	607	47.42%
False step-siblings	117	9.14%

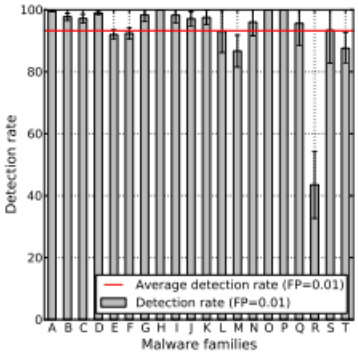
(b) Drebin dataset.

Total apps	5560	100.00%
No. of unique certificates	963	17.32%
Representative apps	3549	63.83%
Unique apps	1441	25.92%
Unique apps with unique certificates	681	12.25%
Twins	519	9.33%
Siblings	1332	23.96%
False Siblings	136	2.45%
Step-siblings	2365	42.54%
Cousins	2214	39.82%
False step-siblings	1386	24.93%

Figure 1.1: Details of a dataset



(a) Detection performance as ROC curve.



(b) Detection per malware family.

Id	Family	#	*Id*	Family	#
A	FakeInstaller	925	*K*	Adrd	91
B	DroidKungFu	667	*L*	DroidDream	81
C	Plankton	625	*M*	LinuxLotoor	70
D	Opfake	613	*N*	GoldDream	69
E	GingerMaster	339	*O*	MobileTx	69
F	BaseBridge	330	*P*	FakeRun	61
G	Iconosys	152	*Q*	SendPay	59
H	Kmin	147	*R*	Gappusin	58
I	FakeDoc	132	*S*	Imlog	43
J	Geinimi	92	*T*	SMSreg	41

(c) Top malware families in our dataset.

Figure 1.2: Drebin detection rate

- **AndroMalShare**

  AndroMalShare is a project focused on sharing Android malware samples. It's

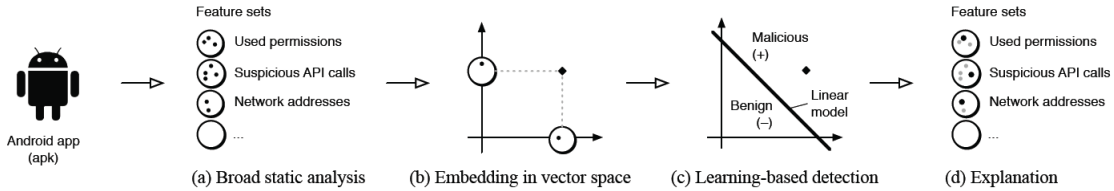


Figure 1. Schematic depiction of the analysis steps performed by Drebin.

Figure 1.3: Analysis steps performed by Drebin

only for research, no commercial use. Present statistical information of the samples, a detail report of each malware sample scanned by SandDroid and the detection results by the anti-virus productions.[6]

- **AMD Project**
  AMD contains 24,553 samples, categorized in 135 varieties among 71 malware families ranging from 2010 to 2016. The dataset provides an up-to-date picture of the current landscape of Android malware, and is publicly shared with the community.[14]

1.6 CLASSIFIER

A Machine Learning algorithm/Mathematical function that maps input data to a category is called a Classifier. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

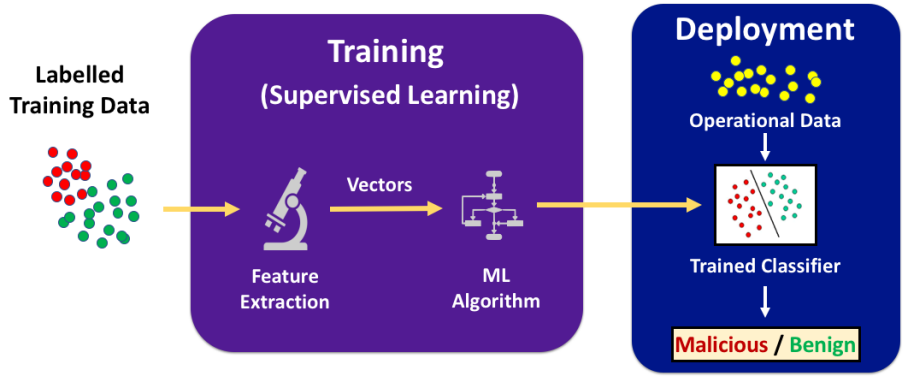


Figure 1.4: Learning process

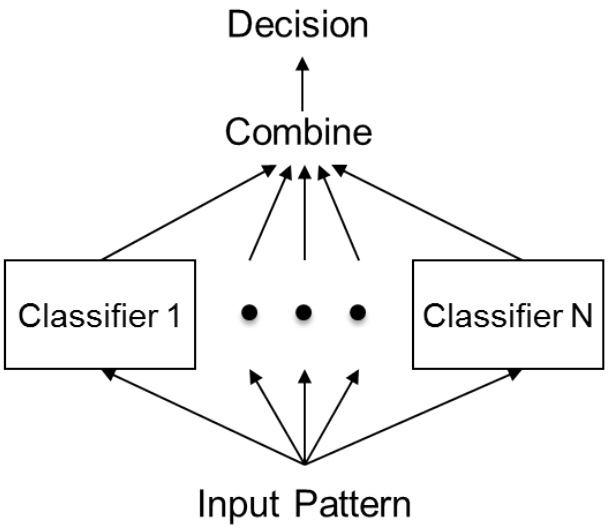


Figure 1.5: Classifiers use

The output of a classifier is a class. In this case the output of the applied classifier will be whether a given application is malicious or not(benign).[15]

Chapter 2

LITERATURE SURVEY

Table 2.1: Literature Survey

Sr. no	Name of Paper	Authors	Description	Algorithms used
1	Malware Detection in Android Mobile Platform using Machine Learning Algorithms	Mariam Al Ali, Davor Svetinovic, Zeyar Aung, Suryani Lukman 2017	A practical and effective anomaly based malware detection framework is proposed with an emphasis on Android mobile computing platform. A dataset consisting of both benign and malicious applications (apps) were installed on an Android device to analyze the behavioral patterns. Various ML algorithms are then applied on this dataset to classify the application as malicious or benign.	Behavior based techniques, LR Classifier, KNN Classifier, Naive Bayes
2	Lightweight Malware Detection based on Machine Learning Algorithms and the Android Manifest File	Monica Kumaran and Wenjia Li 2016	Study aims to learn if the Android manifest file provides enough information to classify an app as malicious or benign. In particular it compares the efficacy of using requested permissions versus inter-app intent communication. It also improves static malware detection by comparing and refining different machine learning algorithms on the manifest file dataset.	Signature based technique, LR algorithm, Naive Bayes, KNN, Decision Tree

Table 2.2: Literature Survey - continued

Sr. no	Name of Paper	Authors	Description	Algorithms used
3	Machine learning aided Android malware classification	Nikola Milosevic , Ali Dehghantanha , Kim-Kwang Raymond Choo 2017	Present two machine learning aided ap- proaches for static analysis of Android malware. The first approach is based on permissions and the other is based on source code analysis utilizing a bag-of-words representation model.	Static techniques, LR algorithm
4	Detection of Transformed Malwares using Permission Flow Graphs	Ridhima Seth and Rishabh Kaushal 2017	A lightweight static Permission Flow Graph (PFG) based approach to detect malware even when they have been transformed (obfuscated).	Static techniques, Permission flow graphs
5	Machine learning-assisted signature and heuristic-based detection of malwares in Android devices	Zahoor-Ur Rehman , Sidra Nasim Khan , Khan Muhammad , Jong Weon Lee , , Zhihan Lv , Sung Wook Baik , Peer Azmat Shah , Khalid Awan , Irfan Mehmood 2018	An efficient hybrid framework is pre- sented for detection of malware in Android Apps. The proposed framework considers both signature and heuristic-based analysis for Android Apps. We have reverse engineered the Android Apps to extract manifest files, and binaries, and employed state-of-the-art ma- chine learning algorithms to efficiently detect malwares.	Hybrid framework, LR classifier, Naive Bayes

Chapter 3

RELATED WORKS

Every day, approximately 1.3 million Android devices are being activated according to Google Chairmen Erich Schmidt . Android provides their users a rich media support, optimized graphic system and powerful browser. Apart from this, Android OS also provides support for 24 h GPS tracking, video camera, compass and 3D-accelerometer. It yields rich Application Pro- gram Interfaces (APIs) for location and map functions. Users can easily control or process Google map on Android devices and access location at low cost. Due to the high usage of Smartphone's, every individual user is exposed to the threat of unwanted and malicious applications. Malware authors are busy in writing malicious applications with an increase in the number of Android users. The recent research illustrated that Android Apps are repacked by malicious ELF binaries for hid- ing calls to external binaries. Similarly, researchers are trying to find out the best malware detection methods like memory forensic technique and secure data communication methods that can prevent Android devices. Whenever a user wants to install an application from play store, Application is downloaded first and then asked for installation after accepting all permissions. User can't install an application without accepting all permissions required by developer (hacker). Hackers usually ask for permission through which they can access user's camera, audio, text messages and all other private information. Users are uninformed of this purpose of hackers and they accept all permissions to install application. In this way, they become victim of hacker's attack. Moreover, hackers can make changes in constant strings to attack on mobile devices . Several techniques for detecting malware have been proposed in literature which can be divided into two broad classes: Static and Dynamic Analysis-based methods. Dynamic Analysis also known as behavioural-based analysis collects informa- tion from the OS at runtime such as system calls, network access and files and memory modifications. Hybrid apps consist of both native apps, and web apps. Like native apps, they live in an app store and can take advantage of the many de- vice features available. Like web apps, they rely on HTML being rendered in a browser, with the caveat that the browser is embedded within the app. Often, companies

build hybrid apps as wrappers for an existing web page. In that way, they hope to get a presence in the app store without spending significant effort f or developing a different app. Hybrid apps are also popular because they allow cross platform development. Thus significantly reduce development costs: that is, the same HTML code components can be reused on different mobile operating systems. Tools such as PhoneGap and Sencha Touch allow people to design and code across platforms, using the power of HTML. However, developers rush to exploit offthe shelf libraries in hybrid apps. Great new features are freely available without fully understanding, addressing, the security implications, increasing the chances of malware penetration in mobile devices. In Static Analysis (signature-based analysis), information about the App and its expected behaviour consists of explicit and implicit observations in its binary/source code. Static Analysis methods are fast and effective, but various techniques can be used to dodge Static Analysis and thus render their ability to cope with polymorphic malware. There are number of signature and behaviour-based detection tools available on play store for detection of malicious Android applications. Recent study has shown that signature based malware detection tools works till a certain level. They become ineffective when malware authors make changes in apps. Such type of signature-based tools and anti-viruses could not provide protection to Android users. Since Android is an open source and extensible platform, it allows to extract as many features as we would like. This enables to provide richer detection capabilities, not relying merely on the standard call records or power consumption pat- terns. The proposed method is novel in the context that it evaluates the ability to detect malicious activity on an Android device by employing Machine Learning algorithms using a variety of monitored features like permissions, providers, intent filters, process name and constant strings extracted from Android Apps. The proposed malware detection technique can also be used on diverse environments like BlackBerry, iOS etc. This work aims to find solution for following challenges:

- How to develop a malware detection system that can adapt to any kind of malware?
- How to detect malware before actual installation?
- How to scrutinize hybrid mobile apps for possible malware threat?
- How to warn Android users about malware after a download?
- Why extracting combined features of Android Apps is better way to detect malware than signature based and behaviour- based techniques?

Selection of good features from Android applications and their combination can lead to a robust malware detection system. Most of the malware detection techniques with

dynamic analysis detect malware after installation of an Android App which can affect devices. To install an application, user has to allow all malicious permissions. It is not a secure way that a malware detection technique identifies malware after a device has been affected. We performed static analysis in the proposed malware detection technique to detect malware after downloading application. In this way, security of Smartphone does not compromise. When a user downloads an App from play store and identified malicious by the proposed malware detection technique, Apps will be prompted by detection system and user will be informed about malicious app before installation.[14]

Chapter 4

SIGNATURE-BASED METHODS FOR MALWARE DETECTION

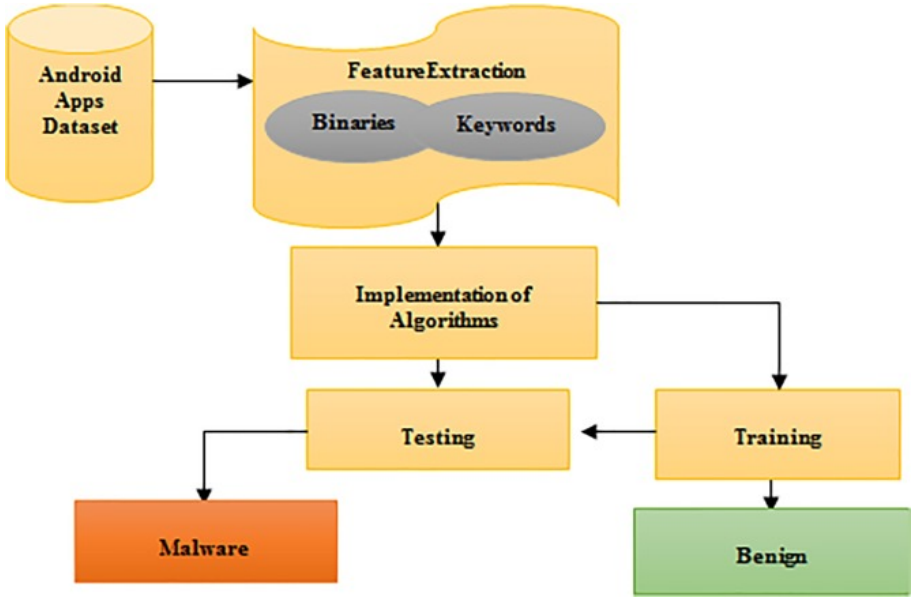


Figure 4.1: Basic process of Static method of detection

To detect Android malware most static approaches use requested permissions as the backbone of analysis. This study aims to learn if parsing only the Android manifest file is enough information to classify an app as malicious or benign. Another purpose of the study is to discern if permission sets or app intent filters are more effective indicators of malware. Finally, the study compares several different machine learning algorithms to find one which creates the most effective classifier. Android apps are stored as .apk files, which were decompiled using the open source apktool. For each app the manifest file, a .xml file, was further decomposed into permissions and intent filters using Python's ElementTree XML API. Information was stored in a matrix of indicator variables with each row representing an Android app and columns

representing what features it declares. The 183 features extracted were from three categories:

- **Requested Permissions:** Android apps must request user consent to access functionalities such as location, camera, or contacts. They must also list all their requests in the manifest file. A complete list of Android permissions was acquired from the Android developer website. This accounted for 154 features.
- **Declared permissions:**Android apps can create their own permissions, which can be used as an extra security layer against other apps accessing their data. A single variable was marked true if an app created any of its own permissions.
- **Intent Filters:** Intents are used to request actions from other app components. Apps can send implicit intents to other apps requesting information. Intent filters tell the system which implicit intents the app can handle and what overall phone states the app wants to know. A list of common intent filters was acquired from the Android developer website. This accounted for 28 features.[4]

4.1 PERMISSION-BASED ANALYSIS

Since Android security model is based on app permissions, we use permission names as features to build a machine learning model. Every app has to acquire the required privileges to access the different phone features. During an app installation, a user is asked whether to grant the app access to the permissions requested. Malicious apps usually require certain permissions. For example, in order to access and exfiltrate sensitive information from the SD card, a malicious app would require access to both the SD card and Internet. Our approach is to model combinations of the Android permissions requested by such malicious apps. We propose an approach that uses the appearance of specific permissions as features for a machine learning algorithm. In this approach, we first extract the permissions from our dataset and create a model. For training, we use Weka toolkit and evaluate several machine learning algorithms, including SVM, Naive Bayes, C4.5 Decision trees, JRIP and AdaBoost. Classification algorithms we chose differ in their underlying concept. Support Vector Machines is a non-probabilistic supervised machine learning binary classification algorithm. SVM is capable of nonlinear classification that maps inputs into high di- mensional feature space. C4.5 decision tree is a statistical classifier that builds a decision tree based on information entropy. Each node of the tree algorithm selects a feature and splits its sets of samples into subsets until classes can be inferred. Random forest is an ensemble classification algorithm that combines a number of decision trees and returns the mode of individual decisions by decision trees. Naive Bayes is a simple probabilistic classifier that is based on applying Bayes theorem with strong independence

assumption between features. Bayesian network is a probabilistic graphical model that represents a set of random variables and their inter-dependencies in directed acyclic graph. JRIP is a propositional rule learner that tries every attribute with every possible value and adds a rule which results to the greatest information gain. Logistic regression is a statistical regression model where dependent variable is used to estimate the probability of binary response based on multiple features. AdaBoost is a meta algorithm that can be used with many other algorithms to improve their performance by combining their outputs into a weighted sum which represents the final output. We then used the modified Weka 3.6.6 library 1 for Android to develop the OWASP Seraphimdroid Android app, which is using support vector machines with sequential minimal optimization model. 2 We also apply several clustering techniques in order to evaluate the performance of our unsupervised and supervised learning algorithms. Training, testing and evaluation of our model are performed using Weka Toolkit by applying the Far- thest First, Simple K-means and Expectation maximization (EM) algorithms. Simple K-means is a clustering algorithm where samples are clustered into n clusters, in which each sample belongs to a cluster with the nearest mean. Farthest First al- gorithm uses farthest-first traversal to find k clusters that minimize the maximum diameter of a cluster, and Expectation maximization (EM) assigns a probability distribution to each instance which indicates the probability of it belonging to each of the clusters.[4]

4.2 SOURCECODE-BASED ANALYSIS

The second approach is a static analysis of the app's source code. Malicious codes generally use a combination of services, methods, and API calls in a way that is not usual for non-malicious app . Machine learning algorithms are capable of learning common combinations of malware services, API and system calls to distinguish them from non-malicious apps. In this approach, Android apps are first decompiled and then a text mining classification based on bag-of-words tech- nique is used to train the model. Bag-of-words technique has already showed promising results for classification of harmful apps on personal computers . Decompiling Android apps to conduct static code analysis involves several steps. First, it is necessary to extract the Dalvik Executable file (dex file) from the Android application package (APK file). The second step is to transform the Dalvik Executable file into a Java archive using the dex2jar tool. 3 Afterwards, we extract all .class files from the Java archive and utilize Procyon Java decompiler (version 0.5.29) to decompile .class files and create .java files. Then, we merge all Java source code files of the same app into one large source file for further processing. Since Java and natural language text have some degree of similarity, we apply the technique used in natural language processing, known as "a bag-of-words".

In this technique, the text, or Java source code in our case, is represented as a bag or set of words which disregards the grammar or word order. The model takes into account all words that appear in the code. Our approach considers the whole code including import statements, method calls, function arguments, and instructions. The source code obtained in the previous step is then tokenized into unigrams that are used as a bag-of-words. We use several machine learning algorithms for classifications, namely: C4.5 decision trees (in Weka toolkit, it is known as J48), Naive Bayes, Support Vector Machines with Sequential Minimal Optimization, Random Forests, JRIP, Logistic Regression and AdaBoostM1 with SVM base. We performed our training, testing and evaluation using Weka Toolkit. For source code analysis, we also utilized ensemble learning with combinations of three and five algorithms and majority voting decision system. Ensemble learning combines multiple machine learning algorithms over the same input, in hope to improve the classification performance. The number of algorithms is chosen in a way that system is able to unambiguously choose the output class based on majority of votes.[4]

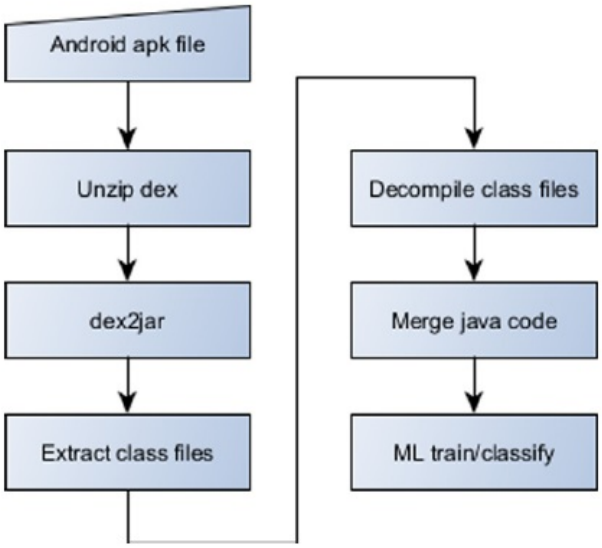


Figure 4.2: Flow diagram of process

We also experiment with clustering on the source code. Clustering algorithms we use include the Farthest First, Simple K-means and Expectation maximization (EM). A flow diagram of the process is presented in Fig. 1 .

4.3 EVALUATION AND DISCUSSION

Evaluated the performance of the approaches using 10-fold cross validation. In 10-fold cross validation, the original sample was randomly partitioned into ten equal sized

sub-samples. A single sub-sample was retained for the testing, while the remaining nine were used for training. The process was repeated ten times, and each time using a different sub-sample for testing. The results were then averaged to produce a single estimation. The main advantage of this method is that all samples were used once only for validation. The metrics we used for the evaluation of the algorithms are precision, recall and F-measure, which are widely used in the text mining and machine learning communities. Classified items can be true positive (TP –items correctly labeled as belonging to the class), false positive (FP - items incorrectly labeled as belonging to a certain class), false negative (FN - items incorrectly labeled as not belonging to a certain class), and true negative (TN - items correctly labelled as not belonging to a certain class). Given the number of true positives and false negatives, recall is calculated using the following formula:

$$Recall = TP/(TP + FN)$$

The recall is sometimes referred to as "sensitivity"or the "true positive rate". Given the number of true positive and false positive classified items, precision (also known as "positive predictive rate") is calculated as follows:

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

The measure that combines precision and recall is known as F-measure, given as:

$$F = ((1 + B^2) * Recall * Precision)/(B^2 * (Precision + Recall))$$

, where B indicates the relative value of precision. A value of = 1 (which is usually used) indicates the equal value of recall and precision. A lower value indicates a larger emphasis on precision and a higher value indicates a larger emphasis on recall.

4.3.1 Evaluation of permission-based classification:

Table 4.1: Permission based classification

Evaluation results of permission-based classification using single machine learning algorithms.			
Algorithm	Precision	Recall	F-Score
Logistic regression	0 .823	0 .822	0 .821

The evaluation of machine learning algorithms performing permission-based classification is presented in Table 1 . As observed from Table 1 , support vector machines with sequential minimal optimization has the best performance with a F-measure value of 0.879. In other words, this algorithm correctly classified 87.9percent of test

instances in 10-fold cross validation. The algorithm is also efficient, in terms of speed, as it took only 0.04 s to train the model. Instances were also classified faster; thus, making this approach suitable for real-time classification of (malicious) apps. Details for the implementation of LR classifier is given in the table.[4]

Table 4.2: Sourcecode based classification

Evaluation results of Sourcecode-based classification using single machine learning algorithms.			
Algorithm	Precision	Recall	F-Score
Logistic regression	0 .935	0 .935	0 .935

4.3.2 Evaluation of Source-Code based Classification:

Of the 400 apps in the data set, unable to decompile 32 of them (10 non-malicious and 22 malicious), perhaps due to code encryption and obfuscation or instability of our Java decompiler. Nevertheless, the remaining 368 source files were sufficient to train a good model. The evaluation of the classification for the analysis of the app's source code is presented in Table 4 . As Over 90percent of instances were correctly classified using LR. The high accuracy of source code-based classification reveals that the machine can infer app behavior from its source code. Even though the bag-of-words model disregards grammar and word order in text (in our context, the source code), it is possible to train a successful machine learning model that is able to distinguish malicious app from non-malicious app. Also, with the machine learning-based source code analysis, it is possible to analyze whether an Android package (apk) is malicious in less than 10 s, which is significantly faster than a human analyst.[4]

Chapter 5

DETECTION OF TRANSFORMED MALWARES USING PERMISSION FLOW GRAPHS

With growing popularity of Android, it's attack surface has also increased. Prevalence of third party android marketplaces gives attackers an opportunity to plant their malicious apps in the mobile eco-system. To evade signature based detection, attackers often transform their malware, for instance, by introducing code level changes. Here we see a lightweight static Permission Flow Graph (PFG) based approach to detect malware even when they have been transformed (obfuscated). A number of techniques based on behavioral analysis have also been proposed in the past; however our interest lies in leveraging the permission framework alone to detect malware variants and transformations without considering behavioral aspects of a malware. The proposed approach constructs Permission Flow Graph (PFG) for an Android App. Transformations performed at code level, often result in changing control flow, however, most of the time, the permission flow remains invariant. As a consequences, PFGs of transformed malware and non-transformed malware remain structurally similar as shown in this paper using state-of-theart graph similarity algorithm. Furthermore, propose graph based similarity metrics at both edge level and vertex level in order to bring forth the structural similarity of the two PFGs being compared. validate the proposed methodology through machine learning algorithms. Results prove that our approach is successfully able to group together Android malware and its variants (transformations) together in the same cluster. Further, we demonstrate that our proposed approach is able to detect transformed malware with a detection accuracy of 98.26percent, thereby ensuring that malicious Apps can be detected even after

transformations.[8]

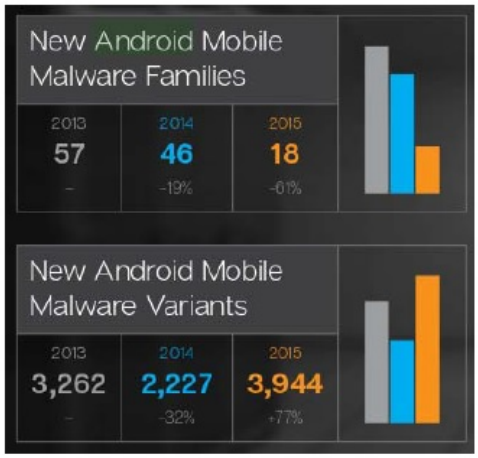


Figure 5.1: Trends diagram

This trend means that attackers are not investing their efforts in developed new malwares from scratch but they are rather modifying the existing malware families to create malware variants. Although, there exist an official Android market (Google Play store), however, at the same time a large number of third party marketplaces have also sprung up which have given the attackers an opportunity to host such malware variants.4 For instance, in a report published by SC Media claims that Turkish Android App store spread malware.5 Despite advisories to not download apps from third party App Stores6, users do end up downloading apps from third party stores.

5.1 METHODOLOGY

The approach involves construction of Permission Flow Graphs (PFGs) and leverages from the concept of graph similarity to create the feature vectors for machine learning algorithms. PFGs represent the sequence in which the various permissions are used in a mobile application. Attacker, while making transformations, may or may not affect the flow of permissions. However, as it gets demonstrated through our work, the resulting PFG of the transformed malware still remains structurally similar to that of the original malware, even if not an exact match.[8]

5.1.1 Permission Flow-Graph Construction:

The permission framework in Android has been extensively studied, in fact the set of permissions requested by an application have been extensively used as feature vectors in prior work. Our work differs in the fact that we also take into consideration the transformation attacks, thereby capturing the variants of a given malware. A lot of

prior works have constructed API graphs in their approach, however, API graphs are typically large in size thereby increasing computational costs. Instead, we pre-process and convert the API graphs into Permission Control Graph as explained in this section. We begin by defining the concept of Permission Flow Graph (PFG) as below.

- **Definition**
  Permission Flow Graph (PFG) is a directed graph that can be defined as an ordered pair G = (V,E) where, V is set of permission nodes and E is a set of directed permission edges. Each edge (pi, pj) implies that permission pj is requested after pi as per program control flow.

Given an Android application, first an API level Control Flow Graph (CFG), referred as (Ax), needs to be constructed. CFG represents the control flow of APIs used by the application as depicted in Figure. Each node of an API control flow graph represents an API call, and an edge between node i and node j represents the control flow between the two API calls. The Android permission framework is such that each API call may or may not require one or more permissions. Hence a one to many mapping exists between an API call and an Android Permission. Using the API graph (Ax), we construct, its corresponding Permission Flow Graph (PFG) (Px). Thus for each API graph Ax, all nodes that require a certain permission need to be extracted, and depending on the control flow between them, a Permission Flow graph Px is constructed. For each node ai in Ax, a set of nodes (Pi, Pi1, Pi2,. Pik) are present in the PFG, where k is the number of permissions required by API call ai. The conversion between and API graph and Control Flow Graph can be seen in Figure. Once the Permission Flow Graphs (PFGs) for the Android Apps are generated and stored, a similarity feature vector is created for each record in the training data using Edge and Vertex similarity metrics. This is a one dimensional feature vector with N elements, where N is the number of Android Apps under consideration. Each element of the feature vector contains a similarity score, which represents the degree of similarity of the Permission graph of this application with respect to another application in our database. The ith element of this feature vector represents the similarity score of PFG of Android App with respect to PFG of ith Android App in our malware database. Thus the structural similarity between applications in the context of the sequence in which the two applications request permissions is incorporated in these similarity feature vectors. The metrics used for calculating the similarity score are such that they take into account the partial similarity between graphs as well and not just the exact match. Thus these feature vectors would capture structural similarities between malwares and their transformed variants.[8]

5.1.2 Proposed solution pipeline

A broad pipeline of our approach is shown in Figure which depicts following key components:-

- Android App: The input to the proposed system is an Android App which needs to be checked whether it is malicious or not.
- API Generator: Converts the input Android App into its corresponding call flow graph using the API calls.
- Constructing PFG: Next step involves converting the API call graph into its corresponding permission flow graph.
- Feature Calculation: Two key features namely edge level metric (ELM) and vertex level metric (VLM) are computed at this stage.
- Clustering / Classification: Finally the input Android App is tested whether it belongs to an existing malware family or whether it is malicious or not.

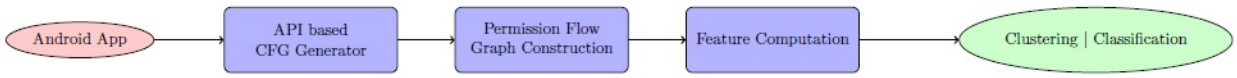


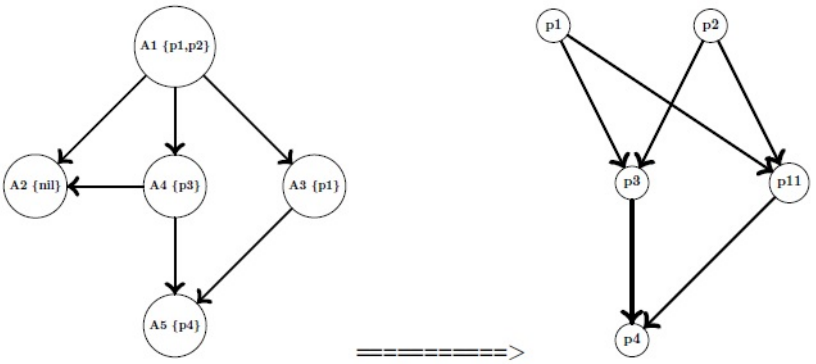
Figure 5.2: Proposed solution pipeline

5.1.3 Graph Similarity Metrics

Given two graphs G1 and G2, the two key metrics used in our work are edge level metric (ELM) and vertex level metric (VLM), computed as below.

$$ELM = numcommonedges(G1, G2)edges(G1) + edges(G2)(1)$$

V LM is similarity score between each vertex in G1 and G2, which is computed as per the Similarity Flooding algorithm. Briefly, two vertices are similar if their permission labels as well as their neighbors are similar. Using the Similarity flooding algorithm a MXN matrix (Similarity-matrix) is constructed containing the similarity score of node i in G1 with node j in G2, where M and N are the total number of nodes in G1 and G2 respectively. Using the pseudo code presented in Algorithm 1, we reduce this MXN matrix to a MX1 matrix. Each element in this matrix contains the similarity score of a node in graph G1 to its most similar counterpart in graph G2.



(a) API level Control Flow Graph (CFG) depicting the set of permissions corresponding to each API call. For instance, for API call A1, permissions are p1 and p2 and so on.

(b) Permission Flow Graph (PFG) corresponding to the API CFG depicted in Fig. 2, it may be noted that permissions p1 and p2 on the top correspond to API A1 and so on.

Figure 5.3: Permission flow graphs

Algorithm 1 Proposed Algorithm

1: **procedure** PERMISSIONSIMPROXIMITY(()G_{1i}, G_{2j}) ▷ Computing the similarity score of the most similar counterpart of node i in Graph G1 with node j in Graph G2
2: $S_{ij} \leftarrow SimScore(G_{1i}, G_{2j})$
3: **for** *Each row* R_i *in* S_{ij} **do**
4: $max_i \leftarrow 0$
5: **for** *Each score* S_i *in* R_i **do**
6: **if** *Perm(i,G1) = Perm(i,G2)* **then**
7: **if** *Score Si* $>= max_i$ **then**
8: $max_i \leftarrow S_i$
9: **end if**
10: **end if**
11: **end for**
12: **end for**
13: **end procedure**

Figure 5.4: Algorithm for PFG

5.2 RESULTS

5.2.1 Clustering Results

LR clustering was applied to group the different malwares, detailed findings are presented

- **DroidKungFu:** Is a Malware that affects Android OS and it targets the mobile platform in China. First piece of malware found in the Android Market is in March 2011.
- **DroidDream:** Is a mobile botnet type of malware that appeared in spring 2011. The DroidDream Trojan gained root access to Google Android mobile

Table 5.1: Clustering accuracy using various similarity metrics

CLUSTERING ACCURACY USING VARIOUS SIMILARITY METRICS			
Dataset	Edge Metric	Vertex Metric	Combination
AnserverBot	100	100	100
DroidKungfu	91.42	77.14	100
ADRD	75	75	75
DroidDream	66.66	86.66	60
GoldDream	100	38.46	61.54

devices in order to access unique identification information for the phone.

- **Android.Golddream:** Is a Trojan horse that steals information from Android devices.

  Android package file: This Trojan disguises itself on certain marketplaces as game software, but it comes bundled with a Trojan.

- **AnserverBot:** One of the most sophisticated bot program infecting Android devices. This particular bot piggybacks on legitimate apps and communicates with remote CandC servers for further instructions. Based on our current investigation, AnserverBot is being injected into a number of (20+) legitimate apps, which are then distributed in alternative Android markets in China. Different from earlier ones with bot capabilities such as Pjapps and BaseBridge, this bot program exploits several techniques, including deep code obfuscating and (Plankton-like) dynamical code loading to thwart reverse engineering efforts as well as anti-tampering to protect itself. To the best of our knowledge, this is indeed one of the most sophisticated bot program on Android we have ever seen to date.

Chapter 6

CONCLUSION

Development of a defect prediction model helps in ascertaining software quality attributes and focused use of constraint resources.They also guide researchers and practitioners to perform pre-ventive actions in the early phases of software development and commit themselves for creation of better quality software. In this work, we not only conduct an extensive empirical experimentation on publicaly available application packages of Android software systems but also provide a repeatable and pragmatic approach to achieve such models. Specifically, the main contributions of the paper are: (i) create defect prediction models using various ML techniques on collected multiple data sets from the Google code repository for the Android platform, (ii) pre-process the data using the attribute reduction techniques, (iii) empirically validate the constructed models on various releases of the application packages of Android data sets in order to obtained generalized results, (iv)perform statistical tests to evaluate the significance of the obtained results and, (v) analyze the outcomes to deduce meaningful and generalized conclusions.

The paper focuses on the various ways of detecting malware on android platform in detail.

6.0.1 Future scope

Future research includes the evaluation of the proposed approaches using a significantly bigger labeled balanced data sets and utilizing online learning. Another research focus is combining static and dynamic software analysis in which multiple machine learning classifiers are applied to analyze both source code and dynamic features of apps in run-time.

Bibliography

[1] Mariam Al Ali, Davor Svetinovic, Zeyar Aung, Suryani Lukman Malware Detection in Android Mobile Platform using Machine Learning Algorithms.

[2] Monica Kumaran Electrical Engineering and Computer Sciences University of California, Berkeley Berkeley, U.S.A. monica.kumaran@berkeley.edu Wenjia Li School of Engineering and Computing Sciences New York Institute of Technology New York, U.S.A. wli20@nyit.edu Lightweight Malware Detection based on Machine Learning Algorithms and the Android Manifest File

[3] Nikola Milosevic , Ali Dehghantanha , Kim-Kwang Raymond Choo , a School of Computer Science, University of Manchester, UK b School of Computing, Science and Engineering, University of Salford, UK c Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249-0631, USA Machine learning aided Android malware classification

[4] Songyang Wu, Pan Wang, Xun Li, Yong Zhang Effective Detection of Android Malware Based on the Usage of Data Flow APIs and Machine Learning

[5] Zhenxiang Chen, Qiben Yan, Hongbo Han, Shanshan Wang, Lizhi Peng, Lin Wang, Bo Yang Machine Learning Based Mobile Malware Detection Using Highly Imbalanced Network Trafc

[6] Ridhima Seth and Rishabh Kaushal Detection of Transformed Malwares using Permission Flow Graphs

[7] An empirical framework for defect prediction using machine learningtechniques with Android software Ruchika Malhotra Q1Department of Software Engineering, Delhi Technological University, Bawana Road, Delhi, India

[8] MalDozer: Automatic framework for android malware detection using deep learning ElMouatez Billah Karbab , *, Mourad Debbabi , Abdelouahid Derhab , Djedjiga Mouheb Concordia University, Canada King Saud University, Saudi Arabia University of Sharjah, United Arab Emirates

[9] What is Malware? How Malware Works and How to Remove it AVG https://www.avg.com/en/signal/what-is-malware

[10] What is Machine Learning? A definition Expert System www.expertsystem.com/machine-learning-definition

[11] Reverse engineering - Wikipedia https://en.wikipedia.org/wiki/Reverse_engineering

[12] Data sets and machine learning - Deeplearning4j: Open-source ... https://deeplearning4j.org/data-sets-ml

[13] Machine learning - What is a Classifier? - Cross Validated https://stats.stackexchange.com/questions/69602/what-is-a-classifier

[14] Machine learning-assisted signature and heuristic-based detection of malwares in Android devices Zahoor-Ur Rehman , Sidra Nasim Khan , Khan Muhammad , Jong Weon Lee , Zhihan Lv , Sung Wook Baik , Peer Azmat Shah , Khalid Awan , Irfan Mehmood

[15] 6 Easy Steps to Learn Naive Bayes Algorithm https://www.analyticsvidhya.comMachineLearning