A SURVEY ON ANDROID MALWARE DETECTION USING MACHINE LEARNING

Dr. M. Amutha[1], A. Mohanraj[2], S. Vinu[3], V. Balamurugan[4] and A. Aarthi[5]

[1]Professor, HINDUSTHAN COLLEGE OF ENGINEERING AND TECHNOLOGY, Coimbatore,

[2]Assistant professor, SRI ESHWAR COLLEGE OF ENGINEERING, Coimbatore,

[3]Assistant Professor, HINDUSTAN COLLEGE OF ENGINEERING AND TECHNOLOGY, Coimbatore,

[4]Assistant Professor, AKSHAYA COLLEGE OF ENGINEERING AND TECHNOLOGY, Coimbatore,

[5]Assistant Professor, KIT- KALAIGNARKARUNANIDHI INSTITUTE OF TECHNOLOGY, Coimbatore,

[1]amuthabps@gmail.com, [2]mohanraj.a@sece.ac.in, [3]vinuja@gmail.com, [4]balamuruganvsrit@gmail.com, [5]aarthicse4@gmail.com

Abstract

With the unfold of humanoid software package, humanoid malware has been increasing in recent years. humanoid malware poses a heavy risk to shoppers, like loss of non-public information and advanced fraud, and is put in and dead on the device while not the express permission or information of the user. Varied ways are planned by researchers and practitioners to counter these dangers. One in every of these strategies, static analysis, is usually accustomed determine malware on humanoid devices and may forestall malware installation. A comprehensive literature review of ninety eight surveys from January 2014 to March 2020 are going to be conducted to focus on the newest work on humanoid malware detection mistreatment static analysis. First, we tend to classify the static analysis of humanoid malware detection into four teams in line with the characteristics of the applying. These teams embrace humanoid property-based strategies, opcode-based strategies, program graph-based strategies, and symbolic execution-based strategies. Next, we tend to measure the power of static analysis to observe malware, scrutinize the results of empirical studies, and compare the effectiveness of assorted humanoid malware detection models. By distinguishing ninety eight studies from January 2014 to March 2020, we've conducted an intensive literature review that clearly demonstrates the newest work on humanoid malware detection mistreatment static analysis. First, we tend to categorise static analysis of humanoid malware detection into four categories: humanoid feature-based approach, operation code-based methodology, program graph-based methodology, and application property-based symbolic execution-based methodology. Next, by examining the results of empirical studies, we tend to measure the power of static analysis to observe malware and compare the effectiveness of assorted humanoid malware detection models. Finally, confine mind that humanoid malware may be detected mistreatment static analysis. Additionally, preliminary results show that neural network models square measure superior to non-neural network models in detective work humanoid malware. However, there square measure still several problems with static analysis. Therefore, some current analysis communities have to be compelled to come back up with some distinctive ways to enhance the detection of humanoid malware. Additionally, AN integrated platform has to be created to objectively assess the effectiveness of assorted humanoid malware detection strategies.

1. Introduction

Android has overtaken different subtle in operation systems as a results of the mobile market's recent spectacular enlargement. Quite eightieth of all smartphones oversubscribed up to now are steam-powered by the humanoid software package. Once the humanoid software package is wide utilized by several users and application developers, it additionally becomes a favourite attack target for malevolent people. Malware was at the start discovered within the humanoid system in Sept 2010. Soon after, the amount of malicious programmes starts to rise.

The information a pair of reports that within the third quarter of 2018, there have been 3.2 million instances of humanoid malware, up four-hundredth year over year. In everyday life, humanoid applications square measure wide used. As of Sept 2018, there have been a pair of .6 million humanoid applications accessible to users, in line with Google Play three. The humanoid market conceals an outsized range of harmful apps that is kind of dangerous for shoppers. On smartphones, humanoid malware is put in and launched while not expressly asking the shoppers for permission or with their information. It usually exhibits one or additional subsequent actions: strong-arm connection, browser capture, information thieving and modification, malicious data assortment, malicious installation, hateful hurrying, and different malicious actions.

Static analysis will observe malware by scanning the complete APK while not corporal punishment the APK, whereas active study instruments and implements the APK for malware discovery. Static analysis will instantly determine malware and forestall malware from being put in, not like dynamic analysis that has bound difficulties resisting malicious deformation techniques such java reflection and dynamic code loading. To boot, static analysis might travel through all potential APK execution routes, creating it helpful and ascendible for batch unknown APK identification. Static analysis is presently a key methodology for locating malware on humanoid devices.

For instance, Kirin [1] suggests a tool that identifies vulnerable programmes largely supported mixtures of risky permissions. It’s early example of static analysis that applies permissions to seek out malware on humanoid. For the aim of finding personal data leaks in humanoid applications, ScanDal [2] accepts Dalvik bytecode as input. To seek out malicious payloads, IccTA [3] collects parts from apps. Through the utilization of management flow graphs and taint information analysis, Shahid et almethod's [4] mechanically identifies sensitive information leak channels. There are some surveys and reviews written concerning humanoid malware discovery mistreatment static analysis. From the perspective of humanoid security design, Faruki et al. [5] explore the glitches of malware diffusion and concealing ways between 2010 and 2014.

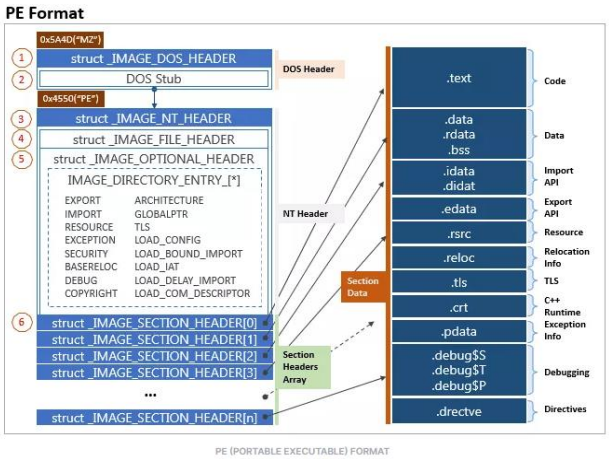
1.1 PE file format

All executable files use the Common Object File Format (COFF), a file format for executable, object code, and shared library computer files used on UNIX systems. And PE is one such COFF format for executable, object code, DLLs, FON font files, and core dumps that is now offered for 32-bit and 64-bit Windows operating systems (Portable Executable).

The PE (Portable Executable) file type instructs the Windows OS loader what information is required to manage the wrapped executable code. Dynamic library references, API export, import

tables, resource management information, and TLS information are all included for linking. Because the data structures used on disc are the same ones used in memory, if you know how to find anything in a PE file, you can almost certainly find the exact information after the file has been loaded into memory.

It matters that PE files are not just mapped into memory as a single memory-mapped file. Instead, the Win32 loader looks over the PE file to decide which sections of the file to map in.ll the information from an executable file that a process needs is represented in memory by a module. It is possible to read other PE file sections but not map them in (for instance, relocations). Some components, such as those that are inserted at the end of the file with debug information, might not be mapped in at all. How much memory must be left aside for the executable's memory mapping is specified in a field in the PE header. The end of the file, after any parts that will be mapped in, is where data that won't be mapped in is inserted.



Procedures:

Step 1: Data pre-processing: Import libraries

Step 2: Data import in

Step 3: Checking for missing data

Step 4: Examining data for categorization

Step 5: Scaling the features

Step 6: Dividing data into sets for training, validation, and evaluation.

Procedure for choosing features

One of the crucial elements in creating a model is feature selection. Its objective is to identify the ideal combination of features for constructing a model.

The following are some well-liked methods for feature selection in machine learning:

- Filter techniques
- Wrapper techniques
- Embedded techniques

1.2 Filter methods

Filter methods are frequently employed during the pre-processing step. No matter the machine learning method employed, these algorithms pick traits from the dataset. They are good at minimising redundant, correlated, and duplicate features and very efficient in terms of compute, but they do not completely remove multicollinearity. When features are evaluated alone (without relying on other qualities), the choice of features can occasionally be advantageous, but it will lag when a combination of features can improve the model's overall performance.

Among the strategies employed are: Information Gain, which evaluates the decrease in entropy values and is defined as the amount of information provided by the feature for recognising the target value. The goal values for feature selection are taken into account when calculating the information gain of each characteristic.

Chi-square test – The Chi-square method (X2) is frequently used to examine the correlation between category variables. It contrasts the observed values for the dataset's various properties with their expected values.

Chi-square Formula

Fisher's Score: A suboptimal set of features is produced by this method, which selects each feature independently based on how well it meets the Fisher criterion. The Fisher's score increases with the quality of the selected feature.

With values ranging from -1 to 1, the Pearson's correlation coefficient is a metric for assessing the correlation between two continuous variables and the direction of the relationship.

Variance Threshold: This method involves eliminating all features whose variance falls below a predetermined threshold. This approach eliminates characteristics with zero variance by default. This technique makes the assumption that characteristics with larger variation are more likely to contain information.

Mean Absolute Difference (MAD): This approach resembles the variance threshold approach, however MAD does not include a square. With this approach, the mean absolute deviation from the mean value is calculated.

Dispersion Ratio: It is the ratio of the arithmetic mean (AM) to the geometric mean (GM). For a specific characteristic, it has a value that goes from +1 to GM as AM. A more relevant feature is implied by a higher dispersion ratio.

Mutual Dependence: If two variables are interdependent, this technique determines how much can be learned about one variable by observing the other. When a feature is present or absent, it is measured how much information it adds to the target prediction.

Relief : This method evaluates the quality of characteristics by randomly choosing a sample instance from the dataset, updating each feature, and comparing the chosen instance to the two nearest instances of the same and opposing classes to determine which instances are similar to one another.

In addition, a recent study [6] details the understanding of static package study (ie, flow sensitivity, context sensitivity, pathway kindliness, ground understanding, thing understanding), and static analysis within the humanoid domain like code. However, there square measure gaps in humanoid malware detection analysis in new yonks. Additional significantly, with the fast development of humanoid malware in recent years, some humanoid malware detection studies have clearly magnified and a number of other new humanoid malware detection solutions have emerged for instance, by analogy with the graph arrangement technique, [7] S.Y.Yerima et al. [7] mix management flowcharts and diagram cores to investigate the variations between malicious and gentle submissions and determine humanoid malware.

Consequently, it's imperative to use trendy static analysis to summarize humanoid malware detections. This systematic literature review (SLR) are going to be conducted when an intensive identification of relevant studies to supply a transparent summary of humanoid malware detection mistreatment static analysis over the previous few years. The most assistances of this SLR camera square measure as tracks.

- Achieve this SLR supported a crucial side of humanoid malware detection by static analysis.
- This white book breaks down static analysis of humanoid malware into four classes supported the capabilities of your application. It then evaluates the capabilities of

fixed analysis by analysing empirical proof and relates the routine of various replicas in humanoid malware detection.

- Supported the consequences of pragmatic proof, static investigation is operative and neural network models square measure superior to non-neural network models in detective work humanoid malware.

Conduct a review. In this phase, we will show you the main contents of this SLR research.

This can be divided into 6 steps:

1. The issues that need to be examined in this SLR are identified by survey questions, and the survey questions' responses serve as the foundation for the discussion part.
2. A search plan. Finding search terms and sources for gathering primary research is the goal of this step.
3. Selection criterion for research. Both inclusion and exclusion criteria are used in the selection of studies. These criteria screen unrelated studies according to inclusion and exclusion criteria and decide which papers are included or excluded from this SLR.
4. Standards for evaluating quality. The relevance of particular research to the objectives of this SLR is evaluated at this step.
5. Data extraction creating a data extraction form with the aim of obtaining correct information about the survey question is the objective of this phase.
6. Data composition and to collect and summarize the outcomes of the main study.

Report evaluation: according to these guidelines, that phase should be performed on this SLR camera.

2. Common Statistical Analysis Techniques for Detecting Android Malware

The source or binary code of a programme is the input for the code analysis method known as static analysis. Examine this code without running the programme to ensure that it adheres to the program's specifications (security, reliability, etc.). Due to the fact that applications do not need to execute the primary advantages of static analysis over dynamic analysis, they are extremely efficient and quick. Static analysis methods are frequently used to identify Android malware due to these benefits. The META-INF, reserve files, collection directories, resource directories, AndroidManifest.xml, and binaries make up the majority of the APK, in general.

Static analysis is divided into four areas based on the traits taken from the APK:

- Android attributes,
- Opcodes,
- Programmed diagrams,
- Symbolic execution.

A procedure using Android characteristics. This category primarily compiles Android property-related functions. Both the configuration file and the binary bytecode programme

provide access to these functions. Feizollah etal, for illustration. [6] Extrapolate application intent as a malware classification trait. Yerima and co. [7] Experimental results indicate that sensitive API calls and permissions are promising characteristics for identifying Android malware. Approach based on opcodes.

The APK binary file's opcode is extracted. Binaries are composed of numerous little files, with each small file serving as an application's class file. This is a collection of opcodes. This approach considers opcode sequences as text and focuses on identifying malware by integrating deep learning and natural language processing algorithms. J Yan, etc. [9] the n-gram model is based on the presumption that the occurrence of the nth word is related exclusively to the previous n-1 word. All feasible n-grams from the operation code sequence as the initial feature. Model for natural language processing) to categorise malware, create and integrate machine learning models. S.Rasthofer [10] to recognise malware, learn the contextual meanings of opcode sequences through long short-term memory. The program graph is extracted from the APK binary file. Program graphs can seizure more syntactic information associated to the above dual approaches. As a result, several research use malware analysis and detection techniques based on programme graphs. For instance, Allix et al. It gathers all the fundamental building pieces that make up them, calls them functions, and combines by relying on an abstract representation of the application's control flow diagram.

Execution-based technique using symbols. Execution-based technique using symbols using procedural variables in place of abstract symbols, symbolic execution accurately calculates procedural variables to simulate the execution of a programme. These abstract symbols are used to produce expressions and constraints that can be used in the constraint solver in accordance with conditional branching of a certain path in your application. The representations and limitations for harmful applications are subsequently compiled into a rules library.

The application is regarded as dangerous if the application's terms and restrictions match those in this rule collection by M. Fan, J. Liu, and others [11]. For instance, TASMAN [12] employs symbolic execution to further detect Android malware and increase the efficacy of Flowdroid in order to lower the high false positive rate of Android malware detection using Flowdroid. Studies employing symbolic execution-based methods to recognise Android malware are required because symbolic execution currently confronts a number of difficulties, including path selection and constraint resolution. The sum is really little.

It includes a list of categories for static analysis methodologies together with the number and percentage (PCT) of the respective study categories. According to this, which represents around 53% of all polls, the Android property-based static analysis method is the most widely used one. Additionally, there are around 19% and 20%, respectively, of primary studies pertaining to opcode-based and programme graph-based approaches. There aren't many studies on approaches based on symbolic execution. This implies that the property-based technique of Android was the main focus [14].

3. Android Malware Detection Using Static Experiments

Empirical Studies

Detailed evaluation and précis of empirical evidence. Figure four suggests the 5 steps of the demonstrative test technique for detecting Android malware. First, facts series is aimed toward amassing benign and malicious datasets. In general, the greater experimental datasets you have the greater compelling consequences you may get. Second, function extraction goals to extract capabilities from the APK thru a static evaluation help tool [15]. Third, function discount strategies are carried out to decide and pick out critical capabilities. Fourth, version choice goals to discover the proper version to differentiate among malicious and benign applications. These fashions consist of statistical and system mastering fashions. Fifth, version assessment goals to assess version generalization via overall performance measurements. Following the empirical experimental technique, the subsequent segment discusses experimental datasets, static evaluation help tools, usually used functions, characteristic discount approaches, fashions used, and Android malware detection overall performance measurements.

4. Databases Used for Detecting Android Malware

Researchers and practitioners have used a variety of harmful datasets in their first studies. The dataset can be separated into two groups based on its dataset source [19]: in-lab datasets and in-wild datasets. The datasets used in the lab are primarily Drebin 4 and Genome 5, and are usually regarded as the foundation for Android malware detection. The first malware dataset to be compiled and made public is the genome. Drebin suggests evaluating Android malware detection techniques using the genome [2]. This is the first effective strategy that can be explained [35]. Records from the wild are updated and kept up to date constantly. The public can access the aforementioned dataset. Nevertheless, some studies continue to employ proprietary databases like [20] and [23]. Applications for benign datasets are mostly found in the Chinese mobile application market and Google Play Store 13, and according to the majority of studies, benign datasets are not open source.

5. Support Tools Used in Android Malware Detection for Statistical Analysis

Numerous tools are employed in the primary study to support static analysis. The implementation of preliminary static analysis can be done using a number of standard tools. The intermediate plot [20], which is used in the static analysis phase, differs between these programmes. These support tools have intermediate representations in smali, dex assembler, jimple, and java class. To obtain the Android application's device representation format, more analysis and processing are needed [25].

6. A Common Use for Features in Android Malware Detection

The behaviour of the APK at various levels can be represented by various sorts of functions. This leads to a lack of consistency in these features' malware detection abilities. The primary investigation employs a set of traits to represent the behaviour of the APK in order to identify

malware. The context semantics of apks can be captured by opcode sequences, according to four categories of RQ1 static analysis techniques. Opcode-based methods treat opcode sequences as text. This section focuses on the features of Android and programme graph-based techniques for malware detection because symbolic execution is a sophisticated technique that has been used in two studies and is also a difficult methodology. Features associated with Android property-based approaches can be divided into eight categories a security-related resource called permissions is one that the user pre-grants during the installation procedure. Some API requests are bound to the Android permissions system because they are permission-related. For instance, in order to connect to the network, you must call the get Network Operator () API and seek permissions from access network state.

Additionally, some API calls, such get Device ID, are linked to sensitive resources or data (). In general, a good application uses an API request less frequently when it is sensitive than a malicious application does. The image also demonstrates that network- and code-related patterns are among the core survey's neglected characteristics (such as [29]). The network has features that can be used to retrieve files and data and may be connected to botnets. By combining good and bad APK code patterns, code-related patterns are gathered. Ratings, Downloads, Developer Reputation, etc., are descriptive of the APKs released in the application market. Five different types of programme graphs are displayed, including API dependency graphs, component call graphs, control flow graphs, function flow graphs, and data flow graphs. You can get these programme graphs via the tool that supports static analysis. Context-dependent API actions are represented as nodes in the graph of API dependencies, and data dependencies between operations are represented as edges [30]. Control flow graphs and data flow graphs combine to generate inter-component call graphs [67].

The connection between the most recent executed basic block and the one that is now active in a control flow graph is known as an edge. The function is represented as a node in the function call graph, and the relationships between function calls are represented as edges, denoted by call statements like invoke. Contrary to control flow graphs, data flow graphs illustrate programmes using data flow and data processing techniques. Data flow diagrams are used in most static error analysis approaches. As a result, when using a programme graph-based approach, the quantity of data flow graphs takes up the largest share.

7. Feature Reduction Techniques for Android Malware Detection

In important investigations, a range of feature reduction strategies were employed to pinpoint and choose the essential features for Android malware detection. The two primary types of feature reduction techniques are feature extraction and feature selection. By removing unnecessary and distracting elements, feature selection enables you to choose the most important features, such as: B. Information gathering (IG). The dimensions of these original features can be reduced by feature extraction to produce new combinations of characteristics, such as: Principal component analysis (B) (PCA). Approximately 24% of main studies, excluding the four SLRs, employ conventional feature reduction methods that can be applied

immediately in the study. 12 percent of studies have enhanced common feature reduction methods and suggested a number of novel feature reduction strategies, such [37].

Additionally, word embedding techniques are most frequently utilised in roughly 17% of research that use neural networks to automatically extract features from applications , Extract up to 34,000 static features in 7 categories. Apply a Deep Auto encoder (DAE) to recreate the original features in order to handle these features effectively. Additionally, about 47% of the surveys employed Origin's characteristics directly to identify Android malware like [38] without using feature reduction techniques. The list and distribution of function reduction techniques that are offered for sale.

Seven common feature reduction techniques are created: IG, 2, Fisherscore (Fisher), GR, correlation-based feature selection (CFS), evolutionary algorithm (EA), and PCA. It shows that IG, where the IG of a feature f is defined as the difference between the uncertainty before utilising the feature f and the uncertainty after it is expected, is the most widely used feature reduction approach. If feature f1's IG is higher than feature f2's IG, then feature f1 is seen as being better to feature f2. Be aware that several researches compare many techniques to determine the most suitable feature reduction technique. Martnetal, for instance, to discover an appropriate technique to decrease characteristics. The results of IG, chi-square (2), and gain ratio should be compared (GR).

8. Models for Detection Android Malware

The model for identifying Android malware is introduced in this section. Machine learning models and statistical models are the two groups into which the models are split model for machine learning. Machine learning techniques are rapidly becoming more and more commonly employed in recent years to identify Android malware due to the rapid development of machine learning techniques such as natural language processing and picture recognition. Support Vector Machines (SVMs), Naive Bayes (NB), Logistic Regression (LR), Ensemble Learning (EL), and Neural Networks are the machine learning models most frequently utilised in primary investigations. Numerous research communities, including those concerned with image recognition and detection, heavily rely on neural networks, a recent development in machine learning.

It also aids in the detection of Android malware. Xiao and co. [30] we propose employing artificial neural networks (ANN) and associating API calls from static analysis of Android applications to detect malware. The evaluation's findings indicate a high F value. Convolutional neural networks can be trained using Deep Autoencoder (DAE) as a pre-training technique (CNN). This strategy enables you to learn more flexible patterns faster by combining DAE with CNN (DAE-CNN). Using DAE-CNN increases accuracy by 5% when compared to SVM and Statistical framework. Making a library of rules from a malicious application is the statistical methodology for Android malware detection.Then determines how similar the unidentified APK and this rule library are. This apk is regarded as harmful if the similarity rises above a particular level. In contrast, this APK is regarded as innocuous if the resemblance is below a specific threshold. Ali-Gombeetal is a good example. [38] Uses a predefined library of rules to detect malware that are based on permissions and the extraction

of opcode sequences from vulnerable function modules. With the exception of the four SLRs, 82 studies have been associated to machine learning models as opposed to 12 researches that have been related to statistical models. It's very almost 13 percent. This demonstrates how machine learning models have lately taken centre stage in the detection of Android malware.

Classifications should also be done for machine learning models. Based on Malhotraetal, the classification technique. [39] Machine learning models can be divided into the following nine categories: Decision Trees (DT), Basin Learning (BL), Ensemble Learning (EL), Neural Networks (NN), Support Vector Machines (SVM), and Evolutionary Algorithms (EA). It is divided into categories such as rule-based learning (RBL), logistic regression (LR), and more. The number and percentage of studies belonging to each category. We discover that the three most often employed models are neural networks (17%), SVM (19%), and ensemble learning (22%). It depicts a more comprehensive distribution of machine learning models, and the most well-liked.

9. Static Analysis-Based performance Measures for Android Malware Detection

Performance evaluations are used to evaluate the model's generalizability in identifying Android malware. The definitions of key performance indicators and the number of key performance indicator surveys are displayed. The definition of performance metrics is illustrated using the terms True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The number of correct benign applications for which the application has been used is denoted by TP. FP: Malware that was mistakenly identified as innocuous programmes; TN: Malware that was correctly identified when it turned out to be malware. FN: Malware that was accurately identified when it turned out to be benign applications [34].

The model's capacity for generalisation is improved by reduced performance measurements for FPR, FNR, and error rates. For significant surveys like [8] and [34], the Jaccard, MCC (Matthews Correlation Coefficient), standard deviation, and p-values are the most common correlation coefficients. In addition to the performance metrics mentioned above, the model is more generic the higher the performance measurements, such as accuracy and F measurements. As seen by the amount of studies that employ precision and recall, precision is the most often used measure of performance.

10. Static Analysis Techniques General Performance in Android Malware Detection Based on Empirical Evidence

This section's objective is to assess static analysis methods' potency in spotting Android malware. Only the numbers from the performance measurements are recorded in another dataset for some studies that integrate numerous datasets into one large dataset. Then count the number of surveys using the same static analysis method and choose accuracy, precision, and recall as metrics. These three performance metrics are the most often employed ones. Finally, using the same static analysis method, the minimum, maximum, mean, and standard deviation of the three performance indicators chosen for the study are determined.

In contrast to Model B, analysis and comparison revealed that Model A was superior to it despite the dataset's sample size. Furthermore, we divide these models into two groups: neural network models and non-neural network models. Then, using the same dataset, determine the average accuracy, fit, and recall of these two models. We finally reached the preliminary conclusion that neural network models are superior to non-neural network models after evaluating the performances of the two models.

It depicts how Android property-based techniques perform on Drebin, Genome, Virusshare, and VirusTotal. Neural network models perform significantly better than non-neural network models, with the exception of the correctness of the genome. Neural network models'[40] accuracy clearly outperforms non-neural network models, especially in regard to Virus share. While neural network models' accuracy on the genome is superior to non-neural network models', the two types of models' accuracy is comparable. This graph shows that the neural network model outperforms the non-neural network model in terms of overall performance.

11. Limitations

Construct validity, internal validity, and external validity pose the most challenges to the validity. The collection of studies is what determines the composition's validity. While this document makes every effort to assemble pertinent material from journals and conferences from seven internet databases, it's still likely that some of the publications have been missed. It exists. The possibility of process mistakes exists when studies are excluded based on inclusion or exclusion criteria, which is another component of construct validity. Use the cross-check procedure to examine the list of primary research publications in order to further avoid these mistakes. Data extraction and analysis are related to internal validity.

Data extraction and analysis are challenging tasks; therefore we also conduct cross-checks, accept the findings of comparisons, and receive the final data. You can still be extracting and interpreting your data incorrectly, though. These data should also be checked by the primary study's original author in order to prevent inaccuracies. An analysis of the findings of the main study is called external validity. There is a risk to the reliability of the RQ3 and RQ4 results. Because the research utilised for the comparison were inconsistent, this might not produce conclusive conclusions. As a result, we suggest developing a uniform platform to minimise inconsistencies in primary research. Additionally, we must gather additional data on static analysis's effectiveness in detecting Android malware.

It provides a thorough overview of Android malware detection using static analysis and summarises the most recent methodologies. Specifically, 98 studies between 2014 and 2020 carried out this SLR. The SLR also examines the different types of static analytics approaches, the procedure for conducting empirical trials, the effectiveness of various models for Android malware detection, and the capabilities of static analytics techniques for malware detection. We will employ static analysis to discuss and impact the detection of Android malware based on the findings of the primary investigation.

This SLR states that

1. Android characteristics are the most used static analysis technique for identifying Android malware.
2. In-lab data sets like Drebin and genome make up the majority of the demonstration experiment. Apktool is typically used as a support tool for static analysis in studies. The most used feature reduction method is IG. The most often utilised features are sensitive and privileged API requests. Out of all the models utilised, machine learning models account for the majority. The most widely used performance metric is accuracy.
3. The results of empirical research indicate that malware can be found using static analysis techniques.
4. We arrive at the preliminary conclusion that neural network models are superior to non-neural network models by examining and contrasting main studies. The SLR draws the conclusion that there are still some difficulties in identifying Android malware via static analysis based on the responses of the survey questions. We have instructions that recommend creating a number of novel approaches to enhance Android malware detection performance as well as setting up an integrated platform to properly evaluate the performance of the various techniques in order to address this issue.

Survey about Android Malware Detection:

Approach	Author	Description of method	Strengths	Weakness
Static analysis	[26]	Signature based	React efficiently to Android malware threats	High false negative
	[27]	Signature based	Detect unknown malware	Generate false positive
	[28]	Filtering	The degree of correctness is high	Resist latest malicious application
	[29]	Signature generation	Low computational overhead, the false positive rate is zero	It intercepts binders call and system calls
	[30]	Permission-based	Detect unknown malware	High false positive
	[31]	Permission and Application Programming Interface (API) based	Low cost of deployment	Cannot correctly detect Android malware
	[32]	Permission-based	Detect well-known malware	Contain some external dependencies
	[33]	Permission-based	Analyze manifest files to detect malware	Inadequate for detecting adware samples
	[34]	Contrasting Permission pattern	Error is minimal	The interval of the contrasting pattern is very wide
Dynamic Analysis	[35]	Boot sequence	Good compliments	High false negative
	[36]	Honey pot	Detect intent and session hijacking malware	Cannot detect root exploits and scripts malware
	[37]	Anomaly behavior monitoring	High efficacy with low overhead	Low performance and energy overhead
	[38]	System call logs	Run Android applications in an isolated environment	

Approach	Author	Description of method	Strengths	Weakness
Machine learning	[39]	Hybrid	Capture instantaneous attacks	Slightly lower True Positive Rate (TPR)
	[40]	Hybrid	It consumes low resources	Performance is reduced due to communication dependent on the server
	[41]	Back propagation Neural network	Achieves 0.982773 F-score	The number of the states of Markov chains quite large
	[42]	Ensemble learning	Detection with low false positive	Large feature space is required
	[43]	Deep learning (Deep Belief neural network)	Detects illegitimate applications with 96.7% accuracy	Unrealistic dynamic analysis of malicious may evade the detection system.
	[44]	Recurrent Neural Network and Convolutional Neural Network		
	[45]	Deep learning (Convolutional Neural Network)	A malicious application can be identified with an accuracy of 96–97%	Features captured were not confirmed.
	[46]	Artificial Immunity	High False negative rate	Inadequate dataset for experimentation
	[47]	Hidden Markov Model	Identify malicious applications with a precision of 0.96	The verification of the technique is done on a large dataset
	[48]	Feature vector	Detect the unknown malware effectively	Insufficient behavior features

12. Conclusion

On the basis of the dataset used in the literature, a survey was conducted while taking the accuracy of the procedures that were already in use into consideration. A promising strategy for identifying Android malware is proposed by the study. In terms of detecting accuracy, machine learning beat other strategies, particularly hybrid-based ones. Researchers must investigate further deep learning methods for the identification of Android malware in order to fully utilise the model. They also need to use a tonne of data to train the model.

References

[1] W.Enck, M.Ongtang, and P.McDaniel, “On lightweight mobile phone application certification,” in Proc. the 16th ACM conference on Computer and communications security (CCS), Chicago, USA, pp. 235-245, 2009.

[2] J.Kim, Y.Yoon, K.Yi, and J.Shin, “Static Analyzer for Detecting Privacy Leaks in Android Applications,” MoST, vol. 12, no. 110, 2012.

[3] L.Li, A.Bartel, T.F.Bissyande, J.Klein, and p.Mcdaniel.“IccTA: Detect- ing Inter-Component Privacy Leaks in Android Apps,” in Proc. the 37th International Conference on Software Engineering (ICSE), 2015, pp. 280-291.

[4] S.Alam, I.Traore, and I.Sogukpinar, “Annotated control flow graph for metamorphic malware detection,” The Computer Journal, vol. 58, no. 10, pp. 2608-2621, 2012.

[5] P.Faruki, A.Bharmal, V.Laxmi, V.Ganmoor, M.S.Gaur, M.Conti, and M.Rajarajan, “Android security: a survey of issues, malware penetration, and defenses,” IEEE communications surveys and tutorials, vol. 17, no. 2, pp. 998-1022, 2014.

[6] A.Feizollah, N.B.Anuar, R.Salleh, G.Suarez-Tangil, and S.Furnell, "An- droDialysis: Analysis of Android Intent Effectiveness in Malware Detec- tion," Computers and Security, vol. 65, pp. 121-134, 2017.

[7] S.Y.Yerima, S.Sezer, and I.Muttik, "High accuracy android malware detection using ensemble learning," IET Information Security, vol. 9, no. 6, pp. 313-320, 2015.

[8] Q.Jerome, K.Allix, R.State, and T.Engel, "Using opcode-sequences to detect malicious Android applications," in 2014 IEEE International Conference on Communications (ICC), pp. 313-320, Jun. 2014.

[9] J.Yan, Y.Qi, R.State, and Q.Rao, "LSTM-based hierarchical denoising network for Android malware detection," Security and Communication Networks, pp. 1-18, 2018.

[10] S.Arzt, S.Rasthofer, C.Fritz, E.Bodden, A.Bartel, J.Klein, Y.L.Traon, D.Octeau, and P.Mcdaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," Acm Sigplan Notices, vol. 49, no. 6, pp. 259-269, Jun. 2014.

[11] M.Fan, J.Liu, X.Luo, K.Chen, Z.Tian, Q.Zheng, and T.Liu "Android malware familial classification and representative sample selection via frequent subgraph analysis," IEEE Transactions on Information Forensics and Security, vol. 13, no. 8, pp. 1890-1905, 2018.

[12] S.Arzt, S.Rasthofer, R.Hahn, and E.Bodden, "Using targeted symbolic execution for reducing false-positives in dataflow analysis," in Proc. the 4th ACM SIGPLAN International Workshop on State of the Art in Program Analysis, pp. 1-6, Jun. 2015.

[13] S.Ni, Q.Qian, and R.Zhang, "Malware identification using visualization images and deep learning," Computers and Security, vol. 77, pp. 871-885, 2018.

[14] J.Y.Kim, S.J.Bu, and S.B.Cho, "Zero-day malware detection using trans- ferred generative adversarial networks based on deep autoencoders," Information Sciences, vol. 460, pp. 83-102, 2018.

[15] W.Wang, Y.Li, X.Wang, J.Liu, and X.Zhang, "Detecting Android mali- cious apps and categorizing benign apps with ensemble of classifiers," Future Generation Computer Systems, vol. 78, pp. 987-994, 2018.

[16] F.Dong, Y.Li, X.Wang, J.Liu, and X.Zhang, "Defect prediction in android binary executables using deep neural network," Wireless Personal Com- munications, vol. 102, no. 3, pp. 987-994, 2018.

[17] D.Gallingani and R.Gjomemo, "Static detection and automatic exploita- tion of intent message vulnerabilities in Android applications," 2015.

[18] Z.U.Rehman, S.N.Khan, K.Muhammad, J.W.Lee, Z.Lv, S.W.Baik, P.A.Shah, K.Awan, and I.Mehmood, "Machine learning-assisted signa- ture and heuristic-based detection of malwares in Android devices," Computers and Electrical Engineering, vol. 69, pp. 828-841, 2018.

[19] Allix, K., Bissyandé, T. F., Jérome, Q., Klein, J., & Le Traon, Y, "Empirical assessment of machine learning-based malware detectors for Android. Empirical Software Engineering, " 21(1), 183-211, 2016.

[20] H.Zhu, Z.Zhu, W.Shi, X.Chen, and L.Cheng, "DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model," Neurocomputing, vol. 272, pp. 638-646, 2018.

[21] S.Arshad, M.A.Shah, A.Wahid, A.Mehmood, H.Song, and H.Yu, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," IEEE Access, vol. 6, pp. 4321-4339, 2018.

[22] A.Saracino, D.Sgandurra, D.Dini, and F.Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 1, pp. 83-97, 2016.

[23] Z.Yuan, Y.Lu, and Y.Xue, "Droiddetector: android malware characteriza- tion and detection using deep learning," Tsinghua Science and Technol- ogy, vol. 21, no. 1, pp. 114-123, 2016.

[24] T.Ban, T.Takahashi, S.Guo, D.Inoue, and K.Nakao, "Integration of multi- modal features for android malware detection using linear svm," in Proc. the 11th Asia Joint Conference on Information Security (AsiaJCIS), pp. 141-146, Aug. 2016.

[25] S.Morales-Ortega, P.J. Escamilla-Ambrosio, A.Rodriguez-Mota, and L.D.Coronado-De-Alba, "Native malware detection in smartphones with android OS using static analysis, feature selection and ensemble clas-sifiers," in Proc. the 11th International Conference on Malicious and Unwanted Software (MALWARE), pp. 1-8, Oct. 2016.

[26] Kang, H., Jang, J.W., Mohaisen, A., Kim, H.K.: Detecting and classifying android malware using static analysis along with creator information. Int. J. Distrib. Sens. Netw. 11(6), 479174 (2015).

[27] Faruki, P., Laxmi, V., Bharmal, A., Gaur, M.S., Ganmoor, V.: AndroSimilar: robust signature for detecting variants of Android malware. J. Inf. Secur. Appl. 22, 66–80 (2015).

[28] Song, J., Han, C., Wang, K., Zhao, J., Ranjan, R., Wang, L.: An integrated static detection and analysis framework for Android. Pervasive Mob. Comput. 32, 15–25 (2016).

[29] Sun, M., Li, X., Lui, J.C., Ma, R.T., Liang, Z.: Monet: a user-oriented behavior-based malware variants detection system for Android. IEEE Trans. Inf. Forensics Secur. 12(5), 1103–1112 (2017).

[30] Rovelli, Paolo, Vigfússon, Ýmir: PMDS: permission-based malware detection system. In:Prakash, Atul, Shyamasundar, Rudrapatna (eds.) ICISS 2014. LNCS, vol. 8880, pp. 338– 357. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13841-1_19.

[31] Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M. and Wu, K.P.: DroidMat: android malware detection through manifest and API calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security (Asia JCIS), pp. 62–69. IEEE, August 2012.

[32] Talha, K.A., Alper, D.I., Aydin, C.: APK Auditor: permission-based Android malware detection system. Digit. Investig. 13, 1–14 (2015).

[33] Sato, R., Chiba, D., Goto, S.: Detecting Android malware by analyzing manifest files. Proc. Asia Pac. Adv. Netw. 36(23–31), 17 (2013).

[34] Ping, X., Xiaofeng, W., Wenjia, N., Tianqing, Z., Gang, L.: Android malware detection with contrasting permission patterns. China Commun. 11(8), 1–14 (2014).

[35] Vidal, J.M., Monge, M.A.S., Villalba, L.J.G.: A novel pattern recognition system for detecting Android malware by analyzing suspicious boot sequences. Knowl. Based Syst.150, 198–217 (2018).

[36] Shankar, V.G., Somani, G.: Anti-Hijack: runtime detection of malware initiated hijacking in Android. Procedia Comput. Sci. 78, 587–594 (2016).

[37] Saracino, A., Sgandurra, D., Dini, G., Martinelli, F.: Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Trans. Dependable Secur. Comput. 15(1), 83–97 (2016).

[38] Bläsing, T., Batyuk, L., Schmidt, A.D., Camtepe, S.A., Albayrak, S.: An android application sandbox system for suspicious software detection. In: 2010 5th International Conference on Malicious and Unwanted Software (MALWARE), pp. 55–62. IEEE, October 2010.

[39] Wei, L., Luo, W., Weng, J., Zhong, Y., Zhang, X., Yan, Z.: Machine learning-based malicious application detection of Android. IEEE Access 5, 25591–25601 (2017).

[40] Arshad, S., Shah, M.A., Wahid, A., Mehmood, A., Song, H., Yu, H.: SAMADroid: a novel 3-level hybrid malware detection model for android operating system. IEEE Access 6, 4321–4339 (2018).

[41] Xiao, X., Wang, Z., Li, Q., Xia, S., Jiang, Y.: Back-propagation neural network on Markov chains from system call sequences: a new approach for detecting Android malware with system call sequences. IET Inf. Secur. 11(1), 8–15 (2016).

[42] Yerima, S.Y., Sezer, S., Muttik, I.: High accuracy android malware detection using ensemblelearning. IET Inf. Secur. 9(6), 313–320 (2015).

[43] Yuan, Z., Lu, Y., Xue, Y.: Droiddetector: Android malware characterization and detection using deep learning. Tsinghua Sci. Technol. 21(1), 114–123 (2016).

[44] Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., Yagi, T.: Malware detection withdeep neural network using process behavior. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 577–582, June 2016.

[45] Hasegawa, C., Iyatomi, H.: One-dimensional convolutional neural networks for Android malware detection. In: 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA), pp. 99–102. IEEE, March 2018.

[46] Brown, J., Anwar, M., Dozier, G.: Detection of mobile malware: an artificial immunity approach. In: 2016 IEEE Security and Privacy Workshops (SPW), pp. 74–80. IEEE, May 2016.

[47] Canfora, G., Mercaldo, F., Visaggio, C.A.: An HMM and structural entropy based detector for android malware: an empirical study. Comput. Secur. 61, 1–18 (2016).

[48] Li, Y., Jin, Z.: An Android malware detection method based on feature codes. In: 4thInternational Conference on Mechatronics, Materials, Chemistry and Computer Engineering (2015).