Android Malware Detection Based on Static Analysis and Data Mining Techniques: A Systematic Literature Review

Hemant Rathore(✉), Soham Chari, Nishant Verma, Sanjay K. Sahay, and Mohit Sewak

Department of CS and IS, BITS Pilani, K K Birla Goa Campus, Goa, India
{hemantr,h20210029,h20200056,ssahay,p20150023}@goa.bits-pilani.ac.in

Abstract. Android applications are proliferating, which has led to the rise of android malware. Many research studies have proposed various detection frameworks for android malware detection. Literature suggests that static malware detection techniques are practical and assuring for detecting android malware. This paper presents a thorough survey of data mining-based static malware detection. We briefly discuss the growth of android malware and current detection techniques and offer a comprehensive analysis and summary of studies for each data mining-based malware detection phase, such as data acquisition, preprocessing, feature extraction, learning algorithms, and evaluation. Finally, we highlight some challenges and open issues in data mining-based android malware detection. This review will help understand the complete picture of static android malware detection and serve as a basis for malware detection in general.

Keywords: Android · Deep Learning · Machine Learning · Malware Detection · Static Analysis

1 Introduction

Android is the most popular operating system for smartphone devices. As of June 2022, Android owns 71.47% of the market share in mobile OS, with more than 2 billion users worldwide [1]. The Google Play Store, the official app store for Android, consists of 2.65 million applications for download. In addition, Android OS is also gaining popularity for tablet devices and various Internet-of-Things (IoT) devices, such as smart wearables and smart TVs. This ever-growing android market share has compromised android security to some extent.

Android provides various security features, such as the permissions system, to enhance the system's security. However, statistics show that as of March 2020, 482,579 new malware is generated monthly [2]. Due to an open market model, lack of isolation from third-party libraries, and easy-to-reverse-engineer applications, there has been significant growth in android malware. Various methods

W. Wang and J. Wu (Eds.): BROADNETS 2023, LNICST 511, pp. 51–71, 2023.
https://doi.org/10.1007/978-3-031-40467-2_4

have been proposed to enhance the security of the android system, including android malware detection.

Android malware detection can generally be classified into three categories: static, dynamic, and hybrid. Static detection analyzes suspicious android features and does not require running the android application. Dynamic detection performs analysis of the apps by executing them but demands much higher computational resources and time than static analysis. Hybrid detection combines both static and dynamic detection to obtain a balance between detection efficiency and effectiveness. Static analysis achieves high code coverage, and literature suggests it is effective for android malware detection [31].

Some previous studies have discussed static android malware detection. However, there are some limitations in previous analyses. With the emerging novel solutions to tackle android malware, it is necessary to be up-to-date with the current research progress in this field. However, previous studies have discussed now outdated research based on the surveyed literature. In addition, the surveys cover a narrow scope of data mining-based studies, specifically, novel unsupervised learning methods. Data mining-based solutions have proven effective for android malware detection as they can detect new malware, unlike signature-based solutions, which analyze specific patterns with existing known malware. They perform better in efficiency and effectiveness than the previously known approaches. Hence, this study presents a comprehensive survey of data mining-based research to overcome these limitations. The main contributions are:

- We present a thorough and systematic review of data mining-based android malware detection. This paper briefly discusses the background of android applications and malware and focuses on each phase of data mining-based malware detection, such as data acquisition, feature extraction, learning algorithms, and evaluation.
- We extensively discuss the novel unsupervised learning-based, reinforcement learning-based malware detection frameworks and supervised learning solutions to fill some research gaps in previous studies.
- We categorize the proposed solutions and learning algorithms for each section to better understand the advancement in data mining-based research in the field.
- Finally, we discuss new challenges and open issues in data mining-based solutions, such as explainability and IoT solutions, and provide insights for further research.

The rest of this paper is organized as follows. Section 2 describes the evolution of the malware industry, types of malware, and techniques used by malware designers. Section 3 describes the current malware detection techniques and frameworks. Performance metrics used to evaluate the models are discussed in Sect. 4. Section 5 presents the complete data mining-based framework for android malware detection. Previous studies in literature using machine learning and deep learning techniques are discussed in Sects. 6 and 7, respectively. Some open issues and challenges in android malware detection are presented in Sect. 8. Finally, Sect. 9 presents our conclusions.

2 Malware Evolution and Taxonomy

The term *Malware* is derived from *Mal*icious soft*ware*. Malware is a software program designed with malicious intent by the adversary or malware developer. Malicious software enters the computing environment without the user's knowledge and performs undesirable actions like stealing information, corrupting/deleting files, observing behavior like a spy, etc. Malware can be categorized into different types according to its intent and behavior.

2.1 Evolution of Android Malware

Initially, malware developers were focused on computing devices like desktops, laptops, etc. However, as mobile phone technology evolved and the number of mobile phone users increased, many malware developers shifted dynamically to attack mobile phones. Symbian and Windows were the starting mobile OS and were lucrative for malware developers. However, after the evolution and massive adoption of android mobile phones, it became a hotspot for attacks and the malware industry. Android phones gained popularity because of features such as WiFi, 3G, 4G, and 5G, which helped attackers quickly get into the user system. Another motive behind the increasing mobile malware was the growing and flourishing cyber attack industry selling users' private data in the black markets.

2.2 Malware and Its Classification

Malware is a malicious file or program, usually delivered over a network. It is often used to infect, explore, steal, or perform virtually any behavior that an attacker wants to perform against the target. Malware are categorized into different types according to their intent and behavior:

- A **virus** inserts itself in other standalone software programs and is invoked by that program to perform malicious activities or to spread itself [45]. The virus remains inactive until it is activated by the driver program.
- **Worm** is a standalone malicious program that runs independently to perform malicious activities [45]. It needs no driver program to run.
- **Trojan Horse** pretends itself to be a legitimate software program but acts maliciously in the background without user authorization or knowledge [9]. These are hard to recognize, and they attack in the backend.
- **Spyware** is a malicious program that spies on the user or computer system and monitors its activities without the user knowledge or consent [6].
- **Ransomware** installs covertly on a victim's computer and executes a crypto-virology attack that adversely affects it.
- **Bots** is a malicious program that allows the bot-master to remotely control the infected system [46].
- **Hybrid malware** combines two or more other forms of the above malware properties into a new type to achieve more powerful attack functionalities which drastically impact the infected system.

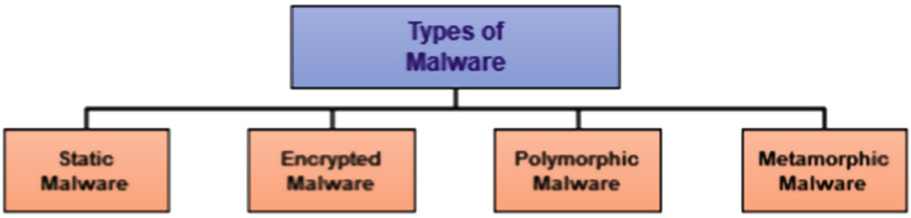

Fig. 1. Types of malware.

2.3 Obfuscation Techniques Used by Malware Designers

Malware can be detected using various malware detection techniques that prevent them from performing malicious activity against the target system. However, creating new malware does not require writing the whole malware code from scratch; instead, various obfuscation techniques can be used to generate many new variants of the same malware with minimum effort. Malware designers can use obfuscation techniques like garbage code insertion, changing control/data flow, package/class/method renaming, string/class encryption, or using reflection APIs. Some of the methods used to create new variants of malware are:

- **Encryption technique** is used to encrypt the malware program so that the signature of the encrypted malware does not match with the signature of the original malware. Identification and decryption of such malware is a challenging task.
- **Polymorphism** includes malware that constantly changes its identifiable features based on a mutation engine that uses various obfuscation techniques to avoid detection, making it challenging to comprehend.
- **Metamorphism** includes body polymorphic malware, as it replicates malware into a new malicious code with minimum to no resemblance to the older version. The mutation engine is also mutated to evade any signature-based detection.

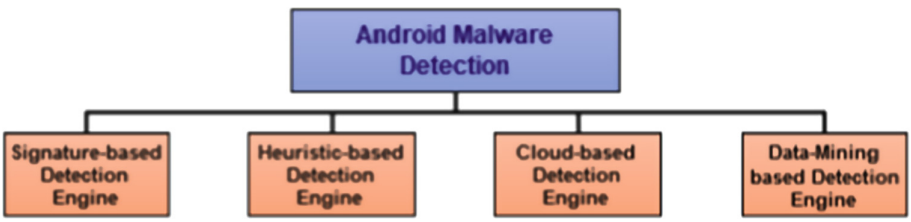

Fig. 2. Android malware detection engines.

3 Current Malware Detection Engines

The anti-malware industry and research community design, develop and deploy various malware detection engines to detect new as well as old malware. These currently include signature, heuristic, and cloud based detection engines.

- **Signature Engine:** All malicious software, programs, and files have a digital footprint/behavior. A signature is a unique string from binary code that identifies a malicious sample/footprint/behavior [47]. If the test file's signature matches the known malware's signature, then the test file is classified as malware otherwise benign.
  Signatures are often manually generated, disseminated, and maintained by domain experts, which are human-driven processes and thus are a bottleneck. The malware designers have also developed tools/techniques that modify a malware sample using obfuscation while maintaining the malicious behavior to generate a new malware variant. The newly created malware variants will be able to evade/fool/bypass the signature-based detection engine, thus making it less effective.
- **Heuristic Engine:** follows a proactive approach where the domain experts design and develop rules/patterns to discriminate malware and benign files/properties/behavior, etc. [47]. The design rules/patterns are generic enough to detect variants of the same malware family. However, rule/pattern generation is again a human driven process and is slow and error-prone.
  Figure 3 shows the design of traditional anti-malware/anti-virus software

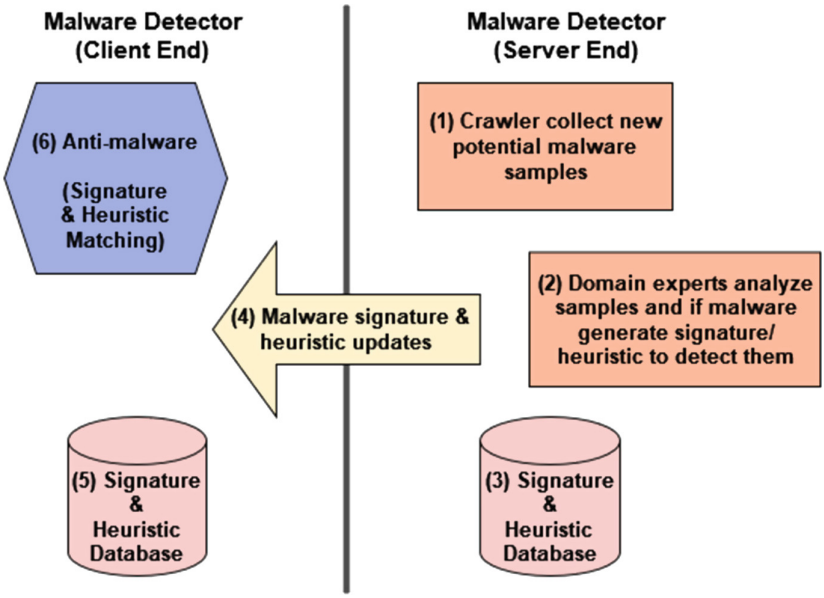

Fig. 3. Design of traditional anti-malware system.

based on signature and heuristic engines. It consists of two parts: malware detector (client end) deployed on smartphone/laptop/desktop etc., and malware detector (server end) deployed on anti-virus company infrastructure. First, crawlers deployed by the anti-virus company scan the internet to find new potential malware samples. Then, domain experts analyze these samples to find new malware samples. Later, signature/heuristic is developed to detect these new malware samples and is stored in the signature & heuristic database. The database is synced from the server end to the client end using regular updates. The anti-virus on the client end executes continuously by taking test samples and generating the signature of the test sample. This signature is compared with the existing malware signature in the database (client side), and if matched, then the test sample is declared malware otherwise benign.

These traditional anti-malware/anti-virus software has two bottlenecks. First, malware signature/heuristic generation is human-driven and cannot cope with the exponential growth in malware numbers seen in the last decade. Second, with the ever-increasing number of malware signatures/heuristics, searching in the database has become slow.

- **Cloud-based Approach:** is a two-layer malware detection strategy comprising a client machine at the first layer and server infrastructure at the second layer. First, the client machine analyzes and verifies the signature/heuristic of the test sample. If found malicious, then the test sample is declared as malware. Otherwise, the test sample is sent to the server for deeper analysis. The server infrastructure has additional malware detection mechanisms from different antivirus engines, and they use voting methods to classify the test sample as malicious or benign. Kedarnath et al. used system calls and network traffic to develop a prototype of cloud-based android botnet malware detection to predict the botnet family of an application [48]. Although the cloud-based approach provides two-layer detection, it is computationally very expensive and needs high-speed network connectivity.
- **Data mining-based Approach:** is used to develop the next-gen state-of-the-art malware detection systems, and it follows a two-step process: feature extraction followed by classification/clustering. The feature extraction step involves performing static/dynamic/hybrid analysis of samples to extract relevant features that help in malware detection. The extracted features are then passed to data mining techniques like classification (supervised learning) or clustering (unsupervised learning) algorithms to develop malware detection models. Abhishek et al. performed static analysis to extract android permissions and used random forest classifiers to develop an android malware detection model that achieved an AUC of 0.817 [48]. Chao et al. used a support-based permission method to mine unique patterns for malware detection [48]. The model achieved 94% accuracy and a very low false positive rate of 5%. The anti-malware industry has already started exploring data mining-based malware detection engines along with traditional engines.

4 Performance Metrics for Data-Mining Based Approach

The data mining-based malware detection models can be developed using various supervised or unsupervised learning algorithms. The performance of these detection models can be measured using the following performance metrics:

4.1 Performance Metrics for Supervised Learning Based Malware Detection Models

Supervised learning algorithms require a dataset that contains samples (malware and benign) and their class labels. The malware detection model is then trained on the labeled data using supervised learning algorithms, and the model is used to perform predictions for unlabelled data in the future.

Supervised learning requires dividing the dataset into two parts-*training set* and *testing set*. The sets must be mutually exclusive to remove any radicalization or biasness in the final results. Algorithms like *hold-out*, *cross-validation* and *bootstrapping* can be used to divide the dataset. For example, the dataset can be divided such that 70% of samples are used for training, and the remaining 30% are used for testing. After dividing the dataset, the malware detection model is trained on *training set* using supervised learning (classification algorithm). The performance of the malware detection model is then evaluated on *test set* using the following performance metrics:

- **True Positive** (TP) is the number of samples that are actually malicious and are also classified as malware by the malware detection model.
- **True Negative** (TN) is the number of samples that are actually benign and are also classified as benign by the malware detection model.
- **False Positive** (FP) is the number of samples that are actually benign but are wrongly classified as malware by the malware detection model.
- **False Negative** (FN) is the number of samples that are actually malicious but are wrongly classified as benign by the malware detection model.
- **Accuracy** is the percentage of samples in the dataset that are correctly classified (malware or benign) by the malware detection model.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- **Error Rate** is the percentage of samples in the dataset that are wrongly classified (malware or benign) by the malware detection model.

$$ErrorRate = \frac{FP + FN}{TP + TN + FP + FN} \tag{2}$$

- **Precision** is the percentage of correctly predicted malware samples over all the samples that are predicted as malicious by the malware detection model.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

– **Recall** is the percentage of samples that are correctly predicted as malware by the malware detection model over all the malware samples in the dataset.

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

– **F-score** (F1 score) is the harmonic mean of precision and recall.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5}$$

– **AUC** is the area under the receiver operating characteristic, and it is used to show the prediction ability of the malware detection model under various discrimination thresholds.

4.2 Performance Metrics for Unsupervised Learning Based Malware Detection Models

Unsupervised learning based malware detection models can be evaluated using *internal* and *external* evaluation metrics. Internal metrics include *Rand index*, *Fowlkes-Mallows scores*, *Silhouette Coefficient*, *Calinski-Harabasz Index*, *Davies-Bouldin Index* etc. Rathore et al. performed clustering for malware detection on the Drebin dataset and used silhouette coefficient, and calinski-harabasz index for determining the quality of clusters [35]. Lou et al. in TFDroid use clustering to first divide android applications into various clusters and then detect malware applications in each cluster [27]. On the other hand *external* evaluation is used when both unsupervised and supervised learning are employed to detect malware. Here, the performance is determined by observing the system's overall improvement with or without unsupervised learning. Rathore et al. developed a malware detection model using a random forest classifier that achieved an AUC of 99.4%. However, when they first performed data clustering followed by a classification model in each cluster, the AUC was boosted to 99.6% [35].

4.3 Other Performance Metrics

The data preprocessing step includes methods to improve the understanding of feature vectors before constructing malware detection models. *Physical* and *Logical distance* can be used to compare different tuples/attributes in the feature vector. Physical distance can be measured using *Euclidean distance*, *Manhattan Distance*, *Mahalanobis distance* etc. In contrast, logical distance can be calculated using probabilistic behavior in which past uncertainties are used to calculate future uncertainties. *Pearson correlation coefficient* followed by *Heat Map* can also be employed to understand the relationship between different features or classes. Other correlation metrics includes *Chi-square*, *Fisher score and Matthews correlation* etc. Shabtai et al. employed chi-square, information gain, and fisher score along with Bayesian Network to construct a malware detection model and achieved the highest accuracy of 92% using information gain [43]. Rathore et al. developed heat maps to understand the correlation between android permissions in malware and benign applications [34].

5 Data Mining-Based Malware Detection Models

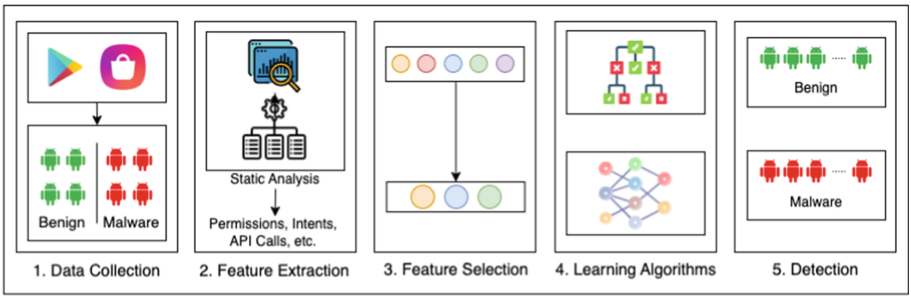

Fig. 4. Framework for developing malware detection system based on data mining techniques.

Figure 4 illustrates the framework for constructing malware detection models based on data mining techniques. The first step is *Data collection* which involves collecting malicious and benign applications for the dataset. The second step is *Feature Extraction*, wherein the application files are reverse-engineered, and features are extracted. The next step is *Feature Selection*, in which the less critical features are removed, and the most relevant features are selected. Further, these features are passed onto the *Learning Algorithm*, wherein a classification model is trained. Additionally, this model is used to classify applications as malware or benign during the testing phase. Each of the steps is discussed in detail in the following sections.

5.1 Data Collection: Android Applications Acquisition

The quality of data (android applications) plays a crucial role while constructing malware detection models. The malware and benign android applications (APKs) in the dataset are generally collected from official app stores such as *Google Play* store and other third party app stores like *apkmirror*, *apkpure*, *F-droid*, *GetJar* etc. Santos et al. collected 13,189 malicious applications from *VxHeavens* and 13,000 benign applications from various trusted sources and scanned using *ESET NOD32* antivirus to ensure that the benign applications were safe [48]. Gao et al. used a benign dataset of 49,000 applications downloaded from *PlayDrone* whereas the malicious applications were obtained from the *Android Malware Genome Project* [11]. Various authors have collected malicious android applications and shared them with the research community like *Drebin* by Arp et al. [4], *Android Malware Dataset* by argus lab [49], *PRAGuard* by University of Cagliari [28], *Kharon* by INRIA [20], etc.

Table 1. Various feature extraction techniques for android applications.

Static Analysis	Dynamic Analysis	Hybrid Analysis
Android application is analysed by parsing its code without executing it	Android application is executed in controlled environment	Android application is analysed using both static and dynamic analysis
Static Features: Permission, Intent, Opcode, etc.	Dynamic Features: API Calls, System Calls etc.	Both static and dynamic features
Scan the code using various parsers	Execution environments: debugger, simulator, emulator, virtual machine	Polyunpack: hidden codes compared with runtime execution + instances with runtime codes
Computationally expensive	Poor performance with trigger based malware	Sometimes computationally very expensive
Better overall code coverage	Poor/Limited coverage problem	Good overall code coverage
Inability due to undecidability, lack of support for runtime packages, limitation to obfuscation	Efficient in analyzing packed malware, more time consuming, requires more resources than static	80:40 ratio makes it more efficient as compared to both the approaches

5.2 Feature Extraction and Representation

Feature Extraction. involves finding features or data patterns that can represent android applications during model development. Features can be extracted using static, dynamic, or hybrid analysis of android applications. *Static analysis* requires parsing the code without executing it. Static analysis of android applications will generate static features like permissions, intent, opcodes, etc. Rathore et al. performed the static analysis of android applications and extracted android permission for malware detection models [34]. On the other hand, *Dynamic analysis* requires executing the code in a controlled environment. Dynamic analysis of android applications will result in dynamic features like system calls, API calls, etc. Fan et al. performed the dynamic analysis and extracted API calls for malware detection. The *hybrid analysis* allows both static as well as dynamic analysis of android applications to extract features. Table 1 illustrates a brief comparison between static, dynamic, and hybrid analysis of android applications. This survey paper extensively focuses on static analysis and static features for android malware detection. Figure 5 shows the percentage of existing papers using a particular static feature for malware detection. Android permission is the most used static feature, followed by API calls and opcode sequences. Other static features include intent, components etc.

Feature Representation. The extracted features comprise raw opcode sequence and *classes.dex* with bytecode. The *classes.dex* contains all the compiled objects used at the run time in the Dalvik environment. These features need to be fed to the classification model in certain representations. These representations are then directly passed to the deep learning models (neural models with n-gram encoding technique) for better classification and characterization. Three different feature representation forms are used: vectorized, tree-based, and graph-based.

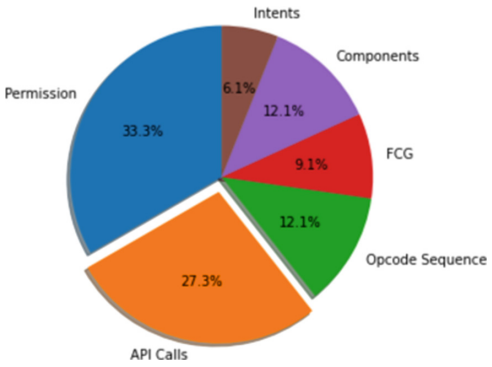

Fig. 5. Distribution of android features for malware detection.

- **Vectorized** representations are the most widely used feature representations in android malware detection and classification. It uses a one-hot encoding technique for vector construction. 'Term Frequency-Inverse Document Frequency' (TF-IDF) and word embedded technique are some of the popular methods. Abhilash et al. used the Doc2Vec technique to create a vectorized representation of a doc file [13]. They obtained a detection accuracy of 95.3% using the LSTM model when trained on 3500 malicious and 2700 benign samples. Karbab et al. proposed the MalDozer framework, which achieved an accuracy of 99.84% [18]. The one-hot encoding technique was used for the vector representation of 33,000 malicious and 33,000 benign applications.
- An abstract syntax tree is formed for android applications' source code in **tree-based** representation. The tree nodes represent the constructs of each source code obtained from the decompression of APK files. Fan et al. used tree-based feature representation for 251 benign and 7000 malicious applications [48]. The decision tree classifier was used with the API calls as the feature set and obtained 95.9% accuracy.
- In **graph-based** representation, android apps are represented using graphs like CFG and IDFG, which help in categorization and are challenging to be attacked by malware. These graphs can be directly fed to the DL models or represented in vectorized form. Jerome et al. used CFG as the feature selection algorithm and created a model with an F-measure score of 0.89 [17]. Gao et al. used CFG for 49,000 benign and 1249 malicious applications [11]. SVM-GFD and SVM-DFD classifiers were used in which SVM-GFD performed better, with an accuracy of 87.85%.

  All of these approaches mentioned above are used to improve the performance and efficiency of the models. The ideal feature representation for detection helps in faster training and better accuracy of the classification models.

5.3 Data Pre-processing

Data Preprocessing and feature selection are two critical aspects of machine learning algorithms. The dataset obtained generally contains a lot of inconsistencies, noise, and missing values which makes the data representation skewed. Hence, it is necessary to preprocess the data using the following steps:

- **Data Cleaning:** The data inconsistencies created due to noise are removed, and missing values are handled.
- **Data Integration:** Data obtained from various sources are merged, and redundant data is removed.
- **Data Reduction:** The main aim is to reduce the size of the data by removing irrelevant features. Less important features are identified using the correlation information and removed. Some algorithms used for data reduction are Wavelet transformation, PCA, clustering, and sampling.
- **Data Transformation:** Normalization of data is performed using various techniques. Data with similar functionality are aggregated, and metadata and multi-valued data are handled. Massarelli et al. calculated DFA coefficients for features and given to wrappers for classifier exploitation [32]. Further reduction in the subset is performed after mutual information is collected and passed to the PCA algorithm.

5.4 Feature Selection

Feature selection aims at the identification and elimination of redundant features which have little to no role in the performance improvement of the original model. VetDroid performs analysis on fine-grained android permissions [53]. It proved its usefulness by digging out all the essential syscalls required by the system. They analyzed 1,249 free apps obtained from the Google Play Store. The feature selection process can be done by ranking each feature based on its importance and removing the less critical features or identifying subsets of the most relevant features.

Feature Ranking: Each feature of the set is ranked based on its behavior and importance using pre-existing feature selection algorithms. Some algorithms use mathematical calculations for this determination, while others use rough set theory for the selection process. Using fuzzy rough sets for feature selection has proved its efficiency over rough set theory [16]. Feature grouping was advised to perform better than individual ranking, suggesting that grouped features reduce the time to reach the optimal subset than individual features. Shabtai et al. used 12,075 sample vectors from two devices comprising 1,781 malicious vectors [32]. They suggested that the focus should be on monitoring the malicious processes and permissions rather than the whole system. In addition, they also presented the usage of unsupervised learning over supervised learning to increase malware detection accuracy.

Subset Determination and Verification: Feature selection algorithms are used to identify feature subsets consisting of the most significant features and excluding the least important ones. Tam et al. demonstrated the ensemble feature selection that considers all the feature subsets ranked using different feature selection algorithms [48]. Three significant approaches that are used for feature selection are: filter, wrapper, and embedded.

- **Filter** is the low cost operation focusing on ranking the features based on a score. These rankings are independent of the later stage computation and hold good generalization ability.
- **Wrapper** is a black box technique to determine the best performing subset. It is costly and time consuming. It indentifies different subsets and generates predictions to evaluate the quality of the generated feature subset. The issue of overfitting persists in this approach.
- **Embedded** approach captures the dependencies in a better way and consumes less time than wrappers but is still slower than the filter approach. In addition, it may also suffer from overfitting.

Filter methods are mostly used in malware detection as they are the most scalable and efficient. Some of the examples of filter method include *Document Frequency*, *Information Gain*, *Fisherman Selection* and *Maximum Relevance*.

5.5 Different Learning Techniques

Following are various learning approaches that are used to train a malware detector. Each of these methods is suitable for different scenarios.

Supervised Learning: A labeled dataset is used in supervised learning that helps to predict either a discrete variable (classification) or a continuous variable (regression). Liu et al. used supervised learning algorithms like J48 decision tree algorithm and obtained an accuracy of 98.4% [26]. Shabtai et al. extracted 22,000 features, but used 9,898 after data preprocessing using feature selection [43]. They used a top-performing feature set from a group of 50, 100, 200, 300, 500, or 800 features. These datasets were provided to the decision tree and naive bayes algorithm, which gave an accuracy of 90% and 82% with FPR of 0.38 and 0.22, respectively.

Unsupervised Learning: Unsupervised learning is used for unlabeled data where samples are grouped at the time of feature selection and clustering. This approach is quite popular in problems like feature dimension reduction. K. Liu et al. differentiated and mentioned that algorithms like PCA and K-means perform better in unsupervised learning [26]. TFDroid used k-means to cluster data [27]. They tested 2 to 70 data clusters and finalized 32 clusters. They used 5,161 benign applications and 3,247 malicious applications. Results showed that clustering worked better than the non-clustering approach. The clustering approach achieved an accuracy of 90% and a TP of 93.65%, whereas non-clustering gave 84.31% accuracy and a TP of 85.51%.

Semi-supervised Learning: Semi-supervised learning is a combination of supervised and unsupervised learning. Santos et al. proved that mixing supervised and unsupervised performed better. They used the Learning with Local and Global Consistency (LLCG) algorithm with 1,000 malicious and 1,000 benign

Table 2. Malware detection using machine learning algorithms.

Year	Author(s)	Dataset	Performance	Comment(s)
2012	Sarma et al. [37]	121 (M) 1,58,062 (B)	Detection Rate 81% (SVM)	(+) extensive study of amdroid permission (−) malware dataset is too small
2013	Gascon et al. [12]	12,158 (M) 1,35,792 (B)	Detection Rate: 89% (SVM)	(+) can detection local obfuscation (−) expensive (−) evasion possible
2014	Jerome et al. [17]	11,960 (M) 12,905 (B)	F-measure 0.8931 (SVM)	(+) used digital certificate to improve classification performance (−) use only opcode sequence as feature
2014	Chan et al. [8]	175 (M) 621 (B)	Accuracy 86.33% (NB)	(+) used two features (permission & API calls) (−) limited work, small dataset
2016	Fereidooni et al. [10]	Dataset 1: 11,000 (M) 11,000 (B) Dataset 2: 18,766 (M) 11,187 (B)	F1 score Dataset1: 96% (XGB) Dataset2: 97% (XGB)	(+) used many features (+) performed feature selection (−) investigated static features only
2016	Li et al. [25]	388 (M) 322 (B)	AUC 0.8248 (NB)	(+) feature reduction (−) poor performance (−) no comparison with other classifiers
2017	Yanf et al. [50]	640 (B) 640 (M)	Accuracy 95.42% (RF)	(+) use image classification (−) limited dataset
2017	Narayanan et al. [29]	5560 (M) 5000 (B)	Accuracy 89.92% (OL)	(+) use dependency graph with context and sensitive information (−) curse of dimensionality due to many feature
2018	Koli [21]	120 (B) 175 (M)	Accuracy 97.77% (DT)	(+) static and dynamic analysis (−) small dataset
2019	Lou et al. [27]	5560 (M) 5161 (B)	Accuracy 93.7% (k-means)	(+) used clustering for malware detection (+) bag of words as feature (−) curse of dimensionality
2019	Sharma et al. [44]	5531 (M) 4235 (B)	Accuracy 96.32% (RF)	(+) group formation based on dangerous permissions (−) some subgroup has limited samples
2020	Rathore et al. [34]	5,560 (M) 5,721 (B)	Accuracy 94% (RF) AUC 98.1% (RF)	(+) exhaustive study of android permissions (+) used many ML and DL classifiers (−) investigated only one feature
2020	Rathore et al. [35]	5,560 (M) 5,592 (B)	Accuracy 95.7% (RF) AUC 99.4% (RF)	(+) exhaustive feature selection (+) combined clustering and classification (−) only opcode used
2021	Lee et al. [22]	2,500 (M) 5,000 (B)	Accuracy 98.1% (MLP) F1 score 97.1% (MLP)	(+) genetic algorithm-based feature selection (+) used many ML classifiers (−) lower performance than non-selection
2022	Zhang et al. [52]	125 (M) 300 (B)	Accuracy 99.34%	(+) early detection (+) high performance (−) small dataset

applications and achieved an accuracy of 88% on 60% of labeled and 40% of unlabelled data. Xabier et al. also suggested that the model performed better with an accuracy of over 90% with 10% labeled data and 90% unlabeled data, and it depreciated once the amount of labeled data increased [48].

6 Malware Detection Using Machine Learning

Table 2 summarizes various machine learning-based approaches for malware detection. The extensive study of various approaches has led to specific observations, which are highlighted as the advantages and areas of improvement. Fereidooni et al. achieved high accuracy of 97% even though they used unseen malware samples, which imbalanced the dataset during the testing phase [10]. Some works [21,25] achieved good detection accuracy using NB and DT techniques. However, the models were trained on a minimal amount of data. In [29], a combination of dependency graphs is proposed to classify applications as malware or benign. The algorithm using digital certificates proposed in [17] obtained a confidence interval of 99% but is ineffective with advanced obfuscation and growth of feature space dynamically. A novel clustering-based malware detection is proposed in [27]. They investigated 10000 samples and achieved a high detection accuracy of 93.7%. In [37], authors achieved a detection accuracy of 81% with SVM, which used signal trigger function as a metric. However, the dataset was highly imbalanced, with around 158000 benign samples and only 121 malicious samples. This posed a challenge to the efficacy of the model. Rathore et al. combined clustering and classification techniques on the opcodes feature set and obtained the highest accuracy of 94% using Random Forests [35]. Lee et al. performed feature selection using Genetic algorithm and showed that time costs of nine classification algorithms significantly reduced while maintaining good performance [22]. Zhang et al. proposed an early detection framework using system calls and six ML algorithms. They achieved an average early detection accuracy of 99.34%.

7 Malware Detection Using Deep Learning

Similarly, various Deep Learning methods proposed in previous work for malware detection are summarized in table 3. Huang et al. have used one convolution layer with API calls to build neuron matrices [15]. However, the approach followed enabled only offline detection. Lee et al. used the n-gram based RNN and CNN networks for detection which proved to be effective and reliable [23]. Hota et al. used multi-layer data passing technique with doc2vec as a learning algorithm and feature recombination helped in achieving a detection accuracy higher than 95% [13].

Some of the previous research also includes inherent relationship methodologies along with various algorithms of one hot encoding vector, bag of words, and doc2vec learning. Yuan et al. used the Boltzmann machine which uses permissions as the static feature and showed that it performed better than ML

Table 3. Malware detection using deep learning algorithms.

Year	Author(s)	Dataset	Algorithm	Performance
2014	Yuan et al. [51]	250 (M) 250 (B)	Android permission Restricted boltzman machine Deep belief network	Accuracy 96.5%
2016	Hou et al. [14]	2500 (M) 2500 (B)	API calls API call blocks Deep belief network	Accuracy 96.66%
2017	li et al. [24]	5,560 (M) 1,23,453 (B)	Permission API calls and other information Deep neural network	F1 96.08%
2018	Karbab et al. [18]	33,000 (M) 33,000 (B)	API calls one hot encoding Convolutional neural network	F1 96-99% FPR 0.06%–2%
2018	Booz et al. [5]	48,643 (M + B)	Permission Deep neural network	Accuracy 95% F1 93%
2019	Hota and Irolla [13]	Dataset 1: 3500 (M) 2700 (B) Dataset 2: 1,50,000 (M) 1, 50, 000 (B)	Doc2Vec Long short term memory	Accuracy 95.3%
2019	Huang et al. [15]	3,697 (M) 3,312 (B)	API based feature graph Feature selection-Top-20 APIs Convolutional neural network	F1 94.3%
2019	Oak et al. [30]	120,780 (M) 60,390 (B)	Imbalanced data Sequence Modeling Permission BERT and LSTM	F1 score 0.919
2019	Lee et al. [23]	12 lakh (M) 10 lakh (B)	Permission Intent Ngram Convolutional neural network Recurrent neural network	AUC 0.9986
2020	Sewak et al. [38]	5,560 (M) 5,721 (B)	Intent Deep neural network	AUC 0.814 Accuracy 77.2%
2021	Almahmoud et al. [3]	1,430 (M) 1,390 (B)	Permission API Calls System Events Permission Rate Recurrent neural network	Accuracy 98.58%
2021	Cai et al. [7]	7,362 (M) 35,948 (B)	Function embedding Function Call Graph Graph convolutional network	Accuracy 99.81%
2022	Kim et al. [19]	9,000 (M) 9,000 (B)	API call Graph Convolutional neural network	Accuracy 91.27%

models [51]. The DBN outperformed several DL models with a detection accuracy of 96.66%. Li et al. also used API calls and permissions as the feature set with the backpropagation Deep neural network model and achieved a detection accuracy of 97.16% [24]. This model also succeeded in correctly identifying malware families. The dataset contained more than 120000 benign and 5560 malware samples. Almahmoud et al. used permissions, API calls, system events and permission rate as the feature set and proposed a novel RNN architecture that outperformed ML algorithms witn an accuracy of 98.58% [3]. Cai et al. proposed Enhanced Function Call Graphs (E-CFGs) and Graph convolutional networks to generate vectorized representations for app runtime behaviors. Results showed that various ML algorithms performed better with E-CFGs than static features [7]. Kim et al. proposed MAPAS, a CNN-based approach using API call graphs as the feature set. It performed 145.8% faster and consumed less memory than previous state-of-the-art approach and achieved higher accuracy of 91.27% [19].

8 Issues, Challenges and Future Work

Existing literature suggests that data mining-based frameworks for android malware detection have proven effective and efficient. However, some ongoing challenges still need to be addressed. This section discusses open issues and challenges in static android malware detection and provides future research directions.

- **Lack of Representative Datasets:** All android malware detection frameworks require a representative dataset of benign and malware (or malicious) android applications. The dataset collection required collecting android applications from various markets, such as Google Play Store and other third-party markets, and web crawling. This usually led to an unbalanced set of malware and benign applications with very few samples from the malware class. The DREBIN provided a collection of 5560 malware applications from 179 various malware families [4]. However, up-to-date representative datasets covering a wide range of malware categories are necessary to develop data mining-based malware detection frameworks.
- **Incremental Learning** The trained malware detection models must be effective against the repository of known malware as well as newly generated malware. The data mining-based algorithms should also consider the recent malware samples to maintain their effectiveness. The frameworks should dynamically update the training sets to include the new malicious examples while retaining the properties of historical data.
- **Active Learning** Selecting representative samples from a sizeable collection of unknown samples helps improve detection accuracy. Active learning, an effective learning strategy to reduce the cost of acquiring labeled examples and addressing data scarcity problems, has still been used in limited malware detection research. Moscovitch et al. used active learning techniques to showcase the efficiency of the acquisition process and improvement in the

classifier. The framework enabled the selection of representative samples from more than 30,000 files.

- **Adversarial Learning** The rise of data mining-based malware detection techniques has led to malware attackers fooling trained machine learning models using adversarial attacks. These attacks perform perturbations in the training/testing data and force the misclassification of malware samples as benign. Sewak et al. suggested a framework to reduce the impact of adversary attacks in intrusion detection, spam detection, fraud detection, and counter-terrorism [39,40,42]. Hence, the detection models must be highly accurate in predicting malicious applications and robust to such attacks [33,36,41].
- **Explainability** Data mining-based malware detection models trained on a dataset comprising malicious and benign applications can detect malware in any given application during the testing phase. However, most of these methods cannot justify the decision to classify the application as malware or benign, i.e., the results obtained through primary data mining-based algorithms are not explainable enough. Therefore, the research should focus on the explainability of the models as equally as the performance.

9 Conclusion

The application of data mining-based android malware detection is a significantly developing research topic with many unsolved and unexplored challenges. In this article, we studied the evolution and growth of android malware and the current malware detection techniques and frameworks. We surveyed several data mining-based static malware detection techniques and presented a complete framework for data mining-based android malware detection. Our extensive survey highlights critical observations for each literature and provides insights for further research. Finally, we discuss some research gaps and open issues in this field. We hope this work will boost data mining-based malware detection and inspire researchers to pursue new research avenues.

References

1. Android - Statistics & Facts. https://www.statista.com/topics/876/android/
2. Development of new android malware worldwide. https://www.statista.com/statistics/680705/global-android-malware-volume/
3. Almahmoud, M., Alzu'bi, D., Yaseen, Q.: ReDroidDet: android malware detection based on recurrent neural network. Procedia Comput. Sci. **184**, 841–846 (2021)
4. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: DREBIN: effective and explainable detection of android malware in your pocket. In: Network and Distributed System Security Symposium (NDSS), vol. 14, pp. 23–26 (2014)
5. Booz, J., McGiff, J., Hatcher, W.G., Yu, W., Nguyen, J., Lu, C.: Tuning deep learning performance for android malware detection. In: 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 140–145. IEEE (2018)

6. Borders, K., Prakash, A.: Web tap: detecting covert web traffic. In: 11th ACM Conference on Computer and Communications Security (CCS), pp. 110–120 (2004)
7. Cai, M., Jiang, Y., Gao, C., Yuan, W.: Learning features from enhanced function call graphs for android malware detection. Neurocomputing **423**, 301–307 (2021)
8. Chan, P.P., Song, W.K.: Static detection of Android malware by using permissions and API calls. In: International Conference on Machine Learning and Cybernetics, vol. 1, pp. 82–87. IEEE (2014)
9. Craig-Lees, M.: Sense making: trojan horse? Pandora's box? Psychol. Mark. **18**(5), 513–526 (2001)
10. Fereidooni, H., Conti, M., Yao, D., Sperduti, A.: ANASTASIA: ANdroid mAlware detection using STatic analySIs of applications. In: 8th International Conference on New Technologies, Mobility and Security, pp. 1–5. IEEE (2016)
11. Gao, T., Peng, W., Sisodia, D., Saha, T.K., Li, F., Al Hasan, M.: Android malware detection via graphlet sampling. IEEE Trans. Mob. Comput. **18**(12), 2754–2767 (2018)
12. Gascon, H., Yamaguchi, F., Arp, D., Rieck, K.: Structural detection of android malware using embedded call graphs. In: ACM Workshop on Artificial Intelligence and Security, pp. 45–54 (2013)
13. Hota, A., Irolla, P.: Deep neural networks for android malware detection. In: International Conference on Information Systems Security and Privacy (ICISSP), pp. 657–663. IEEE (2019)
14. Hou, S., Saas, A., Ye, Y., Chen, L.: DroidDelver: an android malware detection system using deep belief network based on API call blocks. In: Song, S., Tong, Y. (eds.) WAIM 2016. LNCS, vol. 9998, pp. 54–66. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47121-1_5
15. Huang, N., Xu, M., Zheng, N., Qiao, T., Choo, K.K.R.: Deep android malware classification with API-based feature graph. In: IEEE TrustCom/BigDataSE, pp. 296–303. IEEE (2019)
16. Jensen, R., Shen, Q.: Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches. IEEE Trans. Knowl. Data Eng. **16**(12), 1457–1471 (2004)
17. Jerome, Q., Allix, K., State, R., Engel, T.: Using opcode-sequences to detect malicious android applications. In: IEEE ICC, pp. 914–919. IEEE (2014)
18. Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: MalDozer: automatic framework for android malware detection using deep learning. Digit. Investig. **24**, S48–S59 (2018)
19. Kim, J., Ban, Y., Ko, E., Cho, H., Yi, J.H.: MAPAS: a practical deep learning-based android malware detection system. Int. J. Inf. Secur. **21**, 1–14 (2022)
20. Kiss, N., Lalande, J.F., Leslous, M., Tong, V.V.T.: Kharon dataset: android malware under a microscope. In: The LASER Workshop 2016, pp. 1–12 (2016)
21. Koli, J.: RanDroid: android malware detection using random machine learning classifiers. In: IEEE ICSESP. pp. 1–6. IEEE (2018)
22. Lee, J., Jang, H., Ha, S., Yoon, Y.: Android malware detection using ml with feature selection based on the genetic algorithm. Mathematics **9**(21), 2813 (2021)
23. Lee, W.Y., Saxe, J., Harang, R.: SeqDroid: obfuscated android malware detection using stacked convolutional and recurrent neural networks. In: Alazab, M., Tang, M.J. (eds.) Deep Learning Applications for Cyber Security. ASTSA, pp. 197–210. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-13057-2_9
24. Li, D., Wang, Z., Xue, Y.: Fine-grained android malware detection based on deep learning. In: IEEE CNS, pp. 1–2. IEEE (2018)

25. Li, X., Liu, J., Huo, Y., Zhang, R., Yao, Y.: An android malware detection method based on AndroidManifest file. In: IEEE CCIS, pp. 239–243. IEEE (2016)
26. Liu, K., Xu, S., Xu, G., Sun, D., Liu, H.: A review of android malware detection approaches based on machine learning. IEEE Access **8**, 124579–124607 (2020)
27. Lou, S., Cheng, S., Huang, J., Jiang, F.: TFDroid: android malware detection by topics and sensitive data flows using machine learning techniques. In: International Conference on Information and Computer Technologies, pp. 30–36. IEEE (2019)
28. Maiorca, D., Ariu, D., Corona, I., Aresu, M., Giacinto, G.: Stealth attacks: an extended insight into the obfuscation effects on android malware. Comput. Secur. **51**, 16–31 (2015)
29. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y.: Context-aware, adaptive, and scalable android malware detection through online learning. IEEE Trans. Emerg. Top. Comput. Intell. **1**(3), 157–175 (2017)
30. Oak, R., Du, M., Yan, D., Takawale, H., Amit, I.: Malware detection on highly imbalanced data through sequence modeling. In: 12th ACM Workshop on Artificial Intelligence and Security, pp. 37–48 (2019)
31. Pan, Y., Ge, X., Fang, C., Fan, Y.: A systematic literature review of android malware detection using static analysis. IEEE Access **8**, 116363–116379 (2020)
32. Rathore, H., Agarwal, S., Sahay, S.K., Sewak, M.: Malware detection using machine learning and deep learning. In: Mondal, A., Gupta, H., Srivastava, J., Reddy, P.K., Somayajulu, D.V.L.N. (eds.) BDA 2018. LNCS, vol. 11297, pp. 402–411. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04780-1_28
33. Rathore, H., Nikam, P., Sahay, S.K., Sewak, M.: Identification of adversarial android intents using reinforcement learning. In: International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2021)
34. Rathore, H., Sahay, S.K., Rajvanshi, R., Sewak, M.: Identification of significant permissions for efficient android malware detection. In: Gao, H., J. Durán Barroso, R., Shanchen, P., Li, R. (eds.) BROADNETS 2020. LNICST, vol. 355, pp. 33–52. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68737-3_3
35. Rathore, H., Sahay, S.K., Thukral, S., Sewak, M.: Detection of malicious android applications: classical machine learning vs. deep neural network integrated with clustering. In: Gao, H., J. Durán Barroso, R., Shanchen, P., Li, R. (eds.) BROADNETS 2020. LNICST, vol. 355, pp. 109–128. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68737-3_7
36. Rathore, H., Samavedhi, A., Sahay, S.K., Sewak, M.: Robust malware detection models: learning from adversarial attacks and defenses. Forensic Sci. Int.: Digit. Invest. **37**, 301183 (2021)
37. Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Android permissions: a perspective combining risks and benefits. In: 17th ACM symposium on Access Control Models and Technologies, pp. 13–22 (2012)
38. Sewak, M., Sahay, S.K., Rathore, H.: DeepIntent: implicitintent based android IDS with E2E deep learning architecture. In: IEEE PIMRC, pp. 1–6. IEEE (2020)
39. Sewak, M., Sahay, S.K., Rathore, H.: Value-approximation based deep reinforcement learning techniques: an overview. In: International Conference on Computing Communication and Automation, pp. 379–384. IEEE (2020)
40. Sewak, M., Sahay, S.K., Rathore, H.: Deep reinforcement learning for cybersecurity threat detection and protection: A review. In: Krishnan, R., Rao, H.R., Sahay, S.K., Samtani, S., Zhao, Z. (eds.) SKM 2021. CCISv, vol. 1549, pp. 51–72. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-97532-6_4
41. Sewak, M., Sahay, S.K., Rathore, H.: DRLDO: a novel DRL based de-obfuscation system for defence against metamorphic malware. Def. Sci. J. **71**(1), 55–65 (2021)

42. Sewak, M., Sahay, S.K., Rathore, H.: Policy-approximation based deep reinforcement learning techniques: an overview. In: Joshi, A., Mahmud, M., Ragel, R.G., Thakur, N.V. (eds.) Information and Communication Technology for Competitive Strategies (ICTCS 2020). LNNS, vol. 191, pp. 493–507. Springer, Singapore (2022). https://doi.org/10.1007/978-981-16-0739-4_47
43. Shabtai, A., Fledel, Y., Elovici, Y.: Automated static code analysis for classifying android applications using machine learning. In: International Conference on Computational Intelligence and Security, pp. 329–333. IEEE (2010)
44. Sharma, A., Sahay, S.K.: Group-wise classification approach to improve android malicious apps detection accuracy. arXiv preprint arXiv:1904.02122 (2019)
45. Spafford, E.H.: The internet worm program: an analysis. ACM SIGCOMM Comput. Commun. Rev. **19**(1), 17–57 (1989)
46. Stinson, E., Mitchell, J.C.: Characterizing bots' remote control behavior. In: M. Hämmerli, B., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 89–108. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73614-1_6
47. Szor, P.: The Art of Computer Virus Research and Defense. Addison-Wesley Professional (2005)
48. Tan, D.J., Chua, T.W., Thing, V.L.: Securing android: a survey, taxonomy, and challenges. ACM Comput. Surv. (CSUR) **47**(4), 1–45 (2015)
49. Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current android malware. In: Polychronakis, M., Meier, M. (eds.) DIMVA 2017. LNCS, vol. 10327, pp. 252–276. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60876-1_12
50. Yang, M., Wen, Q.: Detecting android malware by applying classification techniques on images patterns. In: IEEE ICCCBDA, pp. 344–347. IEEE (2017)
51. Yuan, Z., Lu, Y., Wang, Z., Xue, Y.: Droid-sec: deep learning in android malware detection. In: ACM Conference on SIGCOMM, pp. 371–372 (2014)
52. Zhang, X., Mathur, A., Zhao, L., Rahmat, S., Javaid, A., Yang, X.: An early detection of android malware using system calls based machine learning model. In: International Conference on Availability, Reliability and Security, pp. 1–9 (2022)
53. Zhang, Y., Yang, M., Yang, Z., Gu, G., Ning, P., Zang, B.: Permission use analysis for vetting undesirable behaviors in android apps. IEEE Trans. Inf. Forensics Secur. **9**(11), 1828–1842 (2014)