Android Malware Detection Using ML

Subjects: Computer Science, Artificial Intelligence | Computer Science, Software Engineering

Contributor: Janaka Senanayake

This systematic review discussed ML-based Android malware detection techniques. It critically evaluated 106 carefully selected articles and highlighted their strengths and weaknesses as well as potential improvements. The ML-based methods for detecting source code vulnerabilities were also discussed, because it might be more difficult to add security after the app is deployed. Therefore, this paper aimed to enable researchers to acquire in-depth knowledge in the field and to identify potential future research and development directions.

Android security | malware detection | code vulnerability | machine learning

1. Introduction

Numerous industrial and academic research has been carried out on ML-based malware detection on Android, which is the focus of this review paper. The taxinomical classification of the review is presented in **Figure 1**. Android users and developers are known to make mistakes that expose them to unnecessary dangers and risks of infecting their devices with malware. Therefore, in addition to malware detection techniques, methods to identify these mistakes are important and covered in this paper (see **Figure 1**). Detecting malware with ML involves two main phases, which are analysing Android Application Packages (APKs) to derive a suitable set of features and then training machine and deep learning (DL) methods on derived features to recognize malicious APKs. Hence, a review of the methods available for APK analysis is included, which consists of static, dynamic, and hybrid analysis. Similar to malware detection, vulnerability detection in software code involves two main phases, namely feature generation through code analysis and training ML on derived features to detect vulnerable code segments. Hence, these two aspects are included in the review's taxonomy.

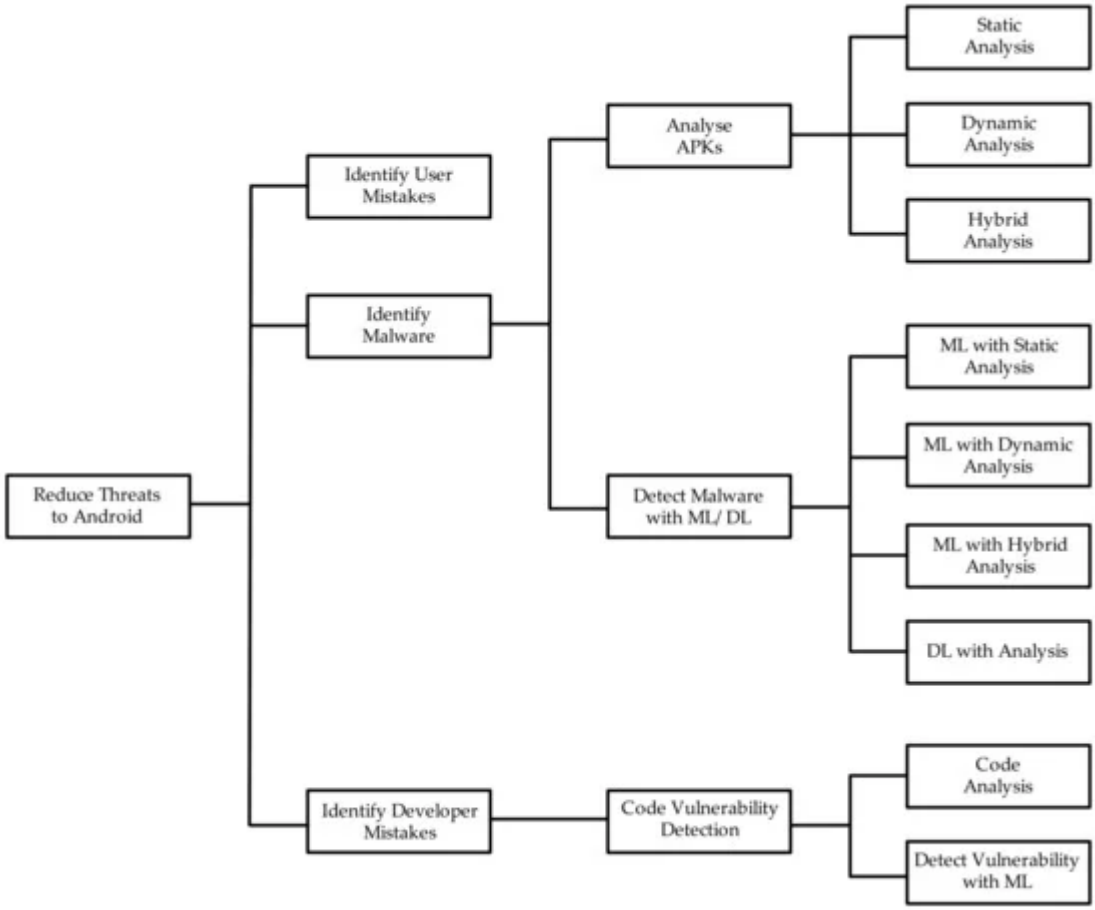

Figure 1. Taxonomy of the review.

2. Background

2.1. Android Architecture

Android is built on top of the Linux Kernel. Linux is chosen because it is open source, verifies the pathway evidence, provides drivers and mechanisms for networking, and manages virtual memory, device power, and security [5]. Android has a layered architecture [6]. The layers are arranged from bottom to top. On top of the Linux Kernal Layer, the Hardware Abstraction Layer, Native C/C++ Libraries and Android Runtime, Java Application Programming Interface (API) Framework, and System Apps are stacked on top of each. Each layer is responsible for a particular task. For example, the Java API Framework provides Java libraries to perform a location awareness application-related activity such as identifying the latitude and the longitude.

2.2. Threats to Android

While Android has good built-in security measures, there are several design weaknesses and security flaws that have become threats to its users. Awareness about those threats is also important to perform a proper malware detection and vulnerability analysis. Many research and technical reports have been published related to the Android threats [13] and classified Android threats based on the attack methodology. Social engineering attacks,

physical access retrieving attacks, and network attacks are described under the ways of gaining access to the device. For the vulnerabilities and exploitation methods, man in the middle attacks, return to libc attacks, JIT-Spraying attacks, third-party library vulnerabilities, Dalvik vulnerabilities, network architecture vulnerabilities, virtualization vulnerabilities, and Android debug bridges and kernel vulnerabilities are considered.

2.2.1. Malware Attacks on Android

Malware attacks are the most common case that can be identified as a threat to Android. There are various definitions for malware given by many researchers depending on the harm they cause. The ultimate meaning of the malware is any of the malicious application with a piece of malicious code [16] which has an evil intent [17] to obtain unauthorised access and to perform neither legal nor ethical activities while violating the three main principles in security: confidentiality, integrity, and availability.

2.2.2. Users and App Developers' Mistakes

The mistakes can happen knowingly or unknowingly from the developers as well as users. These mistakes may lead to threats arising to Android OS and its applications. It has been identified that users are responsible for most security issues [25]. Some common mistakes done by the users will lead to serious threats in an Android application. At the time of installing Android applications, users will be asked to allow some permissions. However, all the users may not understand the purpose of each permission. They allow permission to run the application without considering the severity of it. Fraudulent applications might steal data and perform unintended tasks after getting the required permissions. It is possible to arise threats to the Android systems due to the mistakes performed by the app developers at the time of developing applications. In the publishing stage of the Android apps, Google Play will have only limited control over the code vulnerabilities in the applications. Sometimes developers are specifying unwanted permissions in the Android manifest file mistakenly, which encourages the user to grant the permissions if the permissions were categorised as not simple permissions [26]. Though the app development companies and some of the app stores are advising about following the security guidelines implemented at the time of development, many developers still fail to write secure codes to build secured mobile applications [27].

2.3. Machine Learning Process

ML is a branch of artificial intelligence that focuses on developing applications by learning from data without explicitly programming how the learned tasks are performed. The traditional ML methods make predictions based on past data. ML process lifecycle consists of multiple sequential steps. They are data extraction, data preprocessing, feature selection, model training, model evaluation, and model deployment [9]. Supervised learning, unsupervised learning, semisupervised learning, reinforcement learning, and deep learning are the different subcategories of ML [28]. The supervised learning approach uses a labelled dataset to train the model to solve classification and regression problems depend on the output variable type (continuous or discreet). Unsupervised learning is used to identify the internal structures (clusters), the characteristics of a dataset, and a labelled dataset is not required to train the model. A mix of both supervised and unsupervised learning techniques are applied in

semisupervised learning and used in a case of limited labelled data in the used dataset [29]. The learning model and the data used for training are inferred. The model parameters are updated with the received feedback from the environment in reinforcement learning where no training data is involved. This ML method proceeds as prediction and evaluation cycles [30]. DL is defined as learning and improving by analysing algorithms on their own. It works with models such as artificial neural networks (ANN) and consists of a higher or deeper number of processing layers [31].

3. Methodology

Android was first released in 2008. A few years later, the security concerns were discussed with the increasing popularity of Android applications [2]. More attention was received towards applying ML for software security in the last five years because many researchers continuously identify and propose novel ML-based methods [9]. This review was conducted according to the Preferred Reporting Items for Systematic reviews and Meta-Analysis (PRISMA) model [32]. Based on the objective of this study, first we formulated several research questions. Next, a search strategy was defined to identify the conducted studies which can be used to answer our research questions. The database usage and inclusion and exclusion criteria were also defined at this stage. The study selection criteria were defined to identify the studies aiming to answer the formulated research questions as the third stage. The fourth stage is defined as data extraction and synthesis, which describes the usage of the collected studies to analyse for providing answers to the research questions. We reviewed threats to the validity of the review and the mechanism to reduce the bias and other factors that could have influenced the outcomes of this study as the last step of the review process. **Figure 2** shows a summary of the paper selection method for this systematic review.

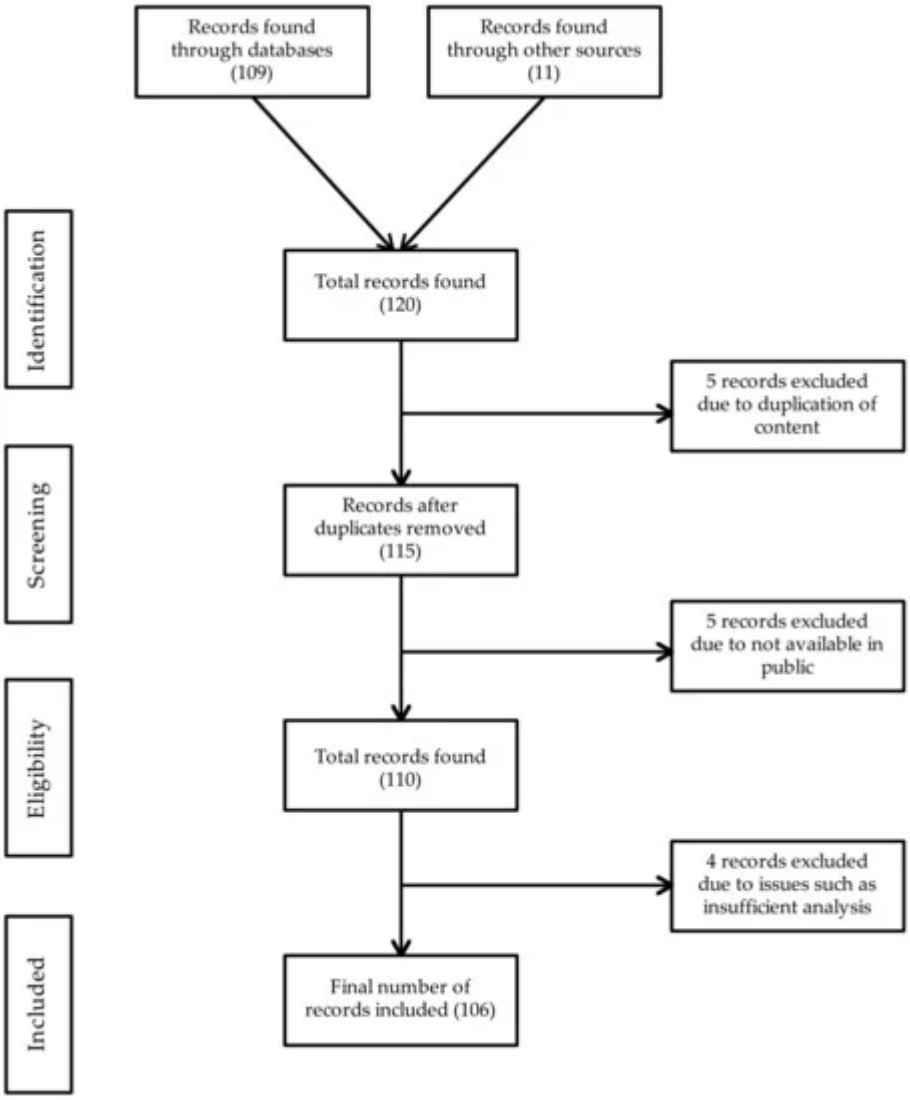

Figure 2. PRISMA method: collection of papers for the review.

3.1. Research Questions

This systematic review aims to answer the following research questions.

RQ1: What are the existing reviews conducted in ML/DL based models to detect Android malware and source code vulnerabilities?

RQ2: What are code/APK analysing methods that can be used in malware analysis?

RQ3: What are the ML/DL based methods that can be used to detect malware in Android?

RQ4: What are the accuracy, strengths, and limitations of the proposed models related to Android malware detection?

RQ5: Which techniques can be used to analyse Android source code to detect vulnerabilities?

4. Related Work

Previous reviews in [9,13,17,34,35,36,37] discussed various ML-based Android malware detection techniques and ways to improve Android security. However, several limitations have been identified in the above works, such as not covering recent proposals on ML methods to detect malware, narrow scopes, and lack of critical appraisals of

suggested detection methods. The lack of a thorough analysis of ML/DL-based methods was also identified as a limitation of existing works. Android malware detection and Android code vulnerability analysis have a lot in common. ML methods used in one task can be customised for use in the other task. However, as per our understanding, there are no reviews that cover these two areas together. These shortcomings have been addressed in this work and therefore our work is unique.

5. Machine Learning to Detect Android Malware

Malware detection in Android can be performed in two ways; signature-based detection methods and behaviour-based detection methods [39]. The signature-based detection method is simple, efficient, and produces low false positives. The binary code of the application is compared with the signatures using a known malware database. However, there is no possibility to detect unknown malware using this method. Therefore, the behaviour-based/anomaly-based detection method is the most commonly used way. This method usually borrows techniques from machine learning and data science. Many research studies have been conducted to detect Android malware using traditional ML-based methods such as Decision Trees (DT) and Support Vector Machines (SVM) and novel DL-based models such as Deep Convolutional Neural Network (Deep-CNN) [40] and Generative adversarial networks [41]. These studies have shown that ML can be effectively utilised for malware detection in Android [9].

5.1. Static, Dynamic, and Hybrid Analysis

As mentioned earlier, analysing APKs to extract features is required to use some of the proposed ML techniques in the literature. To this end, three analysis techniques are identified as static, dynamic, and hybrid analysis method [62,63,64]. Static analysis can be performed by analysing the bytecode and source code (or re-engineered APK) instead of running it on a mobile device. Dynamic analysis detects malware by analysing the application while it is running in a simulated or real environment. However, there is a high chance of exposing the risks to a certain extent to the runtime environment in the dynamic analysis since malicious codes will be executed which can harm the environment. The hybrid analysis involves methods in both static and dynamic analysis.

5.2. Static Analysis with Machine Learning

Static analysis is the widely used mechanism for detecting Android malware. This is because malicious apps do not need to be installed on the device as this approach does not use the runtime environment [67].

5.2.1. Manifest Based Static Analysis with ML

Manifest based static analysis is a widely used static analysis technique.

Table 1. Manifest based static Analysis with ML.

Year	Study	Detection Approach	Feature Extraction	Used Datasets	ML Algorithms/Models	Selected ML Algorithms/Models	Model Accuracy	Strengths	Limitations/Drawbacks

			Method						
2018	[68]	Developing 3 level data purring method and applying ML models with SigPID	Manifest Analysis for Permissions	Google Play	NB, DT, SVM	SVM	90%	High effectiveness and accuracy	Considered only the permission analysis which may lead to omit other important analysis aspects
2021	[69]	Analysing permission and training the model with identified ML algorithm	Manifest Analysis for Permissions	Google Play, AndroZoo, AppChina	RF, SVM, Gaussian NB, K-Means,	RF	81.5%	The model was trained with comparatively different datasets	Did not consider other static analysis features such as OpCode, API calls, etc.
2021	[70]	Reducing dimension vector generation and based on that perform malware detection using ML models	Manifest Analysis for permissions	AMD, APKPure	MLP, NB, Linear Regression, KNN, C.4.5, RF, SMO	MLP	96%	Efficiency, applicability and understandability are ensured	Hyper-parameter selections are not made in the use
2021	[71]	Selecting feature using dimensionality reduction algorithms and using Info Gain method	Manifest Analysis for permissions and intents	Drebin, Google Play	RF, NB, GB, AB	RF, NB, AB	RF-98%, NB-92%, AB-97%	Analysed the features as individual components and not as a whole	Did not consider about other features such as API calls, Opcode etc.
2021	[72]	Feature weighting with join optimisation of weight mapping with proposed JOWMDroid framework	Manifest Analysis for permission, Intents, Activities and Services	Drebin, AMD, Google Play APKPure	RF, SVM, LR, KNN	JOWM-IO method with SVM and LR	96%	Improved accuracy and efficiency	Correlation between features were not considered

References

1. Number of Mobile Phone Users Worldwide from 2016 to 2023 (In Billions). Available online: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (accessed on 19 May 2021).
2. Mobile Operating System Market Share Worldwide. Available online: https://gs.statcounter.com/os-market-share/mobile/worldwide/ (accessed on 19 May 2021).
3. Number of Android Applications on the Google Play Store. Available online: https://www.appbrain.com/stats/number-of-android-apps/ (accessed on 19 May 2021).
4. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. J. Netw. Comput. Appl. 2020, 153, 102526.
5. Khan, J.; Shahzad, S. Android Architecture and Related Security Risks. Asian J. Technol. Manag. Res. [ISSN: 2249–0892] 2015, 5, 14–18. Available online: http://www.ajtmr.com/papers/Vol5Issue2/Vol5Iss2_P4.pdf (accessed on 19 May 2021).

5.2.2. Code Based Static Analysis with ML

Code based analysis is the other way of performing the static analysis to detect Android malware with ML.

6. Platform Architecture. Available online: https://developer.android.com/guide/platform (accessed on 10 May 2021).

Table 2. Code based static Analysis with ML.

Year	Study	Detection Approach	Feature Extraction Method	Used Datasets	ML Algorithms/Models	Selected ML Algorithms/Models	Model Accuracy	Strengths	Limitations/Drawbacks
2016	[78]	Transforming malware detection problem to matrix model using Wxshall algo and extracting Smali codes and generated the API call graph using Androguard	Code analysis for API Calls and code instrumentation for network traffic	MalGenome	Custom build ML based Wxshall algorithm, Wxshall extended algorithm	Wxshall extended algorithm	87.75%	Few false alarms	Required to expand the behaviour model and improve the efficiency
2017	[74]	Using the combination of system functions to describe the application behaviours and constructing eigenvectors and then using Androidetect	Code analysis for API calls and Opcodes	Google Play	NB, J48 DT, Application functions decision algorithm	Application functions decision algorithm	90%	Can identify the instantaneous attacks. Can judge the source of the detected abnormal behaviour High performance in model execution	Did not consider some important static analysis features such as OpCode, API calls, etc.
2018	[39]	Using TinyDroid framework, n-Gram methods after getting the Opcode sequence from .smali after decompiling .dex	Code Analysis for Opcode	Drebin	NLP, SVM, KNN, NB, RF, AP	RF and AP with TinyDroid	87.6%	Lightweight static detection system High performance in classification and detection	Malware samples were taken only from few research studies and some organisations which lack metamorphic malware samples
2018	[73]	Analysing Package level	Code Analysis for API calls and	Drebin, Contagio, Google Play	DT, RF, KNN, NB	RF	86.89%	Model performs well even when	Other information contained in operands were not considered

android analysis techniques. ACM Comput. Surv. (CSUR) 2017, 49, 1–41.

16. Li, L.; Li, D.; Bissyandé, T.F.; Klein, J.; Le Traon, Y.; Lo, D.; Cavallaro, L. Understanding android app piggybacking: A systematic study of malicious code grafting. IEEE Trans. Inf. Forensics Secur. 2017, 12, 1269–1284.
17. Ashawa, M.A.; Morris, S. Analysis of Android malware detection techniques: A systematic review. Int. J. Cyber-Secur. Digit. Forensics 2019, 8, 177–187.
18. Suarez-Tangil, G.; Tapiador, J.E.; Peris-Lopez, P.; Ribagorda, A. Evolution, detection and analysis of malware for smart devices. IEEE Commun. Surv. Tutor. 2013, 16, 961–987.
19. Mos, A.; Chowdhury, M.M. Mobile Security: A Look into Android. In Proceedings of the 2020 IEEE International Conference on Electro Information Technology (EIT), Chicago, IL, USA, 31 July–1 August 2020; pp. 638–642.

		information extracted from API calls using decompiled Smali files	Information flow					the length of the sequence is short	which affect to the overall model
2016	[77]	Using Deterministic Symbolic Automaton and Semantic Modelling of Android Attack	Code Analysis for Opcode/Byte code	Drebin	AB, C4.5, NB, LinearSVM, RF	RF	97%	Use a combined approach of ML and DSA inclusion	Unable to detect new malware patterns since this will not perform complete static analysis
2017	[80]	Training Hidden Markov Models and comparing detection rates for models based on static data, dynamic data, and hybrid approaches	Code analysis for API calls and Opcode in static analysis and System call analysis	Harebot, Security Shield, Smart HDD, Winwebsec, Zbot, ZeroAccess	HMM	HMM	90.51%	Check the difference approaches available to detect ML	Did not consider other ML algorithms or other important features
2019	[75]	Determining the apps call graphs as Markov chain Then obtaining	Code Analysis	Drebin,	RF, KNN, SVM	RF	94%	the system is trained on older samples and	Requires a high memory

Year	Study	Detection Approach	Feature Extraction Method	Used Datasets	ML Algorithms/Models	Selected ML Algorithms/Models	Model Accuracy	Strengths	Limitations/Drawbacks
2017	[81]	Using customized method named Waffle Director	Manifest Analysis for Sensitive permissions and API calls	Tencent, YingYongBao, Contagio	DT, Neural Network, SVM, NB, ELM	ELM	97.06%	Fast Learning speed and Minimal human intervention	Combination of permissions and API calls are not refined
2017	[82]	Using a code-heterogeneity-analysis framework to classify Android repackaged malware by Smali code intermediate representation	Manifest Analysis for Intents, Permissions and API calls	Genome, Virus-Share, Benign App	RF, KNN, DT, SVM	RF with custom model proposed	FNR-0.35%, FPR-2.96%	Provide in-depth and fine-grained behavioural analysis and classification on programs	Detection issues can happen when the malware use coding techniques like reflection and cannot handle if the encryption techniques used in DEX
2018	[84]	Extracting features and transforming into binary vectors and training using	Manifest Analysis for Permissions Code Analysis for API calls,	Drebin	SVM, DT, RF NBs	DT	97.7%	Highly accurate to analyse permission, API calls, opcode an	Broadcast receivers, filtered intend, Control Flow Graph analysis, deep native code analysis were not considered

373–440.

30. Alauthman, M.; Aslam, N.; Al-Kasassbeh, M.; Khan, S.; Al-Qerem, A.; Choo, K.K.R. An efficient reinforcement learning-based Botnet detection approach. J. Netw. Comput. Appl. 2020, 150, 102479.
31. Shrestha, A.; Mahmood, A. Review of deep learning algorithms and architectures. IEEE Access 2019, 7, 53040–53065.

		ML with RanDroid Framework	opcode and native calls					native calls toward malware detection	
2018	[86]	Creating the binary vector, apply ML models, evaluate performance of the features and their ensemble using DroidEnsemble	Manifest analysis for permissions, code analysis for API calls and system calls analysis	Google Play, AnZhi, LenovoMM, Wandoujia	SVM, KNN, RF	SVM	98.4%	Characterises the static behaviours of apps with ensemble of string and structural features.	Mechanism will fail if the malware contains encryption, anti-disassembly, or kernel-level features to evade the detection
2019	[83]	Extracting applications features from manifest while decompiling classes.dex into jar file and applying ML models	Manifest Analysis for permissions, activities and Code Analysis for Opcode	Drebin, playstore, Genome	KNN, SVM, BayesNet, NB, LR, J48, RT, RF, AB	RF with 1000 decision trees	98.7%	High efficiency, Lightweight analysis and fully automated approach	Did not consider about the API calls and other important features when analysing the DEX.
2019	[85]	Using FlowDroid for static analysis and proposing TFDroid framework to detect malware using sensitive data flow analysis	Manifest Analysis for permission and Code Analysis for information flow	Drebin, Google Play	SVM	SVM	93.7%	Analysed the functions of applications by their descriptions to check the data flow.	Did not consider the improving clustering techniques and applicability of other ML models

38. Ghaffarian, S.M.; Shahriari, H.R. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Comput. Surv. (CSUR) 2017, 50, 1–36.
39. Chen, T.; Mao, Q.; Yang, Y.; Lv, M.; Zhu, J. TinyDroid: A lightweight and efficient model for Android malware detection and classification. Mob. Inf. Syst. 2018, 2018.
40. Nisa, M.; Shah, J.H.; Kanwal, S.; Raza, M.; Khan, M.A.; Damaševičius, R.; Blažauskas, T. Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features. Appl. Sci. 2020, 10, 4966.
41. Amin, M.; Shah, B.; Sharif, A.; Ali, T.; Kim, K.I.; Anwar, S. Android malware detection through generative adversarial networks. Trans. Emerg. Telecommun. Technol. 2019, e3675.
42. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. In Proceedings of the 2014 Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014.

5.3. Dynamic Analysis with Machine Learning

The second analysis approach is dynamic analysis. Using this approach it is possible to detect malware with ML after running the application in a runtime environment.

Table 5. Dynamic analysis based malware detection approaches.

Year	Study	Detection Approach	Feature Extraction	Used Datasets	ML Algorithms/Models	Selected ML Algorithms/Models	Model Accuracy	Strengths	Limitations/Drawbacks

43. Google Play. Available online: https://play.google.com/ (accessed on 19 May 2021).
44. AndroZoo. Available online: https://androzoo.uni.lu/ (accessed on 19 May 2021).
45. AppChina. Available online: https://tracxn.com/d/companies/appchina.com (accessed on 19 May 2021).
47. YingYongBao. Available online: https://android.myapp.com/ (accessed on 19 May 2021).

			Method						
2017	[87]	Extracting the DNS, HTTP, TCP, Origin based features of the network used by apps	Network traffic analysis for network protocols	Genome	DT, LR, KNN, Bayes Network, RF	RF	98.7%	Work with different OS versions, Detect unknown malware, and infected apps	If the malware apps using encrypted, not possible to detect malware properly
2017	[88]	Using Markov Chain-based detection technique, to compute the state transitions and to build transition matrix with 6thSense	System resources analysis for process reports and sensors	Google Play	Markov Chain, NB, LMT	LMT	95%	Highly effective and efficient at detecting sensor-based attacks while yielding minimal overhead	Tradeoffs such as frequency accuracy, battery frequency are not discussed which can affect the malware detection accuracy
2017	[89]	Using Dynamic based permission analysis using a run-time and detect malware using ML calculate the accuracy	Code instrumentation analysis Java classes and dynamic permissions	Pvsingh, Android Botnet, DroidKin	NB, RF, Simple Logistic, DT K-Star	Simple Logistic	99.7%	High Accuracy	Need to address the app crashing issue in the selected emulators in dynamic analysis
2019	[90]	Using dynamically tracks execution behaviours of applications and using ServiceMonitor framework	System call analysis	AndroZoo, Drebin and Malware Genome	RF, KNN, SVM	RF	96.7%	High accuracy and high efficiency	Not detecting difference in some system calls of malware and benign apps since signature based verification was not applied
2020	[91]	Extracting the features and permissions from Android app. Performing feature selection and	System call analysis, Code instrumentation for network traffic analysis and System resources analysis	APKPure, Genome	RF, SVM	RF	91.7%	High efficiency	Impact from features like HTTP, DNS, TCP/IP patterns are not considered

58. Google Playstore Appsin Kaggle. Available online: https://www.kaggle.com/gauthamp10/google-playstore-apps (accessed on 19 May 2021).
59. CICMaldroid Dataset. Available online: https://www.unb.ca/cic/datasets/maldroid-2020.html (accessed on 19 May 2021).
60. AZ Dataset. Available online: https://www.azsecure-data.org/other-data.html/ (accessed on 19 May 2021).
61. Github Malware Dataset. Available online: https://github.com/topics/malware-dataset (accessed on 19 May 2021).
62. Alqahtani, E.J.; Zagrouba, R.; Almuhaideb, A. A Survey on Android Malware Detection Techniques Using Machine Learning Algorithms. In Proceedings of the 2019 Sixth International Conference on Software Defined Systems (SDS), Rome, Italy, 10–13 June 2019; pp. 110–117.

		proceed to classification with DATDroid							
2021	[92]	Using decompilation, model discovery, integration and transformation, analysis and transformation, event production	Code instrumentation for java classes, intents	AMD	ML algorithms used in MEGDroid, Monkey, Droidbot	MEGDroid	91.6%	Considerably increases the number of triggered malicious payloads and execution code coverage	System calls are not monitored

65. Kouliaridis, V.; Kambourakis, G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. Information 2021, 12, 185.

66. Chen, L.; Hou, S.; Ye, Y.; Chen, L. An adversarial machine learning model against android malware evasion attacks. In Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data; Springer: Cham, Switzerland, 2017; pp. 43–55.

67. Lubuva, H.; Huang, Q.; Msonde, G.C. A review of static malware detection for Android apps permission based on deep learning. Int. J. Comput. Netw. Appl. 2019, 6, 80–91.

68. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant permission identification for machine-learning-based android malware detection. IEEE Trans. Ind. Inform. 2018, 14, 3216–3225.

5.4. Hybrid Analysis with Machine Learning

Hybrid analysis is the third approach which can be used in ML-based Android malware detection.

Table 6. Hybrid analysis based malware detection approaches

Year	Study	Detection Approach	Feature Extraction Method	Used Datasets	ML algorithms/Models	Selected ML algorithms/Models	Model Accuracy	Strengths	Limitations/Drawbacks
2017	[96]	Using a set of Python and Bash scripts which automated the analysis of the Android data.	Manifest analysis for permissions and System call analysis for dynamic analysis	Andrototal	NB, DT	DT	80%	Model execution is efficient	Consider system call appearance rather than frequency and Lower number of samples used to train
2018	[95]	Using Binary feature vector and permission vector datasets were created using the analysis techniques and was used with the ML algorithms	Manifest analysis for permissions and system call analysis	Drebin	RF, J.48, NB, Simple Logistic, BayesNet TAN, BayesNet K2, SMO PolyKernel, IBK, SMO NPolyKernel	RF	Static-96%, Dynamic-88%	Compared with several ML algorithms	Accuracy depends on the 3rd party tool (Monkey runner) used to collect features.

73. Zhang, P.; Cheng, S.; Lou, S.; Jiang, F. A novel Android malware detection approach using operand sequences. In Proceedings of the 2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC), Shanghai, China, 18–19 October 2018; pp. 1–5.

74. Wei, L.; Luo, W.; Weng, J.; Zhong, Y.; Zhang, X.; Yan, Z. Machine learning-based malicious application detection of android. IEEE Access 2017, 5, 25591–25601.

2019	[94]	Preparing a JSON file after reverse engineering, decompiling, and analysing the APK by running in a sandbox environment and then extracting the key features and applied ML	Manifest analysis for permissions, code analysis for API calls and System call analysis	MalGenome, Kaggle, Androguard [79]	SVM, LR, KNN, RF	LR for static analysis and RF for dynamic analysis	Static-81.03%, Dynamic-93%	Dynamic analysis performed was better than the static analysis approach in terms of detection accuracy	Did not perform a proper hybrid analysis approach to increase the overall accuracy
2017	[99]	Using import term extraction, clustering and applying genetic algorithm with MOCODroid	Code analysis for API calls and information flow and system call analysis	Virus-total, Google Play	Genatic algorithm, Multiobjective evolutionary algorithm	Multiobjective evolutionary classifier	95.15%	Possible to avoid the effects of the concealment strategies	Did not consider about other clustering methods.
2020	[97]	Extracted 261 combined features of the hybrid analysis with using the support of datasets and performed the ML/DL models	Manifest analysis for permissions and system call analysis	MalGenome, Drebin, CICMalDroid	SVM, KNN, RF, DT, NB, MLP, GB	GB	99.36%	Hybrid analysis is having higher accuracy comparing to static analysis and dynamic analysis individually	Runtime environment and configuration is not considered
2020	[98]	Using Conditional dependencies among relevant static and dynamic features. Then trained ridge regularised LR classifiers and modelled their output	Manifest analysis for permissions, code analysis for API calls and system call analysis	Drebin, AMD, AZ, Github, GP	TAN	TAN	97%	Highly accurate	Possibility of some malwares remain undetected

82. Tian, K.; Yao, D.; Ryder, B.G.; Tan, G.; Peng, G. Detection of repackaged android malware with code-heterogeneity features. IEEE Trans. Dependable Secur. Comput. 2017, 17, 64–77.
83. Kabakus, A.T. What static analysis can utmost offer for Android malware detection. Inf. Technol. Control 2019, 48, 235–249.
84. Koli, J. RanDroid: Android malware detection using random machine learning classifiers. In Proceedings of the 2018 Technologies for Smart-City Energy Security and Power (ICSESP), Bhubaneswar, India, 28–30 March 2018; pp. 1–6.
85. Lou, S.; Cheng, S.; Huang, J.; Jiang, F. TFDroid: Android malware detection by topics and sensitive data flows using machine learning techniques. In Proceedings of the 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), Kahului, HI, USA, 14–17 March 2019; pp. 30–36.
86. Wang, W.; Gao, Z.; Zhao, M.; Li, Y.; Liu, J.; Zhang, X. DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features. IEEE Access 2018, 6, 31798–31807.

		relationships as a TAN							
2021	[100]	Using exploit static, dynamic, and visual features of apps to predict the malicious apps using information fusion and applied Case Based Reasoning (CBR)	Manifest analysis for permissions and System call analysis	Drebin	CBR, SVM, DT	CBR	95%	Require limited memory and processing capabilities	Require to present the knowledge representation to address some limitations

90. Salehi, M.; Amini, M.; Crispo, B. Detecting malicious applications using system services request behavior. In Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Houston, TX, USA, 12–14 November 2019; pp. 200–209.

91. Thangavelooa, R.; Jinga, W.W.; Lenga, C.K.; Abdullaha, J. DATDroid: Dynamic Analysis Technique in Android Malware Detection. Int. J. Adv. Sci. Eng. Inf. Technol. 2020, 10, 536–541.

92. Hasan, H.; Ladani, B.T.; Zamani, B. MEGDroid: A model-driven event generation framework for dynamic android malware analysis. Inf. Softw. Technol. 2021, 135, 106569.

93. Raphael, R.; Mathiyalagan, P. An Exploration of Changes Addressed in the Android Malware Detection Walkways. In Proceedings of the International Conference on Computational Intelligence, Cyber Security, and Computational Models, Coimbatore, India, 19–21 December 2019; Springer: Singapore, 2019; pp. 61–84.

5.5. Use of Deep Learning Based Methods

It is possible to use deep learning techniques also for detecting Android malware. In MLDroid, a web-based Android malware detection framework.

Table 8. Deep learning based malware detection approaches

Year	Study	Detection Approach	Feature Extraction Method	Used Datasets	ML/DL Algorithms/Models	Selected DL Algorithms/Models	Model Accuracy	Strengths	Limitations/Drawbacks
2017	[104]	Using n-Gram methods after getting the Opcode sequence from .smali after dissembling .apk	Code Analysis for Opcodes	Genome, IntelSecurity, MacAfee, Google Play	CNN, NLP	Deep CNN	87%	Automatically learn the feature indicative of malware without hand engineering	Assumption of all APKs are benign in Google Play dataset while all are malicious in malware dataset
2021	[108]	Using DL based method which uses Convolution Neural Network based approach to	Code Analysis for API calls, Opcode and Manifest Analysis for Permission	Drebin, AMD	CNN	CNN	91% and 81% on two datasets	Reduce over fitting and possible to train to detect new malware just by collecting	Did not compared with other ML/DL methods

Efficiency Android Malware Detection. In Proceedings of the 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), Jakarta, Indonesia, 19–20 November 2020; pp. 8–12.

98. Surendran, R.; Thomas, T.; Emmanuel, S. A TAN based hybrid model for android malware detection. J. Inf. Secur. Appl. 2020, 54, 102483.

		analyse features						more sample apps	
2018	[102]	Applying LSTM on semantic structure of bytecode with 2 layers of detection and validating with DeepRefiner	Code Analysis for Opcode/bytecode	Google Play, VirusShare, MassVet	RNN, LSTM	LSTM	97.4%	High efficiency with average of 0.22 s to the 1st layer and 2.42 s to the 2nd layer detection	Need to train the model regularly to update the training model on new malware
2020	[105]	Detecting Malware attributes by vectorised opcode extracted from the bytecode of the APKs with one-hot encoding before apply DL Techniques	Code Analysis for Opcode	Drebin, AMD, VirusShare	BiLSTM, RNN, LSTM, Neural Networks, Deep Convents, Diabolo Network model	BiLSTMs	99.9%	Very high accuracy, Able to achieve zero day malware family without overhead of previous training	Did not analyse complete byte code
2020	[106]	Using DynaLog to select and extract features from Log files and using DL-Droid to perform feature ranking and apply DL	Code instrumentation analysis for java classes, intents, and systems calls	Intel Security	NB, SL, SVM, J48, PART, RF, DL	DL	99.6%	Experiments were performed on real devices High accuracy	Could have implemented the intrusion detection part also to make it more comprehensive malware detection tool
2021	[101]	Selecting features gained by feature selection approaches. Applying ML/DL models to detect malware	Code instrumentation for java classes, permissions, and API calls at the runtime	Android Permissions Dataset, Computer and security dataset	farthest first clustering, Y-MLP, nonlinear ensemble decision tree forest, DL	DL with methods in MLDroid	98.8%	High accuracy and easy to retrain the model to identify new malware	Human interaction would be required in some cases. Can contain issues in the datasets

107. Millar, S.; McLaughlin, N.; del Rincon, J.M.; Miller, P. Multi-view deep learning for zero-day Android malware detection. J. Inf. Secur. Appl. 2021, 58, 102718.
108. Qaisar, Z.H.; Li, R. Multimodal information fusion for android malware detection using lazy learning. Multimed. Tools Appl. 2021, 1–15.
109. Acar, Y.; Stransky, C.; Wermke, D.; Weir, C.; Mazurek, M.L.; Fahl, S. Developers need support, too: A survey of security advice for software developers. In Proceedings of the 2017 IEEE Cybersecurity Development (SecDev), Cambridge, MA, USA, 24–26 September 2017; pp. 22–26.
110. Mohammed, N.M.; Niazi, M.; Alshayeb, M.; Mahmood, S. Exploring software security approaches in software development lifecycle: A systematic mapping study. Comput. Stand. Interfaces 2017, 50, 107–115.
111. Weir, C.; Becker, I.; Noble, J.; Blair, L.; Sasse, M.A.; Rashid, A. Interventions for long-term software security: Creating a lightweight program of assurance techniques for developers. Softw. Pract. Exp. 2020, 50, 275–298.
112. Alenezi, M.; Almomani, I. Empirical analysis of static code metrics for predicting risk scores in android applications. In Proceedings of the 5th International Symposium on Data Mining

2021	[107]	Characterising apps and treating as images. Then constructing the adjacency matrix. Then applying CNN to identify malware with AdMat framework	Code Analysis for API calls, Information flow, and Opcode	Drebin AMD	CNN	CNN	98.2%	High Accuracy and efficiency	Performance is depending on number of used features

11 ... 34–94.

114. Pustogarov, I.; Wu, Q.; Lie, D. Ex-vivo dynamic analysis framework for Android device drivers. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 1088–1105.

115. Amin, A.; Eldessouki, A.; Magdy, M.T.; Abdeen, N.; Hindy, H.; Hegazy, I. AndroShield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach. Information 2019, 10, 326.

6. Machine Learning Methods to Detect Code Vulnerabilities

Hackers do not just create malware. They also try to find loopholes in existing applications and perform malicious activities. Therefore, it is necessary to find vulnerabilities in Android source code. A code vulnerability of a program can happen due to a mistake at the designing, development, or configuration time which can be misused to infringe on the security [38]. Detection of code vulnerability can be performed in two ways. The first method is reverse-engineering the APK files using a similar approach discussed in Section 3. The second method is identifying the security flaws at the time of designing and developing the application [109].

6.1. Static, Dynamic, and Hybrid Source Code Analysis

Similar to analysing APKs for malware detection, there are three ways of analysing source codes. They are static analysis, dynamic analysis, and hybrid analysis. In static analysis, without executing the source code, a program is analysed to identify properties by converting the source to a generalised abstraction such as Abstract Syntax Tree (AST) [113].

The number of reported false vulnerabilities depends on the accuracy of the generalisation mechanism. The runtime behaviour of the application is monitored while using specific input parameters in dynamic analysis. The behaviour depends on the selection of input parameters. However, there are possibilities of undetected vulnerabilities [114].

In hybrid analysis, it provides the characteristics of both static analysis and dynamic analysis, which can analyse the source code and run the application to identify vulnerabilities while employing detection techniques [115].

6.2. Applying ML to Detect Source Code Vulnerabilities

116. Tahaei, M.; Vaniea, K.; Beznosov, K.; Wolters, M.K. Security Notifications in Static Analysis Tools: Developers' Attitudes, Comprehension, and Ability to Act on Them. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, Yokohama, Japan, 8–13 May 2021; pp. 1–17.

117. Goaër, O.L. Enforcing green code with Android lint. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops, Melbourne, VIC, Australia, 21–25 September 2020; pp. 85–90.

118. Habchi, S.; Blanc, X.; Rouvoy, R. On adopting linters to deal with performance concerns in android apps. In Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 3–7 September 2018; pp. 6–16.

119. Wei, L.; Liu, Y.; Cheung, S.C. OASIS: Prioritizing static analysis warnings for Android apps based on app user reviews. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017; pp. 672–682.

120. Luo, L.; Dolby, J.; Bodden, E. MagpieBridge: A General Approach to Integrating Static Analyses into IDEs and Editors (Tool Insights Paper). In Proceedings of the 33rd European Conference on Object-Oriented Programming (ECOOP 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 15–19 July 2019.

121. Wang, Y.; Xu, G.; Liu, X.; Mao, W.; Shi, G.; Pedrycz, W.; Wang, W. Identifying vulnerabilities of SSL/TLS certificate verification in Android apps with static and dynamic analysis. J. Syst. Softw. 2020, 167, 110609.

122. Gupta, A.; Suri, B.; Kumar, V.; Jain, P. Extracting rules for vulnerabilities detection with static metrics using machine learning. Int. J. Syst. Assur. Eng. Manag. 2021, 12, 65–76.

123. Kim, S.; Yeom, S.; Oh, H.; Shin, D.; Shin, D. Automatic Malicious Code Classification System through Static Analysis Using Machine Learning. Symmetry 2021, 13, 35.

124. Bilgin, Z.; Ersoy, M.A.; Soykan, E.U.; Tomur, E.; Çomak, P.; Karaçay, L. Vulnerability Prediction From Source Code Using Machine Learning. IEEE Access 2020, 8, 150672–150684.
125. Russell, R.; Kim, L.; Hamilton, L.; Lazovich, T.; Harer, J.; Ozdemir, O.; Ellingwood, P.; McConley, M. Automated vulnerability detection in source code using deep representation learning. In Proceedings of the 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 17–20 December 2018; pp. 757–762.
126. Chernis, B.; Verma, R. Machine learning methods for software vulnerability detection. In Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics, Tempe, AZ, USA, 21 March 2018; pp. 31–39.
127. Wu, F.; Wang, J.; Liu, J.; Wang, W. Vulnerability detection with deep learning. In Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 13–16 December 2017; pp. 1298–1302.
128. Pang, Y.; Xue, X.; Wang, H. Predicting vulnerable software components through deep neural network. In Proceedings of the 2017 International Conference on Deep Learning Technologies, Chengdu, China, 2–4 June 2017; pp. 6–10.
129. Garg, S.; Baliyan, N. A novel parallel classifier scheme for vulnerability detection in android. Comput. Electr. Eng. 2019, 77, 12–26.
130. Ponta, S.E.; Plate, H.; Sabetta, A.; Bezzi, M.; Dangremont, C. A manually-curated dataset of fixes to vulnerabilities of open-source software. In Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), Montreal, QC, Canada, 26–27 May 2019; pp. 383–387.
131. Namrud, Z.; Kpodjedo, S.; Talhi, C. AndroVul: A repository for Android security vulnerabilities. In Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, Toronto, ON, Canada, 4–6 November 2019; pp. 64–71.
132. Cui, J.; Wang, L.; Zhao, X.; Zhang, H. Towards predictive analysis of android vulnerability using statistical codes and machine learning for IoT applications. Comput. Commun. 2020, 155, 125–131.
133. Zhuo, L.; Zhimin, G.; Cen, C. Research on Android intent security detection based on machine learning. In Proceedings of the 2017 4th International Conference on Information Science and Control Engineering (ICISCE), Changsha, China, 21–23 July 2017; pp. 569–574.

Year	Study	Code Analysis Method	Approach	Used ML/DL Methods/Frameworks	Accuracy of the Model
2017	[127]	Dynamic Analysis	Collected 9872 sequences of function calls as features. Performed dynamic analysis with DL methods	CNN-LSTM	83.6%
2017	[133]	Hybrid Analysis	Decompiled the apk file. Performed static analysis of the manifest file to obtain the components/permissions. Dynamic analysis and fuzzy testing were conducted and obtained system status.	AB and DT	77%
2019	[115]	Hybrid Analysis	Reverse engineered the APK, performed static analysis of manifest files & codes and extracted meta data from it. Performed Dynamic analysis to identify intent crashing and insecure network connections for API calls. Generated the report.	Androshield	84%
2020	[124]	Hybrid Analysis	Performed intelligent analysis of generated AST. Checked ML can differentiate vulnerable and nonvulnerable.	MLP and a customised model	70.1%
2017	[113]	Static Analysis	Generated the AST, navigated it, and computed detection rules. Identified smells when training with manually created dataset.	ADCCTOR framework	98%
2017	[128]	Static Analysis	Combined N-gram analysis and statistical feature selection for constructing features. Evaluated the performance of the proposed technique based on a number of Java Android programs.	Deep Neural Network	92.87%
2019	[129]	Hybrid Analysis	Decompiled the APK and selected the features and executed the APK and generated log files with system calls. Generated the vector space and	MLP, SVM, PART, RIDOR, MaxProb, ProdProb	98.37%

			trained with ML algorithms as parallel classifiers.		
2020	[121]	Hybrid Analysis	In static analysis, vulnerabilities of SSL/TLS certification were identified. Results from static analysis about user interfaces were analysed to confirm SSL/TLS misuse in dynamic analysis.	DCDroid	99.39%
2021	[122]	Static Analysis	32 supervised ML algorithms were considered for 3 common vulnerabilities: Lawofdemeter, BeanMemberShouldSerialize, and LocalVariablecouldBeFinal	J48	96%
2021	[123]	Static Analysis	Classified malicious code using a PE structure and a method for classifying it using a PE structure	CNN	98.77%

7. Results and Discussion

Based on the reviewed studies in ML/DL based methods to detect malware, it is identified that 65% of studies related to malware detection techniques used static analysis, 15% used dynamic analysis, and the remaining 20% followed the hybrid analysis technique. This high attractiveness of static analysis may be due to the various advantages associated with it over dynamic analysis, such as ability to detect more vulnerabilities, localising vulnerabilities, and offering cost benefits.

Many ML/DL based malware detection studies used the code analysis method as the feature extraction method. Apart from that, manifest analysis and system call analysis methods are the other widely used methods. It is possible to detect a substantial amount of malware after analysing decompiled source codes rather than analysing permissions or other features. That may be the reason for the high usage of code analysis in malware detection.

By using the feature extraction methods, permissions, API calls, system calls, and opcodes are the most widely extracted features. Many hybrid analysis methods extracted permissions as the feature to perform static analysis. It is easy to analyse permissions when comparing with the other features too. These could be reasons for the high usage of permissions as the extracted feature. Services and network protocols have low usage in feature extractions. The reason for this may be it is comparatively not easy to analyse those features.

Drebin was the most widely used dataset in Android Malware Detection, and it was used in 18 reviewed studies. Google Play, MalGenome, and AMD datasets are the other widely used datasets. The reason for the highest usage

of the Drebin dataset may be because it provides a comprehensive labelled dataset. Since Google Play is the official app store of Android, it may be a reason to have high usage for the dataset from Google.

It is identified that the RF, SVM, and NB are at the top of widely studied ML models to detect Android malware. The reason may be that the resource cost to run RF, SVM, or NB based models is low. Models like CNN, LSTM, and AB have less usage because to run such advanced models, good computing power is required, and the trend for DL-based models was also boosted in recent years. **Table 12** summarises widely used ML/DL algorithms with their advantages and disadvantages.

The majority of the studies used hybrid analysis and static analysis as the source code analysis techniques in vulnerability detection in Android. To perform a highly accurate vulnerability analysis, the source code should be analysed and executed too. Therefore, this may be the reason to have hybrid analysis and static analysis as the widely used source code analysis methods to detect vulnerabilities.

8. Conclusions and Future Work

Any smartphone is potentially vulnerable to security breaches, but Android devices are more lucrative for attackers. This is due to its open-source nature and the larger market share compared to other operating systems for mobile devices. This paper discussed the Android architecture and its security model, as well as potential threat vectors for the Android operating system. Based upon the available literature, a systematic review of the state-of-the-art ML-based Android malware detection techniques was carried out, covering the latest research from 2016 to 2021. It discussed the available ML and DL models and their performance in Android malware detection, code and APK analysis methods, feature analysis and extraction methods, and strengths and limitations of the proposed methods. Malware aside, if a developer makes a mistake, it is easier for a hacker to find and exploit these vulnerabilities. Therefore, methods for the detection of source code vulnerabilities using ML were discussed. The work identified the potential gaps in previous research and possible future research directions to enhance the security of Android OS.

Both Android malware and its detection techniques are evolving. Therefore, we believe that similar future reviews are necessary to cover these emerging threats and their detection methods. As per our findings in this paper, since DL methods have proven to be more accurate than traditional ML models, it will be beneficial to the research community if more comprehensive systematic reviews can be performed by focusing only on DL-based malware detection on Android. The possibility of using reinforcement learning to identify source code vulnerabilities is another area of interest in which systematic reviews and studies can be carried out.