Android Malware Detection A Survey

Raima Zachariah*, Akash K†, Mohammed Sajmal Yousef‡ and Anu Mary Chacko§
Department of Computer Science and Engineering, NIT Calicut
P.O. Chatamangalam, Kerala - 673601
Email: *raima.zachariah@yahoo.co.in, †akashkbaburajan@gmail.com, ‡yousefsajmal1@gmail.com, §anu.chacko@nitc.ac.in

***Abstract*—The popularity of Android mobile devices has gone up in our lives and are being used for handling a lot of our personal and confidential i nformation. H ence t hey are now an ideal target for attackers. Android based smart-phone users can download a lot of free applications from Android Application Market/Play Store. These downloaded apps sometimes contain malware applications that can take possession of private information from users. Apps in queue for launching on Play Store generally undergo an automatic security test where crawl operations like typing, tapping and swiping are performed periodically and analysis results are generated. In this survey paper, we aim to briefly d iscuss t he d ifferent t echniques u sed in Android malware detection and highlight their advantages and limitations.**

***Keywords*—Android Malware Detection Techniques, Machine Learning**

I. Introduction

Android platform offers a range of capabilities at very low cost and evolved to be the most preferred operating system for smart phones. Android being open source increases the risk and raises serious issues related to malicious applications. Also the increase in the number of applications in Android market makes it an easy target for malware authors.

Having an accurate and deep understanding of the working of malwares is crucial to develop preventive measures. The primary signs shown by devices when it is infected with malware are slow performance, drop in battery percentage and unusual app behavior.

Malwares come in different forms like Trojans, Back-doors, Worms, Botnets, Spywares, Ransomwares and Riskwares. Various approaches have to be devised to fight t hem b ased on their nature. The two main approaches are static analysis and dynamic analysis. The malware detection applications use these techniques in their working to alert the users if the application shows malicious behavior.

In this paper we propose to discuss the malware penetration techniques, and android malware detection techniques along with their advantages and limitations. This paper is structured as follows: In section II, we discuss various android malware penetration techniques. In section III, various android malware detection approaches are analyzed in depth followed by a tabulation of results.

II. Android Penetration Techniques

Malware applications use the following penetration techniques for infecting the device. Repackaging, Updating and Downloading.[1]

One of the most common techniques adopted by malware developers is Repackaging, which involves installing malicious application on Android Platform. It involves downloading a popular app, disassembling them, adding malicious code, repacking the code and uploading again. The signature of this repackaged app may be changed by malware authors to prevent detection. Updating is another method that malware authors use and is more difficult to detect. This involves wrapping the inflicted code in an update component that will download malware at run time instead of directly including it in the application. The technique where users are attracted or tricked to download malicious apps is very effective for malware penetration. Unintentional downloads are the malware injected into the system when they visit an online site that contains malicious content. In another technique, the encrypted malicious content within the apk files are decrypted after the installation of the app on the users device. Some malwares download the malware from remote servers instead of enclosing payload as resource dynamically and such malwares cannot be discovered by static analysis approach.

III. Android Malware Detection

The following section intends to discuss the various approaches that have been used to detect android malwares.

A. Static Approach

Static approach involves disassembling and analyzing the source code to check functionalities and the presence of malware without running the code.

1) Signature based detection: Signature based methods primarily focuses on categorizing a code as malware based on the signatures by comparing it with existing malware family signatures. It is highly efficient but fails to detect the unknown malwares due to limited size of signature database. Two examples are

- AndroSimilar [2] uses a syntactic foot-printing mechanism that identifies regions of similarity by statistical methods using known malwares to discover the unknown variants of existing malwares. It classifies an app as malware or benign based on variable length signatures which are further compared with signatures in the AndroSimilar database. The paper reports an accuracy of 60%.
- DroidAnalytics [3] extracts and analyzes the apps at op-code level. First a signature is generated at the

method level. Later, signatures of methods, application signatures along with traces from API calls are used to generate signatures at the class level which is used for further analysis. Out of the 150,368 Android applications that were collected, 2,494 malware samples from 102 families were determined. Among those, there were 342 zero-day malware samples from six different malware families.

2) Permission based detection: This is a quicker method analyzing only the manifest file. Due to the minor differences in permissions requested by malicious and benign apps, it requires a second pass to increase the efficiency of malware detection.

- Stowaway [4] Compiled Android apps are tested for over-privilege using this tool. It determines the API calls made by the app and identifies the permissions needed for each API call using a permission map. Stowaway was applied to a set of 940 applications and it was found that about 33% were over-privileged. The cause this developer error may be because of lack of reliable permission information.
- R.Sato in [5] proposed a light weight malware detection technique which analyzes only the manifest.xml file. It extracts specific information described in the manifest file, compares the extracted information with the keyword lists and then computes the malignancy score to judge the sample as malware or not. An accuracy of 90% was reported by authors.
- C.Y.Haung proposed a technique for classifying labeling into three [6]. *Site based labeling* identifies apps as benign if it is downloaded from official Google market. *Scanner based labeling* depends on the antivirus scanner. *Mixed labeling* is carried out on both scanner based and site based labels. Machine learning algorithms are used to further analyze permissions. The author claimed that the method effectively detects around 81% of the malware samples.
- PUMA [7], studies the permissions extracted from the application and uses machine learning approaches to detect malicious applications. Accuracy rates higher than 80% were achieved according to authors. A 50-tree Random Forest classifier was observed to give the best results in terms of accuracy.
- Tang, Wei [8] proposed a Security Distance Model which uses the idea that it requires more than one permission to challenge the security of Android devices. Authors classified the combinations of permissions into four categories , namely Safe, Normal, Dangerous and Severe Security Distance.The idea was to create an awareness among users about the dangers of giving permissions blindly.
- Enck developed KIRIN [9], a light weight tool that provides certification at installation time. An app is categorised as malware only when it is unable to pass all of its security tests. These tests are a comparison of the security rules and the permissions requested by the app. 10 apps were found to request for dangerous permissions out of the 311 apps tested spanning 16 categories.
- DroidMat [10] works by extracting information from the androidmanifest.xml file. It further analyses the app using permissions and API call tracing mechanism. To improve the working of the classifier, K-means classifier is used along with K-nearest neighbours algorithm.

3) Dalvik Bytecode detection: Bytecode analysis is used to differentiate malicious apps from benign by carrying out a study on their behaviour and obtaining information from control and data flow graphs. This method has a disadvantage of requiring more power and storage space.

- Jinyung Kim developed SCANDAL [11]. It identifies an app as malicious based on the leakage of private information from the information source to remote server. It is a static analyzer and does not support data leakage due to reflections.
- Karlsen [12] produced the first ever formalized version of Dalvik Bytecode including java reflective features. This approach provided an easy way to perform control data flow analysis and to identify malware. It also tracks the sensitive API calls performed during execution.
- Zhou [13] implemented DroidMOSS which is a system that can measure the similarity of apps. Fuzzy Hashing is used to detect changes made in the app by repackaging. The significant limitation of this tool is the limited size of the malware database.
- DroidAPIMiner [14]: DroidRanger combines permission based behavioral footprint and a filtering scheme depending on heuristics to detect the presence of malware. API level information is used to differentiate between malicious and benign apps. Through different analysis steps, critical APIs which occur in malicious apps at a higher frequency are identified. This tool has the following limitations: Malware authors can use reflections to obfuscate dangerous API calls. It might give a poor result if malware authors include more benign API calls into their application. 99% accuracy with a false positive rates of 2.2% was reported.
- Fuchs presented Scandroid [15] which allows incremental checking of apps during the installation time. Security specifications are extracted from the android manifest xml files, and it is checked whether the data flowing through these apps are uniform with the specifications.
- Chin presented ComDroid [16] that detects application communication vulnerabilities. The intent control across flow functions are tracked. It fails to differentiate between if and switch statements but passes through all branches. The location of the potential vulnerability, the type, and whether data leakage/injection could be involved is supplied by ComDroid.

4) Static Analysis of Executables for Collaborative Malware Detection on Android [17]: This method is based on

a lightweight and static method to detect malwares on android systems. This tool consumes less resources making it fit for mobile users. The approach involves a function call analysis, data extraction, training set creation and classification of executables using classifiers like PART, prism and nearest neighbor classifier.

5) DREBIN [18]: Its a light weighted method which allows to identify malicious apps on a smartphone and does a broad static analysis. The features which are enclosed in a joint vector space allowing the ordinary patterns that indicates the malware to be identified automatically and are also used for explaining the decisions. DREBIN tracks down 94% of the malicious cases with a few false alarms outperforming the other approach of malware detection. This method is suitable for analyzing downloaded applications as the checking time is only about 10s. In order to alleviate the absence of dynamic analysis, it provided features that enabled us to spot the execution of hidden code. The quality of results produced by DREBIN relies on the accessibility of malicious and benign apps. Use of machine learning brings in the prospect of poisoning and mimicry attacks like renaming of activities and components during the learning stage.

B. Dynamic Approach

1) Anomaly based detection: It can detect unknown malwares and their variants in run time. It gives wrong result if a benign app displays similar behavior as that of the malicious app.

- CrowDroid by Iker [19] uses dynamic analysis of app behavior to detect malwares. The detector is embedded in a overall framework used for collecting traces from a large number of real users based on crowd-sourcing. This data is then clustered using 2-means clustering algorithm and results are saved at the server.
- Andromly by Shabtai [20] It uses machine learning algorithms to continuously monitor device state to differentiate between benign and malicious apps.
- Zhao proposed AntiMalDroid [21] it monitors the behavior of apps and classsifes apps as benign or malicious. Signatures of the apps are generated from the behavior characteristics obtained from the learning module. It saves the signatures in a database on which a comparison is done with the already present benign and malware application signatures. It is useful in detecting unknown malwares and its variants but is time consuming but it also provides a very high detection rate.

2) Taint Analysis: provided an efficient tracking of sensitive information but is not suited for real time analysis as it reduced performance to almost 20 times slower.

- Enck proposed TaintDroid [22]. Androids virtualized execution environment is leveraged to provide a real time analysis in this tool. It simultaneously tracks multiple sources of sensitive data and identifies the data leakage. Data from privacy sensitive sources are automatically labeled (tainted) and labels are transitively applied as sensitive data moves through interprocess messages, program variables and files. It does real time analysis but cannot perform control flow tracking.

3) Emulation based Detection: It can detect privilege escalation attacks on the kernel but it consumes more resources.

- DroidScope by Yan [23] In this method both the Java-level and OS level semantics are reconstructed seamlessly and simultaneously. This tool also exports the 3-tiered APIs which mirror the 3 levels of an android device, that is, the hardware, the operating system and the Dalvik Virtual Machine which enables in facilitating custom analysis. The whole operating system facilitates this process by allowing the monitoring and by staying out of the execution environment and moreover, privilege based attacks can also be detected. This approach also has a drawback of having very limited code coverage.

C. Other Approaches

1) Apposcopy [24]: A high-level language to specify signatures which discuss semantic properties of malware families and a static analyser to check if an application matches a malware signature is composed. It combines taint analysis, static analysis and Inter-Component Call Graph which effectively detects Android apps which have certain control as well as data flow properties. The two popular techniques are taint analyzers or signature based detectors. It shows resistance to low level code transformation.

2) CASSANDRA [25]: (Context-aware Adaptive and Scalable ANDroid mAlware detector): It makes it possible to capture the security-sensitive behaviors and their context information from dependency graphs. cannot detect all types of malicious behaviors but it is adaptive to evolving malwares, it uses concepts of static analysis and does not use dynamic inspection.Certain attacks like attacks based on reflection and byte-code encryption could go undetected. The accuracy of the tool can get reduced,if benign CADG sub-graph features or fake invariants are added into malicious apps.

3) DroidDetector by Z.Hyuan [26]: It is an android malware detection engine based on deep learning. These techniques were applied on about 192 features that were obtained from dynamic and static app analysis to differentiate between benign and malware appslications. The authors arrived at a conclusion that deep learning was a good method in characterizing malwares based on the inferences made on a large number of applications. This tool also claims to achieve a 96.76% detection accuracy which is much greater than the accuracy rates that were obtained from traditional machine learning techniques.

4) Blaising: proposed Android Application Sandbox [27] which carry out both static and dynamic analysis to identify apprehensive apps. The static approach is carried out on the extracted .dex file and the dynamic analysis is carried out on an android emulator which is normally used for the purpose of debugging. It further uses an AASandbox technique. Due to the use of AASandbox it adds entropy to the dataset. The limitation of this tool is that it is not able to find new malware types.

TABLE I

SUMMARY OF RESULTS

Approach	Name	Description	Advantages	Limitations	Accuracy/ Results
Static analysis	AndroSimilar 2013 [2]	Uses a signature based approach to detect unknown variants of Existing malware.	Useful to detect malwares generated by obfuscation and Repackaging.	Limited signature Databases.	60%
	DroidAnalytics 2013 [3]	Generates 3 level signatures of apps to detect malware	Analyses app at opcode level. More robust than cryptographic hash methods. Can detect zero day repackaged malware.	Relatively higher rates of false positives.	**2494** malware samples detected from **102** families. **150368** collected samples
	Stowaway 2011 [4]	Uses static analysis and permission maps to detect over privileges in apps	Generates maximum set of permissions needed for an application.	Unable to handle complex reflective calls.	About one-third of 940 apps tested are over privileged.
	R Sato, D Chiba, S Goto 2013 [5]	Light weight malware detection mechanism.	Faster as it analyses only manifest file.	Does not analyze the entire code.	90%
	C.-Y. Huang, Y.-T. Tsai C.-H. Hsu 2013 [6]	It labels the apps as benign or malware after analyzing easy to retrieve features.	A quick filter to identify malicious apps.	Requires a second pass to make complete analysis to a reported malicious app. Uses machine learning algorithms as Naive Bayes, ADABOOST etc	81%
	PUMA 2013 [7]	Uses machine learning techniques to analyze extracted permissions from the app.	High accuracy rates	High false positive. Lacks dynamic analysis.	80%
	Security Distance Model 2011 [8]	Uses the idea that more than one permission is required to threaten the security of android devices	Identifies malware at install time. Highly Scalable	Not easy to classify if The threat point is between 50 and 100.	Threat point less than 20 for most of the 100 apps tested.
	KIRIN 2009 [9]	Gives a set of rules which is compared with the set of requested permissions and is declared malicious if the rules are not satisfied.	Lightweight certification at install time.	Legitimate apps maybe declared as malware	10 dangerous/300 evaluated

	DroidMat 2009 [10]	Extracts information from manifest file and analyses the behaviour of application	Lesser time and cost saving as it does not require dynamic simulation and manual efforts. Light weight.	Inadequate for detecting adware samples.	90%
	SCANDAL 2012 [11]	A static analyzer that detects the privacy leakage in apps	Dalvik byte code is available.	Does not support apps which use reflections for data leakage. Semantic and interprocedural properties are not analyzed.	Privacy leaks found in 11/90 apps. Detected privacy leaks in 8 malicious apps.
	E. R. Wognsen, H. S. Karlsen, M. C. Olesen, R. R. Hansen 2013 [12]	The first to have Dalvik bytecode formulation with reflective java features.	Identifies sensitive API calls and performs control and data flow analysis.	Extension required in analyzing reflection and concurrency handling.	Studied 1700 apps.
	DroidMoss 2012 [13]	Uses baksmali tool and generates fingerprints for each app to create signatures.	Effective detection of repackaged apps	Limited databases	5%-13% of apps in the analyzed market places are repackaged.
	ScanDroid 2009 [15]	Checks consistency of the data flow through the application.	Provide security at install time	Java source code or bytecode in jvml should be available.	Performs well on test data
	ComDroid 2011 [16]	Detects communication based vulnerabilities in android apps	Notifies the user when a threat is observed	Requires users to manually check on warnings	12 of the 20 apps tested have atleast one vulnerability.
	DroidAPIMiner [14]	combines permission-based behavioral footprint and a heuristic based filtering scheme	Uses a variety of classifiers for better results.	Malware authors can fool the app using obfuscation	99% accuracy with 2.2% false positive rates.
	Static Analysis of Executables for Collaborative Malware Detection on Android [17]	Uses collaboration for security approach to extent malware detection results.	Lightweight Uses many classifiers	Higher rate of false positives. Susceptible to attacks	Collaboration performs better than existing algorithm to reduce false negative rates.

	DREBIN 2014 [18]	Detects malware on the smartphone itself	Lightweight app. Features are embedded in a joint vector space. Suitable for checking downloaded applications	Lacks dynamic analysis. Use of machine learning brings about mimicry and poisoning attacks	94%
Dynamic Analysis	CrowDroid 2011 [19]	Anomaly detection.	Deep Analysis Uses stracetool and 2-mean clustering algorithm and sends the results to a remote server	Does not work efficiently if the app invokes a lot of system calls.	85% - 100% depending on the malware type.
	Andromaly 2012 [20]	Anomaly detection approach.	Uses machine learning algorithms while continously monitoring machine state.	Battery drainage issues.	80-90%
	AntiMalDroid 2011 [21]	Monitors the behavior of applications ro classify apps.	Can detect unknown malware and variants.	Time consuming	Very high detection rates
	TaintDroid 2010 [22]	Taint Analysis is used	Real time analysis. Tracks multiple sources of sensitive data. Identifies data leakage.	Control flow is not performed. If an information leaves the device and returns to the network, it cannot be kept track of.	High efficiency with no false positives.
	DroidScope 2012 [23]	Emulation based technique	Monitors the whole operating system. Privilege based attacks can be detected.	Limited code coverage.	Accuracy depends on the quality of the features.
Other Approaches	Apposcopy 2014 [24]	Blend of taint analysis and ICCG(Inter Component Call Graph)	Resistant to low level code transformations. Works well with obfuscated code.	Not useful against dynamic code-loading and user reflection. Unfit for instant malware detection	90% with false alarm rates of 10%
	CASSANDRA 2017 [25]	Captures security sensitive behaviors from dependency graphs	Adaptive to evolution of malware features.	Transformational apps are non Detectable. Adversaries can reduce accuracy by including benign CADG features.	89.92%
	DroidDetector 2016 [26]	Uses online deep learning	Detects automatically if app is malware or not.	Accuracy depends on the availability of reliable training data.	96.76%
	Sandboxing 2010 [27]	Performs both static and dynamic analysis.	Sandboxing adds entropy to the data set.	New malware cannot be detected	Performs relatively well.

IV. Discussion and Conclusion

In this paper the various kinds of malwares, their penetration techniques and various approaches used to detect malware are discussed. It is observed that detection of malicious contents using static approach is less efficient when compared to dynamic approach which keeps monitoring the apps remotely or otherwise. But in general most of these approaches fail to detect the parts of the malicious code that is not executed. So it is better to combine the best features in both the approaches and to develop a hybrid method which provides much better performance even though it is expensive because of the limited resources. It has been noted that most of these detection techniques fail or don't perform really well when a zero-day attack has been launched. It has been observed that static approaches are generally faster, provide higher accuracy rates and is effective against existing malware attacks whereas dynamic approaches perform well and can detect malware variants and uses a real time analysis. We can conclude that no single approach is enough to make a system secure.

References

[1] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.

[2] Parvez Faruki, Ammar Bharmal, Manoj Singh Gaur, Vijay Laxmi, and Vijay Ganmoor. Androsimilar: robust statistical feature signature for android malware detection. *Proceedings of the 6th International Conference on Security of Information and Networks*, pages 152–159, 2013.

[3] Min Zheng, John CS Lui, and Mingshen Sun. Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware. *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 163–171, 2013.

[4] Adrienne Porter Felt, Steve Hanna, Erika Chin, Dawn Song, and David Wagner. Android permissions demystified. *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638, 2011.

[5] Ryo Sato, Shigeki Goto, and Daiki Chiba. Detecting android malware by analyzing manifest files. *Proceedings of the Asia-Pacific Advanced Network*, 36:23–31, 2013.

[6] Chun-Ying Huang, Chung-Han Hsu, and Yi-Ting Tsai. Performance evaluation on permission-based detection for android malware. *Advances in Intelligent Systems and Applications-Volume 2*, pages 111–120, 2013.

[7] Borja Sanz, Carlos Laorden, Igor Santos, Xabier Ugarte-Pedrero, Gonzalo Álvarez, and Pablo Garcia Bringas. Puma: Permission usage to detect malware in android. *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pages 289–298, 2013.

[8] Wei Tang, Jiaming He, Guang Jin, and Xianliang Jiang. Extending android security enforcement with a security distance model. *Internet Technology and Applications (iTAP), 2011 International Conference on*, pages 1–4, 2011.

[9] William Enck, Patrick McDaniel, and Machigar Ongtang. On lightweight mobile phone application certification. *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245, 2009.

[10] Dong-Jie Wu, Te-En Wei, Ching-Hao Mao, Kuo-Ping Wu, and Hahn-Ming Lee. Droidmat: Android malware detection through manifest and api calls tracing. *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69, 2012.

[11] Jinyung Kim, Kwangkeun Yi, Yongho Yoon, Junbum Shin, and SWRD Center. Scandal: Static analyzer for detecting privacy leaks in android applications. *MoST*, 12, 2012.

[12] Erik Ramsgaard Wognsen, Mads Chr Olesen, Henrik Søndberg Karlsen, and René Rydhof Hansen. Formalisation and analysis of dalvik bytecode. *Science of Computer Programming*, 92:25–55, 2014.

[13] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326, 2012.

[14] Yousra Aafer, Heng Yin, and Wenliang Du. Droidapiminer: Mining api-level features for robust malware detection in android. *International Conference on Security and Privacy in Communication Systems*, pages 86–103, 2013.

[15] Adam P Fuchs, Jeffrey S Foster, and Avik Chaudhuri. Scandroid: Automated security certification of android. 2009.

[16] Erika Chin, Kate Greenwood, Adrienne Porter Felt, and David Wagner. Analyzing inter-application communication in android. *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252, 2011.

[17] A-D Schmidt, H-G Schmidt, Rainer Bye, Jan Clausen, Kamer A Yuksel, Osman Kiraz, Seyit Ahmet Camtepe, and Sahin Albayrak. Static analysis of executables for collaborative malware detection on android. *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5, 2009.

[18] Daniel Arp, Malte Hubner, Michael Spreitzenbarth, Konrad Rieck, CERT Siemens, and Hugo Gascon. Drebin: Effective and explainable detection of android malware in your pocket. *NDSS*, 2014.

[19] Iker Burguera, Simin Nadjm-Tehrani, and Urko Zurutuza. Crowdroid: behavior-based malware detection system for android. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26, 2011.

[20] Asaf Shabtai, Yuval Elovici, Uri Kanonov, Yael Weiss, and Chanan Glezer. andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[21] Min Zhao, Tao Zhang, Fangbin Ge, and Zhijian Yuan. Antimaldroid: An efficient svm-based malware detection framework for android. *International Conference on Information Computing and Applications*, pages 158–166, 2011.

[22] William Enck, Seungyeop Han, Peter Gilbert, Vasant Tendulkar, Landon P Cox, Byung-Gon Chun, Jaeyeon Jung, Anmol N Sheth, and Patrick McDaniel. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014.

[23] Lok-Kwong Yan and Heng Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. *USENIX security symposium*, pages 569–584, 2012.

[24] Yu Feng, Isil Dillig, Saswat Anand, and Alex Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 576–587, 2014.

[25] Annamalai Narayanan, Lihui Chen, Mahinthan Chandramohan, and Yang Liu. Context-aware, adaptive and scalable android malware detection through online learning (extended version). *arXiv preprint arXiv:1706.00947*, 2017.

[26] Zhenlong Yuan, Yibo Xue, and Yongqiang Lu. Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Science and Technology*, 21(1):114–123, 2016.

[27] Thomas Bläsing, Aubrey-Derrick Schmidt, Leonid Batyuk, Sahin Albayrak, and Seyit Ahmet Camtepe. An android application sandbox system for suspicious software detection. *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, pages 55–62, 2010.