Turkish Journal of Computer and Mathematics Education Vol.12 No.3 (2021), 1785-1792

Research Article

Security Exploitation for Online Meeting Applications: Proof of Concept

Rozahi Istambul[1], Madihah Mohd Saudi [2], Ucu Nugraha[3], Muhammad Yusof [4]

[1]Widyatama University
[2]CyberSecurity and Systems (CSS) Research Unit, Faculty of Science and Technology (FST), Universiti Sains
[3]Islam Malaysia (USIM), 71800 Nilai, Negeri Sembilan, Malaysia
[4]Institut Latihan Perindustrian Kuala Langat,42700 Banting, Selangor, Malaysia
[1]madihah@usim.edu.my

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 20 April 2021

Abstract: With the increase in the demand for online meetings and online learning, there are many security issues and challenges related to it. For example in the year 2020, more than 500,000 Zoom accounts credentials were discovered in the Dark Web due to security exploitation and the default setting used by users. Hence this paper presents a proof of concept for online meeting possible security exploitation by using our developed model called Mobotder for mobile phone. This model is built to detect security exploitation specifically based on geolocation (GPS), permissions, Application Programming Interface (API) calls, and system calls. This model was developed in a controlled lab environment, by applying hybrid analysis and by using open source tools and datasets from Drebin and Google Play Store for training and evaluation. In terms of practicality and as proof of concept (POC), this paper presents the findings for twenty (20) online meeting applications that are currently used worldwide. We discussed in detail how the GPS privacy exploitation occurred and for future work, this model could be used as guidance to defend against malware on a mobile phone.

Keywords: online meeting exploitation, geolocation (GPS), malware, mobile security, security exploitation.

1. Introduction

There are many different techniques were used to exploit software and applications by the attackers. Nowadays, with the increase of online meeting and online learning demand by the users, the attackers are shifting their targets to these users [1]. Those who used unpatched or defaults setting for online applications are exposed to data breaches and security exploitations [2]. For example in 2020, more than 500,000 Zoom accounts credentials were discovered in the Dark Web due to security exploitation and the default setting used by users [3]. Furthermore, during February 2021, there were 2,323,326,953 breached records in the United Kingdom due to ransomware attacks against the cloud service provider Accellion[4]. There were 118 incidents reported and 43 were ransomware attacks. Malware and ransomware attacks are among the most commonly used by attackers to launch their security exploitations. Ransomware is software designed with malicious intention where it will block the victim's access to a computer system until a certain amount of money is paid. Usually, the attacker will use malware to trigger the ransomware for a different type of mobile device used. At the moment, with a new norm of COVID-19 pandemic, many users are working from home and heavily depending on their computer or any mobile devices to be connected online. Hence, based on the existing security challenges, this paper presents a new model called Mobotder to detect possible security exploitation for online meeting applications on a mobile phone. This model can detect and predict any cyberattacks by using permission, API, and system call on a mobile phone. To prove the efficiency of this model, twenty (20) online meeting mobile apps across the world were evaluated to check their level of possible data security exploitation.

This paper is organized as follows: Section 2 explains the related works and methods used, Section 3 discusses the experimental results, Section 4 presents the findings on 20 online meeting apps, and finally Section 5 with conclusions and future work.

2. Methods

Prior to the developments of the Mobotder model, relevant existing works related to mobile malware exploitation were studied for further improvement. Different existing works were carried out for mobile malware detection such as from [5-10]. Notably, the existing works were lack of discussion on feature selection before the formation of the mobile detection model. The feature selection is significant to produce a more accurate detection model before the formation of data mining and classification by using machine learning algorithms [11]. Therefore, this paper presents selected features with mobile malware characteristics to improve the accuracy of the detection. Furthermore, based on our analysis, GPS is one of the surveillance features in a mobile phone that is commonly used by attackers for exploitation, apart from the camera, Bluetooth, Wi-Fi and audio [12,13].

2.1 Data collection

There were two types of datasets used to build Mobotder model, where the botnet dataset from Drebin comprising 5560 malware from 179 different botnet families [14]. We used 2694 botnet samples from 44 different

botnet families from the Drebin dataset. This paper used the percentage split method for the validation purpose, where both datasets were split into 70% for training and 30% for testing for more comprehensive classification and to avoid overfitting. Percentage split is suitable to be implemented in real mobile devices [15]. The Drebin dataset was chosen due to it being easily obtainable from the Internet and used in most mobile malware research works such as by [16-20]. As for proof of concept for this paper, we used 20 anonymous datasets from Google Apps Store that were widely used for online meetings, discussions, or online learning.

2.2 Lab Architecture

Figure 1 demonstrates the lab architecture of this paper which was conducted in a controlled lab environment by using open source tools. Table 1 summarizes the software used for this paper.

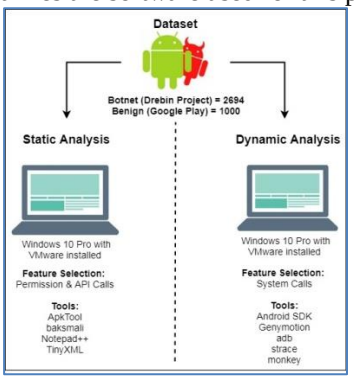

Figure 1. Experiment Lab Architecture.

Table 1. Software Installed for Experiment Lab

Software	Function
VMware Workstation	For virtual workstation in allowing multiple operating systems to run on a single computer.
Genymotion Emulator	It was used as a smartphone android emulator.
VirusTotal (online)	It was used as a scan tool.
ApkTool	It was used for reverse engineering.
TinyXML	It was used to parse Java code.
Smali/baksmali	It was used for assembler/disassembler dex format file.
Android SDK	It was used for android application development.
Java (Weka)	It was used for data testing and simulation.
Notepad++	It was used as a text editor to view source code.

2.3 Data Extraction and Analysis

The feature selection process is illustrated in Figure 2. For the development of the Mobotder model, there are three important elements used which are the permission, API, and system call as summarized in Table 2. This paper used hybrid analysis which consists of static and dynamic analyses. While Figure 3 depicted the summarization of the whole research processes involved to build the Mobotder model.

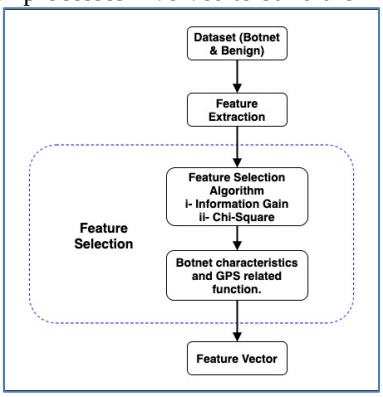

Figure2. Feature Selection Process.

Table 2. Description of Permission, API Call and System Call

Feature	Description
Permission	Permission is derived from AndroidManifest.xml. It protects the privacy of Android users and any android applications require permission consent from the user before installing any application.
API	Application Program Interface (API) call is derived from classes.dex file after the reverse engineering process. It is a set of routines, protocols, and tools used to develop an application.
System call	Application in a mobile phone uses system call to perform specific tasks such as read, write and open since it cannot directly interact with the Android operating system. It is referring to the fundamental interface between an application and the Linux kernel, where this kernel system call to run the services for any application.

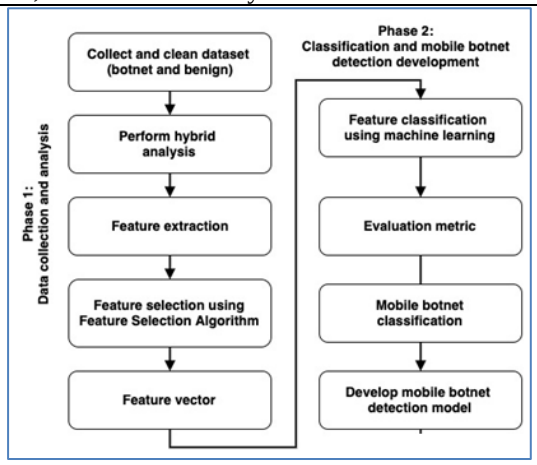

Figure3. Whole Experiment Processes Involved.

3. Findings

In this section, the experiment results are discussed in detail. During the Mobotder model development, there were selected API, permission and system calls that have been tested and verified. Based on the experiments and analysis conducted, it showed that combining selected permissions and API calls produced better accuracy result compared with the benchmark works. Later these selected features were being classified by using a Random Forest classifier. Then the accuracy result was being compared with the existing works by [21-24] and as summarized in Table 3. All the compared works used permission and API calls for feature selection. Based on Table 3, Mobotder has outperformed other existing benchmark works with an accuracy rate of 99.1%.

Table 3. Comparison with Existing Works.

Feature	Work by [21]	Work by [22]	Work by [23]	Work by [24]	Mobotder
Number of Samples (Malware/Benign)	1929/150	250/250	5560/5560	1931/1150	2694/1000
Number of Features (Permission/API calls)	63/1414	12/8	Permission, API calls, hardware components, intents	Permission, API calls, intents, metadata, system calls, network	30/38
ML Classifier	Random Forest	PSO-ANFIS	Random Forest	Random Forest	Random Forest
Accuracy Rate(%)	93.9	89	97.24	97.48	99.1

*ML=Machine Learning

In terms of the practicality of the Mobotder model, this model has been transformed into a mobile app to make the scan job easier. For this evaluation, 20 online meeting applications that are mostly and widely used worldwide were being tested with the Mobotder model. The related API calls and permissions involved as depicted in Table 4. The detailed description for each API and permission is displayed in Table 5 and Table 6.

Table 4. Online Meeting Apps Experiment Results.

Feature	Risk	Description
Application1	High	A3+ A4+ A5+ A6+ A7+ A9+A10 + A12+ A14+ A28+ A29+ A32+ A34
Application2	Medium	P1+ P2+ P4+ P5+ P6+ P7+ P10+ P12+ P16+ P17+ P20+ P25+ A3+ A4+ A5+ A6+ A7+ A9+ A10+ A12+ A28+ A29+ A30 +A31+ A32+ A34+ A35+
Application3	Medium	A36
Application4	High	P1+ P2+ P4+ P5+ P6+ P9+ P10+ P12+ P16+ P17+ P18+ P20+ P25+ P30+ A3+ A4+ A5+ A6+ A7+ A9+ A10+ A11+ A12+ A14+ A15+ A18+ A21+ A22+ A24+ A25+ A26+ A28+ A29+ A32+ A34+ A36+ A38
Application5	Medium	P4+P5+P6+P11+P12+P14+P16+P17+P18+P19+P20+P21+P22+P24+P30+ A1+A3+A4+A5+A6+A7+A8+A9+A10+A12+A14+A15+A18+A10+A20+A24 +A28+A29+A30+A32+A34+A36+A38
Application6	Medium	P1+P2+P4+P5+P6+P11+P12+P16+P17+P18+P25+P26
Application7	Medium	P4+ P6+ P12+ P16 +P17+ P25 + A1+ A3+ A4 + A5 + A6+ A7 + A9 + A10+ A12 + A13 +A 28 + A29 + A32 + A34 + A36
Application8	Medium	P1 +P2 +P4+ P5+ P6+ P7 +P10+ P12 +P16 + P17 + P18 + P20 +P25 + A1 +A3 + A4 + A5+ A6 +A7 +A9 +A10 + A12 + A13 + A14 + A15+ A28 + A29 + A30 + A31 + A32 + A34 + A36
Application9	Medium	P4 +P5+P6 + P10 + P12 + A1 + A3 +A4 + A5 + A6 + A7 + A9 + A10 + A12 +A14 + A15 + A24 +A28 + A29 + A31 + A32 + A34 + A36
Application10	Medium	P4+P5+P6+P12+ P18+ P20+ P25
Application11	Medium	P1 +P2 +P4 + P5+ P6 + P7+ P12 + P18 + P25 + 1 + A3 + A4 + A5 +A6 +A7 + A9 +A10 + A12 + A14 +A28 + A29 + A30 + A32 + A34 + A35 + A36
Application12	Medium	P1+P2 +P4+ P5+P9 +P10 +P11 + P12 + P14 + P16 + P17 + P18 + P20 + P25 + A1 + A3 + A5 + A6 + A7 + A9 + A11 + A12 +A14 + A15 + A28 +A29 + A30 + A31 +A34 + A35 + A36
Application13	Low	P4+P5 +P6 +P7 + P12 + P17 + P20 + A1 + A3 + A4 + A5 + A6 + A7 + A9 + A10 + A12 + A14 + A15 + A28 +A29 + A32 + A34
Application14	Medium	P6+P12 +P17
Application15	Medium	P1 +P4 +P5 +P9 +P12 + P16 + P17 + P18 + P25 + A3 + A4 + A5 +A6 + A7 +A9 +A10 + A12 + A14 +A15 + A17 + A22 + A28 + A29 + A30 + A31 + A32 + A34
Application16	Medium	P4 + P5 + P6 + P9 + P12 + P16 + P18 + A1 + A3 + A4 + A5 + A6 + A7 +A9 +A10 +A11 + A12 + A14 + A15 + A19 + A24 + A28 +A29 + A30 + A31 + A34 + A35 + A36
Application17	Low	P4 + P5 + P6 + P7 + P12 + P 16 + P17 + P18 +P20 + P25
Application18	Medium	P2 + P4 + P6 + P 12 + P14 + P16 + P18 + P20 + A1 + A3 + A4 + A5+ A6 + A7 +A9 + A10 +A12 + A28 + A29 + A32 + A34 + A36
Application19	Medium	P4 +P5 + P6 +P12 +P25
Application20	Medium	P12 +P18+ A3 +A4 + A5 +A6 + A7 + A9+ A10 + A11 + A12 + A13 + A15 + A28 + A29 +A32 +A34
		P1 +P2 +P4 +P5 + P6 +P9 + P11+P12 +P16 +P17 +P18 + P20+ P25 +A1 +A3 +A4 + A5 +A6 +A7 +A9+ A10+ A11 +A12 +A14 + A15+A28+ A29 +A31 +A32+ A34+ A36

Table 5. Description of Related Permissions for Mobotder Model

Feature	Description
P1: Access_Coarse_Location	Allows an app to access approximate location.
Access_Fine_Location	Allows an app to access precise location.
P2: Access_Fine_Location	Allows an app to access extra location provider commands
P3 :Access_Location_Extra_Commands	Allows an app to access information about Wi-Fi networks.
P4: Access_Wifi_State	Allows an app to connect to paired bluetooth devices.

P5: Bluetooth	Allows an app to discover and pair bluetooth devices.
P6: Bluetooth_Admin	Allows an app to enter Wi-Fi Multicast mode.
P7: Change_Wifi_Multicast_State	Allows an app to change Wi-Fi connectivity state.
P8: Change_Wifi_State	Allows an app to install packages.
P9: Install_Packages	Allows an app to install a shortcut in Launcher.
P10: Install_Shortcut	Allows an app to open network sockets.
P11: Internet	Allows an app to kill the background process
P12: Kill_Background_Processes	Allows an app to perform I/O operations over NFC
P13: NFC	Allows an app to read the user's call log.
P14: Read_Call_Log	Allows an app to read the user's contact data.
P15: Read_Contacts	Allows an app to only read the external storage
P16: Read_External_Storage	Allows read-only access to phone state, including the
P17: Read_Phone_State	phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any phone accounts registered on the device.
	Allows an app to read SMS messages.
P18: Read_SMS	Allows an app to receive the
P19: Receive_Boot_Completed	Intent.ACTION_BOOT_COMPLETED that is broadcast after the system finishes booting.
P20: Receive_MMS	Allows an app to monitor incoming MMS messages.
P21: Receive_SMS	Allows an app to receive SMS messages.
P22: Restart_Packages	Allows an app to close processes of other applications.
P23: Send_SMS	Allows an app to send SMS messages.
P24: System_Alert_Window	Allows an app to create windows using the type WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY, shown on top of all other applications.
P25: Uninstall_Shortcut	Allows an app to uninstall shortcut.
P26: Update_Device_Stats	Allows an app to update device statistics
P27: Write_Apn_Settings	Allows an app to write the APN settings.
P28: Write_Call_Log	Allows an app write to user's call log
P29: Write_Contacts	Allows an app to write the user's contact data.
P30: Write_SMS	Allows an app to send SMS

Table 6. Description of Related API Calls for Mobotder Model

Feature	Description
A1: getAccounts	Allows an app to lists all accounts visible to the caller regardless of type.
A2: restartPackage	Allows an app to kill other applications' services
A3: notify	Allows an app to tell the user that something has happened in the background.
A4: query	Allows an app to query the content provider
A5: sendBroadcast	Allows an app to broadcast the given intent to all interested BroadcastReceivers.
A6: startActivity	Allows an app to launch a new activity.
A7: startService	Allows an app to request that a given application service be started.
A8: getBestProvider	Allows an app to returns the name of the provider that best meets the given criteria.
A9: getLastKnownLocation	Returns a Location indicating the data from the last known location fix obtained from the given provider.
A10: isProviderEnabled	Returns the current enabled/disabled status of the given provider.
A11: requestLocationUpdates	Register for location updates using the named provider, and a pending intent.
A12: getActiveNetworkInfo	Returns details about the currently active default data network.
A13: getAllNetworkInfo	Returns connection status information about all network types supported by the device.
A14: getNetworkInfo	Returns connection status information about a particular network type.
A15: getConnectionInfo	
A16: getWifiState	

A17: setWifiEnabled	Return dynamic information about the current Wi-Fi connection, if any is active.
A18: sendTextMessage	Gets the Wi-Fi-enabled state.
A19: sendMultipartTextMessage	Enable or disable Wi-Fi.
A20: sendTextMessage	Send a text-based SMS.
A21: getCellLocation	Send a multi-part text-based SMS.
A22: getDeviceId	Send a text-based SMS
A23: getDeviceSoftwareVersion	Allows an app to return the current location of the device.
A24: getLine1Number	Allows an app to return the unique device ID, for example, IMEI for GSM.
A25: getSimSerialNumber	Allows an app to return the software version number for the device.
A26: getSubscriberId	Allows an app to return the phone number string for line 1.
A27: CryptoCipher	Allows an app to return the serial number of the SIM.
A28: getPackageInfo	Allows an app to return the unique subscriber ID.
A29: getSystemService	Allows an app to use cryptographic operations
A30: HttpPost	Allows an app to retrieve overall information about an app package that is installed on the system.
A31: exec	Allows an app to access application-specific resources and classes.
A32: java/net/HttpURLConnection;->connect	Allows an app to request that a specific web server receive and store data submitted within a request form
A33: getContent	Allows an app to execute the specified command and arguments in a separate process.
A34: openConnection	Allows an app to returns a HttpURLConnection instance that represents a connection
A35: java/net/URLConnection;->connect	Allows an app to gets the contents of this URL.
A36: getInputStream	Allows an app to returns a URLConnection instance that represents a connection to the remote object referred to by the URL.
A37: execute	Allows an app to returns a URLConnection instance that represents a connection
A38: sendSMS	Allows an app to return an input stream for reading from the URL connection.
	Allows an app to executes the specified command for Apache client webserver.
	Allows an app to send SMS.

4. Discussion

Based on the results displayed in Table 4, 10% of the online meetings were exposed to possible high-risk security exploitation by the attackers, followed by 10% with low risk and 80% with medium risk. To avoid these security applications, all mobile phone users must always check and understand the permission they grant for any new mobile app installation into their mobile phone. It is highly recommended to allow only related permission or API for the dedicated mobile app. It is also advisable for users to install a mobile app from the genuine mobile app store to avoid being the victim of malware exploitation. By using the developed Mobotder model, any possible security exploitations could be identified by the users. The Mobotder will trigger an alert to the user once it identifies any potential harm from the mobile app installed. Users and developers must be aware that API and permission concerning online meeting apps could be exploited by malware.

5. Conclusions

In this paper, based on the Mobotder model that has been developed, it is proven that possible security exploitation via permission and API could be detected for the online meeting apps. It is the right solution in detecting any new mobile apps with potential security exploitation. Based on the evaluation conducted, 10% of the tested mobile apps were high risk and have the potential to be exploited by the attackers. Hence, user awareness and solution such as Mobotder are main ingredients in mitigating security exploitation. In the future, Mobotder could be used as a basic guideline and solution to scan any other mobile app category in identifying any potential security exploitation.

Acknowledgment

The authors would like to express their gratitude to Widyatama University, Indonesia and Universiti Sains Islam Malaysia (USIM) (USIM grant no: P1-17-16120-UNI-CVD-FST) for the funding, support, and facilities provided.

References

1. Laplante, P. (2020). Contactless U: Higher education in the postcoronavirus world. IEEE Annals of the History of Computing, 53(07), 76-79.
2. Humayun, M., Niazi, M., Jhanjhi, N. Z., Alshayeb, M., & Mahmood, S. (2020). Cyber security threats and vulnerabilities: a systematic mapping study. Arabian Journal for Science and Engineering, 45(4), 3171-3189.
3. Wagenseil, P. (2020). Zoom security issues: Here's everything that's gone wrong (so far). Tom's Guide, 11.
4. Irwin,L. (2021). List of data breaches and cyber attacks in February 2021 – 2.3 billion records breached, IT Governance UK Blog, Available online: https://www.itgovernance.co.uk/blog/list-of-data-breaches-and-cyber-attacks-in-february-2021-2-3-billion-records-breached [accessed: 28th March 2021]
5. Alqatawna, J. F., Ala'M, A. Z., Hassonah, M. A., & Faris, H. (2021). Android botnet detection using machine learning models based on a comprehensive static analysis approach. Journal of Information Security and Applications, 58, 102735.
6. Yerima, S. Y., & Alzaylaee, M. K. (2020, June). Mobile botnet detection: A deep learning approach using convolutional neural networks. In 2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA) (pp. 1-8). IEEE.
7. Takahashi, T., & Ban, T. (2019). Android Application Analysis Using Machine Learning Techniques. In L. F. Sikos (Ed.), AI in Cybersecurity. Intelligent Systems Reference Library (pp. 181–205). Springer, Cham. https://doi.org/10.1007/978-3-319-98842-9.
8. Alshahrani, H., Mansourt, H., Thorn, S., Alshehri, A., Alzahrani, A., & Fu, H. (2018). Defender: Android Application Threat Detection Using Static and Dynamic Analysis. In Proceeding of 2018 IEEE International Conference on Consumer Electronics, ICCE 2018 (Vol. January, pp. 1–6). https://doi.org/10.1109/ICCE.2018.8326293
9. Sun, M., Li, X., Lui, J. C. S., Ma, R. T. B., & Liang, Z. (2017). Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android. IEEE Transactions on Information Forensics and Security, 12(5), 1103–1112. https://doi.org/10.1109/TIFS.2016.2646641
10. Prakash, G., Darbandi, M., Gafar, N., Jabarullah, N. H., & Jalali, M. R. (2019). A New Design of 2-Bit Universal Shift Register Using Rotated Majority Gate Based on Quantum-Dot Cellular Automata Technology. International Journal of Theoretical Physics, 58(9), 3006-3024.
11. Bhattacharya, A., & Goswami, R. T. (2017). Comparative Analysis of Different Feature Ranking Techniques in Data Mining-Based Android Malware Detection. In J. K. Mandal, S. C. Satapathy, M. K. Sanyal, & V. Bhateja (Eds.), Proceedings of the First International Conference on Intelligent Computing and Communication (Vol. 515, pp. 39–49). Springer, Singapore. https://doi.org/10.1007/978-981-10-3153-3
12. Pieterse, H., & Olivier, M. (2013). Design of a hybrid command and control mobile botnet. Journal of Information Warfare, 12(1), 70–82. Retrieved from https://researchspace.csir.co.za/dspace/bitstream/handle/10204/7385/Pieterse2_2013.pdf
13. Saudi, M. M., Amran, L., & Ridzuan, F. (2020). Go-Detect Application Inspired by Apoptosis to Detect SMS Exploitation by Malwares. In RITA 2018 (pp. 101-116). Springer, Singapore.
14. Arp, D., Spreitzenbarth, M., Malte, H., Gascon, H., & Rieck, K. (2014). Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceeding of the Symposium on Network and Distributed System Security (NDSS) (pp. 23–26). https://doi.org/10.14722/ndss.2014.23247
15. Allibhai, E. (2018). Hold-out vs. Cross-validation in Machine Learning. Available online: https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f [accessed: 28th March 2021]
16. Zhang, H., Luo, S., Zhang, Y., & Pan, L. (2019). An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis. IEEE Access, 7, 69246–69256. https://doi.org/10.1109/ACCESS.2019.2919796
17. Shi-Qi et al. (2019)Shi-Qi, L., Bo, N., Ping, J., Sheng-Wei, T., Long, Y., & Rui-Jin, W. (2019). Deep Learning in Drebin: Android Malware Image Texture Median Filter Analysis and Detection. KSII Transactions on Internet and Information Systems, 13(7), 3654–3670. https://doi.org/10.3837/tiis.2019.07.018

18. Onwuzurike, L., Mariconti, E., Andriotis, P., De Cristofaro, E., Ross, G., & Stringhini, G. (2019). Mamadroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. ACM Transactions on Privacy and Security, 22(2). https://doi.org/10.1145/3313391
19. Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. Computers and Security, 65, 121–134. https://doi.org/10.1016/j.cose.2016.11.007
20. Karbab, E. M. B., Debbabi, M., Derhab, A., & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning. In Proceedings of the 5th Annual DFRWS 2018 Europe (Vol. 24, pp. S48–S59). Elsevier. https://doi.org/10.1016/j.diin.2018.01.007
21. Tansettanakorn, C., Thongprasit, S., Thamkongka, S., & Visoottiviseth, V. (2016). ABIS: A prototype of Android Botnet Identification System. Proceedings of the 2016 5th ICT International Student Project Conference, ICT-ISPC 2016, 1–5. https://doi.org/10.1109/ICT-ISPC.2016.7519221
22. Altaher, A., & Mohammed, O. (2017). Intelligent Hybrid Approach for Android Malware Detection based on Permissions and API Calls. International Journal of Advanced Computer Science and Applications, 8(6).
23. Rana, M. S., Motiur Rahman, S. S. M., & Sung, A. H. (2018). Evaluation of Tree Based Machine Learning Classifiers for Android Malware Detection. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 11056 LNAI, pp. 377–385). Springer International Publishing. https://doi.org/10.1007/978-3-319-98446-9_35
24. Qamar, A., Karim, A., & Shamshirband, S. (2019). A Learning Based Framework for Detection of Android C&C Enabled Applications Using Hybrid Analysis. Preprints. https://doi.org/10.20944/preprints201906.0060.v1