International Journal of Engineering & Technology, 7 (4.15) (2018) 49-54

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

RAPID-Risk Assessment of Android Permission and Application Programming Interface (API) Call for Android Botnet

Zubaile Abdullah[1, 2]*, Madihah Mohd Saudi[2,3]

[1]*Universiti Tun Hussein Onn Malaysia (UTHM), Johor, Malaysia*
[2]*Faculty of Science and Technology, Universiti Sains Islam Malaysia (USIM), Malaysia*
[3]*Cybersecurity and Systems Research Unit, Islamic Science Institute (ISI), Universiti Sains Islam Malaysia (USIM), Malaysia*
**Corresponding author E-mail: zubaile@uthm.edu.my*

Abstract

Android applications may pose risks to smartphone users. Most of the current security countermeasures for detecting dangerous apps show some weaknesses. In this paper, a risk assessment method is proposed to evaluate the risk level of Android apps in terms of confidentiality (privacy), integrity (financial) and availability (system). The proposed research performs mathematical analysis of an app and returns a single easy to understand evaluation of the app's risk level (i.e., Very Low, Low, Moderate, High, and Very High). These schemes have been tested on 2488 samples coming from Google Play and Android botnet dataset. The results show a good accuracy in both identifying the botnet apps and in terms of risk level.

Keywords: *Android Analysis; Android Botnet; Feature Selection; Risk Assessment.*

1. Introduction

In recent years, there has been a tremendous growth of smartphone users around the world. According to a report by the International Data Corporation (IDC), the global shipment of smartphones grew 3.4% in first quarter of 2017 compared to the year 2016 [1]. Among others smartphone's operating software (OS), Android operated smartphone had dominated the market with 85% of market shares. The combination with other software or an application (app), smartphone users now can conveniently store and process sensitive information such as pictures, personal credentials and online banking transaction in their smartphones. As a result, they became an ideal target for cyber-criminal activities by malicious person. Currently, the cyber-criminal incidents of Android operated smartphone occurred frequently compared to other OS operated smartphone [2].

Users are supposed to download and install Android application (app) from Google official applications market named Google Play Store; however, users also can download an application from other third party market. These unofficial markets provide free or non-paying apps and games in which if users download from Google Play Store, they have to pay for those apps. This attracts users to the unofficial market [3] and possibly exposes smartphone users to download and install malicious applications (malware), which camouflaged as benign software from these markets. Furthermore, anybody can upload any type of app to this unofficial app market which is low in security implementation [4], thus it is easier for malware authors to upload their malware app to this market. By default, Android app has limited capabilities in using smartphone resources, sensitive data and system functions and needs to request permissions to do so. To perform certain tasks on the smartphone such as sending messages through Short Message System (SMS), an app must request specific permission from a smartphone user during installation.

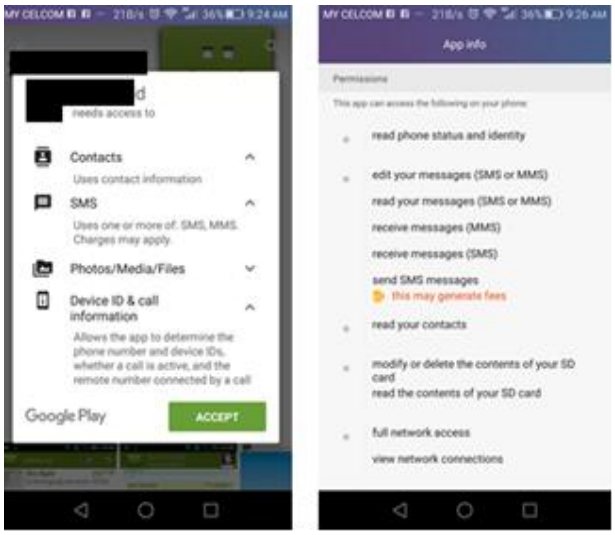

Fig. 1: Permission request during app installation

For example, as shown in Figure 1, an Android app can only send text messages if it has the SEND_SMS permission granted (ACCEPT button) by smartphone user during installation. In this process, smartphone user will be prompted with a list of the permissions required by an app just before the installation. The permission system purpose is to help users avoid privacy or security invasive applications. However, many users do not pay attention to or understand this permission system [5], thus blindly grant permissions to possibly a malware app. As an example, a calculator app which request SEND_SMS or READ_SMS probably is a suspected malicious app because by mean this kind of app need neither send SMS nor read SMS for normal behaviour. Granting such permission to this kind of request possibly expose the user to

subscribe to a premium rate number without their knowledge until they received the phone bill.

The malware apps payload of smartphone come in various forms, such as viruses, Trojans, worms and mobile botnet. However, mobile botnet is more dangerous as they pose serious threats [6-8]. The effect of mobile botnet attacks is disastrous as credential information is exposed to an attacker, user activities and location are leaked, a smartphone user being overbilled because of unauthorized Short Message System (SMS) used, a smartphone can be remotely control by an attacker for other various malicious deeds and smartphone resources are overloaded by malicious activities. In short, privacy and financial of user will be compromised while the performance of the user's smartphone system possibly will be downgraded.

Fortunately, many researches on Android malware detection had been done recently. However, only few research contributions dealt with Android botnet analysis and detection. Furthermore, most of these researches concentrated on numbers of features (permission and API) appearances in Android malware rather than associating these features with their risk to smartphone users. These are the motivation of this research in filling the gap which is not covered by previous research.

2. Related Work

Plenty of research has been made to lessen the increasing threat to the Android system brought by malware. Kirin detected Android malware based on dangerous permission combinations or suspicious action strings [9]. Another research approach by [10] extended Kirin method by increasing the number of permissions to define more permission combinations. However, as there are a few differences in requested or used permissions between benign apps and malware, permission-based approaches suffer the problem of low detection rate.

To overcome the shortcomings of permission based approaches, multi-category feature based approaches were proposed by researchers. This method extracted other static features other than permissions such as an application imported package, application programming interface (API) call, Java code, intent, string, data flow, control flow and hardware components. DroidAPIMiner performed frequency analysis and data flow analysis to all APIs used in an app to calculate most frequently used APIs [11]. Other research that used multi-category features is Drebin which performed a broad static analysis to extract Android application features consists of permissions, sensitive APIs, network address and application hardware components [12]. Although more features are extracted to overcome the shortcoming of permission based approaches, multi-category feature approaches still suffered same shortcoming which is these features cannot associate Android application risk to smartphone users.

Due to permission based and multi category based shortcomings, some researchers attempted to detect Android malware from the viewpoint of Android application risks. In [13] observed that Android apps in the same category usually request similar permissions. They proposed Rare Critical Permission (RCP), which is a set of permission that less requested by same app category. They concluded that an Android app is suspicious to be malware if its request permissions, which matched the RCP list, thus will trigger the Rare Pairs of Critical Permissions (RPCP). RPCP functions is to calculate the app's risk and the threshold to determine if the app if benign or malicious. In [14] propose probabilistic generative models to rank risks of Android apps including the simple Naive Bayes, Mixture of Naive Bayes, and Hierarchical Mixture of Naive Bayes models. Each model estimates the probability that an application would request the permissions. They concluded that the Naive Bayes model gives a promising risk scoring by proving it with real-world datasets. However, both [13-14] approaches have limitation as their approaches cannot state what threat an Android application may cause such as financial losses, leakage of user privacy, or degrading the operating system to smartphone users. This research on the other hand motivated by works done in [15-16]. Both works used permission as features and calculate the requested permission (by an application) risk towards smartphone's user based on risk impact to financial, privacy and smartphone system. As discussed earlier, the differences of the requested permissions between benign apps and malware are relatively small, the above approaches are deficient inherently thus can further be improved. To solve such problem, this research added an additional feature which is API calls to gain better result.

3. Methodology

In this section, the process of risk assessment and the botnet detection scheme based on permission requested and API call is presented. To detect whether an application is benign or botnet, it requires a thorough analysis of features (permission and API call) of applications. This research employed static analysis towards Android APK files from two datasets: botnet and benign Android applications. This process is illustrated in Figure 2 and outlined in the following section.

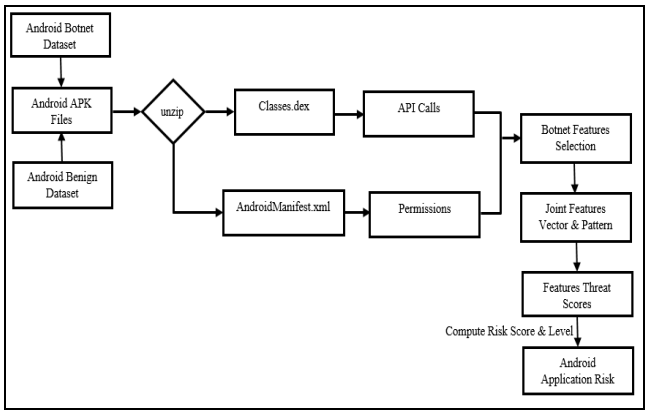

Fig. 2: Features extraction and risk computation phase and process

3.1. Dataset

a) Botnet Dataset: The Android botnet dataset used in this research is downloaded from [17]. The dataset consists of 1929 Android botnet samples in 14 different Android botnet families. These samples cover the majority of existing Android botnet from year from 2010 (the first appearance of Android botnet) to 2014. For the purpose of this research, 1500 Android botnet samples were randomly selected and analyzed.

b) Benign Dataset: A total of 1000 benign apps were downloaded from Google Play Store, an official market that host Android application. It is reasonable to assume that all apps from Google Play Store perform no malicious activities and can be used to construct the benign app dataset. The reason is a malicious app detection system, called Google Bouncer, had deployed by Google to detect the malicious apps uploaded by developers [18]. Any malicious app found by Google Bouncer that may be harmful to users will be removed by Bouncer from Google Play Store. The collected benign apps belong to different categories such as games, applications, education, health, fitness and communications.

3.2. Features Extraction

In this phase, an APK file is unzipped. An APK is an acronym word for Android Application Package, a compressed (zipped) file of any Android application. Generally, an APK file contains .dex files, resources, assets, certificates, and Android manifest file. The Android manifest file contains Android application package's

name, permission request and the minimum API version that the application needs. APKTool is used to unzip the APK file [19]. Once the APK file is unzipped, the AndroidManifest.xml is transformed into readable format. The readable manifests for each 1500 Android botnet and 1000 benign applications are then recorded to a new document format and used for features extraction works.

Features that being extracted are requested permissions and API calls of 2500 Android applications (botnet and benign). The requested permissions of each of these applications are then compared with 138 Android permission in Android system [20] using string similarity method. For each sample, if requested permission matches with the Android permission that permission is noted as 1 to indicate its presence in the sample while 0 indicate the absence of the permission.

Let R be a vector containing set of 138 Android permission. For every ith application in the Android application dataset (botnet and benign), RRiR = {rR1R, rR2R, rR3R,…rj} and r is determined by Equation (1):

$$r_j = \begin{cases} 1, & \text{if jRthR permission exist} \\ 0, & \text{Otherwise} \end{cases} \quad (1)$$

The process to detect for the API call presence in Android apps on the other hand is slightly different from the process of permission occurrence detection. There are hundreds of API calls of Android app however, it is not documented systematically. Thus, in this research, an extensive analysis of each 2500 class.dex is done to search for suspicious API call and generated an API call list of the Android botnet. The list is compared with research done by [3, 21-25]. The API calls of each 2500 Android applications are then compared with the cumulative API call list using the same method as permission occurrence detection above and with the changes of permission to the API call in Equation (1). Further in this phase, the number of botnet apps is reduced to 1499 and the number of benign apps is reduced to 989 because 12 samples are found corrupted and cannot be processed. The result from this phase is used in botnet features selection process.

3.3. Botnet Features Selection

As previously mentioned, android system has a total of 138 permissions and hundreds of API calls. However, not all of these features are requested and being called by the applications in the dataset. The feature selection plays an import role in risk assessment of Android botnets. In this research, the relevance of selected features is based on their impact to privacy and financial of smartphone users and smartphone system. Through a comprehensive analysis on the applications, 20 permissions and 20 API calls which are of interest for botnet writers are presented in Table 1.

Table 1: Selected Feature

Feature Category	Feature Name
PERMISSION	ACCESS_COARSE_LOCATION ACCESS_FINE_LOCATION ACCESS_NETWORK_STATE CALL_PHONE DISABLE_KEYGUARD INSTALL_PACKAGES INTERNET MOUNT_UNMOUNT_FILESYSTEMS PACKAGE_USAGE_STATS READ_CONTACTS READ_EXTERNAL_STORAGE READ_LOGS READ_PHONE_STATE READ_SMS REBOOT RECEIVE_BOOT_COMPLETED RECEIVE_SMS RESTART_PACKAGES SEND_SMS SET_ALARM UPDATE_DEVICE_STATS WRITE_APN_SETTINGS WRITE_CONTACTS WRITE_EXTERNAL_STORAGE
API CALLS	invalidateAuthToken android/app/Activity;->setContentView startActivityForResult openInputStream sendOrderedBroadcast startService android/media/MediaPlayer;->stop isConnectedOrConnecting sendMultipartTextMessage sendTextMessage getDeviceId getDeviceSoftwareVersion getLine1Number getSimSerialNumber getSubscriberId getVoiceMailNumber Cipher(AES/CBC/PKCS5Padding) getSystemService java/net/URL;->openStream sendSMS

The most frequent permission requested is the INTERNET while the frequent APIs used by botnet are getSystemService, get DeviceId, and getSubscriberId. Table 2 lists the top 10 of the features frequently called and used by Android botnet with their possible threat.

Table 2: Top 10 of Features, Description and Possible Threat

Feature	Description	Possible Threat
INTERNET	Allows an app to open network sockets	An app can connect to Internet and communicates with malicious remote server.
READ_PHONE_STATE	Allows an app read only access to phone state.	An app can read the phone number of the device, current cellular network information and the status of any ongoing calls
getSystemService	Allow an app to access to system services	An app can access phone system service capabilities
getDeviceId	Allow an app to get unique device ID such as the IMEI	An app can access sensitive data
startService	Allow an app to request to be started	An app can be executed when phone booting
ACCESS_NETWORK_STATE	Allows app to access information about networks	An app can view information about device communication
RECEIVE_BOOT_COMPLETED	Allows an app to receive the broadcast after the system finished booting	An app to run itself every time smartphone is started
getSubscriberId	Allow an app to get unique subscriber ID	An app can access sensitive information
SEND_SMS	Allows an app to send SMS messages	An app can send SMS to premium rate number or spam another user in smartphone contact list
READ_SMS	Allows an app to read SMS messages	An app can read SMS received including Transaction Authorization Code (TAC) sent by bank

Table 3: Features Patterns (Partial List)

Feature Representation		Patterns
P1 ACCESS_COARSE_LOCATION	AP1 invalidateAuthToken	1. P3, P4, P5, P7, P10, P12, P13, P14, P16, P17, P18, P19, P22, P23, AP6, AP11, AP15, AP18
P2 ACCESS_FINE_LOCATION	AP2 setContentView	
P3 ACCESS_NETWORK_STATE	AP3 startActivityForResult	2. P3, P7, P11, P13, P14, P16, P17, P19, P24, AP6, AP9, AP10, AP11, AP18, AP20
P4 CALL_PHONE	AP4 openInputStream	
P5 DISABLE_KEYGUARD	AP5 sendOrderedBroadcast	3. P3, P7, P16, AP6
P6 INSTALL_PACKAGES	AP6 startService	4. P3, P4, P7, P8, P10, P13, P14, P16, P17, P19, P24, AP3, AP4, AP6, AP10, AP11, AP13, AP15, AP18, AP20
P7 INTERNET	AP7 MediaPlayer;->stop	
P8 MOUNT_UNMOUNT_FILESYSTEMS	AP8 isConnectedOrConnecting	5. P3, P4, P7, P10, P11, P12, P13, P14, P16, P17, P19, P24, AP6, AP9, AP10, AP11, AP13, AP18, AP20
P9 READ_CONTACTS	AP9 sendMultipartTextMessage	
P10 READ_EXTERNAL_STORAGE	AP10 sendTextMessage	6. P1, P2, P3, P4, P5, P7, P8, P10, P13, P14, P16, P17, P18, P22, P24, AP2, AP3, AP4, AP5, AP6, AP7, AP8, AP11, AP17, AP18, AP19
P11 READ_LOGS	AP11 getDeviceId	
P12 READ_PHONE_STATE	AP12 getDeviceSoftwareVersion	7. P1, P2, P3, P4, P5, P7, P8, P11, P12, P13, P14, P16, P18, P22, P24, AP3, AP4, AP5, AP6, AP7, AP8, AP11, AP14, AP15, AP17, AP18
P13 READ_SMS	AP13 getLine1Number	
P14 REBOOT	AP14 getSimSerialNumber	8. P1, P2, P3, P4, P5, P7, P8, P9, P10, P11, P12, P13, P14, P16, P17, P18, P22, P23, P24
P15 RECEIVE_BOOT_COMPLETED	AP15 getSubscriberId	
P16 RECEIVE_SMS	AP16 getVoiceMailNumber	9. P1, P2, P3, P4, P5, P7, P8, P9, P10, P13, P14, P16, P17, P18, P19, P21, P22, P23, P24, AP3, AP4, AP6, AP7, AP8, AP11, AP15, AP18, AP19
P17 RESTART_PACKAGES	AP17 Cipher(AES/CBC/PKCS5Padding	
P18 SEND_SMS	AP18 getSystemService	10. P1, P2, P3, P4, P5, P7, P9, P10, P11, P13, P14, P16, P17, P24, AP3, AP4, AP6, AP7, AP8, AP11, AP14, AP15, AP17, AP18
P19 UPDATE_DEVICE_STATS	AP19 java/net/URL;->openStream	
P20 WRITE_APN_SETTINGS	AP20 sendSMS	
P21 WRITE_CONTACTS		
P22 WRITE_EXTERNAL_STORAGE		

3.4. Joint Feature Vector and Pattern

Based on the result in features extraction and features selection, each of the botnet and benign app features are transferred to the feature vector table to view their features pattern. After removing duplicates patterns, there are 348 unique features patterns (from 1499 patterns) of Android botnet and 548 unique features pattern (from 989) of benign apps. Table 3 shows a partial list of both botnet and benign apps pattern. The patterns are used to calculate the risk score and risk level of each app.

3.5. Features Threat Score

In particular, the Android malware and botnet distribution by malicious persons are always motivated by privacy exposure of victim, financial profit to attacker and degrading operating system of smartphone [16, 26, 27, 15]. Thus, this research had classified the app features into three risks which are privacy risk, financial risk and system risk.

Privacy risk refers to potential leakage of user privacy when features are granted (permission) by the user during Android app installation and used (API calls) during app execution. Financial risk refers to the potential financial losses of the victim. System risk refers to potential smartphone system components performance degrading and unauthorized modification to storage and files of smartphone. These classifications are presented in Table 4.

Table 4: Risk Classification and Threat Score

Risk Classification	Features	Threat Score
Privacy	READ_EXTERNAL_STORAGE	0.2
	setContentView	0.2
	ACCESS_COARSE_LOCATION	0.4
	INTERNET	0.4
	UPDATE_DEVICE_STATS	0.4
	openInputStream	0.4
	ACCESS_FINE_LOCATION	0.8
	READ_LOGS	0.8
	READ_CONTACTS	1
	READ_PHONE_STATE	1
	READ_SMS	1
	RECEIVE_SMS	1
	getDeviceId	1
	getDeviceSoftwareVersion	1
	getLine1Number	1
	getSimSerialNumber	1
	getSubscriberId	1
	getVoiceMailNumber	1
Financial	ACCESS_NETWORK_STATE	0.6
	CALL_PHONE	1
	SEND_SMS	1
	sendMultipartTextMessage	1
	sendTextMessage	1
	sendSMS	1
System	invalidateAuthToken	0.2
	startActivityForResult	0.2
	MediaPlayer;->stop	0.2
	isConnectedOrConnecting	0.2
	Cipher(AES/CBC/PKCS5Padding)	0.2
	DISABLE_KEYGUARD	0.4
	RESTART_PACKAGES	0.4
	sendOrderedBroadcast	0.4
	java/net/URL;->openStream	0.4
	INSTALL_PACKAGES	0.6
	MOUNT_UNMOUNT_FILESYSTEMS	0.6
	WRITE_CONTACTS	0.8
	WRITE_EXTERNAL_STORAGE	0.8
	startService	0.8
	REBOOT	1
	RECEIVE_BOOT_COMPLETED	1
	WRITE_APN_SETTINGS	1
	getSystemService	1

Each of the features in Table 4 is given a threat score value constructed from works done in [16] and also adapted from the NIST Special Publication 800-30 Revision 1 [28] based on impact towards privacy, financial and system as shown in Table 5.

Table 5: Impact Level and Threat Score

Likehood of Impact	Capabilities	Threat Score
Very Low	Features as risk sources do not have any capabilities to perform threat	0.2
Low	The capabilities of risks sources to perform a threat are low	0.4
Moderate	The capabilities of risks sources to carry out a threat are moderate	0.6
High	The capabilities of risk sources to carry out a threat are real and high	0.8
Very High	The capabilities of risk sources to carry out a threat are definite and very high	1

3.6. Computation of Apps Risk and Level

In this phase, each Android app risk score and level is calculated. The first step is to calculate each app's risk toward privacy, financial and system based on Equation (2)-(4):

$$RSP = \frac{fp.tsp}{F_{fp}} \quad (2)$$

$$RSF = \frac{ff.tsf}{F_{ff}} \quad (3)$$

$$RSS = \frac{fs.tss}{F_{fs}} \quad (4)$$

where RSP = risk score for privacy, RSF = risk score for financial, RSS = risk score for system, Fp = privacy related features, ff = financial related features, fs = system related features, tsp = threat score for privacy, tsf = threat score for financial, tss = threat score for system and F = frequency of features in an app.
Further, the Total Risk Score (TRS) of an application is calculated based on the Equation (5):

$$TRS = \frac{RSP+RSF+RSS}{1+(RSP+RSF+RSS)} * 100 \quad (5)$$

An app Total Risk Score (TRS) is the normalization of risk score of privacy, financial and system of the given app. The TRS value is then compared with NIST Risk Score Guide in [28] as shown in Table 6.

Table 6: Risk Score and Risk Level

Risk Score	Risk Level
0 - 20	Very Low
21 - 40	Low
41 - 60	Moderate
61 - 80	High
81 - 100	Very High

4. Results and Discussion

The proposed risk assessment is evaluated within two datasets previously mentioned. The graph in Figure 3 shows the results of the evaluation.

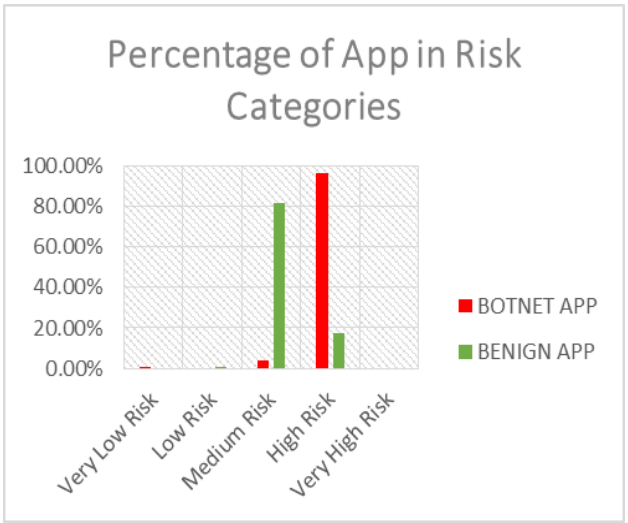

Fig. 3: Android app Risk.

From 1499 Android botnet samples, it is found that 1440 is categorized as High Risk (96.06%), 58 samples are in Medium Risk (3.87%) and only 1 sample in Very Low Risk Category (0.07%). Whereas, for 989 Android benign samples, only 172 samples categorized in High Risk (17.39%), 808 samples in Medium Risk (81.70%) and 9 samples in Low Risk (0.91%).

5. Conclusion

This research has presented a risk-based approach to differentiate a botnet app and benign. Features of the app are extracted to describe on how an application can manipulate user granted permissions and coded API call. This research classifies an application risks into privacy risk, financial risk and system. The results show that this proposed risk assessment is capable of detecting Android botnet at a satisfying accuracy rate. In addition, a pattern that is generated can also be used in the Android botnet detection.
For future work, to improve detection rate, this research is planning to conduct a hybrid Android botnet analysis such as combining a static analysis with dynamic analysis, to solve the problems entangled with static analysis. Further, this research is going to implement a response mechanism after detection using the Apoptosis Immune System in the future.

Acknowledgement

This work was supported by the Ministry of Higher Education (MOHE), Malaysia [Grant No: USIM/FRGS/FST/32/50114] and Universiti Tun Hussein Onn Malaysia [Tier 1 Vot: U892].

References

[1] IDC: Smartphone OS Market Share. (n.d.). https://www.idc.com/promo/smartphone-market-share/os.
[2] Tong, F., & Yan, Z. (2017). A hybrid approach of mobile malware detection in Android. Journal of Parallel and Distributed Computing, 103, 22–31.
[3] Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., & Nadjm-Tehrani, S. (2016). Detection and visualization of android malware behavior. Journal of Electrical and Computer Engineering, 2016, 1-17.
[4] Zheng, C., Zhu, S., Dai, S., Gu, G., Gong, X., Han, X., & Zou, W. (2012). Smartdroid: an automatic system for revealing ui-based trigger conditions in android applications. Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 93–104.
[5] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., & Wagner, D. (2012). Android permissions: User attention, comprehension, and behavior. Proceedings of the Eighth Symposium on Usable Privacy and Security, pp. 1-14.
[6] Eslahi, M., Salleh, R., & Anuar, N. B. (2012). Bots and botnets: An overview of characteristics, detection and challenges. Proceedings of the IEEE International Conference on Control System, Computing and Engineering, pp. 349–354.
[7] Xiang, C., Binxing, F., Lihua, Y., Xiaoyi, L., & Tianning, Z. (2011). Andbot: Towards advanced mobile botnets. Proceedings of the 4th USENIX Conference on Large-Scale Exploits and Emergent Threats, pp. 1-7.
[8] Yusof, M. Bin, Mohd Saudi, M., & Ridzuan, F. (2017). A systematic review and analysis of mobile botnet detection for GPS exploitation. Advanced Science Letters, 23(5), 4696–4700.
[9] Enck, W., Ongtang, M., & McDaniel, P. (2009). On lightweight mobile phone application certification. Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 235–245.
[10] Liang, S., & Du, X. (2014). Permission-combination-based scheme for android mobile malware detection. Proceedings of the IEEE International Conference on Communications, pp. 2301–2306.
[11] Aafer, Y., Du, W., & Yin, H. (2013). Droidapiminer: Mining api-level features for robust malware detection in android. Proceedings of the International Conference on Security and Privacy in Communication Systems, pp. 86–103.
[12] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., & Siemens, C. (2014). DREBIN: Effective and explainable detection of android malware in your pocket. Proceedings of the Ndss, pp. 23–26.

[13] Sarma, B., Li, N., Gates, C., Potharaju, R., Nita-rotaru, C., & Molloy, I. (2012). Android permissions: A perspective combining risks and benefits. Proceedings of the Symposium on Access Control Models and Technologies, pp. 13-22.
[14] Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C, & Molloy, I. (2012). Using probabilistic generative models for ranking risks of Android apps. Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 1-12.
[15] Ye, Y., Wu, L., Hong, Z., & K Huang. (2017). A risk classification based approach for android malware detection. KSII Transactions on Internet and Information Systems, 11(2), 959–981.
[16] Dini, G., Martinelli, F., Matteucci, I., Petrocchi, M., Saracino, A., & Sgandurra, D. (2018). Risk analysis of Android applications: A user-centric solution. Future Generation Computer Systems, 80, 505–518.
[17] Stakhanova, N., & Ghorbani, A. A. (2015). Android Botnets: What URLs are telling us. Proceedings of the Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, pp. 78–91.
[18] Mahindru, A., & Singh, P. (2017). Dynamic permissions based Android malware detection using machine learning techniques. In Proceedings of the ACM 10th Innovations in Software Engineering Conference, pp. 202–210.
[19] ApkTool. (n.d.). A tool for reverse engineering Android apk files. https://ibotpeaches.github.io/Apktool/.
[20] Android Developers. (n.d.). Permissions overview. https://developer.android.com/guide/topics/permissions/overview
[21] Deepa, K., Radhamani, G., & Vinod, P. (2015). Investigation of feature selection methods for android malware analysis. Procedia Computer Science, 46, 841–848.
[22] Fereidooni, H., Moonsamy, V., Conti, M., & Batina, L. (2016). Efficient classification of Android Malware in the wild using robust static features. In W. Meng, X. Luo, S. Furnell, & J. Zhou (Eds.), Protecting Mobile Networks and Devices: Challenges and Solutions. Florida: CRC Press, pp. 181–209.
[23] Karim, A., Salleh, R., & Shah, S. A. A. (2015). DeDroid: A mobile botnet detection approach based on static analysis. Proceedings of the IEEE 12th International Conference on Ubiquitous Intelligence and Computing and IEEE 12th International Conference on Autonomic and Trusted Computing and IEEE 15th International Conference on Scalable Computing and Communications and Its Associated Workshops, pp. 1327–1332.
[24] Qiao, M., Sung, A. H., & Liu, Q. (2016). Merging permission and api features for android malware detection. Proceedings of the IEEE 5th IIAI International Congress on Advanced Applied Informatics, pp. 566–571.
[25] Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013). A new android malware detection approach using Bayesian classification. Proceedings of the IEEE 27th International Conference on Advanced Information Networking and Applications, pp. 121–128.
[26] Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). A survey of mobile malware in the wild. Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 3–14.
[27] Jorgensen, Z., Chen, J., Gates, C. S., Li, N., Proctor, R. W., & Yu, T. (2015). Dimensions of risk in mobile applications: A user study. Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pp. 49–60.
[28] National Institute of Standards and Technology Gaithersburg. (2012). Guide for conducting risk assessments. NIST Special Publication.