Journal of Engineering Science and Technology
Special Issue on ICCSIT 2018, July (2018) 152 - 161

MOBILE MALWARE CALL LOGS CLASSIFICATION BASED ON ANDROID PACKAGE INDEX (API)

MADIHAH MOHD SAUDI[1,2,*], AMIRUL ADLI ISMAIL[1], FARIDA RIDZUAN[1,2]

[1]Faculty of Science and Technology (FST),
Universiti Sains Islam Malaysia (USIM), 71800 Nilai, Negeri Sembilan.
[2]CyberSecurity and Systems Research Unit, Islamic Science Institute (ISI),
Universiti Sains Islam Malaysia (USIM), 71800 Nilai, Negeri Sembilan.
*Corresponding Author: madihah@usim.edu.my

Abstract

In the cyber world, the trend whereby cybercriminals exploit smartphone vulnerabilities to obtain confidential information or make financial gain is becoming more common. They inject the mobile application with malicious code to exploit the victim's smartphone. Moreover, they exploit mobile platforms, in particular, the Android operating system, using mobile malware, without the victim's consent. To combat this problem, this paper presents a new mobile malware call logs classification based on the Android Package Index (API). This new classification is designed to efficiently detect mobile malware attacks. The experiment was conducted in a controlled lab environment by integrating static analysis and Knowledge Data Discovery (KDD) for data cleaning, transformation, and analysis. The dataset is reverse engineered using static analysis. The data consisted of 5,560 samples from Drebin as the training dataset and 500 samples from the Google Play Store for the testing. As a result, thirty-two mobile applications from the Google Play store matched with the proposed classification for call log exploitation. The proposed call log exploitation classification can be used as a reference for other researchers with the same interest and can be further explored as the input for the formation of a mobile malware detection model.

Keywords: Android package index, Call exploitation, Knowledge data discovery, Mobile malware classification, Static analysis.

1. Introduction

Smartphones are widely used worldwide for effective and easy communication. Android has been recognized as one of the most commonly used mobile platforms, with 86.2% usage. It is being targeted by cybercriminals due to its open source feature [1]. However, call log or SMS is the easiest way to exploit a smartphone since many users use these features. There are several ways in which call log and SMS can be exploited. Mobile malware as a harmful software is created to access a device without the user's knowledge and consent, for malicious intent [2]. It has many ways of propagation, such as embedding itself in a genuine mobile application, defeating the two-way authentication of an online transaction and acting as a backdoor into the smartphone. According to a report by Amro [3], IOS_XAGENT has been identified by Trend Micro as masquerading as a legitimate game and then stealing the victim's SMS, contact lists, GPS, pictures and voice recordings.

As for the Android architecture and framework, system calls, API calls, and permissions are most widely used for analysis. API is the easiest way to exploit Android architecture since it is the most vulnerable layer in the architecture [4]. This paper discusses Android and API as well as conducting a comprehensive analysis of mobile malware architecture, focusing on APIs in the framework layer and their evaluation. As a result, five new call log exploitation classifications have been developed and evaluated in this paper. The evaluation results showed that 2% of the mobile applications matched with the proposed classifications.

This paper is organized as follows: section 2 discusses related previous works, section 3 presents the methodology used, section 4 presents the findings and section 5 concludes the paper and discusses opportunities for future work.

2. Related Works

There are many techniques used to perform reverse engineering for mobile applications, for example, static and dynamic analyses. Static analysis is used to get a manifest file and the application coding, and to identify a pattern of malicious behavior without executing the codes. Whereby, in dynamic analysis, the codes will be executed to identify the malicious behavior. The applicability of using static and dynamic analysis depends on the analyst's goal. Table 1 shows some examples of these analyses. This research has applied static analysis for a more in-depth analysis of the Android architecture and for an optimized result.

Table 1. Summary of analysis technique.

Authors	Feature Extraction	Analysis
[2], [5-9]	API calls	Static analysis
[5-6]	Permissions	Static analysis
[10]	Permissions and API calls	Static and dynamic analyses
This study	API calls	Static analysis

Apart from the existing techniques for analysing malware as listed in Table 1, there were also other related works on malware detection techniques [11-15], which are summarized in Table 2. It is observed that each of the related works has its own strength, yet they have some gaps that need to be filled. Dataset size for training

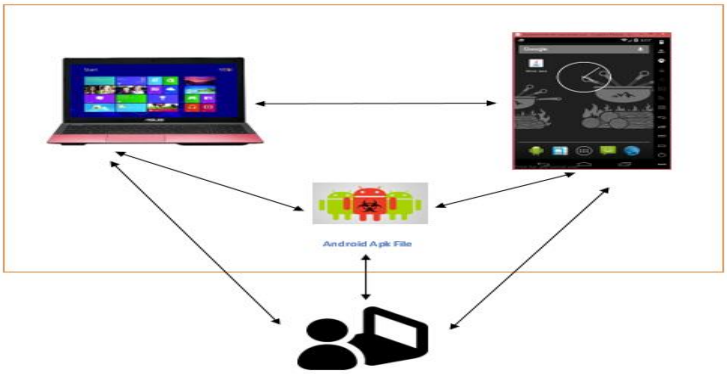
and testing is among the challenges. Therefore, a bigger dataset is recommended for training and testing to gain an optimized result, and we have implemented this in our research.

Table 2. Summary of related works.

Title	Strength	Weakness	Algorithm Used
Exploitation and Detection of a Malicious Mobile Application [11]	Combination of static and dynamic analyses with better performance.	A dataset with file size limit.	Signature-based algorithm.
Android Mobile Malware Surveillance Exploitation Via Call Logs: Proof of Concept [12]	Call logs exploitation using API.	Not in-depth result. The only proof of concept presented.	Reverse engineered the coding.
Crowdroid: Behaviour-based Malware Detection System for Android [13]	100% detection rate for self-written malware based on real time.	Need human intervention to install the Crowdroid application.	The behavior-based detection system, partition-clustering algorithm.
Recognizing API Features for Malware Detection Using Static Analysis [14]	Use APIs and manager classes for malware detection.	Limited based on database provided.	Static analysis.
Manilyzer: Automated Android Malware Detection through Manifest Analysis [15]	Detects malware automatically based on Android manifest file.	To integrate more detection features, such as API.	Naive Bayes, SVM and KNN.

3. Methodology

Based on our analysis, a bigger size of the dataset for training and testing is crucial for producing more accurate and optimized result. As for our training dataset, it is extracted from Drebin with 5,560 samples [16]. Many researchers, such as [17-20], have used the Drebin dataset for their testing since it is freely available, reliable and is the biggest mobile malware dataset to date. For evaluation, 500 different anonymous mobile apps from the Google Play Store have been randomly selected. Our focus for this research is on API calls only. The evaluation was conducted in a controlled lab environment It was conducted using static analysis and the software used in the experiment is shown in Fig. 1. For data cleaning and data transformation, the KDD concept was applied. The raw dataset from Drebin [16] was converted to an APK file, which was later further analysed using static analysis, as shown in Fig. 2.



Asus Core™ i3-3120M
Microsoft Windows 8
MS Excel
WinZip
MS Office

Android 4.3-API level 18,
Virtual Machine
Genymotion

Fig. 1. Setup for the laboratory.

Name	Date modified	Type	Size
06eb19d137ff0a0ccd778b236e8d45c3b9b078115b2ed69baf06aee0244980c1.apk	10/11/2017 12:19..	APK File	478 KB
06f3b82f301c97b2da71f4fdc8ed0e6046381981581dbf7714511abca47d3387.apk	10/11/2017 1:45 PM	APK File	948 KB
06fd5e281179fdfad8c84a4a12977a6942b98992382f6f91b0bb2fc0d4c9e9641.apk	10/11/2017 12:20..	APK File	1,345 KB
6e9848008c42128623d1077f0cc75c0861e2b3d0fb76e68c5fc9ec8f69361572.apk	10/11/2017 12:11..	APK File	1,140 KB
6ea6a44433e321d81fea8fd2c91d5f7d71f57136b979cc85dac90ab5f8f7b070.apk	10/11/2017 1:56 PM	APK File	112 KB
6eb2a685bc4fa7b825fe7e40d1b4263bb38f3dc01f91a578ab9c75049fc4054f.apk	10/11/2017 12:20..	APK File	1,289 KB
6eb9e5b879ab3088872a755b86ae6f928df3927fdbab30ab0d7a8469902c3779.apk	10/11/2017 1:17 PM	APK File	699 KB
6eb51ab4443b237ffdfee833801b39a41e5b6f67d70e38193d96bb61472a4c14.apk	10/11/2017 12:04..	APK File	4,521 KB
6ec9c3033b2051d18762eca6116aec2bf1279360ea358256282da71e1c87eb13.apk	10/11/2017 12:37..	APK File	76 KB
6ed52331a788ef18727c8e34746b59db81acdb261659934be63b0266fb7c19e7.apk	10/11/2017 1:11 PM	APK File	1,818 KB
6f1af02b1836ac348e90b0ac69cd57f1f396b9b1d886be1b007af1b6a5b7008d.apk	10/11/2017 1:07 PM	APK File	2,528 KB
6f2e2f2bac1438cd088de25bb34c6dea20b41ac7756df397e661013664d56d95.apk	10/11/2017 1:53 PM	APK File	3,123 KB
6f3bdc06a6148f184a45bd42a90a5e018ea5d1eddb65bd408c424246fd61233b.apk	10/11/2017 1:22 PM	APK File	3,518 KB

Fig. 2. Data transformation from raw dataset to APK file.

A summary of the whole research process is displayed in Fig. 3. API calls were extracted from a dex class file. For each of the mobile app datasets, if the requested API matches with the Android API call, 1 represents its presence, while 0 indicates its absence. Let M be a vector containing a set of ten Android API calls classifications. For every *i*th application in the Android application dataset (botnet and benign), *Mi = {r1, r2, r3, …, rj}* and

$$r_j = \begin{cases} 1, & if\ permission\ j_{th}\ exist \\ 0, & otherwise \end{cases} \quad (1)$$

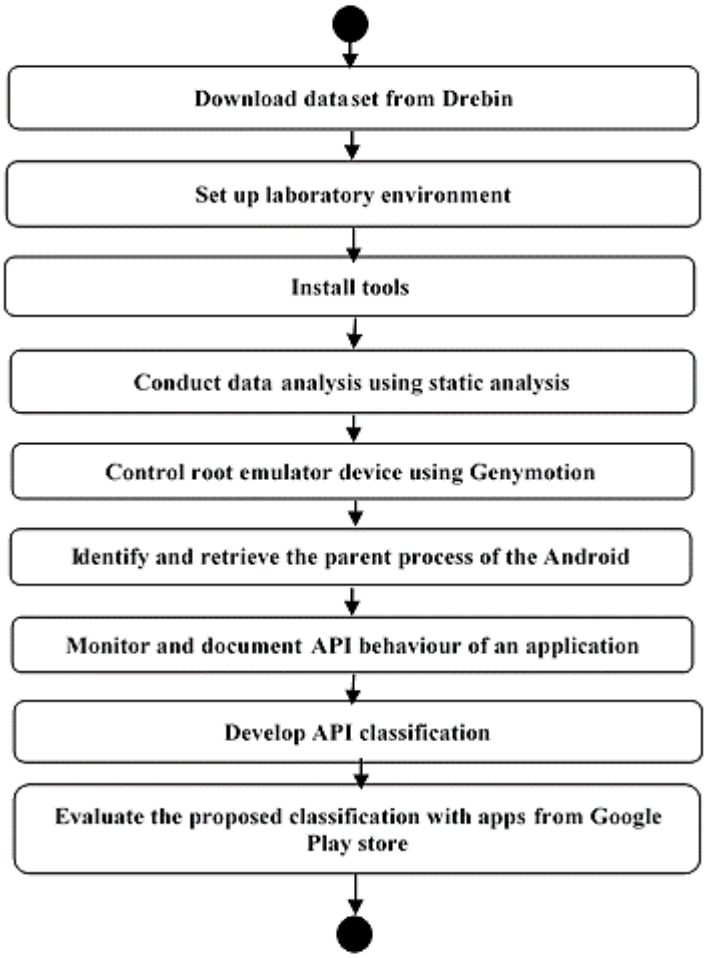


Fig. 3. Flowchart for the conducted experiment.

Equation (1) is meant to be applied for the feature extraction and is used later for data transformation. Next, the KDD and covering algorithm are applied to form the API classification [21]. The covering algorithm is based on the PRISM method and is known as the separate-and-conquer algorithm, where there is a rule for the attributes in each phase. It constructs rules and generates only correct rules with 100% accuracy [22]. The accuracy formula uses PT/DT, where PT represents the positive examples of the class and DT represents the total of the dataset. A summary of the integration of KDD and the covering algorithm is shown in Fig. 4. The pseudocodes for the API classification are displayed in Fig. 5. The pseudocodes explain how the dataset is tested to form a new API call classification based on API calls function, as displayed in Table 3. If the dataset matches with any of the API calls, a new pattern of API call will be developed. The API calls listed in Table 3 are the main features that are used by most smartphone users when they make any phone call. Unfortunately, an attacker with malicious intent can misuse these API calls.

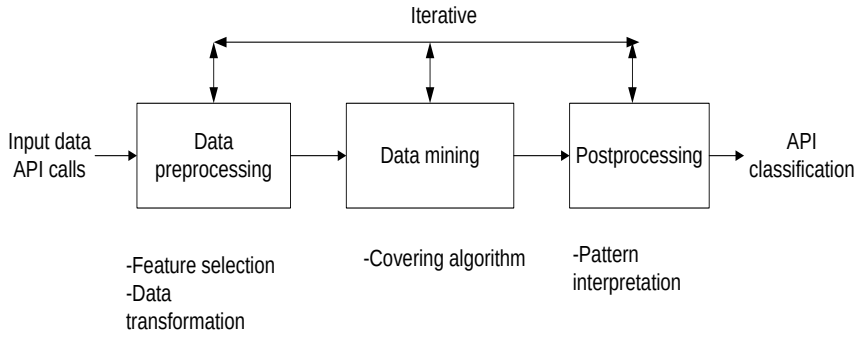


Fig. 4. KDD and covering algorithm.

```
Given:
- Set characteristic's value: {API, not API}
- Set CharFlag = 0
- Set CharFlag = 0
- Dataset A
Output:
- API classification for Dataset B
Algorithms:
While (case ≤ 5560)
{
  - get the worm attributes
     While (worm_attributes != null)
    {
         While (Dataset A != empty)
           {
              - Determine characteristic value for each type of worm attribute
                  from the Dataset A
             - Dataset B = Dataset A (worm_attributes[case,type])
                      If (characteristic_value = API)
                         {
                           CharFlag = 1
                    break
                         }
                      If (characteristic_value = notAPI)
                          CharFlag = 1
          }
                    else
                                - get the next worm attributes from Dataset A
          }
}
```

Fig. 5. Pseudocodes to form API classification using a covering algorithm.

4. Findings

Table 3 displays the API calls that are related to call logs that have been extracted from the Drebin dataset. It contains the nominal data representation by using letter and number, API calls function and description of each API.

Table 3. Extracted API calls.

API nominal representation	API calls function	Description
API137	addToMyContactsGroup	Adds someone in contact group.
API150	startListening	Starts listening to audio speech.
API153	isVoiceMailNumber	Checks a given number against the voicemail number provided by the RIL and SIM card.
API164	getLine1Number	The caller must have the READ_PHONE_STATE credential. Returns the phone number string for line 1, for example, the MSISDN for a GSM phone. Returns null if it is unavailable.
API165	getNeighboringCellInfo	Returns the neighboring cell information of the device.
API166	getSimSerialNumber	Returns the serial number of the SIM, if applicable.
API168	getVoiceMailAlphaTag	Retrieves the alphabetic identifier associated with the voicemail number.
API169	getVoiceMailNumber	Returns the voicemail number. Return null if it is unavailable.
API170	listen	Registers a listener object to receive notification of changes in specified telephony states.
API171	getCallerInfo	Get caller information.

Table 4. New classification related to call logs exploitation.

API nominal representation	Pattern
Pattern1	API137+API150+API153+API164+API165+API 166+API168+API169+API170+API171
Pattern2	API164+API166+API170
Pattern3	API166+API170
Pattern4	API165+API166+API170
Pattern5	API164+API165+API166+API168+API169
Pattern6	API168+API169+API170
Pattern7	API164+API166
Pattern8	API164+API170
Pattern9	API164+API165
Pattern10	API164+API169
Pattern11	API165+API170
Pattern12	API150+API164
Pattern13	API150+API165

Table 4 summarizes the proposed 13 new developed classifications for API calls related to call logs exploitation. After the evaluation was conducted with 500 anonymous mobile apps from the Google Play store, 32 mobile apps were identified and suspected for call log exploitation as presented in Table 5. This is based on the impact in terms of confidentiality, integrity, and availability on the victim's smartphone. Furthermore, based on the evaluation conducted, there are 9 different categories involved, which are game, entertainment, communication, wallpaper, tool, photo, browser, fitness and social media. Based on Table 5, 25% of the matched mobile apps were classified under the category of communication and 15.6% each was under the category of game, entertainment, and tool.

There are a few key findings based on the testing conducted. Firstly, even though the samples for the testing were collected from a genuine mobile app store, malware had still been embedded inside the genuine mobile apps, whether the developers of the apps realized it or not. As a preventive measure, both the developer and end user must be aware of this issue and if possible, the end user should perform an anti-virus scan before installing any mobile apps, which should always be downloaded from a genuine and legal app store. Nevertheless, each of the mobile apps has the potential to be easily exploited by an intruder. Therefore, the end user must make sure that features such as Wi-Fi, Bluetooth, GPS or SMS are only installed when needed. Prevention is better than cure.

Table 5. Evaluation result for 32 matched mobile apps.

Mobile app name	Mobile app category	Mobile app name	Mobile app category
X1	Tool	**X17**	Game
X2	Wallpaper	**X18**	Entertainment
X3	Game	**X19**	Wallpaper
X4	Tool	**X20**	Game
X5	Communication	**X21**	Game
X6	Social Media	**X22**	Game
X7	Fitness	**X23**	Communication
X8	Tool	**X24**	Communication
X9	Communication	**X25**	Communication
X10	Entertainment	**X26**	Communication
X11	Entertainment	**X27**	Entertainment
X12	Photo	**X28**	Communication
X13	Browser	**X29**	Communication
X14	Photo	**X30**	Tool
X15	Photo	**X31**	Entertainment
X16	Wallpaper	**X32**	Tool

5. Conclusion

This paper has presented 13 new API call classifications for call log exploitation, which can be used as a reference and input for the formation of a mobile botnet detection model. Based on the evaluation conducted in this paper, it can be concluded that everyone must be more careful and remind himself that every mobile app created, downloaded or installed has its own risk. This paper can be used as guidance and reference for other researchers with the same interest. For future work, the developed API call classifications for call log exploitation will be used as the database and input for a mobile botnet detection model.

Acknowledgment

This research is sponsored by the Ministry of Higher Education (MOHE), Malaysia under FRGS grant, [grant no: USIM/FRGS/FST/32/50114].

Nomenclatures

Mi	Vector contains a set of Android API calls classification.
*i*th	Application in the Android application dataset.
r_j	Botnet and benign dataset

Abbreviations

API	Android Package Index
APK	Android Package
GPS	Global Positioning System
KDD	Knowledge Data Discovery
KNN	K-Nearest Neighbours
SMS	Short Message Service
SVM	Support Vector Machine

References

1. Kaushik, P.; and Jain, A. (2015). Malware detection techniques in android. *International Journal of Computer Applications*, 122(17), 22-26.
2. Felt, A.P.; Chin, E.; Hanna, S.; Song, D.; and Wagner, D. (2011). Android permissions demystified. *Proceedings of the 18th ACM conference on Computer and communications security*, 627-638.
3. Amro, B. (2018). Malware detection techniques for mobile devices. *arXiv Prepr. arXiv1801.02837*.
4. Shewale, H.; Patil S.; Deshmukh, V.; and Singh, P. (2014). Analysis of Android vulnerabilities and modern exploitation techniques. *ICTACT Journal on Communication Technology*, 5(1), 863-867
5. Aung, Z.; and Zaw, W. (2013). Permission-based android malware detection. *International Journal of Scientific & Technology Research*, 2(3), 228-234.
6. Sahs, J.; and Khan, L. (2012). A machine learning approach to android malware detection. *European Intelligence and Security Informatics Conference* (*EISIC*), 141-147.
7. Yerima, S.Y.; Sezer, S.; McWilliams, G.; and Muttik, I. (2013). A new Android malware detection approach using Bayesian classification. *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference*, 121-128.
8. Almin, S.B.; and Chatterjee, M. (2015). A novel approach to detect android malware. *Procedia Computer Science*, 45, 407-417.
9. Chen, J.; Alalfi, M.H.; Dean, T.R.; and Zou,Y. (2015). Detecting android malware using clone detection. *Journal of Computer Science and Technology*., 30(5), 942-956.
10. Hashim, H.A.B.; Saudi, M.M.; and Basir, N. (2017). Android Botnet features for detection mechanism. *Advanced Science Letters*, 23(6), 5314-5317.

11. Nguyen, T.; McDonald, J.T.; and Glisson, W. B. (2017). Exploitation and detection of a malicious mobile application. in *Proceedings of the 50th Hawaii International Conference on System Sciences*.
12. Abdullah, Z.; Saudi, M.M.; and Anuar, N.B. (2014). Mobile botnet detection: Proof of concept. *Control and System Graduate Research Colloquium (ICSGRC), 2014 IEEE 5th*, 257-262.
13. Burguera, I.; Zurutuza, U.; and Nadjm-Tehrani, S. (2011). Crowdroid: behavior-based malware detection system for Android. *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 15-26.
14. Ghani, S.M.A.; Abdollah, M.F.; Yusof, R.; and Mas'ud M.Z. (2015). Recognizing API features for malware detection using static analysis. *Journal of Wireless Networking and Communications*, 5(2A), 6-12.
15. Feldman, S.; Stadther, D.; and Wang, B. (2014). Manilyzer: automated android malware detection through manifest analysis. *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, 767-772.
16. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; and Rieck, K. (2014). DREBIN: effective and explainable detection of android malware in your pocket.. *NDSS Symposium 2014*, *Briefing Paper*, 1-12.
17. Yusof, M.; Saudi, M.M.; and Ridzuan, F. (2017). A new mobile botnet classification based on permission and API calls. *Emerging Security Technologies (EST), 2017 Seventh International Conference*, 122-127.
18. Li, Z.; Sun, L.; Yan, Q.; Srisa-an, W.; and Chen, Z. (2016). DroidClassifier: efficient adaptive mining of application-layer header for classifying android malware. *International Conference on Security and Privacy in Communication Systems*, 597-616.
19. Lindorfer, M.; Neugschwandtner, M.; Weichselbaum, L.; Fratantonio, Y.; Van Der Veen, V.; and Platzer, C. (2014). Andrubis--1,000,000 apps later: A view on current Android malware behaviors. *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on*, 3-17.
20. Talha, K.A.; Alper, D.I.; and Aydin, C. (2015). APK auditor: Permission-based android malware detection system. *Digital Investigation*, 13, 1-14.
21. Fayyad, U.M.; Piatetsky-Shapiro, G.; and Smyth, P. (2000). The KDD process for extracting useful knowledge from volumes of data. *Communications of the. ACM*, 39(11), 27-34.
22. Witten, I.H.; Frank, E.; Hall, M.A.; and Pal, C.J. (2016). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann.