View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Crossref

Android Malware Classification by Applying Online Machine Learning

Abdurrahman Pektaş[1], Mahmut Çavdar[2], and Tankut Acarman[2(✉)]

[1] The Scientific and Technological Research Council of Turkey, Ankara, Turkey
[2] Computer Engineering Department, Galatasaray University, İstanbul, Turkey
acarmant@gmail.com

Abstract. A malware is deployed to execute malicious activities in the compromised operating systems. The widespread use of android smartphones with high speed Internet and permissions granted to applications for accessing internal logs provides a favorable environment for the execution of unauthorized and malicious activities. The major risk and challenge lies along classification of a large volume and variety of malware. A malware may evolve and continue to hide its malicious activies against security systems. Knowing malware features a priori and classification of a malware plays a crucial role at defending the safety and liability critical user's information. In this paper, we study android malware activities, features and apply online machine learning algorithm to classify a new android malware. We extract a fairly adequate set of malware features and we evaluate a machine learning based classification method. The runtime model is built and it can be implemented to detect variants of an android malware. The metrics illustrate the effectiveness of the proposed classification method.

1 Introduction

According to Internet Security Report, 1.4 billion smartphones were sold in 2015 and 83,3 % phones were running Android, [1]. Their users may save information about their personal identities, online payment system access and user's credentials. Malware authors, cyber criminals aim to steal these information via the distribution and installation of android applications. Overall, 3.3 million applications were classified as malware in 2015. Malware authors deliver this large variety and volume of malicious software by using advanced obfuscation techniques. Therefore, behavior-based malware analysis and classification of a malware sample to its original family plays a crucial and timely role at taking security and protection counter measures.

Android is a complete operating system that uses Android application (app) package (APK) for distribution and installation of mobile apps. APK file contains components which share a set of resources like database, preference, files, classes compiled in the dex file format, etc., App components are divided in four categories: *activities* handling the user interaction; *services* carrying out background tasks; *content providers* managing app's data; *broadcast receivers*

© The Author(s) 2016
T. Czachórski et al. (Eds.): ISCIS 2016, CCIS 659, pp. 72–80, 2016.
DOI: 10.1007/978-3-319-47217-1_8

Table 1. List of system commands and command's execution frequency by our malware test set

Command	Description	Frequency
/system/bin/cat (i.e. cat)	display files	33
logCat	reads the compressed logging files and outputs human-readable messages	13
ping	verifies IP-level connectivity by using ICMP	6
chmod	used to change the permissions of files or directories	4
ln	creates a link to an existing file	3
mount	attaches additional filesystem	2
echo	outputs text to the screen or a file	2
su	used to execute commands with the privileges of another account	2
id	print user ID and group ID of the current user	2

assuring communications between components, app's, even more Android OS. The manifest declares the app's components and how they interact. Also user permissions required by the apps are placed in the manifest file. Android is a privilege-separated operating system, in which each application runs with a distinct system identity (Linux user ID and group ID). Parts of the system are also separated into distinct identities. Linux thereby isolates applications from each other and from the system.

Several commands can be used to infect Android devices. For example, Cat command, i.e., System/bin/cat displays files in the system and it can be executed for malicious purposes. The command-line tool LogCat can be used for viewing the internal logs. Log messages may include privacy-related information. An app can access the log file by giving every app the READ_LOGS permission with aid of the *chmod* command. The list of commands is described in Table 1.

In line with the emerging market of android smartphones, detection and classification of its malware has attracted a lot of attention. Static analysis of the executables by using commands, and modelling of malware features by using permissions and API calls is presented for the detection of a malware in [2,3]. K-means algorithm for clustering and a decision tree learning algorithm for classification of a malware is presented by monitoring various permission based features and events extracted from applications in [4]. A learning model database is obtained by collecting the extracted features and N-gram signatures are created in [5]. Text mining and information retrieval is applied for the static analysis of a malware in [6]. In [7], a heuristics approach by using 39 different behaviour flags such as Java API calls, presence of embedded executables and code size is developed to determine whether an application is malicious or not. A deep learning for automatic generation of malware signature is studied to detect a majority of new variants of a malware in [8]. And, a detection model is trained with the information gathered via the communication among components. A security framework has been

deployed by an European project called NEMESYS for gathering and analyzing information about the nature of cyber-attacks targeting mobile devices and presented a model-based approach for detection of anomalies [9–11].

The paper is organized as follows: In Sect. 2, we present the selected features. In Sect. 3, we implement online machine learning algorithm to the classification of malware samples and we evaluate the results. Finally, we conclude our paper.

2 Feature Set

Cuckoo Sandbox is an open source analysis system and relies on virtualization technology to run a given file, [12]. It can analyze both executable and non-executable files and monitor the run-time activities. In this study, we extracted

Table 2. Features and their types

Feature category	Type	Value
commands	String	/system/bin/cat
services	String	com.houseads.AdService,
		com.applovin.sdk.AppLovinService,'
fingerprint	String	getSimCountryIso, getDeviceId, getLine1Number
permissions	String	INTERNET, ACCESS_NETWORK_STATE, READ_PHONE_STATE, GET_ACCOUNTS
data_leak	String	getAccounts
file_accessed	String	/proc/net/if_inet6, /proc/meminfo ...
httpConnections	String	http://houseads.eu/ads/new_user.php?id=147 &im= 351451208401216 &l=en&c=us&bm =Nexus+5&bv=4.1.2&v=4.2&ct=UMTS &a=null&ts=04032016070451&m=&s=16
send_sms	Boolean	FALSE
receive_sms	Boolean	FALSE
read_sms	Boolean	FALSE
call_phone	Boolean	FALSE
ap_execute_shell_commands	Boolean	TRUE
app_queried_account_info	Boolean	TRUE
app_queried_installed_apps	Boolean	FALSE
app_queried_phone_number	Boolean	TRUE
app_queried_private_info	Boolean	FALSE
app_recording_audio	Boolean	FALSE
app_registered_receiver_runtime	Boolean	TRUE
app_uses_location	Boolean	FALSE
embedded_apk	Boolean	FALSE
is_dynamic_code	Boolean	TRUE
is_native_code	Boolean	FALSE
is_reflection_code	Boolean	TRUE

Table 3. Top 20 requested permissions

Permissions	Frequency
INTERNET	867
READ_PHONE_STATE	826
WRITE_EXTERNAL_STORAGE	764
ACCESS_NETWORK_STATE	744
SEND_SMS	565
INSTALL_SHORTCUT	535
ACCESS_WIFI_STATE	524
WAKE_LOCK	473
RECEIVE_BOOT_COMPLETED	420
VIBRATE	382
RECEIVE_SMS	348
GET_TASKS	337
WRITE_SETTINGS	306
READ_SMS	285
ACCESS_COARSE_LOCATION	281
READ_SETTINGS	278
CHANGE_WIFI_STATE	277
ACCESS_FINE_LOCATION	270
CALL_PHONE	215
SYSTEM_ALERT_WINDOW	182

the most significant and distinguishing behavioral features from the Cuckoo's analysis report. The list of android malware features is given in Table 2. The permissions requested by the applications are ranked according to their persistency in Table 3.

3 Implementation

The testing malware dataset is obtained from "VirusShare Malware Sharing Platform" ([13]), which provides a huge amount of different type malware including PE, HTML, Flash, Java, PDF, APK etc. All experiments were conducted under the Ubuntu 14.04 Desktop operating system with Intel(R) Core(TM) i5-2410M@2.30 GHz processor and 2 GB of RAM. The analysis with 5 guest machines took 5 days to analyze approximately 2000 samples. For labeling malware samples, we used Virustotal, an online web-based multi anti-virus scanner, [14]. The malware classes along their class-specific measures are given in Table 4.

Table 4. Malware families and their class-specific measures

Family	Code	#	Recall	Specificity	Precision	Balanced accuracy
android.trojan.fakeinst	1	193	0.94	0.98	0.94	0.96
android.riskware.smsreg	2	104	0.67	0.99	0.86	0.83
android.trojan.agent	3	79	0.60	1.00	1.00	0.80
android.adware.gingermaster	4	74	0.67	0.99	0.80	0.83
android.adware.adwo	5	69	0.83	1.00	1.00	0.92
android.trojan.smssend	6	66	1.00	0.84	0.35	0.92
android.trojan.smskey	7	48	0.25	1.00	1.00	0.63
android.adware.utchi	8	45	1.00	1.00	1.00	1.00
android.trojan.clicker	9	37	1.00	0.99	0.75	0.99
android.adware.appquanta	10	34	1.00	1.00	1.00	1.00
android.adware.plankton	11	34	0.50	1.00	1.00	0.75
android.trojan.fakeapp	12	19	1.00	1.00	1.00	1.00
android.trojan.boqx	13	18	0.50	1.00	1.00	0.75
android.trojan.killav	14	17	1.00	1.00	1.00	1.00
android.riskware.tocrenu	15	14	0.50	1.00	1.00	0.75
android.exploit.gingerbreak	16	12	1.00	1.00	1.00	1.00
android.trojan.bankun	17	12	1.00	1.00	1.00	1.00
android.trojan.smsspy	18	11	1.00	1.00	1.00	1.00

3.1 Online Classification Algorithms

In general, an online learning algorithm works in a sequence of consecutive rounds. At round t, the algorithm takes an instance $\boldsymbol{x}_t \in \mathbb{R}^d$, d-dimensional vector, as input to make the prediction $\hat{y}_t \in \{+1, -1\}$ (for binary classification) regarding to its current prediction model. After predicting, it receives the true label $y_t \in \{+1, -1\}$ and updates its model (a.k.a. hypothesis) based on prediction loss $\ell(y_t, \hat{y}_t)$ meaning the incompatibility between prediction and actual class. The goal of online learning is to minimize the total number of incorrect predictions; $sum(t : y_t \neq \hat{y}_t)$. Pseudo-code for generic online learning is given in Algorithm-1.

3.2 Classification Metrics

To evaluate the proposed method, the following class-specific metrics are used: ***precision***, ***recall*** (a.k.a. sensitivity), ***specificity***, ***balanced accuracy***, and ***overall accuracy*** (the overall correctness of the model). Recall is the probability for a sample in class c to be classified correctly. On the contrary, specificity is

Algorithm 1. Generic online learning algorithm

Input : $\boldsymbol{w}_{t=1} = (0, ..., 0)$
foreach *round t in (1,2,..,N)* **do**
  Receive instance $\boldsymbol{x}_t \in \mathbb{R}^d$
  Predict label of $\boldsymbol{x}_t$: $\hat{y}_t = sign(\boldsymbol{x}_t.\boldsymbol{w}_t)$
  Obtain true label of the $\boldsymbol{x}_t$: $y_t \in \{+1, -1\}$
  Calculate the loss: ℓ_t
  Update the weights: $\boldsymbol{w}_{t+1}$
end
Output: $\boldsymbol{w}_{t=N} = (w_1, ..., w_d)$

the probability for a sample not in class c to be classified correctly. The metrics are given as follows:

$$precision = \frac{tp}{tp\ +\ fp} \tag{1}$$

$$recall = \frac{tp}{tp\ +\ fn} \tag{2}$$

$$specificity = \frac{tn}{tn\ +\ fp} \tag{3}$$

$$balanced\ accuracy = \frac{recall\ +\ specificity}{2} = \frac{1}{2}\left(\frac{tp}{tp\ +\ fn} + \frac{tn}{tn\ +\ fp}\right) \tag{4}$$

$$accuracy = \frac{correctly\ classified\ instances}{total\ number\ of\ instances} \tag{5}$$

For instance, consider a given class c. True positives (tp) refer to the number of the samples in class c that are correctly classified while true negatives (tn) are the number of the samples not in class c that are correctly classified. False positives (fp) refer the number of the samples not in class c that are incorrectly classified. Similarly, false negatives (fn) are the number of the samples in class c that are incorrectly classified. The terms positive and negative indicate the classifier's success, and true and false denotes whether or not the prediction matches with ground truth label.

3.3 Testing Accuracy Results

The accuracy of testing is computed subject to different value of regularization weight parameter. The regularization weight parameter is denoted by C and determines the size of weight change at each iteration. A larger value means a possibility of a higher change in the updated weight vector and the model is created faster. But as a consequence, the model becomes more dependent to the training set and more susceptible to noise data. 10-fold cross-validation approach is used. The class-wise results for the most successful algorithm (i.e. Confidence-weighted linear classification in [15]) according to the different weight C are given in Table 5.

Table 5. Classification accuracy versus different regularization weight parameter

$C = 1$	$C = 2$	$C = 3$	$C = 4$	$C = 5$	$C = 10$	$C = 100$
0.81	0.83	0.84	0.89	0.80	0.78	0.76

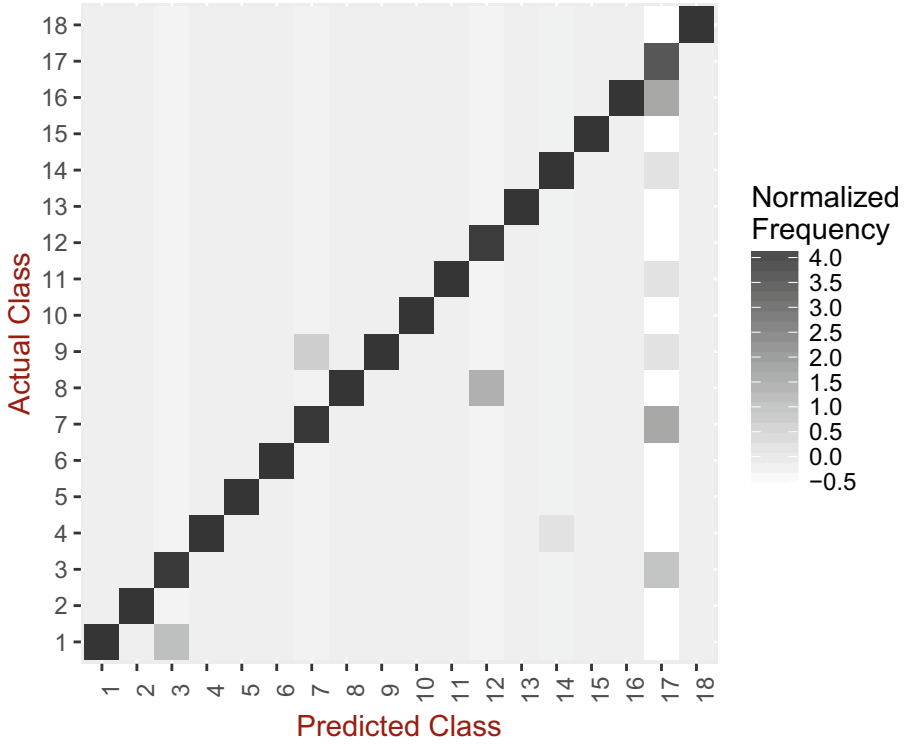

Fig. 1. Normalized confusion matrix

To analyze how well the classifier can recognize instance of different classes, we created the confusion matrix as shown in Fig. 1. The confusion matrix displays the number of correct and incorrect predictions made by the classifier with respect to ground truth (actual classes). The diagonal elements in the matrix represent the number of correctly classified instances for each class, while the off-diagonal elements represent the number of misclassified elements by the classifier. The higher the diagonal values of the confusion matrix are, the better the model fits the dataset (higher accuracy in individual family prediction). Since android.trojan.bankun family combines many functionalities executed also by other families in our dataset, android.trojan.agent, android.trojan.smskey and android.exploit.gingerbreak are incorrectly estimated as android.trojan.bankun.

4 Conclusions

This paper addresses the challenge of classifying android malware samples by using runtime artifacts while being robust to obfuscation. The presented classification system is usable on a large scale in real world due to its online machine learning methodology. The proposed method uses run-time behaviors of an executable to build the feature vector. We evaluated an online machine learning algorithm with 2000 samples belonging to 18 families. The results of this study indicate that runtime behavior modeling is a useful approach for classifying an android malware.

Acknowledgments. The authors gratefully acknowledge the support of Galatasaray University, scientific research support program under grant #16.401.004.

Open Access. This chapter is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, duplication, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, a link is provided to the Creative Commons license and any changes made are indicated.

The images or other third party material in this chapter are included in the work's Creative Commons license, unless indicated otherwise in the credit line; if such material is not included in the work's Creative Commons license and the respective action is not permitted by statutory regulation, users will need to obtain permission from the license holder to duplicate, adapt or reproduce the material.

References

1. Internet Security Threat Report (2016) Available via Symantec. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf. Cited 15 Jun 2016
2. Schmidt, A.D., Bye, R., Schmidt, H.G., Clausen, J., Kiraz, O.: Static analysis of executables for collaborative malware detection on Android. In: 2009 IEEE International Conference on Communications, Dresden, pp. 1–5 (2009)
3. Peiravian, N., Zhu, X.: Machine learning for android malware detection using permission and API calls. In: Proceedings of the ICTAI 2013, The IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 300–305 (2013)
4. Aung, Z., Zaw, W.: Permission-based android malware detection. Int. J. Scient. Technol. Res. **2**, 228–234 (2013)
5. Dhaya, R., Poongodi, M.: Detecting software vulnerabilities in android using static analysis. In: Proceedings of ICACCCT, Communication IEEE International Conference on Advanced Communication Control and Computing Technologies, pp. 915–918 (2014)
6. Tangil, G.S., Tapiador, J.E., Lopez, P.P., Blasco, J.: A text mining approach to analyzing and classifying code structures in android malware families. Expert Syst. Appl. **4**, 1104–1117 (2014)
7. Apvrille, A., Strazzere, T.: Reducing the window of opportunity for Android malware gotta catch em all. J. Comput. Virol. **8**, 61–71 (2012)

8. Xu, K., Li, Y., Deng, R.H.: ICCDetector: ICC-based malware detection on Android. Inf. Forensics Sec. **11**, 1252–1264 (2016)
9. Abdelrahman, O.H., Gelenbe, E., Görbil, G., Oklander, B.: Mobile network anomaly detection and mitigation: the NEMESYS approach. In: Gelenbe, E., Lent, R. (eds.) Information Sciences and Systems. LNEE, vol. 264, pp. 429–438. Springer, Switzerland (2013). doi:10.1007/978-3-319-01604-7_42
10. Gelenbe, E., Görbil, G., Tzovaras, D., Liebergeld, S., Garcia, D., Baltatu, M., Lyberopoulos, G.: NEMESYS: enhanced network security for seamless service provisioning in the smart mobile ecosystem. In: Information Sciences and Systems (2013). doi:10.1007/978-3-319-01604-7_36
11. Gelenbe, E., Görbil, G., Tzovaras, D., Liebergeld, S., Garcia, D., Baltatu, M., Lyberopoulos, G.: Security for smart mobile networks: the NEMESYS approach. In: Proceedings of the Global High Tech Congress on Electronics, pp. 63–69. IEEE (2013)
12. Cuckoo Sandbox (2016). cuckoosandbox.org. Cited 15 Jun 2016
13. Virusshare: Malware Sharing Platform (2016). http://www.virusshare.com/
14. Virustotal: An online multiple AV Scan Service (2016). http://www.virustotal.com/
15. Dredze, M., Crammer, K., Pereira, F.: Confidence-weighted linear classification. In: Proceedings of the 25th International Conference on Machine Learning, pp. 264–271. ACM (2008)