Android Malware Detection Using Permission Analysis

Hossain Shahriar and Mahbubul Islam
Department of Information Technology
Kennesaw State University
GA, USA
hshahria@kennesaw.edu
mislam9@students.kennesaw.edu

Victor Clincy
Department of Computer Science
Kennesaw State University
GA, USA
vclincy@kennesaw.edu

Abstract— **Android applications are widely used by millions of users to perform many different activities. However, many applications have been reported to be malware performing activities not matching with their expected behaviors (e.g., sending SMS message to premium numbers). The existing relevant approaches that identify these applications (malware detection technique) suffer from performance issues where the occurrence of false negatives (FN) remain high. These approaches are not scalable and provides little flexibility to query based on a set of suspicious permission to identify a set of highly relevant known anomalous applications to examine further. This paper proposes a new approach to reduce the number of apps that must be sandboxed in order to determine if they are malicious. We first examine the source of applications to identify list of permissions and find a set of highly close and relevant applications from a given set. The identified most relevant application category's permission is then checked to find if there is significant overlapping or not to identify an application as suspected anomalous. We apply Latent Semantic Indexing (LSI) to identify malware application. Our initial evaluation results suggest that the proposed approach can identify malware applications accurately.**

Keywords — Permission analysis, Android malware, Latent Semantic Indexing.

I. Introduction

It is well known that anomalous (malware) Android applications are repackaged around popular applications available in the market [1]. Typical activities from anomalous applications may also include leveraging of root level exploits, changing of device password, communicating with external servers and making phone calls. A large number of publications have recently addressed Android malware detection based on classification [6, 8] and detection [9, 10, 11-14, 16, 17, 18, 19]. These approaches identify the presence of anomalous activities or risky permission sets present in suspected applications to identify an application as anomalous. These current approaches, however, suffer from performance issues (false negative warnings remain high) and are not scalable. They do not provide the flexibility for a developer to query an application's permission and find the most relevant category an application fits.

A key challenge to identify a suspected application as anomalous is to understand the context of the list of dangerous permissions present in the application source file (e.g., writing to file, reading contact information, sending SMS). A good (benign) application may require similar set permissions present in an anomalous application. A number of research works have addressed this ambiguity [21, 22] of identifying a list of dangerous permissions by identifying top occurring and common permissions based on application categories (*e.g.*, game, communication). Then, an application is labeled as anomalous if its list of permission is deemed rare considering the known set of permissions for a given category. While this approach is practical, the key challenge is to identify the most relevant set of permissions for a given category of application. Information retrieval is widely used to find a set of most relevant documents for a given set of search key words. This work attempts to apply an information retrieval technique by identifying suitable set of applications that are close fit based on permission search with context (category of applications).

In this paper, we propose an approach to identify anomalous applications. We apply two phase analysis: static and dynamic. During static phase, we apply the concept of *Latent Semantic Indexing* (LSI [3, 4]) from information retrieval. A query for a suspected application is formed and applied to find the list of most closely related applications, then see if the applications category of permissions is of highly matching or not. To reduce false positive warnings, we apply a dynamic analysis where we inspect the invoked functionalities to confirm the expected behaviors per permission list.

Our proposed approach has advantages compared to other approaches. Firstly, it uses the hypothesis that most malware applications are repackaged from popular categories. Secondly, it considers multiple categories as possibilities. It allows a tester to check for suspicious activities in lower dimensions, rather than looking at all applications. Thirdly, it improves false positive and false negative rates compared to other approaches.

The paper is organized as follows: Section II provides an overview of related work. In Section III, the proposed approach is discussed in details. Section IV discusses the experimental evaluation. Section V presents the conclusions and discusses future work.

II. Related Work

A number of prior related works have the similar goal to ours, which is to identify anomalous applications. Sarma *et al.* [15] analyzed list of permissions for a new application and then found how close these set of permissions compared to other applications in the same category. Their approach relied on two broader categories of applications and games. The games were further subdivided into eight categories and the applications were further subdivided into 26 categories. The analysis relied on identifying rate permissions (e.g., READ_HISTORY_BOOKMARKS, READ_SMS, WRITE_CONTACTS) that should not appear in an application. Their study considered a small set of malware applications while identifying the list of rare permissions with respect to benign applications.

Later Arp *et al.* [22] addressed this issue through a large set of malware applications. Their approach obtained features from a benchmark consisting of both good and malware applications. They analyzed program sources and develop feature vectors comprising permissions, API calls, network addresses, followed by applying a machine learning approach (support vector machine) to differentiate between the two types of applications. The detection occured on the mobile device where an application was labeled as benign or malware.

Lindorfer *et al.* [20] developed the Marvin tool that generates a risk by analyzing an application using static (e.g., certificate metadata, class structure, app name) and dynamic features (networking activity, SMS activity, file operation, phone activity, data leak). Bhandari *et al.* [21] developed a similar approach that combines static (outside the device) and dynamic analysis (on the device). They generated risk indicators or scores representing how malicious an application could be based on training data, which is learned with machine learning algorithms.

In contrast to these efforts, we propose risk inquiry based on a customized list of terms or words that may indicate risks (permissions, APIs). We explore an information retrieval technique to find how relevant known risky permissions and APIs are used in the list of an available set of malware benchmarks to develop a model-of-learning. Then the model-of-learning is applied to detect actual malware. We show a highlight and recap of these related approaches in Table 1.

Now, we discuss other works that address the broader problem of malware detection and mitigations. Crusselle *et al.* [12] detected malware applications by comparing data dependency graphs between known applications and new application.

Li *et al.* [13] proposed a feature hashing-based technique. They first identify k-grams of various opcode sequence patterns within each basic block and consider them as features. The presence or absence of each feature in DEX files are encoded in a vector. All vectors obtained from the files are merged to obtain the fingerprint of each application.

Zhou *et al.* [14] computed hash values for each local unit of opcode sequence of the classes (DEX files). The long opcode was handled by splitting the opcodes into small units and computing hashes for each split unit. Finally, all individual hashes ~~are~~ were combined into one hash value. By doing so, any additional inserted code ~~is~~ was detected, not only for the overall application, but also for the specific instruction sets. The approach suffered from false positive warnings if the inserted dummy opcode ~~does~~ did not have any negative impact.

We are aware of approaches that classify malware applications and their detection technique. Porter *et al.* [6] performed a survey on malicious characteristics for mobile device malware in 2011. Cooper *et al.* [8] classify android malware detection techniques and comparatively identify the advantages and disadvantages including static analysis, sandboxing, and machine learning approaches. Enck *et al.* [9] analyzed a large set of android applications and identified dataflow, structure, and semantic patterns. The dataflow patterns identify whether any sensitive data information should not be sent outside. Enck *et al.* [10] also proposed a rule-based certification technique to check the presence of undesirable properties in applications suspected as malware.

Batyuk *et al.* [11] performed static analysis on binary code of android applications (after decompressing APK and decoding Java byte code into Smali assembly language. They look for the presence of APIs that may be relevant of reading sensitive information (*e.g.*, IMEI or device identifier, IMSI or subscriber identifier, phone number, writing information to output stream). Yang *et al.* [16] detected money stealing malware by examining the manifest file of android applications to see if billing permission was present.

III. Proposed Detection Approach

A. Latent Semantic Indexing(LSI)-based Permission Analysis

LSI is a well-known information retrieval technique where a set of words are used to identify the most relevant set of documents. The technique relies on computing a matrix where rows are set of words, and columns are set of documents. This matrix is then reduced to find the most important set of documents using singular value decomposition (SVD) technique.

A set of known anomalous applications comprised of documents. In particular, we examine the permission lists present in the XML files after decompiling the apk into source files with appropriate tools (apktool [2]). The terms are defined based on a known set of dangerous permissions. We examine related literature works and identified a set of dangerous permissions.

Queries are formulated based on the list of dangerous permissions of interests. Then, a query vector is used to rank the short listed applications using five steps:

Step1: Obtain a list of known anomalous applications and consider them as documents. Identify the category of these applications and most occurring permissions (to be used in step 4). Obtain the list of permissions from a given application to test and consider them as keywords.

Step2: Generate a matrix M where columns are set of documents (or known malware applications) and rows are permissions keywords**.** Entries in the matrix represent the number of occurrences of the permissions in the documents.

Step3: Apply SVD to matrix M and reduce the dimension of *M* and identify the most relevant application (the first column following SVD would be the highest relevant, flowed by the second column and so on).

Step4: Identify the set of short listed applications (documents) and obtain the common set of permissions from their respective categories.

Step5: If a tested application's permission significantly matches with the common set of permissions from the most relevant categories, we label the application as anomalous and proceed to the dynamic analysis step. If not, we identify it as normal application.

Not all risky permissions may be invoked by applications. Moreover, risky permission may be present in both good and anomalous applications. Thus, only finding the presence of risky permission is not enough and may lead to false positive warning. To reduce false positive warning, we confirm the maliciousness by testing the application within a sandbox (virtual emulator with introspection of invoked methods facility). If in the sandbox environment, the application shows the expected behaviors matching with list of permissions, then we label the application as anomalous.

We now discuss how to compute SVD from M (term-document matrix) in next subsection. Then, we provide an example analysis for the reader.

B. Computation of SVD from term-document matrix

We now discuss the detailed computation steps of SVD [14] from the term document matrix.

Step1: Build Term-Document matrix $A = [m*n]$, where m = number of terms (permissions), n = number of documents; $q = [m*1]$ as query vector. We denote terms as $t_1 \ldots t_m$, documents as $d_1 \ldots d_n$.

Step2: Perform Singular Value Decomposition (SVD) on A to obtain three matrices U, S, and V^T such that $A = U * S * V^T$. Here, * is matrix multiplication operator, U is the Eigen vector of the matrix A, S is the diagonal matrix having singular value, V is Inverse of U matrix, V^T is the transpose of V matrix.

Step3: Choose k out of n ($k < n$) to reduce the dimensionality of the A matrix.

Step 4: Define A_k by reducing the dimensions of A from $m*n$ to $m*k$. We define S_K from S by reducing dimension from $m*n$ to $m*k$ ($k < n$). Finally, define V_K from V by reducing the dimension from $n*n$ to $k*k$. Each document d_i is approximated by the corresponding row vector of v_i from V_K.

Step 5: Obtain the approximate query vector in reduced space from q to q_k, by multiplying the three matrices $q* U_K * S_K^{-1}$. This means the original query vector is reduced from $m*1$ to $k*1$. Here, S_K^{-1} is the inverse matrix of S_K.

Step 6: Measure similarity using cosine distance formula as follows: $Sim(q_k, d_k) = q_k.d_k / (|q_k||d_k|)$

Here, |…| represents the norm operation, $q_k.d_k$ represents the dot product for the two column vectors.

C. Example of finding malware application

We assume d_1 - d_4 are four anomalous apk files (documents). We define a set of terms related to permissions and find the presence of the terms in the documents to define A matrix (step1) as shown in Table I. Here, we show four well known risky permissions.

TABLE I. TERM DOCUMENT MATRIX (A)

Term	d_1	d_2	d_3	d_4	q
SEND_SMS	3	3	2	1	1
CALL_PHONE	0	1	1	4	0
INTERNET	4	3	2	0	1
READ_CONTACT	2	1	3	1	1

Let us assume a query (q). After applying SVD[1] of Step2, we obtain S, U, V^T matrices by performing Singular Value Decomposition of A matrix. These matrices are shown in Tables II, III, and IV. We show the V matrix in Table V.

TABLE II. SINGUALR VALUE MATRIX (S)

8.1	0	0	0
0	4.08	0	0
0	0	1.70	0
0	0	0	0.30

TABLE III. U MATRIX

-0.58	0.05	-0.29	-0.75
-0.25	-0.92	-0.21	0.21
-0.63	0.37	-0.25	0.62
-0.43	-0.09	0.89	-0.09

TABLE IV. VT MATRIX

-0.63	-0.53	-0.49	-0.24
0.36	0.06	-0.08	-0.92
-0.07	-0.57	0.80	-0.14
0.67	-0.61	-0.33	0.25

[1] Online tools for SVD accessed from, http://www.dotnumerics.com/MatrixCalculator/default.aspx

TABLE V. V MATRIX

-0.63	0.36	-0.07	0.67
-0.53	0.06	-0.57	-0.61
-0.49	-0.08	0.80	-0.33
-0.24	-0.92	-0.14	0.25

Let us assume that we reduce the dimensionality of S from 9 X 4 to 2 X 2 (setp3). The S_K, U_K, and V_K are shown in Tables VI, VII, and VIII, respectively (step 4).

TABLE VI. SK MATRIX

8.1	0
0	4.02

TABLE VII. UK MATRIX

-0.58	0.05
-0.25	-0.92
-0.63	0.37
-0.43	-0.09

TABLE VIII. VK MATRIX

-0.63	0.36
-0.53	0.06
-0.49	-0.08
-0.24	-0.92

Now, the new document vectors in the reduced space would be the V_K matrix as shown in Table IX (step 4):

TABLE IX. REFINED DOCUMENT MATRIX

d_1	-0.63	0.36
d_2	-0.53	0.06
d_3	-0.49	-0.08
d_4	-0.24	-0.92

We now find q_K (step 5) by multiplying $q^T*U_K*S_K^{-1}$ which is shown in Table X. Here, S_K^{-1} is the inverse matrix of S_K.

TABLE X. QK VECTOR

q_k	-0.202	0.078

We now apply step 6 to compute the distance between q_k and each of the documents as shown in Table XI. Thus, the given application is closest to d1 type.

TABLE XI. SIMILARITY BETWEEN QUERY AND DOCUMENT

Cosine Similarity	Distance
Sim (q_k, d_1)	0.15
Sim (q_k, d_2)	0.11
Sim (q_k, d_3)	0.09
Sim (q_k, d_4)	-0.02

We have a ranking of documents: $d_1 > d_2 > d_3 > d_4$. This means the highest relevant application is d_1 followed by d_2, d_3, and d_4. Assume, we consider examining only top two relevant application, we now examine the list of most common occurring permissions and see if it matches significantly (*e.g.*, >70%, see evaluation section) or not with the query. If there is no significant match, we label the application as benign. If there is matching, we run the application in sandbox to confirm activities and label it as malware.

IV. EVALUATION

We use apktool [5] to retrieve permission lists from APK sources. To perform the dynamic analysis, we use a modified Dalvik VM emulator to inspect activities and invoked calls from logs generated by the virtual machine.

To evaluate our approach, we gathered 1200 malware applications from Malgenome project (959 samples) [5] and Contagio dump (241 samples) [7] website. The malware samples were analyzed and mapped to five categories (c1-c5): games, entertainment, live wallpaper, music and video, and communication. Most of the applications fit with game, followed by entertainment and communication categories (see Table XII). We define categories based on Google Market Place. Currently, there are total 29 categories of applications available in the Market Place [30]. Table XIII shows the top nine permissions present in these applications (two example categories).

TABLE XII. DISTRIBUTION OF MALWARE APPLICATION CATEGORIES

Application category	Malgenome	Contagio
Games (c1)	68%	84%
Entertainment (c2)	12%	6%
Live Wallpaper (c3)	3%	2%
Music and Video (c4)	4%	2%
Communication (c5)	13%	6%

We mixed the samples from both data sources and split them between training sample and testing samples randomly. We take 80% (960 applications) samples to build term-document matrix for LSI. We validate our approach with 20% sample (240 applications) by forming permission queries and see if the closely related categories of applications fall within the identified category of a test application.

TABLE XIII. TOP 9 PERMISSIONS IN TWO CATEGORIES

Games (c1)	Entertainment (c2)
INTERNET	INTERNET
SEND_SMS	RECORD_AUDIO
GET_ACCOUNTS	GET_ACCOUNTS
CALL_PHONE	SEND_SMS
SEND_SMS	WRITE_SMS
ACCESS_COARSE_LOCATION	CALL_PHONE
RECEIVE_MMS	READ_LOGS
WRITE_CONTACTS	WRITE_CONTACTS
WRITE_EXTERNAL_STORAGE	WRITE_EXTERNAL_STORAGE

To identify the list of known risky permission list, we look at the literature and reuse the one reported in the work of [15]. Table XIV shows the list of risky permissions from three categories: privacy, billing, and file/system.

Table XV summarizes the results (average) for query size nine (q(6)) for test dataset. The first row indicates that the LSI identified on average 76% of times tested applications closely relevant with c1, followed by c2 (12%), c3 (6%), c4 (3%), and c5 (3%).

The permission matching column shows the average of overlapping with actual matching of permission lists between the closest category identified by LSI and an application under test. The sandbox column indicates that all testes samples were confirmed to be anomalous as they displayed expected behaviors inside virtual device emulator. Similarly, the other two rows can be explained.

TABLE XIV. LIST OF RISKY PERMISSIONS

Permission category	Name
Privacy	ACCESS_COARSE_LOCATION
	PROCESS_OUTGOING_CALLS
	READ_CALENDAR
	READ_HISTORY_BOOKMARKS
	READ_PHONE_STATE
	READ_SMS
	RECEIVE_MMS
	RECORD_AUDIO
	READ_LOGS
Billing	CALL_PHONE
	INTERNET
	SEND_SMS
File and system operation	MOUNT_UNMOUNT_FILESYSTEMS
	WRITE_CALENDAR
	WRITE_CONTACTS
	WRITE_HISTORY_BOOKMARKS
	WRITE_SMS
	WRITE_EXTERNAL_STORAGE
	NFC
	GET_ACCOUNTS

TABLE XV. EVALUATION RESULTS Q(6)

c1	c2	c3	c4	c5	Permission matching	Dynamically Verified
76%	12%	6%	3%	3%	84%	100%
82%	10%	5%	2%	1%	93%	100%
89%	5%	3%	2%	1%	99%	100%

V. CONCLUSIONS AND FUTURE WORK

This work proposed a new approach to detect anomalous Android applications by identifying the most relevant category of permissions to match from and then confirming behaviors in an emulator environment. The relevant category is identified using Latent Semantic Index (LSI) analysis. The approach has the potential to discover new anomalous applications. We evaluate the approach using two open source malware repositories. The initial results show that our approach can detect malware applications. Our future work includes evaluating more applications and query types, and apply the approach to address other security vulnerabilities.

REFERENCES

[1] Y. Zhou and X. Jiang, “Dissecting Android Malware: Characterization and Evolution,” *Proc. of IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2012, pp. 95-109.

[2] A tool for reverse engineering Android apk files, https://ibotpeaches.github.io/Apktool/

[3] C. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 2008, Cambridge University Press.

[4] A. Marcus and J. Maletic, “Using Latent Semantic Analysis to Identify Similarities in Source Code to Support Program Understanding,” *Proc. of 12th IEEE International Conference on Tools with Artificial Intelligence*, November 2000, pp. 46–53.

[5] Android Malware Genome Project, http://www.malgenomeproject.org/

[6] A. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, “A Survey of Mobile Malware in the Wild,” *Proc. of the ACM Workshop Security and Privacy in Mobile Devices (SPMD)*, 2011, pp. 3-14.

[7] Contagio malware dump, Accessed from http://contagiodump.blogspot.com/

[8] V. Cooper, H. Shahriar, and H. Haddad, “A Survey of Android Malware Characteristics and Mitigation Techniques,” *Proc. of the 11th International Conference on Information Technology: New Generations*, IEEE CPS, Las Vegas, USA, April 2014, pp. 327-332.

[9] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, “A Study of Android Application Security,” *Proc. of USENIX Security Symposium*, August 2011.

[10] W. Enck, M. Ongtang, and P. McDaniel, “On Lightweight Mobile Phone Application Certification,” *Proc. 16th ACM Conf. Computer and Communications Security (CCS 09)*, ACM, 2009, pp. 235-245.

[11] L. Batyuk, M. Herpich, S. Camtepe, K. Raddatz, A. Schmidt, and S. Albayrak, “Using Static Analysis for Automatic Assessment and Mitigation of Unwanted and Malicious Activities within Android Applications,” *Proc. of 6th*

International Conference on Malicious and Unwanted Software (MALWARE), October 20011, pp. 66-72.

[12] J. Crussell, C. Gibler, and H. Chen. Attack of the clones: Detecting cloned applications on android markets. *Proc. of European Symposium on Research in Computer Security (ESORICS)*, pages 37–54, 2012.

[13] S. Li, *Juxtapp: A scalable system for detecting code reuse among android applications*, Master's thesis, EECS Department, University of California, Berkeley, May 2012. Accessed from http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-111.html

[14] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party android marketplaces," *Proc. of the 2nd ACM conference on Data and Application Security and Privacy (CODASPY)*, pp. 317–326.

[15] Bhaskar Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, "Android Permissions: A Perspective Combining Risks and Benefits," Proc. of SACMAT, June 20–22, 2012, Newark, New Jersey, USA, pp. 13-22.

[16] C. Yang, V. Yegneswaran, P. Porras, and G. Gu, "Detecting Money-Stealing Apps in Alternative Android Markets," *Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS)*, October 2012, Raleigh, North Carolina, USA, pp. 1034-1036.

[17] Y. Zhou and X. Jian, "Detecting Passive Content Leaks and Pollution in Android Applications," Proc. of 20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013.

[18] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic Detection of Capability Leaks in Stock Android Smartphones," In *Proceedings of the 19th Annual Symposium on Network and Distributed System Security* (NDSS), Feb 2012, San Diego, CA, USA.

[19] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards Taming Privilege-Escalation Attacks on Android," *Proc. of the 19th Annual Symposium on Network and Distributed System Security* (NDSS), Feb 2012, San Diego, CA, USA.

[20] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis," *Proceedings of the 39th Annual International Computers, Software & Applications Conference (COMPSAC)*, Taichung, Taiwan, 2015, pp. 422-433.

[21] S. Bhandari, R. Gupta, V. Laxmi, M. Gaur, A. Zemmari, M. Anikeev, "DRACO: DRoid analyst combo an android malware analysis framework," *Proceedings of the 8th International Conference on Security of Information and Networks (SIN)*, September 2015, pp. 283-289.

[22] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," *Proc. of Annual Symposium on Network and Distributed System Security* (NDSS). The Internet Society, 2014.