Detecting Mobile Application Malicious Behaviors Based on Data Flow of Source Code

Chia-Mei Chen
Department of Information Management
National Sun Yat-sen University
Kaohsiung, Taiwan
cchen@mail.nsysu.edu.tw

Je-Ming Lin
Department of Information Management
National Sun Yat-sen University
Kaohsiung, Taiwan
lin.13k@gmail.com

Gu-Hsin Lai
Department of Information Management
Chinese Culture University
Taipei, Taiwan
Lgx4@ulive.pccu.edu.tw

***Abstract*—Mobile devices have become powerful and popular. Most internet applications are ported to mobile platform. Confidential personal information such as credit card and passwords are stored in mobile device for convenience. Therefore, mobile devices become the attack targets due to financial gain. Mobile applications are published in many market platforms without verification; hence malicious mobile applications can be deployed in such marketplaces.**

Two approaches for detecting malware, dynamic and static analysis, are commonly used in the literature. Dynamic analysis requires is that analyst run suspicious apps in a controlled environment to observe the behavior of apps to determine if the app is malicious or not. However, Dynamic analysis is time consuming, as some mobile application might be triggered after certain amount of time or special input sequence. In this paper static analysis is adopted to detect mobile malware and sensitive information is tracked to check if it is been released or used by malicious malware.

In this paper, we present a mobile malware detection approach which is based on data flow of the reversed source code of the application. The proposed system tracks the data flow to detect and identify malicious behavior of malware in Android system. To validate the performance of proposed system, 252 malware form 19 families and 50 free apps from Google Play are used. The results proved that our method can successfully detecting malicious behaviors of Android APPs with the TPR 91.6%.

Keywords—Android, Mobile malware, Static analysis, Data flow

I. INTRODUCTION

As Internet-connected mobile device like smartphone or tablet have become popular, they have become new target for attackers for financial gain. For example, FakeInstaller, a widespread mobile malware family, sends SMS messages to premium rate numbers without the user's consent [4]. Some mobile malware use botnet techniques that not only send SMS messages to premium rate numbers, but also include a backdoor to receive commands from a remote server. FakeInstaller.S a variants of FakeInstaller used "Android Cloud to Device Messaging" to register the infected devices in a database and send them messages (URLs) from malware authors' Google accounts. [4].

Among platform of mobile device, Android holds 79.3% of the total market share [2] in mobile phones and tablet devices in 2013. The great volume of Android devices now becomes targets for attacker. According to F-Secure's report in 3Q 2013, there are 252 new Android threat families and new variants of existing families are discovered. No malware has been yet to be recorded in 2013 on the other platforms (Blackberry, iOS, Windows Phone) [3]. F-Secure's investigation (see figure 1) also shows that profit-motivated threats is increasing, which typically make monetary profit by sending premium-rate SMS messages from infected devices, without the users consent. This rise could be attributed to the continued growth in large SMS-sending trojan families such as FakeInst, OpFake, PremiumSms and SmsSend, whose developers keep churning out new variants. [3]

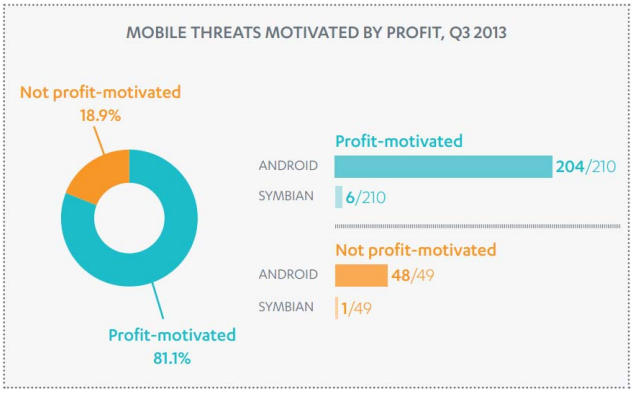

Figure 1: Comparison between new threats discovered in Q3 2013 that are profit-motivated versus non-profit-motivated ones [3]

In the Android Market, accumulated number of applications and games is over 1 million. In December 2013, there are over 80,000 new android apps were released [1]. A challenge for the detection of malware is that the count of apps in market grow so fast that a quick scan mechanism is needed.

There are two ways used to detect malware, they are dynamic and static analysis approach. Dynamic analysis first

978-1-4799-6566-3/14 $31.00 © 2014 IEEE
DOI 10.1109/TSA.2014.10

executes malware in a controlled environment (in most case, in a virtual machine), then, analysts observe and record malware's behavior for further analysis. However, authors of malware now use anti-VM technique to evade detecting. Besides, to observe and record malware's behavior, dynamic analysis systems need a lot of resource (Memory space, disk space and computational resource). Either in a powerful workstation or in a smartphone does not have enough resource to run dynamic analysis for the great volume of apps. Rather than executing malware, static analysis analyzes malware itself without executing it. The advantage of static analysis is that it can scan and check malware quickly. Due to this reason, we use static analysis approach to detect malware in Android system.

Security tools need to know which values in a program an attacker could potentially control. Using dataflow to determine what an attacker can control is used to validate quality of software. It requires knowing where information enters the program and how it moves through the program. Data flow is the key to identifying many input validation and representation defects [11]. In this paper, we observe Android malware's behavior and use the concept of data flow of source code to build attack pattern. The proposed system uses these pattern to check if the suspicious apps are malicious or not.

II. Related Work

In this section, prior works about detection of malware of mobile apps are reviewed and discussion.

Kim et al. proposed a power-aware malware detection framework to monitor, detect, and analyze energy-greedy threats. Kim's system first collected power samples and builds a power consumption history with the collected samples, and the latter generates a power signature from the power consumption history. The data analyzer then detects an anomaly by comparing the generated power signature with those in a database [5]. Chekina et al. proposed a system to detect abnormal network traffic in mobile device for malware detection [6]. Shabtai et al. proposed a system which use knowledge-based, temporal abstraction method to detect previously unknown malware. In the proposed approach, time-stamped security data is continuously monitored within the target mobile device (i.e., smartphones, PDAs) and then processed by the knowledge-based temporal abstraction (KBTA) methodology. K-means, logical regression and histograms are used as training tools [7]. These researches used dynamic analysis approach for malware detection. Dynamic analysis does not suitable for mobile device because most mobile devices just have limited power capability. If author of malware use anti-VM technology, the abnormal could not be triggered. In addition, the purpose of attacker is financial gain. Malware just cause small traffic and resource (send credit card number or premium-rate SMS messages), it is difficult for traditional dynamic analysis to detect state-of-art mobile malware. Therefore, it seems that static analysis is a better approach for mobile malware detection.

Apvrille1 and Strazzere present a heuristics engine which used 39 different flags of different nature such as Java API calls, presence of embedded executables, code size, URLs. Each flag is assigned a different weight, based on statistics we computed from the techniques mobile malware authors most commonly used in their code. The engine outputs a risk score which highlights samples which are the most likely to be malicious [8]. However, the proposed system had a bad detection rate. Grace et al. proposed a system called RiskRanker to spot zero-day Android malware by sifting through the large number of untrusted apps in existing Android markets. The proposed system used vulnerabilities in the OS kernel to detect high-risk apps and used some privilege-related API like permission-group. COST_MONEY as features to detect mid-risk apps [9]. Wu et al. proposed a system called DroidMat, it's a feature-based mechanism to provide a static analyst paradigm for detecting the Android malware. The mechanism considers the static information including permissions, deployment of components, Intent messages passing and API calls for characterizing the Android applications behavior [10]. Prior researches used API or other feature to classify Android which could just identify if an app is malicious or not. These works could not tell you why the app is belong to malicious. In this paper, we use concept of dataflow to detect Android malware, by means of data flow based approach, we could examine the dataflow of Android app to check if the app is malicious or not.

III. Proposed Approach

To detect malicious mobile apps, the proposed system consists of three components. Figure 2 illustrates the process of proposed system. The proposed system first use reservse engineering technology to generate source code from suspicious APK files. Structure mapping compoent then builds structure tree of class and dependency of variables in APK file. Finally we use concept of data flow to build several threat patterns, and use them to detect mobile malware. The details of these component will be described in following.

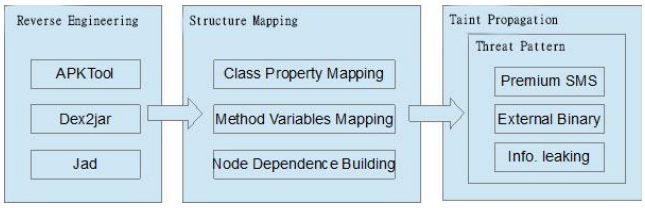

Figure 2: System Process

Figure 3 illustrates how reservse engineering works. In the first step, all APK files will be unpacked using APKTool. In step 2, we use dex2jar tool to transfer .dex file into .jar file. After uncompress .jar file, we could get several .class files. After decompiling process, several java source codes will be re-built in step 3.

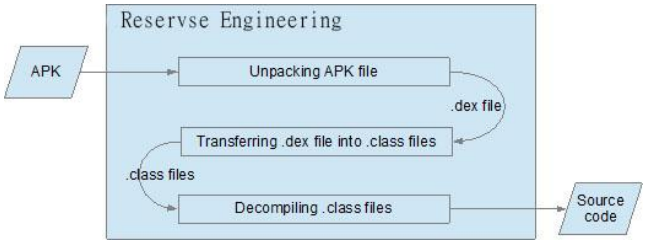

Figure 3: Reservse Engineering Component

The second component is Structure mapping compoent. The purpose of this component is to build the relationship for each class, attribute, method and variable from suspicious source code. Figure 4 illustrates relationships among nodes from a APK file. In figure 4, we could know each APK node consists of several class, and one class consists of several class methods and attributes. Each method contains at least one parameters and varibales.

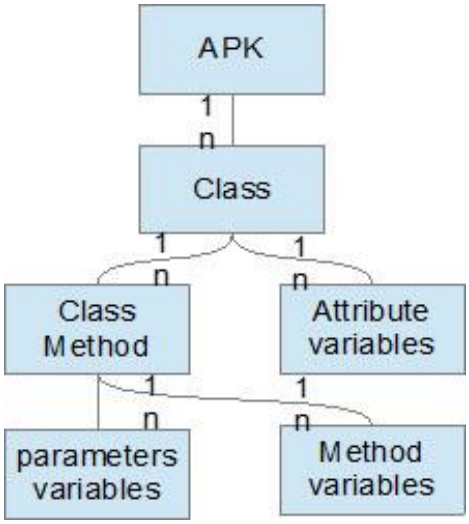

Figure 4: Relationships among nodes

In the proposed system, three type of nodes are defined, they are ClassNode, MethodNode and VariableNode.

ClassNode is used describe information of single java class. For each ClassNode we store the following information:

- className: Name of this class.
- classFilePath: The path of this class in jad file.
- attributesList: All attributes in this class, every elements in this list belong to VariableNode.
- methodsList: All method in this class, every elements in this list belong to MethodNode.

MethodNode describes information of one method within a class. Every MethodNode contains the following information.

- methodName: Name of this method.
- rawCode: Source code of this method.
- parametersList: All parameters in this method, every elements in this list belong to VariableNode.
- resultsList: Every variables that may be returned.
- variablesList: All variables in this method, every elements in this list belong to VariableNode
- parentNode: The parent node of this method.

VariableNode is the description of single variable, it contains:

- variableName: Name of this variable.
- type: Data type of this variable.
- taintTagList: TaintTags which this variable receives.
- ancestorList and descendantList: Dependence relation of this variable. Every elements in this list belong to VariableNode.
- groundApiList: It store API from android framework or constant variable.
- parentNode: Parent node of this variable, it might be a MethodNode or ClassNode.

The first step of structure mapping compoent is Class Property Mapping(CPM). Figure 5 shows the process of CPM.

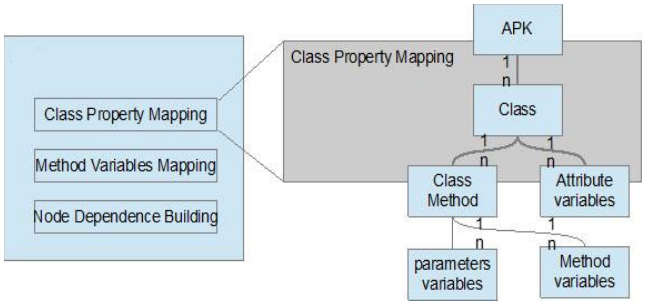

Figure 5: Process of CPM

The task of CPM is to build ClassNode, MethodNode and VariableNode from an APK file. The proposed system first examines all source codes to build ClassNode. We develop a parser to handle inner class and local class. When examining source code, if keyword “interface” is found in a class, we ignore the file because an interface is a group of related methods with empty bodies. After all ClassNode are defined, the elements of MethodNode and VariableNode could be extracted from ClassNode.

The second step of structure mapping component is Method Variables Mapping(MVM). Figure 6 shows the process of MVM.

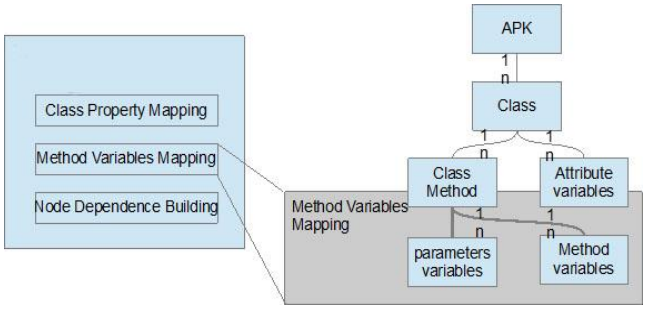

Figure 5: Process of MVM

The purpose of this step is to define VariableNode from each MethodNode. The component first check every variables in rawCode from MethodNode. The found VariableNodes will be added in variablesList of MethodNode.

Once all VariableNodes are found, the third step, Node Dependence Building(NDB), is used to re-build dependent relationships among all ClassNode, MethodNode and VariableNode. Figure 6 shows the process of NDB. In this step, the proposed system re-scan all fragment in rawCode attribute to build dependent relationships.

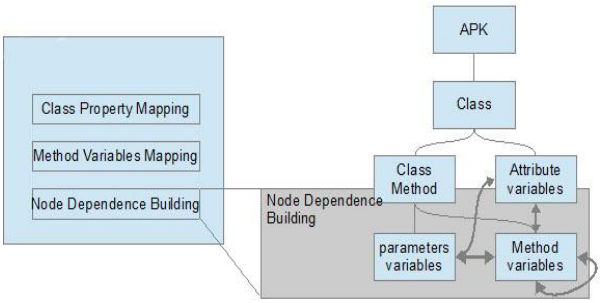

Figure 6: Process of NDB

Up to now, all nodes, and relationships in APK file are defined. Data flow is used to track the data flow to detect and identify malicious behavior of malware in Android system. Original usage of data flow thinks that the threat come is outside the software. But in malicious mobile apps, the threat is inside the software. Figure 7 illustrates the difference between original data flow based approach and data flow approach in our system.

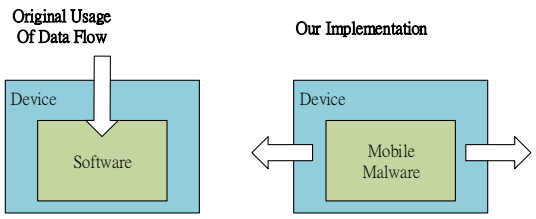

Figure 7: Difference between the proposed system and original data flow.

There are three data flow rules used in the proposed system, they are [11]:

- Source rules: Define program locations where tainted data enter the system.
- Sink rules: Define program locations that should not receive tainted data. For example, SQL injection in Java, Statement.executeQuery() is a sink. In mobile apps, sendTextMessage, is a sink.
- Pass-through rules define the way a function manipulates tainted data. For example, a pass-through rule for the java.lang.String method trim() might explain "if a String s is tainted, the return value from calling s.trim() is similarly tainted.

In this paper, we define eight different patterns. We refer to Spreitzenbarth [13] and Apvrille's [8] researches which summarize the behavior of malware families. After we examine a great number of malicious apps. Eight different threat patterns are defined and illustrated in table 1.

Table 1: Threat Patterns

Threat type	Source	Sink
Send premium-rate SMS messages(String)	String constant	sendTextMessage() sendMultipartTextMessage()
Send premium-rate SMS messages(int)	int constant toString()	
Send premium-rate SMS messages(URL)	HttpClient.execute() HttpResponse.getEntity() toString()	
Steal hardware or software information	TelephonyManager.getDeviceId() TelephonyManager.getSimSerialNumber() TelephonyManager.getDeviceSoftwareVersion() TelephonyManager.getLine1Number() TelephonyManager.getSubscriberId() Build.*	HttpClient.execute() URL.openStream() URL.openConnection() URL.getContent()
Steal information-call record	getContentResolver() "CallLog.*"	
Steal information-contact list	getContentResolver() ContactsContract.*	
Steal information-SMS message	getContentResolver()	

	"content://sms/" or "content://mms-sms/" ContentResolver.query()	
Execute external program	AssetManager.getAssets() AssetManager.open() FileOutputStream.write()	Runtime.exec()

IV. System Validation

To validate the performance of the proposed system, two experiments are conducted. In the first experiment, we use the same samples with Zhou et al.'s research [12]. There are 252 samples with 19 families are used in this experiment. Table 2 illustrates the profile of samples used in this experiment.

Table 2: Profile of Malicious Samples

Family	Samples
ADRD	22
Asroot	8
BeanBot	8
Bgserv	9
CoinPirate	1
DogWars	1
FakePlayer	6
Geinimi	65
GingerMaster	4
GPSSMSSpy	6
HippoSMS	4
Jifake	1
LoveTrap	1
NickiSpy	3
Pjapps	56
RogueSPPush	9
SndApps	10
YZHC	22
Zsone	12

Due to space limitation, we just describe some popular families in this paper.

- ADRD: It steal information from compromised device, encrypt these information and then send out to attackers.
- Asroot: The malicious binary files are contained in this APK file. After installation, the malicious binary files will be executed and the device will be compromised.
- Geinim: Create a back door on compromised device, then send out information to remote server via URL.
- Pjapp: It was spread in 3rd party market. Once a device is compromised, it will retrieval attack command from remote server. Compromised device could also send SMS message to specific number.
- YZHC: It will steal information form compromised device. It could also send premium-rate SMS messages. This malicious had ever found in Google play.

Table 3 show the detection rate for the sample. In our system, we have 91.6 detection rate. The result shows that our system could detect mobile malware effectively.

Table 3: Result of experimrnt1

Family	Samples	Detected Samples	Detection Rate
ADRD	22	10	45.5%
Asroot	8	7	87.5%
BeanBot	8	8	100%
Bgserv	9	9	100%
CoinPirate	1	1	100%
DogWars	1	1	100%
FakePlayer	6	6	100%
Geinimi	65	65	100%
GingerMaster	4	4	100%
GPSSMSSpy	6	6	100%
HippoSMS	4	4	100%
Jifake	1	1	100%
LoveTrap	1	0	0%
NickiSpy	3	2	66%
Pjapps	56	51	91%
RogueSPPush	9	8	88%
SndApps	10	10	100%
YZHC	22	22	100%
Zsone	12	12	100%
Overall	252	231	91.6%

V. Conclusion and Future Work

As mobile device become powerful, most smartphone could handle complex task and be capable of Internet connection. To install application on Android system, user must download apps form google play or 3rd markets. If authors of malware spread malicious app on market, a great number of mobile device will be compromised.

In this paper, according to behavior of malware, we define eight threat patterns. A malware detection system is also developed which uses static analysis approach and concept of data flow. The experiment results show that the

proposed system could not only detect mobile malware but also could identify advertising software with potential risk.

However, in this paper, the threat patterns are created manually. If new type of attack approaches are developed which have a different threat pattern, our system will not detect them. In the future, an auto threat pattern generation should be developed to detect zero day attacks.

REFERENCES

[1] Appbrain, 2013, Number of Android applications, http://www.appbrain.com/stats/number-of-android-apps

[2] IDC, 2013, Apple Cedes Market Share in Smartphone Operating System Market as Android Surges and Windows Phone Gains, According to IDC, http://www.idc.com/getdoc.jsp?containerId=prUS24257413

[3] F-Secure, 2013, Mobile Threat Report, http://www.f-secure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q3_2013.pdf

[4] McAfee Lab , 2012, FakeInstaller' Leads the Attack on Android Phones, https://blogs.mcafee.com/mcafee-labs/fakeinstaller-leads-the-attack-on-android-phones

[5] H. Kim, J. Smith, K. G. Shin, "Detecting Energy-Greedy Anomalies and Mobile Malware Variants" in Proceedings of the 6th international conference on Mobile systems, 2008, pp.239-252

.

[6] L. Chekina, D. Mimran, L. Rokach, Y. Elovici, B. Shapira, "Detection of Deviations in Mobile Applications Network Behavior", CoRR abs/1208.0564, 2012

[7] A. Shabtai, U. Kanonov, Y. Elovici, "Intrusion detection for mobile devices using the knowledge-based, temporal abstraction method", Journal of Systems and Software, Vol. 83, No. 8, 2010, pp. 1524-1537

[8] A. Apvrille, T. Strazzere, "Reducing the window of opportunity for Android malware Gotta catch 'em all", Journal in Computer Virology, Vol.8, No. 1-2, 2012, pp.61-71

[9] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection", The 10th international conference on Mobile systems, applications, and services, 2012, pp.281-294

[10] D.J. Wu, C.H. Mao, T.E Wei, H.M. Lee, K.P. Wu, "DroidMat: Android Malware Detection through Manifest and API Calls Tracing", in Proceedings of 7th Asia Joint Conference on Information Security, 2012. Pp.62-69

[11] B. Chess, J. West, Secure Programming with Static Analysis, 1st ed., Addison-Wesley Professional (June 29, 2007)

[12] Y. Zhou, X. Jiang, "Dissecting Android Malware: Characterization and Evolution", in Proceedings of 33rd IEEE Symposium on Security and Privacy, 2012, pp.95-109

[13] M. Spreitzenbarth, "forensic blog, mobile phone forensics and mobile malware".

[14] http://forensics.spreitzenbarth.de/android-malware/