Mobile Malware Detection in Sandbox with Live Event Feeding and Log Pattern Analysis

Wei-Ting Lin and Jen-Yi Pan
Department of Communications Engineering
National Chung Cheng University
Chia-Yi, Taiwan, R.O.C.
h30299@gmail.com, jypan@comm.ccu.edu.tw

***Abstract*—In recent years, the use of smart devices is becoming increasingly popular. All kinds of mobile applications are emerging. In addition to the official market, there are also many ways to allow users to download the mobile app. As unidentified instances of malware grow day by day, off-the-shelf malware detection methods identify malicious programs mainly with extracted signatures of codes, which only can effectively identify already known malwares, but not new malwares in initial spread. If no samples of these malwares are reported and the virus code library is not patched, users won't be alerted to the malwares. Therefore, this paper proposed a new detection method by live log analysis. A sandbox is conducted to mimic human operations and monitor responses from APPs. Feeding these manual events can excite deactivated malwares and improve the accuracy of log analysis, even though these malware are unknown yet. This study takes recent malwares and benign programs to conduct experiments, and then verifies the effectiveness of the proposed method comparing with those in other papers. The experimental results show that the proposed method outperforms in both hit rate and pass rate.**

Keywords—Sandbox; Malware Detection; Data Mining; live event feeding

I. INTRODUCTION

Recently, the use of smart devices is becoming increasingly popular. All kinds of mobile applications are emerging. Apps can be downloaded from the official market and third-party stores. Therefore, programs that have not been officially certified and even malwares are also increasing. According to Gartner [1], smart device sales reached 1.9 billion units in 2014. Gartner predicts that from 2015 to 2017, annual sales of smart devices will continue to grow. The apps not only can substantially be downloaded from the Internet and installed on smart devices, but also much tightly integrated with the cloud service via a ubiquitous network. With fast development and deployment of 4G network, transmission data rate and network coverage of each operators rapidly increase, which enable smart devices more diversified services, thus significantly changing users' experience on these devices and also the way people work, relax and live.

The market of smart devices rapidly grows, resulting in large and unknown potential security risk. According to IDC survey results [2], Android system already has a market share of 82.8% in the second quarter of 2015. iOS system takes down the second place with a market share of only 13.9%. The results of the survey shows that currently more than eighty percent of smart device users worldwide are using Android. August 2015 McAfee Labs Threats Report [3] points out that there is always a security threat in smart devices. The numbers of mobile malware samples in both the first and second quarters of 2015 increase by nearly 500,000 compared to the prior quarters'.

Existing malware detection techniques identify malicious programs by known code patterns [4]. Hence they can not identify and prevent malwares which have never been reported. For mobile devices, malware scanning consumes a lot of resource, such as computation, memory and power. Therefore, we need an economic mechanism to detect malwares on mobile devices. Furthermore, the mechanism shall quickly identify both known and unknown malware while initially spreading.

Therefore, we propose a method that dynamically analyzes behavior patterns in a sandbox. The sandbox is an isolated and virtualized environment where untrusted applications execute under supervision. Besides, the environment also can mimic users' operations on mobile phones in our plan, such as making phone calls, sending short message and so on. We analyze mobile device log and then compare performance with other studies'. The proposed method is better than traditional ones of static and dynamic analysis on the aspects of hit rate and pass rate.

II. RELATED WORK

In recent years, many studies on smart portable device carry out static analysis and dynamic analysis. The comparison between Android and iOS systems are described in the literatures [5][6][7]. The result shows that Android system is higher risky. Reference [8] mentioned that malwares have a very high proportion based on the Android system. Reference [9] depicts that an attacker can gain a deeper understanding on Android platform because of open sourcing. Furthermore, Google opens market to third parties that are unofficial and independent from Google Play Store. This makes the apps to be released more easily, thus causing users very easy to expose to the virus or malicious apps.

The numbers of mobile malware samples in both the first and second quarters of 2015 increase by nearly 1 million compared to the prior quarters according to the report by McAfee Labs Threats in August 2015. This shows the security issue on the smart device can't be ignored.

A. Static Analysis

The static analysis is to check permission requirement and look for code signature before installing or invoking suspicious applications on mobiles. This approach has the advantage of high accuracy and fast speed. In the case of code signature, static analysis first requires searching for codes that have malicious acts, and then builds a database based on the collected signatures. Once applications executes, it detects malware by matching the executed program with those signatures.

While the Android API keeps opening and increasing, malware developers continuously modify their programs to avoid antivirus scanning. Static analysis would be difficult to scan for malicious means of code content. Most of traditional anti-virus programs are of static analysis, and their discernment depends on the level of latency that database updating follow up initial spread of new malwares. These static analysis approaches might not accurately analyze the malwares that are newly born and whose signatures are not in the database.

Static analysis also includes checking required permissions. Aggressive checking prevents possible malwares in advance but may result users' confusion. On the other hand, mild checking could miss potential malwares.

Liang et al.[10] mentioned the static analysis based on permission[11] combination. They proposed the K-map (K-map is the set where they keep permission combinations of those sampled malwares generating), which is derived from the 1260 malwares and 741 benign programs. They categorize several families of malwares, and obtain sets of permission rules for each family, which can be used to recognize malwares and their family belonged to. Reference [10] uses permissions for analysis, even though it can get very high accuracy. However, as the smart devices' operating system gives more API and functions, apps will require more permission. Malware's permission will easily overlap with benign apps'. The accuracy of this analysis method decreases and false positives rate increases, even worse if mobile apps are injected malicious codes and repackaged again, because permissions of the injected code may overlap with the original apps, resulting in troublesome of differentiating permission of the malicious from the normal.

B. Dynamic Analysis

Dynamic analysis methods judge whether programs are malicious by observing their behavior in isolated machines. Dynamic analysis is roughly classified into two types: "network traffic analysis" and "sequential event analysis". Packet analysis refers to analyze sequences and contents of packets that have passed through supervised networks. Although the packet analysis on network traffic is faster than the event analysis, those actions without the internet will not be effectively analyzed. Besides, many malwares do harmful attacks that are only elicited by human activity. Therefore, analysis on only logged acts is not a thorough solution.

Arora et al.[12] used the method "Network traffic analysis", which divides traffic into a number of traffic features and captures packets according to certain packet signature. Zhao et al.[13] mainly based on generated events and the conditions of how system resources are being used during execution of apps (for example: GPS, SMS, players, etc.). Then the data is recorded and analyzed. Patent [14] proposed a black box detection method of recording various detection operations in a standard mobile terminal detection model machine and comparing these operation records with user service report records provided by a mobile operator. If they are conformed, the result means that the application does not contain malware; on the contrary, if not matched, it means that the application does contain malware.

Patent [15] proposed a static analysis method of scanning uniform resource locators (URL) in apps. It compares the URLs in source codes to those malicious URLs in a well-maintained database, to determine whether the apps are malicious. Patent [16] integrates two parts of static and dynamic analysis. The static analysis part traditionally checks code signatures and permissions in AndroidManifest.xml, and the dynamic analysis part decompiles the application package and adds a monitor code in the source file. After repacking, the app is installed in the sandbox and observed. The recorded events during observation are then further analyzed.

In summary, dynamic analysis has many kinds, such as traffic analysis and sequential event analysis. Traffic analysis methods resolve packet contents to determine whether the apps are malicious. If the content of packets is encrypted or not transmitted through the supervised network, the effectiveness of the analysis is limited. Event analysis installs the app in the virtual machine, and analyzes the generated event. Nevertheless, human behavior, such as turn on/off screens, may activate harmful behavior of malware [17]. To our best knowledge, there is no malware analysis framework that can mimic human activities to trigger malicious actions from malware apps. Hence these frameworks off the shelf might skip these event-triggered malwares. We propose to emulate human behavior on the sandbox by feeding in interaction events of user interface, resulting in that malicious applications highly tend to act.

III. Proposed Method

We proposed a method which analyzes malware's acts in SandBox and DroidBox [18]. The following subsections give an overview of proposed method and detail categorization of acts' signature.

A. System Architecture

The system architecture is as shown in Fig. 1. Firstly, APPs are to be installed in the Android SandBox. Afterwards, the DroidBox will be recorded logged act to the database from the logger. At the same time, the Human Event Emulator will be through Console communication with Android SandBox. Then, Action Counter counts each app's act sequence by Algorithm

1. The next step will be Act Combination for each APP and insert to malware or benign act sets database. When database generated, the malware or benign act sets count to Act Combination Count Table by Algorithm 2. The last, Act Combination Count Table sort and select the top rule to Rule Set. Rule Set is M-map (M-map as a storage of Hash produced by combination of behavior signatures) which will be use it to classify unknown APPs.

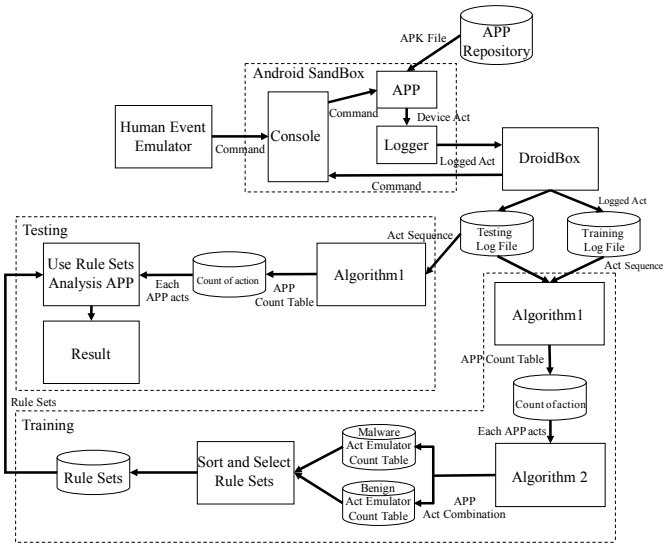


Fig. 1. System Architecture

TABLE I. SELECTION OF ACT

Signature	Description
send SMS	Message sending
read SMS	Message receiving
File access	Operations involving files
recvnet	Receive via network
sendnet	Transfer operations via network
open net	Socket open operations
IMEI read	Read IMEI
IMEI send	Send IMEI using network
IMSI read	Read IMSI
IMSI send	Send IMSI using network
read contact	Read contact
send contact	Send contact using network
read phone number	Read phone number
send phone number	Send phone number using network
Read ICCID	Read Integrate circuit card identity
Phone call	Phone call

Table I is the description of our act selection. We use it to define behavior signatures based on log file contents produced by DroidBox. We define 16 behavior signatures such as send SMS, File access, recvnet, sendnet, open net and Phone call. These behavior signatures stated above are built-in within DroidBox.

In order to facilitate follow-up M-map experiments, we merge 11 feature acts, which may leak private data, to one aggregated feature act called "dataleaks". These acts are "read SMS", "IMEI read", "IMEI send", "IMSI read", "IMSI send", "read Contacts", "send Contacts", "read phone number", "send phone number", "Read ICCID", and "Phone call".

Traditional dynamic analysis is to install the target app in a sandbox and observe the produced acts. The analysis process does not feed in any events through the box. In order to observe the added human actions which affect the outcome of analysis, we implement both the traditional dynamic analysis and event-feeding analysis. In fact, there are really some malicious behaviors waiting for the specific events by human event-triggered [17][20]. Therefore, we propose to add human behavior emulation in the process.

Execution time of a script is 6 hours long. In the first hour, there is no action fed in. In the second hour, the script will make phone calls, send messages, receive SMS, receive calls, and lock and unlock the screen. There are six actions in such a cycle, each action last for 1 minute, separated by 10-second idle intervals. After those operations above have been completed, the script stops for 90 minutes and repeats again and again until the end of four hours, as shown in Fig. 2.

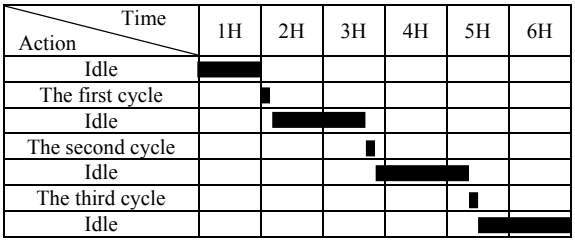


Fig. 2. Gantt chart of script execution

Following the above experiments, emulated human behavior is added. The program that is going to be analyzed possibly increases the number of feature acts generated. For examples in Table II, when the malware sample "Android SMS trojan Flash fake installer" executes standalone, the feature act numbers of "File Access", "recvnet", "sendnet", "open net", and "dataleaks", are 1433, 1, 2, 1, and 1, respectively. After feeding in emulated human events, the feature act numbers become 1649, 7, 10, 7, and 8, respectively. In the other hand, when the benign program "Facebook" executes standalone, the feature act numbers of "File Access" and "dataleaks" are 413 and 1, respectively. After feeding in emulated human events, the numbers become 435 times and 1, respectively. These values show that feeding emulated events effectively raises the degree of malicious activity of malware, but won't affect performance of benign programs. Feeding events thus can help us more accurately determine a malware.

TABLE II. COMPARISON ON THE NUMBER OF BEHAVIOR

APP / Signature	Flash fake installer	Flash fake installer (with live event)	Facebook	Facebook (with live event)
File access	1443	1649	413	435
Recvnet	1	7	0	0
Sendnet	2	10	0	0
Open net	1	7	0	0
Dataleaks	1	8	1	1

B. *Signature Categorization*

TABLE III. NOTATIONS DEFINITIONS

Notation	Description
P	Set of APP
p_i	i-th APP in P
A	Set of probable action
a_i	Action i
S_i	Action sequence of APPi
$S_{i,j}$	j-th action in S_i
$C_{i,j}$	Count of action j in APP i, j∈A, i∈P
Q	A set of action
D_i	Count of action combination i, i∈2^A
2^A	Power set of A

Algorithm 1. Counting of DroidBox generated log

```
Input: Given S_i, i∈P
Output: C_i,j, i∈P, j∈A
for each APP i∈P do
    for each action j in S_i do
        C_i,j++
    end for
end for
```

It has been observed that feature acts that produced by malwares are usually more than one, so this paper proposes to classify those acts by using M-map. We use a combination of features to combine the feature acts produced by single malware. We conduct the analysis with the preceding script and malware features. Then we use M-map to classify (M≥1). M-map automatically generates a rule set in an iterative manner (M = 1, 2, 3, ...) such as Algorithm 1 and Algorithm 2. Symbols used are as shown in Table III. The generated rule set can classify possible malwares.

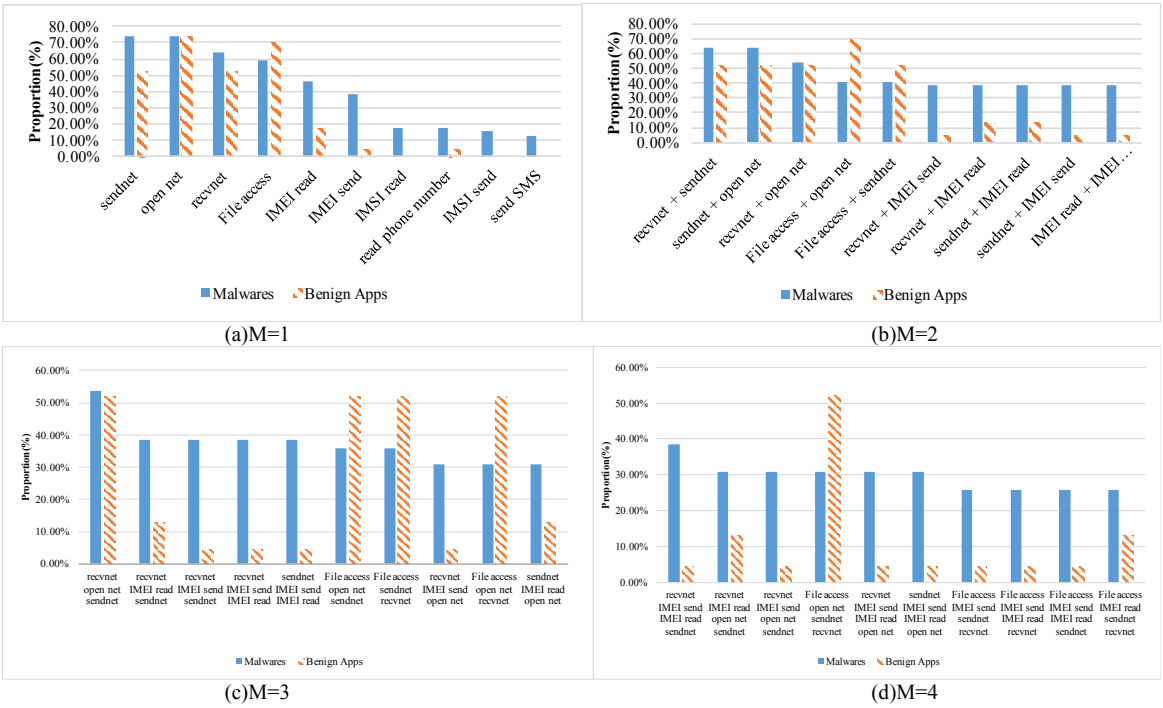
Algorithm 2. Counting all of act combinations and sorting

```
Input: {C_i,j| i∈P, j∈A}
for each element i, i∈(2^A − {{}}) do
    for each APP j ∈ P do
        Q={a | a∈A^C_j,a > 0}
        if i⊆Q then
            D_i++
        end if
    end for
end for

E is sorted sequence of each element i∈(2^A − {{}}) according to
descending order of value D_i
M-map = Sequence of top 10 from selected elements of E, where
each selected element i satisfies |i|=M
```

Algorithm 1 counts DroidBox-generated logged acts for each apps. Algorithm 2 counts act combinations for the set of apps. It also sorts and derives top 10 size-M act combinations to produce M-map. Both malwares and benign programs use Algorithm 1 and 2 to produce M-map. Fig. 3 displays the top ten feature act generated from malwares in the condition of M = 1, 2, 3 and 4. Then we also add act combination result of benign program for comparison.

When M=1, malwares frequently generate “sendnet”, “open net”, “recvnet” and “File access”. Benign programs perform in a similar way, but with a higher proportion of File access. These four features are common behaviors for all applications.



(a)M=1 (b)M=2

(c)M=3 (d)M=4

Fig. 3. Top 10 behavior combinations

When M=2, we find that more than half of behavior combinations from malwares have much higher proportion than those from benign programs, such as "recvnet & IMEI sent", "recvnet & IMEI read", "sendnet & IMEI read", "sendnet & IMEI send" and "IMEI read & IMEI send". Along with the network sending, receiving, and IMEI (International Mobile Equipment Identity) reading, the malware can read the identity of the phone and send it with Internet. Such a behavior can identify each user and further collect personal information.

When the value of M is bigger, the more accurate the difference of proportions can identify malwares. However, the M value is limited to the combination size of benign programs. Malwares usually require more functionality than benign programs do. This also represents that an app contains more types of acts resulting in higher possibility/capability that smart devices are hacked or private data are stolen. Thus this study conducts an experiment with M = 1,2,3,4,5,6 and we discuss which actions often occur in malicious or benign programs.

IV. Experimental Evaluation

A. Analyzed Samples

The sources of experimental data come from the malware research site "contagio mobile" [19]. This site provides malware samples. We analyzed malware samples generated during the period between 2013/01 and 2015/06. The benign application samples are collected in Google Play Store in 2016/01 with more than 100 thousand download times. Above benign application are all published by well-known companies like Facebook. The analysis uses 38 malware samples and 22 samples of benign program collected for analysis.

B. Experiment and Result

Based on M-map generated above, we define the percentage of malware samples detected by the rules as Hit Rate (HR). The percentage of benign samples detected as Pass Rate (PR). False Negative Rate (FNR) is the percentage of undetected malware. False Positive Rate (FPR) is defined as the percentage of benign program of detected malwares. We can organize the results and obtained Table IV. Malware's HR decreases as M increases, but raises the PR of benign program.

TABLE IV. Live Event Feeding Results

M	HR(%)	PR(%)	FNR(1-HR) (%)	FPR(1-PR)(%)
1	97.36	18.19	2.64	81.81
2	78.94	22.73	21.06	77.27
3	65.78	40.91	34.22	59.09
4	47.36	40.91	52.64	59.09
5	36.84	90.91	63.16	9.09
6	26.31	100	73.69	0

In Table IV, malware detection rate (HR) was 97.36% when M=1. When M increases, malware detection rate decreases and the missing rate of benign program (FPR) also decreases. Meanwhile, we observe that the number of actions which malicious and benign programs perform is very different, and we add some number-related features to observe whether the malware detection rates can be enhanced while reducing the false positives rate for benign programs. In this regard, the features of relational operators on actions' counter are added to the experiment. We add "sendnet<=3 & recvnet<=24"(named J481) and "opennet>0 & File access<=27"(named J482) as virtual signatures. Then, the M-map is regenerated afterwards.

Based on Fig. 4 and Fig. 5, we discovered that there are significant differences with the malware and benign programs compared to (c) and (d) in Fig. 3. Based on the M-map tested in this paper, the act combination of the top three can cover most of the malwares and benign programs. Therefore, this experiment just takes the top three combinations of features for analysis. The analyzed results are shown in Table V. It can be learned from the results, when the M = 4, it has the largest malware detection rate, and the lowest false positive rate of benign programs.

TABLE V. Live Event Feeding with Virtual Signature Result

M	HR(%)	PR(%)	FNR(1-HR) (%)	FPR(1-PR)(%)
1	73.68	31.82	26.32	68.18
2	73.68	50	26.32	50
3	68.42	50	31.58	50
4	73.68	100	26.32	0
5	44.73	100	55.27	0
6	23.68	100	76.32	0

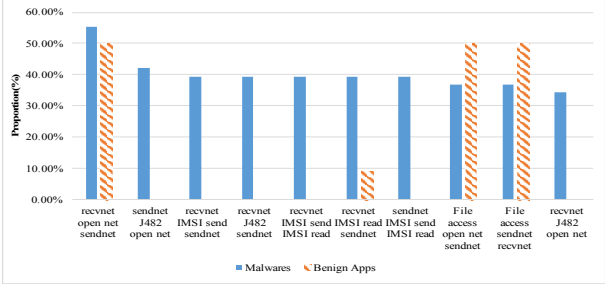


Fig. 4. Regeneration of M-map after adding signature when M=3.

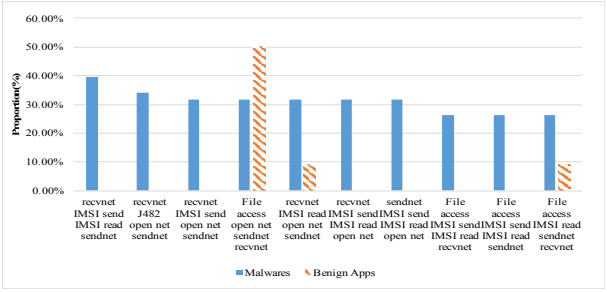


Fig. 5. Regeneration of M-map after adding signature when M=4.

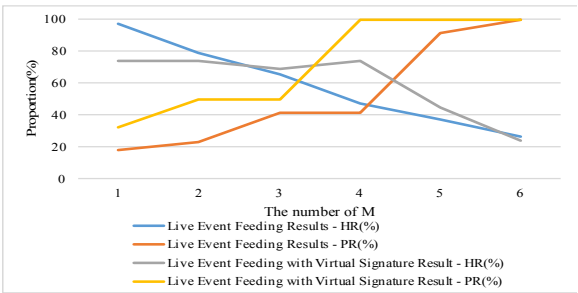


Fig. 6. Live event feeding with and without virtual signature.

Some clues can be learned from the Fig. 6. We compare between two experiments. In the second experiment, we combine few behavior counter relations to a new signature. When M=4, HR and PR are better than of the original.

C. Comparison with Other Method

In order to verify the accuracy and reasonableness of our program, we compare results with reference [10], which assesses the combination of Android APK permissions. Then we implement the method in [10] and feed in our collected samples. The reference described why the K=6 has best analyzed Hit Rate. Analyzed results are in Table VI. Hit Rate is 34.21% and Pass Rate is 95.46%. They are lower than ours.

We investigate the results from reference [10] and deduce why their Hit Rate and Pass rate is lower than the proposed. This is because malware samples declare those unused permissions in the file Manifest.xml, even they don't need them. Our method checks only those used functions at run time in finer granularity. Besides, many benign applications (e.g. Facebook) have more and more functions as well as permissions. Reference [10] thus increases the false positive rate of permission combination used by the benign programs.

The permissions analysis of k-map [10] achieves pretty good performance in malware detection when K=4. However the permission of APP relatively increases as the API and functions of smart devices' operating system are getting more and more. If an application has been injected malicious code and repackaged, its permission is easily overlapped with the original. This permission analysis method then can not effectively apply on malware detection.

TABLE VI. COMPARISON WITH OUR METHOD

	HR(%)	PR(%)	FNR(1-HR) (%)	FPR(1-PR)(%)
K=6	34.21	95.46	65.79	4.54
M=4	73.68	100	26.32	0

V. CONCLUSION

In this paper, we propose a method for malware detection. In the past, if there is no malware's sample signature in the malware database, then malware detection can not quickly find new malwares. Although the official markets like Google Play and Apple APP Store check their apps, technologies may upgrade and vulnerabilities happen at any time, leading to new malwares.

To avoid new malwares which are given birth by new technologies and new vulnerabilities, we propose a method of analyzing malware behavior with higher accuracy because our method considers apps' behavior at run time in finer granularity. Furthermore, the study proposed emulating human operations, the idea of which has not yet been proposed, to activate malwares and hence increase the accuracy of analysis.

ACKNOWLEDGMENT

This work was supported in part by Ministry of Science and Technology, Taiwan, Republic of China, under grant MOST 105-2221-E-194-011.

REFERENCES

[1] "Gartner Says Worldwide Device Shipments to Grow 1.5 Percent, to Reach 2.5 Billion Units in 2015", Gartner.com, 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3088221. [Accessed: 29- Mar- 2016]

[2] "IDC: Smartphone OS Market Share", www.idc.com, 2016. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed: 29- Mar- 2016]

[3] McAfee Labs Threats Report August 2015 http://www.mcafee.com/us/resources/reports/rp-quarterly-threats-aug-2015.pdf

[4] C.-M. Chen, J.-M. Lin, and G.-H. Lai, "Detecting Mobile Application Malicious Behaviors Based on Data Flow of Source Code," in *Trustworthy Systems and their Applications (TSA), 2014 International Conference on*, 2014, pp. 1-6.

[5] F. Al-Qershi, M. Al-Qurishi, S. Md Mizanur Rahman, and A. Al-Amri, "Android vs. iOS: The security battle," in *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*, 2014, pp. 1-8.

[6] I. Mohamed and D. Patel, "Android vs iOS Security: A Comparative Study," in *Information Technology-New Generations (ITNG), 2015 12th International Conference on*, 2015, pp. 725-730.

[7] M. S. Ahmad, N. E. Musa, R. Nadarajah, R. Hassan, and N. E. Othman, "Comparison between android and iOS Operating System in terms of security," in *Information Technology in Asia (CITA), 2013 8th International Conference on*, 2013, pp. 1-4.

[8] Mobile Threat Report Q3 2012 https://www.f-secure.com/documents/996508/1030743/Mobile+Threat+Report+Q3+2012.pdf

[9] D. He, S. Chan, and M. Guizani, "Mobile application security: malware threats and defenses," *Wireless Communications, IEEE,* vol. 22, pp. 138-144, 2015.

[10] S. Liang and X. Du, "Permission-combination-based scheme for android mobile malware detection," in *Communications (ICC), 2014 IEEE International Conference on*, 2014, pp. 2301-2306.

[11] Developer.android.com, " <permission> | Android Developers", 2015. [Online]. Available: http://developer.android.com/intl/zh-tw/guide/topics/manifest/permission-element.html. [Accessed: 21- Dec- 2015].

[12] A. Arora, S. Garg, and S. K. Peddoju, "Malware Detection Using Network Traffic Analysis in Android Based Mobile Devices," in *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, 2014, pp. 66-71.

[13] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "AntiMalDroid: an efficient SVM-based malware detection framework for Android," in *Information Computing and Applications*, ed: Springer, 2011, pp. 158-166.

[14] HUANG YUHUI, "Black box detection method for mobile terminal malicious software behavior", China Patent, CN103369532A, 2013.

[15] C. Alme, "System and method for detecting malicious mobile program code", US20080263659A1, US Patent,2008.

[16] QIN ZHIGUANG, ZHAO YANG, WANG RUIJIN, LIU BINGHOU, HU LONG and GONG XIAOBO, "Android malicious software detecting platform oriented to mobile internet", China Patent, CN103685251A, 2014.

[17] T. Strazzere, "Update: Android Malware DroidDream: How it Works | Lookout Blog", Blog.lookout.com, 2011. [Online]. Available: https://blog.lookout.com/blog/2011/03/02/android-malware-droiddream-how-it-works/. [Accessed: 29- Mar- 2016]

[18] "pjlantz/droidbox", GitHub, 2015. [Online]. Available: https://github.com/pjlantz/droidbox. [Accessed: 06- Apr- 2016].

[19] "contagio mobile", Contagiominidump.blogspot.tw, 2016. [Online]. Available: http://contagiominidump.blogspot.tw/. [Accessed: 29- Mar- 2016].

[20] T. Bradley, "Experts Disagree on Android Call Recording 'Trojan'", PCWorld, 2016. [Online]. Available: http://www.pcworld.com/article/237166/experts_disagree_on_android_call_recording_trojan.html. [Accessed: 02- Apr- 2016].