TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/10 pp418-427
Volume 18, Number 4, August 2013

MobSafe: Cloud Computing Based Forensic Analysis for Massive Mobile Applications Using Data Mining

Jianlin Xu, Yifan Yu, Zhen Chen*, Bin Cao, Wenyu Dong, Yu Guo, and Junwei Cao

Abstract: With the explosive increase in mobile apps, more and more threats migrate from traditional PC client to mobile device. Compared with traditional Win+Intel alliance in PC, Android+ARM alliance dominates in Mobile Internet, the apps replace the PC client software as the major target of malicious usage. In this paper, to improve the security status of current mobile apps, we propose a methodology to evaluate mobile apps based on cloud computing platform and data mining. We also present a prototype system named MobSafe to identify the mobile app's virulence or benignancy. Compared with traditional method, such as permission pattern based method, MobSafe combines the dynamic and static analysis methods to comprehensively evaluate an Android app. In the implementation, we adopt Android Security Evaluation Framework (ASEF) and Static Android Analysis Framework (SAAF), the two representative dynamic and static analysis methods, to evaluate the Android apps and estimate the total time needed to evaluate all the apps stored in one mobile app market. Based on the real trace from a commercial mobile app market called AppChina, we can collect the statistics of the number of active Android apps, the average number apps installed in one Android device, and the expanding ratio of mobile apps. As mobile app market serves as the main line of defence against mobile malwares, our evaluation results show that it is practical to use cloud computing platform and data mining to verify all stored apps routinely to filter out malware apps from mobile app markets. As the future work, MobSafe can extensively use machine learning to conduct automotive forensic analysis of mobile apps based on the generated multifaceted data in this stage.

Key words: Android platform; mobile malware detection; cloud computing; forensic analysis; machine learning; redis key-value store; big data; hadoop distributed file system; data mining

• Jianlin Xu is with Department of Computer Science and Technology and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China. E-mail: xjl11@mails.tsinghua.edu.cn.
• Yifan Yu is with Department of Electronic Engineering and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China. E-mail:yuyf10@gmail.com.
• Zhen Chen and Junwei Cao are with Research Institute of Information Technology and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China. E-mail: {zhenchen, jcao}@tsinghua.edu.cn.
• Bin Cao, Wenyu Dong, and Yu Guo are with Department of Computer Science and Technology, Research Institute of Information Technology and Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China. E-mail: {caobni, dongwy13, guoyu90337}@163.com.
∗ To whom correspondence should be addressed.
Manuscript received: 2013-07-19; accepted: 2013-07-19

1 Introduction

1.1 Mobile threats

These years witness an explosive increase in mobile apps. According to Mary Meeker's report[1] on Mobile Internet trends, more and more PC client softwares are migrating to the mobile device[2, 3]. According to Gartner's statistical prediction[4], the amount of total downloads of mobile apps in 2013 will be about 81 billion. Among these, there are about 800 000 Android apps in Google Play market, and the total download is about 48 billion as of May 2013[5]. In contract with Apple AppStore, there are different sources for Android apps download, such as wandoujia, AppChina, Baidu mobile assistant, etc. While these markets give a good supply and bring more convenience for Android users, they will also bring mobile threats as different market places have different malware detection utilities and methods. Some sophisticated malwares can escape from detection and spread even via such Android markets.

1.2 Some root causes for Android malware origins

It needs some discussions about the malware's origins, provenances and spreading.

(1) Android platform allows users to install apps from the third-party marketplace that may make no efforts to verify the safety of the software that they distribute.

(2) Different market place has different defense utility and revocation policy for malware detection.

(3) It is easy to port an existing Windows-based botnet client to Android platform.

(4) Android application developers can upload their applications without any check of trustworthiness. The applications are self-signed by developers themselves without the intervention of any certification authority.

(5) A number of applications have been modified, and the malwares have been packed in and spread through unofficial repositories.

Some sophisticated malwares detect the presence of an emulated environment and adapt their behavior, e.g., create hidden background processes, scrub logs, and restart on reboot.

1.3 Some known malwares in Android platform

There are a lot of already discovered malwares which include: Drad.A, Fake Player, Geinimi, PJApps, HongToutou, DroidDream trojan, DroidKungFu, SteamyScr, Bgyoulu.A, Cabir, HippoSMS, Fake Netflix, Walk & Text, Dog Wars, DroidDreamLight, BaseBridge, Zsone, jSMSHider, Rageagainstthecage, Zimperlich, Exploid, Plankton, DougaLeaker.A, Rufraud, Gone in 60 s, etc.

1.4 Some malicious behaviors of Android malware

Malware is usually motived by controlling mobile device without user intervention, such as:

(1) Privilege escalation to root,

(2) Leak private data or exfiltrate sensitive data,

(3) Dial premium numbers,

(4) Botnet activity, and

(5) Backdoor triggered via SMS.

1.5 Our work

In this paper, based on home-brewed cloud computing platform and data mining, we propose a methodology to evaluate mobile apps for improving current security status of mobile apps, MobSafe, a demo and prototype system, is also proposed to identify the mobile app's virulence or benignancy. MobSafe combines the dynamic and static analysis methods to comprehensively evaluate an Android app, and reduce the total analyse time to an acceptable level. In the implementation, we adopt the two representative dynamic and static analysis methods, i.e. Android Security Evaluation Framework (ASEF) and Static Android Analysis Framework (SAAF) to evaluate the Android apps and estimate the total time needed to evaluate all the apps stored in one mobile app market, which provide useful reference for a mobile app market owner to filter out the mobile malwares.

2 Related Work

Security analysis of Android apps is a hot topic. More and more researchers use static analysis and dynamic behavior analysis, and even integrate it with machine learning techniques to identify malware.

2.1 Static analysis methods

Barrera et al.[6] made an analysis on permission-based security models and its applications to Android through a novel methodology which applies Self-Organizing Map (SOM) algorithm preserving proximity relationships to present a simplified, relational view of a greatly complex dataset. The SOM algorithm provides a 2-dimensional visualization of the high dimensional data, and the analysis behind SOM can identify correlation between permissions. They discover insights on how the developers use the allowed permission model in developing and underlining the permission model's strengths as well as its shortcomings through their methodology. Based on their results, they propose some enhancements to the Android permission model.

Enck et al.[7] (TaintDroid) built a tool that warns users about applications that request blacklisted sets of permissions. They took both dangerous functionality and vulnerabilities into consideration and applied a wide range of analysis techniques. They designed and implemented a Dalvik decompiler, ded, which can recover application's Java source code only using its installation image. Besides, they analyzed 21 million LOC retrieved from the top 1100 free applications in the Android market using automated tests and manual inspection. Consequently they identified the essential causes of Android application security problems and showed the severity of discovered vulnerabilities. Their results show the wide misuse of privacy sensitive information, the evidence of telephone misuse, wide including of ad libraries in Android application, and the failing to securely use Android APIs of many developers.

Felt et al.[8] developed Stowaway, a tool to detect overprivilege in Android applications, and used this tool to evaluate 940 applications from Android market, finding that about one-third are overprivileged. Additionally, they identified and quantified developer's patterns leading to overprivilege. Moreover, they determined Android's access control policy through automatic testing techniques. Their results present a fifteen fold improvement over the Android documentation and reveal that most developers are trying to follow the principle of least privilege but fail due to the lack of reliable permission information.

Elish et al.[9] implemented an analysis tool to construct data dependence graphs statically with inter-procedural call connectivity information that capture the data consumption relations in programs through identifying the directed paths between user inputs (e.g., data and actions) and entry points to methods providing critical system services. Furthermore, they conducted an initial set of experiments to characterize the data consumption behaviors of legitimate and malicious Android apps with this tool, specifically on how they respond to user inputs and events. Nevertheless, some malwares may attempt to circumvent their data dependence checking by misusing the user's inputs while performing malicious activities, so their work need to be improved in these conditions.

Burguera et al.[10] proposed Crowdroid, which finds that open(), read(), access(), chmod(), and chown() are the most frequently used system calls by malware.

Hoffmann et al.[11] presented SAAF, which provides program analysis such as data-flow analysis and visualization of control flow graph. They analyzed about 136 000 benign apps and 6100 malicious apps, and their results confirm the previous observations for smaller app sets; what's more, their results provide some new insights into typical Android apps.

Nadji et al.[12] proposed airmid, which uses collaboration between in-network sensors and smart devices to identify the provenance of malicious traffic. They created three mobile malware samples, i.e., Loudmouth, 2Faced, and Thor, to testify the correctness of airmid. Airmid's remote repair design consists of an on-device attribution and remediation system and a server-based infection detection system. Once detected, the software executes repair actions to disable malicious activity or to remove malware entirely.

2.2 Dynamic behavior analysis

Portokalidis et al.[13] proposed Paranoid Android, a system where researchers can perform a complete malware analysis in the cloud using mobile phone replicas.

Zhou et al.[14] proposed DroidMOSS which takes advantage of fuzzy hashing technique to effectively localize and detect the changes from app-repackaging behavior.

2.3 Machine learning

Schmidt et al.[15] proposed a solution based on monitoring events occurring on Linux-kernel level. They applied the tool, readelf, to read static information held by executables and used the output of readelf to classify Android software. After applying readelf to both normal apps and malware apps, they used the names of the functions and calls appearing at the output of readelf to form their benign training set and malicious training set. Furthermore, they applied three classifiers, PART (extracting decision rules from the decision tree learner C4.5), Prism (a simple rule inducer which covers the whole set by pure rules), and nNb (a light-weight version of the well-known Nearest Neighbour algorithm), to predict whether an Android software is normal or malicious. The testing result show that their approach is effective. Additionally, they built a system which provides three main functionalities: on-device analysis, collaboration, and remote analysis, to detect malware apps on Android.

Frank et al.[16] investigated the difference between high-reputation and low-reputation applications, and further to identify malware. Their method only uses the permission requests, and doesn't statically analyze applications to extract features when there is no available code.

Shabtai et al.[17] similarly built a classifier for Android games and tools, as a proxy for malware detection.

Sanz et al.[18] applied several types of classifiers to the permissions, ratings, and static strings of 820 applications to see if they could predict application categories, using the category scenario as a stand-in for malware detection.

Zhou et al.[19] found real malware in the wild with DroidRanger, a malware detection system that uses permissions as one input.

3 MobSafe

3.1 Infrastructure cloud platform

3.1.1 CloudStack

Saturn-cloud[20,21], a home-brewed cloud computing platform, is used to conduct security analysis task. Saturn-storage, NFS storage with ZFS file system (openindiana+napp-it)[22], is used to accommodate the virtual machines. It can scale to 16 hard disks, each with 2 TB SATA storage, totally achieve 32 TB store volume. Cloudstack[23] is used to manage a VMware vSphere based computing servers. The whole cloud infrastructure is shown in Fig. 1.

3.1.2 Hadoop storage for mobile apps

There are about 40 servers and 40 TB storage in our experimental research platform based on HDFS.

3.2 Work principle

MobSafe is a system to check whether an Android app is virulence or benignancy based on some customized tools in cloud platform. The procedure of mobsafe is shown in Fig. 2.

MobSafe is an automatize system which can be used to analyze Android apps. When you submit an unknown apk file to MobSafe for analysis, it will check the key-value store whether the apk is already analyzed and its result is stored in hadoop storage. This comparison is based on the hashing of the apk file as the key to query

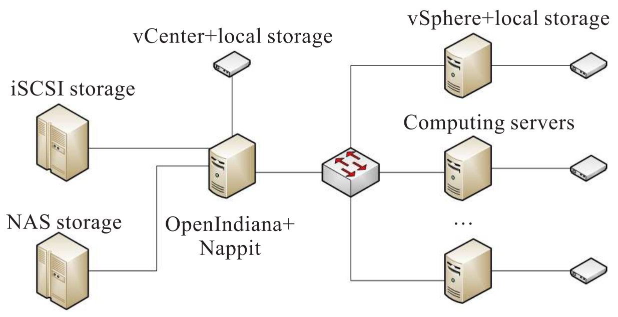

Fig. 1 Infrastructure cloud platform based on CloudStack.

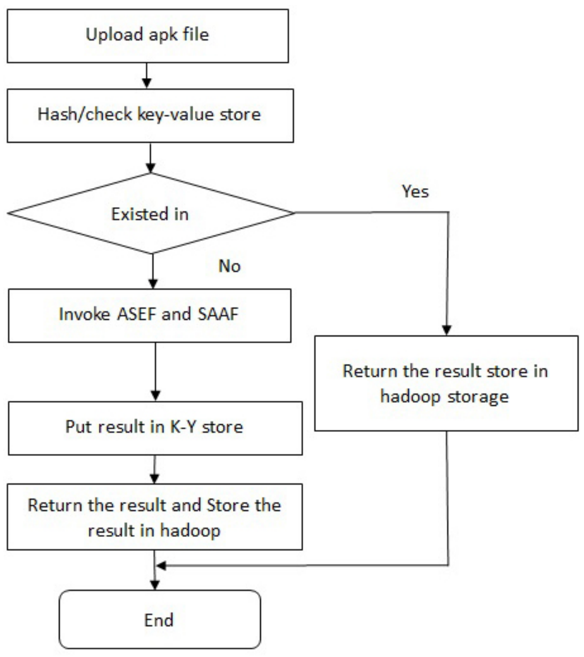
Upload apk file
Hash/check key-value store
Existed in
Yes
No
Invoke ASEF and SAAF
Return the result store in hadoop storage
Put result in K-Y store
Return the result and Store the result in hadoop
End

Fig. 2 The procedure of Android app analysis in Mobsafe.

the redis key value store. In this implementation, the redis version is 2.1.3. If the key is matched in redis, then the result is returned as response to submitter. If the key is not matched, it indicats a new apk file. In such case, the apk is stored in hadoop storage. After that, a daemon invokes the automatize tool, such as ASEF and SAFF, to collect the logs and store them in hadoop specified directory. Also the daemon inserts the key to redis and updates the value with the result directory in hadoop storage.

3.3 Frontend

Mobsafe has a web frontend, which is based on SpringSource's Spring framework, and Twitter Boostrap. It provides suspect apps upload function and returns the analysis result demonstrated in web page.

3.4 Backend

3.4.1 ASEF

ASEF[26] is an automatize tool which can be used to analyze Android application. When you submit an unknown apk file to ASEF for analysis, firstly it will start the ADB logging and traffic sniffing using TCPDUMP, then launch an Android Virtual Machine (AVD) and install the application on it. After that ASEF begins to launch the application to be analyzed and send a number of random gestures to simulate human integration on the application. Meanwhile, ASEF also compares the log of Android virtual machine with a CVE library, and its internet activity with Google Safe browser API[25]. After a certain number of gestures are sent to virtual machine, the test circle is ended and the application will be uninstalled. Then ASEF will begin to analyze the log file and the Internet traffic that the app generated. ASEF uses Google Safe Browsing API to find out whether the URLs the app try to reach are malicious or not. ASEF also checks the existed vulnerability with a known vulnerability list to find out whether the application has some serious vulnerability.

3.4.2 SAAF

SAAF[24] is a static analyzer for Android apk files. It can extract the content of apk files, and decode the content to smali code, then it will apply program slicing on the smali code, to analyze the permissions of apps, match heuristic patterns, and perform program slicing for functions of interest.

3.4.3 Other tools

There are also a lot of other assistance static analysis tools, such as readelf[27], ded[28], apktool[29], androguard[30], and soot[31], to help us analyze the Android apps. Most of these tools are based on reversing engineering. Some dynamic analysis tools like Strace[32] and Randoop[33] to detect Android apps are based on runtime behaivor. Strace watches system call in Linux kernel while Randoop stimulates the Android apps by random inputs and watches the output messages.

4 Evaluation

4.1 Global statistics of the dataset

We have collected data set from AppChina, a China Android market with its Android app installation tool. This assistance tool helps a user to install, upgrade, and remove Android apps quickly and logging such operations for analysis. The data set is collected during the three-month period from May 1st to July 31st in 2012. The size of data set is about 1 TB zipped logs (expanded size above 10 TB). Totally there are about 100 000 active Android apps in logs. We downloaded Android apps from AppChina to verify based on MobSafe. Each downloaded Android app has its web page on the market website. We also crawled the web version of the Android market to supply Android app with text description. We also conduct some correct proof by self-written malware verification.

Figure 3 shows the total number of active apps in AppChina keeps steadily increase during these three months. It maintains a growth rate above 10%.

From Table 1, all these resolution Android devices account for about 90% of total Android devices. We also notice that high resolution display Android device users increase steadily while some middle resolution display Android device users decrease steadily.

We classify the Android devices into three categories: Low class, Middle class, and High class according to the display resolution. It seems that the display resolution of Android devices is increased steadily in these three months as shown in Fig. 4.

It also needs to notice that the number of apps installed in mobile Android devices is about 30 according to three months' statistics in Fig. 5. But as

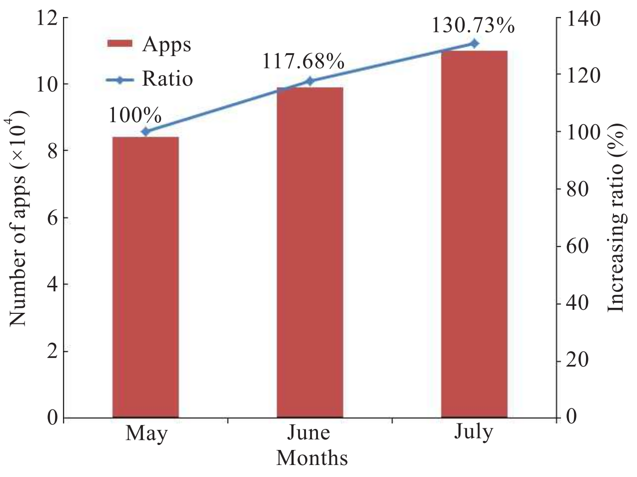

Fig. 3 The trend of Android mobile apps in AppChina in one quarter (in 2012).

Table 1 Different display resolution with different portion accounts for all Android devices in 2012.

	Portion account of different display resolutions (%)						
	240×320	320×480	480×800	480×854	540×960	720×1280	800×1280
May	5.40	26.45	37.60	13.00	7.11	1.22	4.30
June	4.12	29.41	35.31	12.88	5.99	1.93	5.54
July	3.77	31.76	35.1	11.83	5.39	2.73	5.47

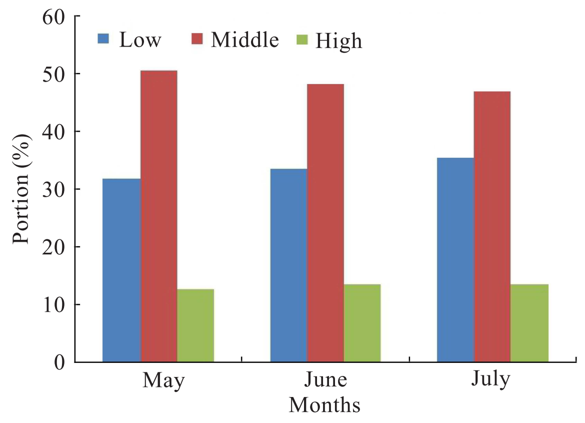

Fig. 4 The portion of three different display's resolutions varied in three month in 2012.

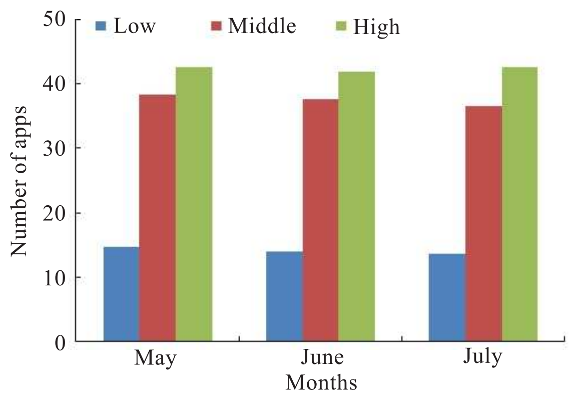

Fig. 5 The average number of apps installed in three category Android devices in 2012.

more and more users chose high resolution Android devices, the number of apps installed in device increases too.

4.2 Performance metrics

4.2.1 ASEF

In order to measure how much time ASEF takes to analyze an app, we write a script which can record the timestamp of the beginning of running a program and use ASEF to analyze 20 different Android apps downloaded from AppChina. The result is shown in Fig. 6, where the time it takes to analyze one application varies from 64 s to 150 s, and the average time is about 100 s. It means that we can finish the analysis and acquire the result in less than 2 min on average.

When we look up the whole analysis procedure in

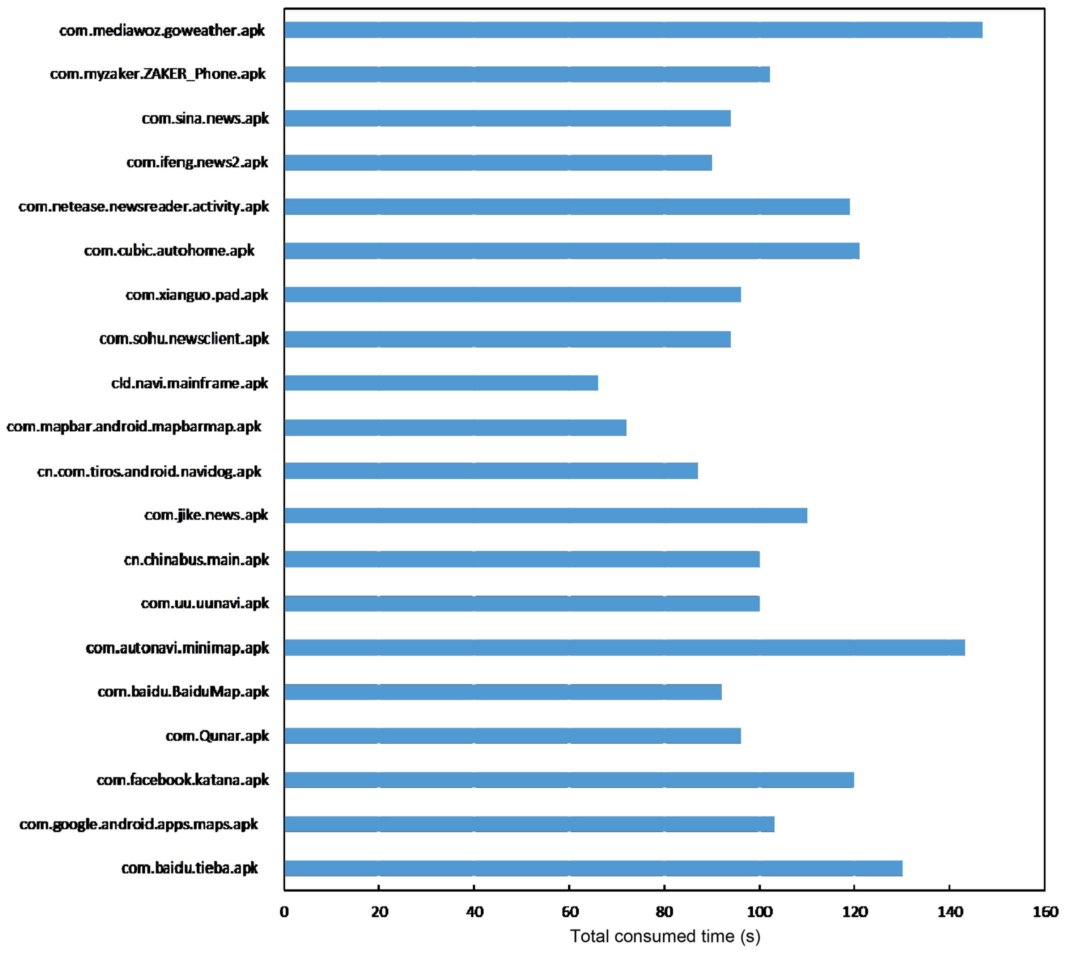

Fig. 6 ASEF: The total consumed time of each app.

detail, we can find out that there are 6 steps during analyzing one app. The preparing step, the starting log service step, the ending process step, and the analyzing step take up 3%, 3%, 5%, and 10% of total time separately. About 80% of time is consumed on the installing and testing stage, shown in Fig. 7. So if we want to reduce the total time, we should try to speed up these two steps.

In the analysis step, the time it takes depends on the random gestures we input. The more gestures, the longer it takes. Figure 8 presents the result of reduced time by cutting down some gestures. We decrease the number of gestures sent to AVD so that the testing time will be shortened. After we decrease the number of gestures from 1000 to 200, the total time decreases by 20 s, which accounts for 20% of the total time. This method is effective and we can also use it to improve user's experience.

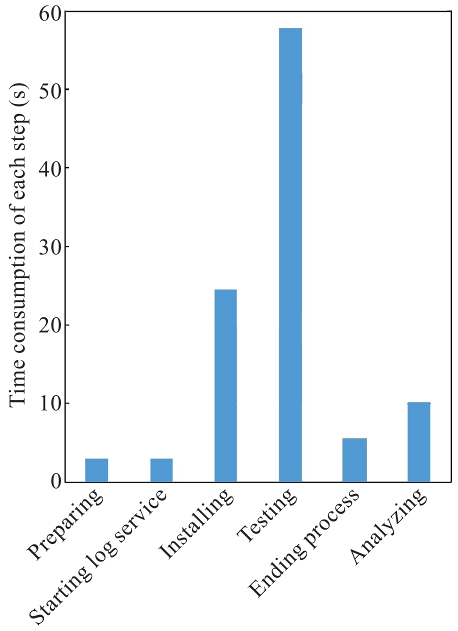

Fig. 7 ASEF: Time consumption of each step (s).

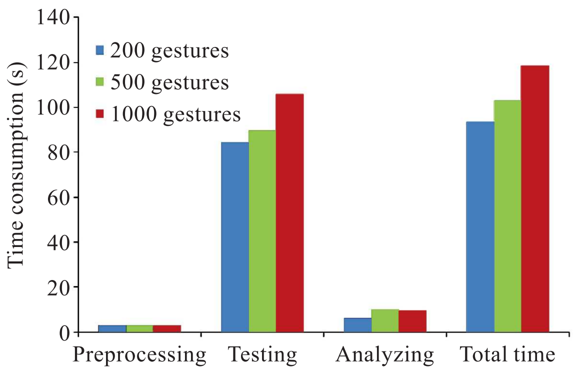

Fig. 8 The time consumption analysis of ASEF framework.

4.2.2 SAAF

We apply SAAF to 25 Android apps downloaded from AppChina for static smali code analysis, to evaluate the performance of this tool. From Fig. 9 below, we can see that the most time consuming step of SAAF is the slicing step, and the second is the permission categorizing step. The average time of analyzing one app consumed by SAAF in one Linux virtual machine, which runs on Intel-i5 four-core CPU with 4 GB of memory, is about 33.93 s.

From Fig. 10, we know that the analyzing of different apps will consume different times, and the total time depends on the complexity of apps, such as the amount of methods etc. But for most apps, SAAF will finish the analysis in an acceptable period.

4.3 Estimated instances

That means if we apply ASEF to all the apps in Google Play market, which has 800 000 apps in total, it will consume about 450 hours by 50 such virtual machines, which runs on Intel-i5 four-core CPU with 4 GB of memory.

If we apply SAAF to all the apps in Google Play market too, it will consume about 151 hours by 50 such virtual machines.

From the above calculation, it also needs to notice that the dynamic method (such as ASEF) costs more

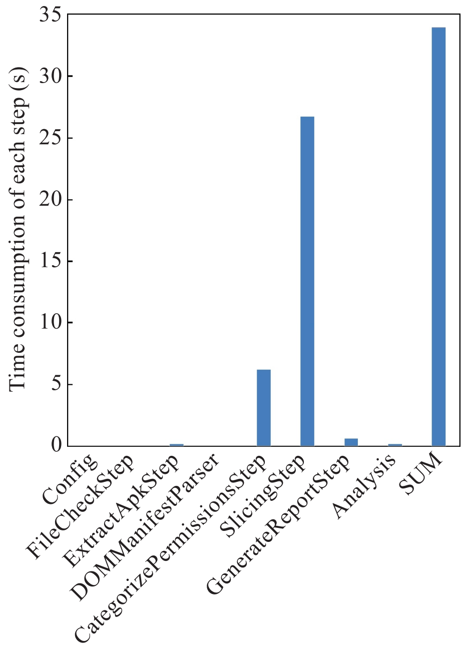

Fig. 9 SAAF: Time consumption of each step (s).

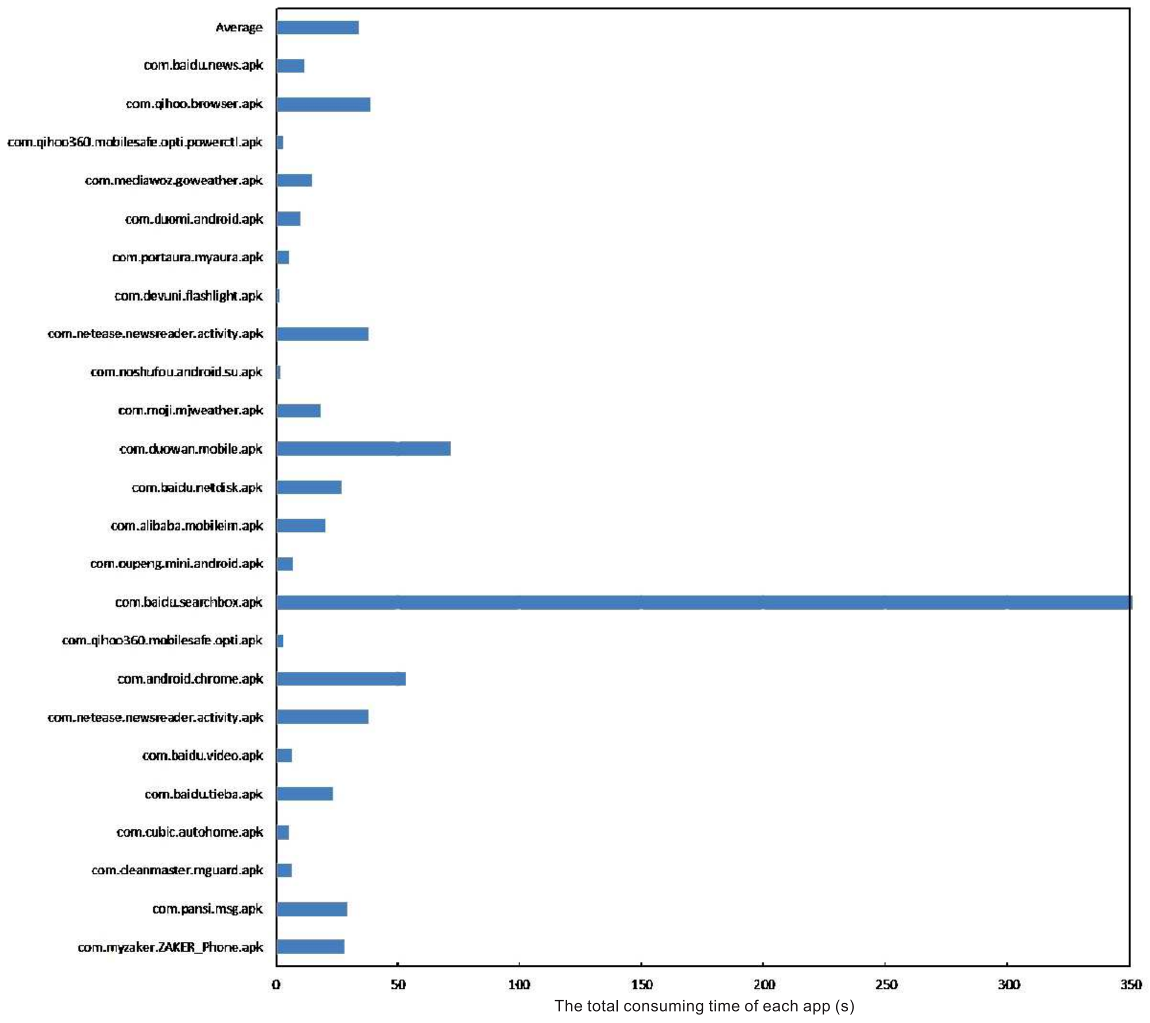

Fig. 10 SAAF: The total consuming time of each app.

time than the static one (such as SAAF) as the former one needs to monitor app's system call and network behaviour.

According to average number of apps installed in one Android device is about 30, it costs about 1 hour to use ASEF and SAAF to finish the analysis in one virtual machine and AVD. But if we can distribute the installed apps into separated individual VMs or AVDs, the whole time can be less than one minute, which is acceptable for user's experience in security check.

5 Conclusions

In this paper, we propose a methodology to evaluate the security of Android mobile apps based on cloud computing platform. We also implement a prototype system, i.e., MobSafe, for automation forensic analysis of mobile apps' static code and dynamical behavior. Based on the real trace from AppChina, a mobile app market, we can estimate that the number of active Android apps and the average number apps installed in one Android device, and the increasing ratio of mobile apps. We adopt ASEF and SAAF, the two representative dynamic analysis method and static analysis method, to evaluate the Android apps and estimate the total time needed to evaluate all the apps stored in a mobile app market. As mobile app market serves as the main line of defence against mobile malwares, it is practical to use cloud computing platform to defence malware in mobile app markets.

6 Future Work

Machine Learning (ML)[34] is a promising technology to identify mobile app's virulence or benignancy based on data mining. As we collect more and more app's logging and network behaviour data, we can further use K-means method to classify apps after training a classifier. In this case, the well-known accuracy metrics includes precision and recall can be measured to evaluate the calssifier algorithm. Other method such as PCA (Primary Component Analysis) and Matrix Factorization also can be used and tested on such data. The final implementation will be shown on the web site (www.mobsafe.net).

Acknowledgements

The authors would like to thank Prof. Jun Li of NSLAB from RIIT for his guidance. This work was supported by the National Key Basic Research and Development (973) Program of China (Nos. 2012CB315801 and 2011CB302805) and the National Natural Science Foundation of China (Nos. 61161140320 and 61233016). This work was also supported by Intel Research Council with the title of Security Vulnerability Analysis based on Cloud Platform with Intel IA Architecture.

References

[1] R. Lawler, Mary Meeker's 2013 Internet Trends report, http://techcrunch.com/2013/05/29/mary-meeker-2013-internet-trends/, May 29, 2013.

[2] J. Wu, *On Top of Tides (Chinese Edition)*, Beijing: China Publishing House of Electronics Industry, January 8, 2011.

[3] S. Q. Feng, *Android software security and reversing engineering analysis (Chinese Edition)*, Beijing: Posts and Telecom Press, Feb. 2013.

[4] Gartner, http://www.gartner.com/it/page.jsp?id=2153215, September 11, 2012.

[5] List of mobile software distribution platforms, http://en.wikipedia.org/wiki/List_of_digital_distribution_platforms_for_mobile_devices, July 19 2013.

[6] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, A methodology for empirical analysis of permission-based security models and its application to Android, in *Proc. 17th ACM Conference on Computer and Communications Security*, Chicago, USA, 2010, pp. 73-84.

[7] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, A study of android application security, in *USENIX Security Symposium*, San Francisco, USA, 2011.

[8] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, Android permissions demystified, in *Proc. 18th ACM Conference on Computer and Communications Security*, Chicago, USA, 2011, pp. 627-638.

[9] K. O. Elish, D. Yao, and B. G. Ryder, User-centric dependence analysis for identifying malicious mobile apps, in *Workshop on Mobile Security Technologies (MoST)*, San Francisco, USA, 2012.

[10] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, Crowdroid: Behavior-based malware detection system for Android, in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, Chicago, USA, 2011, pp. 15-26.

[11] J. Hoffmann, M. Ussath, T. Holz, and M. Spreitzenbarth, Slicing droids: Program slicing for smali code, in *Proc. 28th Annual ACM Symposium on Applied Computing*, Coimbra, Portugal, 2013, pp. 1844-1851.

[12] Y. Nadji, J. Giffin, and P. Traynor, Automated remote repair for mobile malware, in *Proc. 27th Annual ACM Computer Security Applications Conference*, Orlando, USA, 2011, pp. 413-422.

[13] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, Paranoid Android: Versatile protection for smartphones, in *Proc. 26th Annual ACM Computer Security Applications Conference*, Austin, USA, 2010, pp. 347-356.

[14] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets, in *Proc. 19th Annual Network and Distributed System Security Symposium*, San Diego, USA, 2012.

[15] A. D. Schmidt, R. Bye, H. G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, Static analysis of executables for collaborative malware detection on Android, in *Communications, ICC'09, IEEE International Conference on*, Dresden, Germany, 2009.

[16] M. Frank, B. Dong, A. P. Felt, and D. Song, Mining permission request patterns from Android and facebook applications, in *Proc. 12th IEEE International Conference on Data Mining*, Brussels, Belgium, 2012, pp. 870-875.

[17] A. Shabtai, Y. Fledel, and Y. Elovici, Automated static code analysis for classifying Android applications using machine learning, in *Proc. 6th IEEE International Conference on Computational Intelligence and Security (CIS)*, Nanning, China, December, 2010, pp. 329-333.

[18] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P. G. Bringas, On the automatic categorisation of Android applications, in *Proc. 9th IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, Nevada, USA, January, 2012, pp. 149-153.

[19] W. Zhou, Y. Zhou, Y. Jiang, and P. Ning, Detecting repackaged smartphone applications in third-party Android marketplaces, in *Proc. 2nd ACM conference on Data and Application Security and Privacy*, San Antonio, TX, USA, February, 2012, pp. 317-326.

[20] Z. Chen, F. Y. Han, J. W. Cao, X. Jiang, and S. Chen, Cloud computing-based forensic analysis for collaborative network security management system, *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 40-50, 2013.

[21] T. Li, F. Han, S. Ding, and Z. Chen, LARX: Large-scale Anti-phishing by Retrospective Data-Exploring Based on a Cloud Computing Platform, in *Proc. 20th International Conference on. IEEE. Computer Communications and Networks (ICCCN)*, Maui, Hawaii, USA, 2011, pp. 1-5.

[22] IPSAN storage, Openindianna + napit, http://openindiana.org/ and http://www.napp-it.org, June, 2013.

[23] Cloudstack project, http://cloudstack.apache.org, June, 2013.

[24] ASEF project, https://code.google.com/p/asef/, June 2013.

[25] Google Safe Browsing API v2, http://code.google.com/apis/safebrowsing/, June, 2013.
[26] SAAF project, https://code.google.com/p/saaf/, June, 2013.
[27] Readelf, http://sourceware.org/binutils/docs/binutils/readelf.html, June, 2013.
[28] Ded, http://siis.cse.psu.edu/ded/, June, 2013.
[29] Apktool, https://code.google.com/p/Android-apktool/, June, 2013.
[30] Androguard, https://code.google.com/p/androguard/, June, 2013.
[31] Soot, http://www.sable.mcgill.ca/soot/, June, 2013.
[32] Strace, https://zh.wikipedia.org/wiki/Strace, June, 2013.
[33] Randoop, https://code.google.com/p/randoop/, June, 2013.
[34] J. Wu, *Beauty of mathematics (Chinese Edition)*, Beijing: Posts and Telecom Press, January 6, 2012.

Jianlin Xu is an undergraduate student of Department of Computer Science and Technology at Tsinghua University. His research interests include network security and network structure.

Yifan Yu is an undergraduate student of Department of Electronic Engineering at Tsinghua University. His research interests include network security and mobile safety.

Zhen Chen is an associate professor in Research Institute of Information Technology in Tsinghua University. He received his BEng and PhD degrees from Xidian University in 1998 and 2004. He once worked as postdoctoral researcher in Network Institute of Department of Computer Science and Technology in Tsinghua University during 2004 to 2006. He is also a visiting scholar in UC Berkeley ICSI in 2006. He joined Research Institute of Information Technology in Tsinghua University since 2006. His research interests include network architecture, computer security, and data analysis. He has published around 80 academic papers. Now he is leading nslab-saturn team (www.nslab-saturn.net)

Bin Cao is currently a master student of Department of Computer Science and Technology at Tsinghua University. He got his BEng degree from PLA Univ. of Info. & Engi in 2005. His research interests include computer network security and mobile safety.

Wenyu Dong is currently a master student of Department of Computer Science and Technology at Tsinghua University. He got his BEng degree from PLA Univ. of Info. & Engi in 2009. Currently his main research interests focus on computer network security and mobile safety.

Yu Guo is currently a master student of Department of Computer Science and Technology at Tsinghua University. He received his BEng degree from PLA Univ. of Info. & Engi in 2005. His main research interests include network and information security, computer software performance and wireless network.

Junwei Cao is currently Professor and Deputy Director of Research Institute of Information Technology, Tsinghua University, China. He is also Director of Open Platform and Technology Division, Tsinghua National Laboratory for Information Science and Technology. His research is focused on advanced computing technology and applications. Before joining Tsinghua in 2006, Junwei Cao was a Research Scientist of Massachusetts Institute of Technology, USA. Before that he worked as a research staff member of NEC Europe Ltd., Germany. He got his PhD in computer science from University of Warwick, UK, in 2001. He got his MEng and BEng degrees from Tsinghua University in 1998 and 1996, respectively. He has published over 130 academic papers and books, cited by international researchers for over 3000 times. He is a senior member of the IEEE Computer Society and a member of the ACM and CCF.