International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 2 Issue: 5 1069– 1073

Faamac: Forensic Analysis of Android Mobile Applications using Cloud Computing

S.V. Nagendra Prasad Yadav
Student, Dept. of CS&E
AKIT, Tumkur

R.C. Shivamurthy
HOD, Dept. of CS&E
AKIT, Tumkur

Abstract **–** Mobiles have gained widespread usage& in smart phones many interesting applications are made available through Google Play Android is one of the major Smartphone platform today. The explosive increase in mobile apps more and more threats migrate from traditional PC client to mobile device. Smartphone applications can steal users' private data and send it out behind their back Smartphone's which store various personal data, such as phone identifiers, location information and contacts. mobsafe prototype is proposed methodology to evaluate mobile apps based on cloud computing platform and data mining. MobSafe prototype helps 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, Android Security Evaluation Framework (ASEF) and Static Android Analysis Framework(SAAF), the two representative dynamic and static analysis methods is adopted to evaluate the Android apps and estimate the total time needed to evaluate all the apps stored in one mobile app market & evaluation results show 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.

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

*****

I.INTRODUCTION

Powerful and well-connected Smartphone's are becoming increasingly common. Their features are provided by focused applications that users can easily install from application market places. With hundreds of thousands of applications available Smartphone applications can steal users' private data and send it out behind their back. The worldwide Android smartphone market is raises security and privacy concerns. However, current Android's permission based approach is not enough to ensure the security of private information.

A. MOBILE THREATS

The witness in an explosive increase in mobile apps. on Mobile Internet trends more and more PC client software's are migrating to the mobile device. The amount of total downloads of mobile apps in 2013 will be about 81 billion. Among these, there are about 800000 Android apps in Google Play market, and the total download is about 48 billion as of May 2013.

Some malicious behaviors of Android malware is usually motivated by controlling mobile device without user intervention, such as:

(1) Privilege escalation to root,
(2) Leak private data
(3) Dial premium numbers,
(4) Botnet activity and

B. Root causes for Android malware origins are as follows:

(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, reboot.

II 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

A. Static analysis methods

Barrera et al. made an analysis on permission basedsecurity models and its applications to Android through a novel methodology which applies Self- Organizing Map 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.

Nadji et al 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 correctnessof 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

.Felt et al.developed Stowaway, a tool to detect over privilege in Android applications, and used this tool to evaluate 940 applications from Android market, finding that about one-third are over privileged. Additionally, they identified and quantified developer's patterns leading to over privilege. Moreover, they determined Android's access control policy through automatic testing techniques. Their results present a fifteen fold

improvement over the Android documentation andreveal that most developers are trying to follow the principle of least privilege but fail due to the lack of reliable permission information.

B. Dynamic behavior analysis

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

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

C . Machine learning

Schmidt et al. 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.

III THE PROPOSED METHODOLOGY

Home-brewed cloud computing platform and data mining, a methodology is proposed 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 analyze time to an acceptable level. In the implementation, 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.

IV SYSTEM DESIGN

A. System Architecture

mobsafe prototype that defines the structure and /or behavior of a system.. It also defines the system components or building blocks and provides a plan from which the system developed. The architectural design process is concerned with establishing the basic structural framework for a system.

System architecture also involves identifying the major components of the system and communications between these components.

MobSafe is a system to which is used to check an Android app is virulence or benignancy based on some customized tools in cloud platform. The procedure of mobsafe is shown in Fig.1.MobSafe is an automotive 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 the redis key value store.

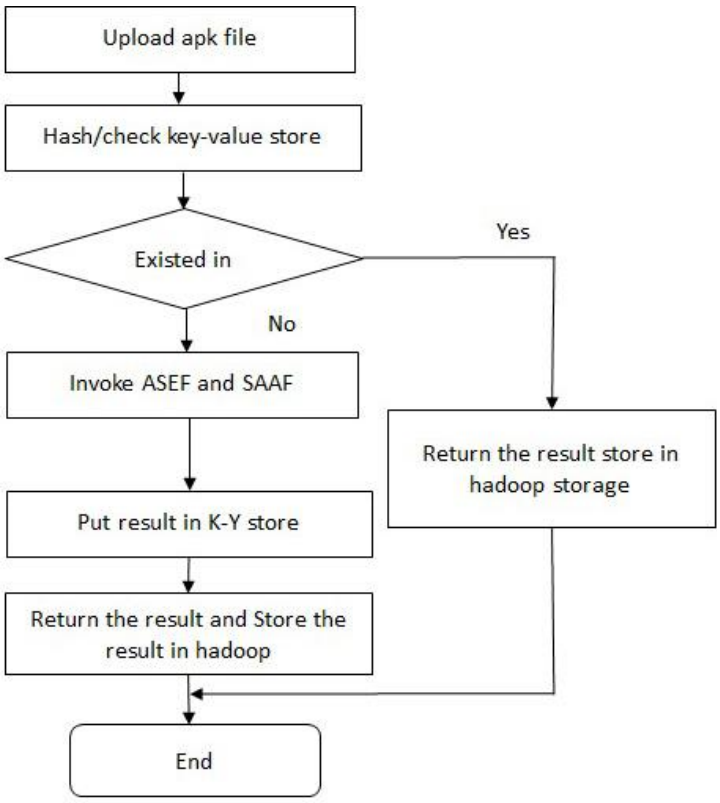

Figure.1 system architecture

 ISSN: 2321-8169
1069– 1073

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 indicates 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.

B. General steps in Mobsafe

There are 5steps as follows: start,upload apk file and check key hash value store, invoke tools ASEF & SAAF,Show result.

user registers for application in cloud and platform sends the credentials via email and sms which will be active for 15 days and user uploads the apk file to find the criticality of app n cloud platform applies ASEF and SAAF an algorithms to find the nature of app and stores result ,if the app is already analyzed the status is sent to user and user requests for registration extension after expiry to admin user. The admin user approves the requests and updates the platform and new credentials will be sent to user and admin user can query the app analysis statistics generate graph for the same.

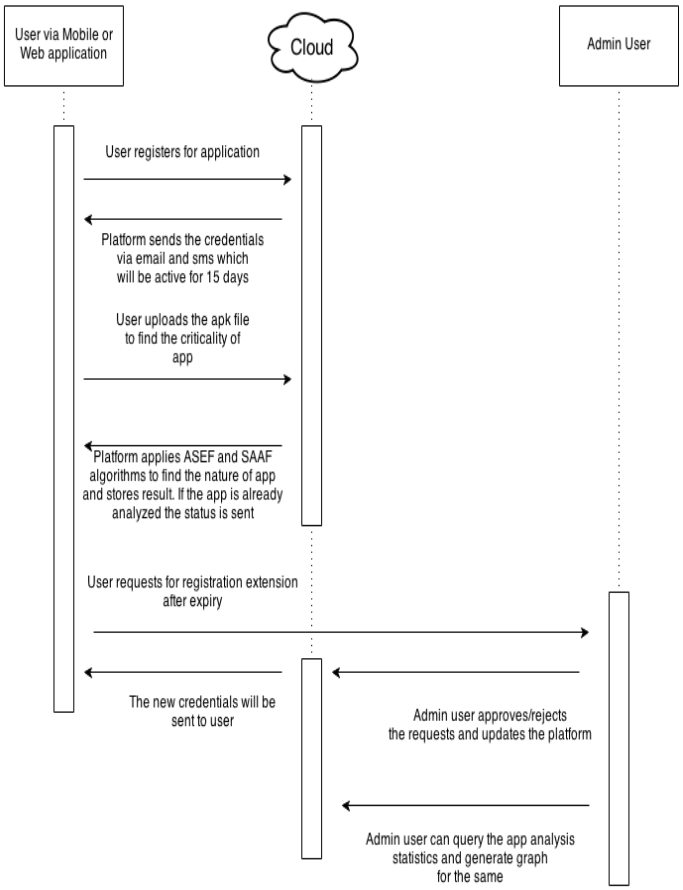

Figure 2.general sequence of design

(1) ASEF

ASEF is an automatize tool which can be used to analyze Android application. When you submit an unknown apk file to ASEF for analysis, As shown in figure 3 it as three phases: active, passive, interpret 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. 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.

(2)SAAF

SAAF 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.

(2) Other tools

There are also a lot of other for cracking techniques that rely on weaknesses in the human being attached to a computer system rather than software; the aim is to trick people into revealing passwords or other information that compromises a target system's security.

V Performance metrics

A. *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 Google app.

The result is shown inFig. 6, where the time it takes to analyze one applicationvaries 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 detail

we can find out that there are 6 steps during analyzing one app. The preparing step, the starting log service step, the

 ISSN: 2321-8169
1069– 1073

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. 4. 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.

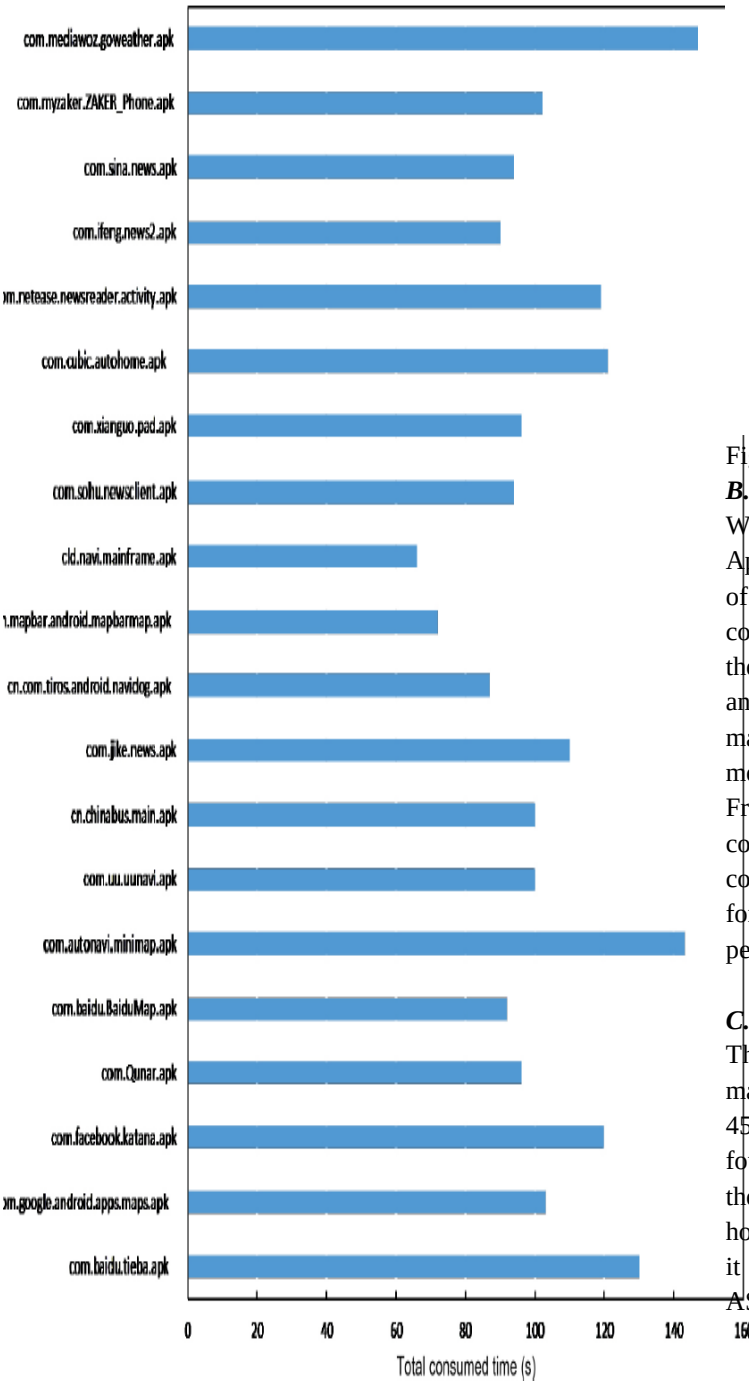

Figure 4 ASEF: The total consumed time of each app

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.

.

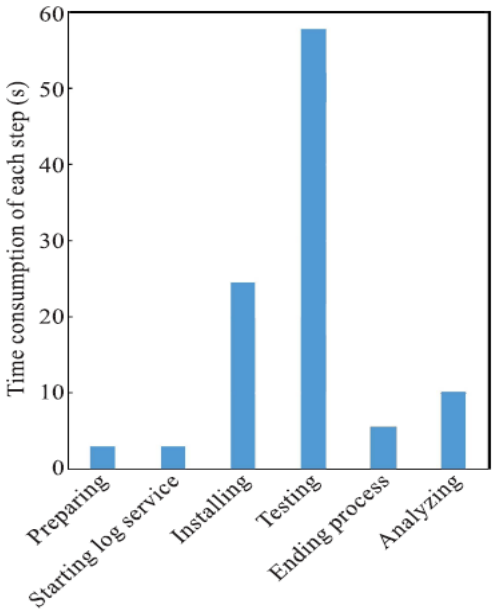

Figure 5.the time consumption for each app

B.SAAF

We apply SAAF to 25 Android apps downloaded from Google App for static smali code analysis, to evaluate the performance of this tool. From Fig. 6 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. 7, 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.

C. 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 4GB 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

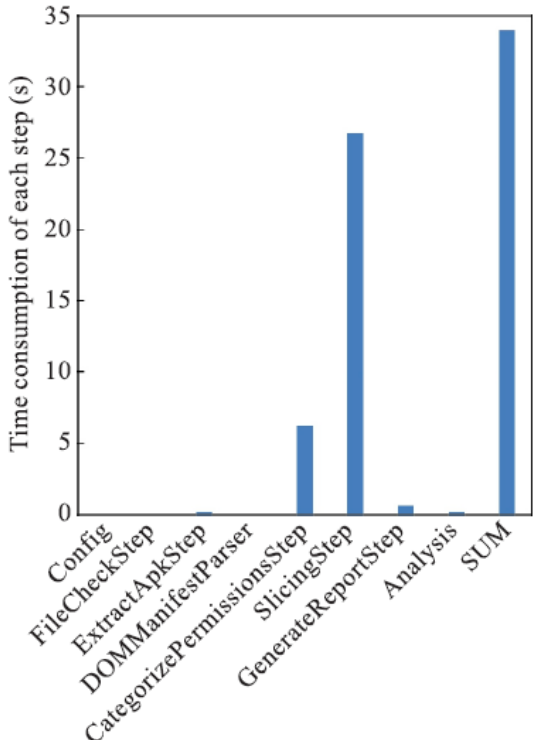

Figure 6. The SAAF time consumption of each step

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

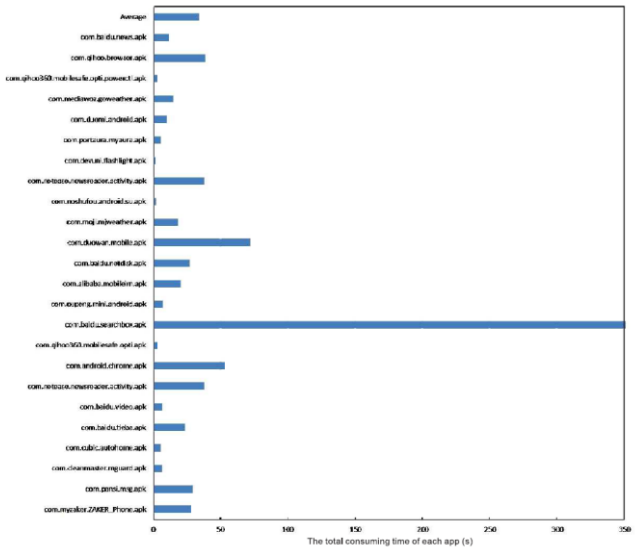

Figure.7 The SAAF:the total time consuming for each app

VI Conclusion

The proposed methodology is to improve security status & evaluate the security of Android mobile apps and to indentify mobile apps virulence or benignancy based on cloud computing platform. The prototype system Mob Safe can be implemented for automation forensic analysis of mobile apps static code and dynamical behavior ASEF and SAAF the two representative dynamic analysis method and static analysis method can be used to evaluate the Android apps and estimate the total time needed to evaluate all the apps stored in a mobile app market

VII Future Work

There is a lot of future work that can be done with this project. Machine Learning (ML) is a promising technology to identify mobile app's virulence or benignancy based on data mining. As we collect more and more app' slogging and network behavior 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 classifier algorithm. Other method such as PCA (Primary Component Analysis) and Matrix Factorization also can be used and tested on such data.

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] Sangho Lee and Da Young Ju International Journal of Security and Its Application Vol.7, No.5 (2013), pp.121-130 http://dx.doi.org/10.14257/ijsia.2013.7.5.10

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

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

[6] List of mobile software distribution platforms, http://en.wikipedia.org/wiki/List of digital distribution platforms for mobile devices, July 19 2013.

[7] 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.

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

[9] 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.