International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 12 | Dec 2017 www.irjet.net p-ISSN: 2395-0072

Energy Efficient Monet for Malware Detection System for Android Smartphones

Shyam S. Gupta[1], Dr. Sanjay Jain[2]

[1]Research Scholar, SJJTU, Jhunjhunu
[2]Ph. D Guide, SJJTU, Jhunjhunu

Abstract - *The Android OS based Smartphones has significant demand in the mobile market because it has amazing applications support. In the smartphones, there is two research problems such as energy efficiency and malware detection. In the smartphone's applications needs the significant amount of energy for malware scanning and processing at runtime. The smart phones have limited battery backup, there is need to energy minimization approach using many applications. Malwares are a huge threat to mobile security, therefore it must have efficient malware detection system. In the literature above both problems are independently studied, and in this paper, we are presenting the EEMonet hybrid approach in which aim is to defend against different types of android malware and less energy consumption performance. For malware detection, approach is consisting of backend server and client. The client model is nothing but in-device app for monitoring behavior and generating the signature based on new interception methods. The backed server is doing the task of large scale malware detection. This approach combines "static logic structures" and "dynamic runtime information for malware detection. For energy efficiency, we are designing an algorithm that adds sophisticated energy-aware computation offloading capabilities to Android apps. The proposed method monitors device and application status and then decides where code should be executed automatically.*

Key Words: Malware detection, Android, Runtime behaviour, Mobile computing, Energy management, Distributed computing.

1. INTRODUCTION

ANDROID is a mobile operating system from Google and it powered mobile devices dominate around 78:7 % of the smartphone OS market in the first quarter of 2016. Android applications (apps for short) can be downloaded not only from the Google's official market Google Play, but also from third-party markets. Although Google Play scans any uploaded apps to reduce malware, other markets/sites usually do not have sufficient malware screening, and they become main hotbeds for spreading Android malware. As a result, Android attracts millions of malware. It is reported that 97 % of mobile malware is on the Android platform. Broadly speaking, there are two types of in-device malware detection systems. The first one is to perform static malware detection. This type of systems uses static information such as API calling information and control flow graphs to generate signatures for detection. For example, anti-virus engines will scan files in apps after their installation. However, studies have shown that these types of anti-virus engines can be easily bypassed using transformation attacks (i.e. code obfuscation techniques like package name substitution and reflection technique). Furthermore, sophisticated signature generation and signature matching techniques based on control flow finding incur considerable computation overhead, and consume energy on mobile devices which have limited battery resource, preventing them from being adopted as in-device detection systems. The second type of in-device detection system is the dynamic intrusion prevention system, as seen in several products and research studies. These systems work in the background and monitor apps at runtime. Once they discover any suspicious behavior, a notification will pop up to alert the users. Note that suspicious behaviors are usually based on sensitive APIs. Most benign apps (e.g., Text message management apps) may also invoke these APIs (e.g., sending text message API) for legitimate reasons. Therefore, this type of systems may introduce false alerts and makes intrusion notifications annoying and less preferable. Moreover, a study also shows that existing products in the market can be easily circumvented.

According to a survey, it was reported that over 98 % of new malware samples are in fact derivatives (or variants) from existing malware families. These malware variants use more sophisticated methods like dynamic code loading, manifest cheating, string and call graph obfuscation to hide themselves from existing detection systems. Although these techniques can help malware to hide their malicious logic, we observe that the "dynamic behaviors" of malware's core functionalities, such as unauthorized subscription of premium services or privilege escalation at runtime, remain unchanged. The runtime behaviors of a new malware variant and its earlier generation are usually very similar. A detection system based on runtime behaviors of malware will be able to detect more malware and their variants more reliably. In addition, the static structures of the malware are often similar within a malware family. However, battery life has become one of the biggest obstacles for mobile device advancements. Performance demanded by smartphones and tablets is increasing at a much faster rate than technological improvements in battery capacity. The need for increased performance of mobile devices directly conflicts with the desire for longer battery life. One popular technique to reduce energy consumption of mobile devices is computation offloading in which an application reduces energy consumption by delegating code execution to other devices. Traditionally, computations are offloaded to remote, resource-rich servers Selection of a proper offloading strategy can reduce power consumption and simultaneously enhance mobile device performance.

2. OBJECTIVE OF THE STUDY

The present study tries to show Energy Efficient Monet for Malware Detection System for Android Smartphones.

3. THE REVIEW

In this chapter, we are presenting the different way those are presented to mine high utility item sets effectively. Mobile devices have limited resources such as battery capacity, storage, and processor performance. Computation offloading is an effective method to alleviate these restrictions by sending heavy computations to resourceful servers and receiving results from these servers. Many issues related to computation offloading have been investigated in the past decade, including feasibility of offloading, offloading decisions, and development of offloading infrastructures. Jade was built upon previous research regarding program partitioning, code offloading, and remote execution. In this section, we provide an overview of proposals by these researches and how they relate to Jade.

Cuervo et al. Proposed MAUI, a system that enables energy-aware offloading of mobile code to the infrastructure. MAUI enables developers to produce an initial partitioning of their applications by annotating methods and/or classes as remotable. At runtime, the MAUI solver decides which remotable methods should execute locally and which should execute remotely. Unlike MAUI, Jade provides a sophisticated programming model with a full set of APIs, so developers have total control on: how application is partitioned, where code is offloaded and how remotable code interacts with local code. In Jade, dependencies do not exist between remotable tasks, the profiler and optimizer do not need to analyze the whole program, thereby, energy cost of program profiling and cost model calculation is lower than MAUI.

Chun et al. proposed Clone Cloud, an application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. In Clone Cloud, threads must be paused, all states of the threads must be transferred to the server, and then threads resume on the server in order to offload computation. Offloading is expensive, however, especially when the client and server are both resource constraint mobile devices. In contrast, code offloading in Jade system is lightweight. Remotable objects are serialized, transferred, and de-serialized, resulting in much lower overhead compared to thread migration. Proposed approach framework and design.

3.1 Problem Definition

Now days use of android operating system (OS) based mobiles is growing and captured around 1/3 of mobile market. In android, there is variety of apps already developed and still daily number of new apps has been introduced. Such android apps can easily download from the Google play store as well as from other third party stores. The Google play store supporting the apps scanning methodologies during their uploading process in order to reduce the malwares, however the third party stores may not be equipped with such provisioning and hence they leads to android malware spreading. Therefore it is must to have methodology to perform the detection of android malwares from end users mobiles during the apps installations or working. There are two approaches of malware detection from users mobile such as static malware detection and dynamic intrusion detection systems. In literature under these two categories number of solutions introduced, but suffered from the limitations in terms of efficiency and scalability. Such methods are not supporting to scan the runtime behaviors of malware in order to detect them successfully. Recently, another hybrid solution introduced in which “runtime behavior” with “static structures” is combined to detect malware variants efficiently, this method called as Monet. The problem identified with Monet is power consumption or energy efficiency is not addressed while malware detection.

3.2 Proposed System Architecture

In this paper, we are presenting the modified version of Monet approach called EE-Monet which stands for energy efficient Monet with goal of user-oriented behavior-based malware variants detection system for android with minimum power consumption. In Monet, we studied that it was introduced to detect the malware variants as well as defend against the transformation attacks. Basically Monet is generating the signature of runtime behavior composed of RBG (runtime behavior graph) and SSS (suspicious system call set) in order to accurately present the malware runtime behavior. We further modifying the Monet to EEMonet in which we are using the Jade system. Jade is a system that adds sophisticated energy-aware computation offloading capabilities to Android apps. This will helps to minimize the system overhead in terms of energy consumption while malware scanning process.

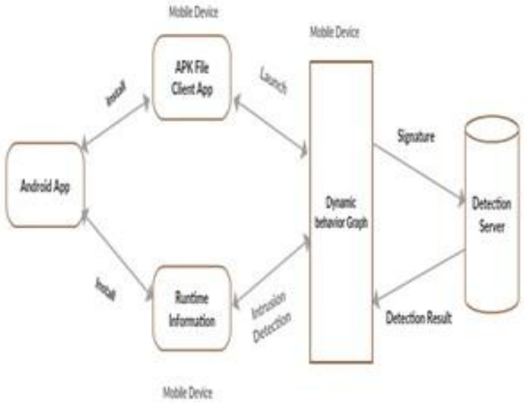


Figure 1. System Architecture

4. ALGORITHMS

4.1 Input Set

Android application for malware detection.

4.2 Malware Detection Algorithm

```
STEP-1 START
STEP-2 Input Android App.
STEP-3 APK file at client side.
STEP-4 Static behavior graph of app.
STEP-5 Static behavior graph generation.
STEP-6 Get runtime information of app.
STEP-7 Apply runtime information collection.
STEP-8 Dynamic behavior graph of app.
STEP-9 Intrusion detection.
STEP-10 Signature graph.
STEP-11 Signature generation.
STEP-12 Signature send to the server.
STEP-13 Server send detection result.
STEP-14 STOP
```

4.3 Code Offloading Algorithm

```
STEP-1 START
STEP-2 Records code information in the offloaded code table (Table 1).
STEP-3 Offloads the code to the server.
STEP-4 the server receives the code and executes it in a new thread.
STEP-5 the server sends the code back to the client after execution is complete.
STEP-6 the client receives the code and updates code information in the offloaded.
STEP-7 STOP
```

Table 1: Example of offloaded code table

Example of offloaded code table				
ID	Offloaded	Server	Returned	Result
1	TRUE	192.168.49.1	TRUE	Finish
2	TRUE	192.168.49.1	TRUE	Finish
3	TRUE	192.168.49.1	TRUE	Finish

4.4 Jade Programming Model

```
STEP-1 START
STEP-2 Profiler.
STEP-3 Program Profiling.
STEP-4 Device Profiling.
STEP-5 Optimizer.
STEP-6 Communication Manager.
STEP-7 STOP
```

4.5 Benchmark results

Table 2: Benchmark result

Test	Baseline	Monet	Overhead
CPU	21043	20015	4:8 %
Memory	14201	13019	8:3 %
I/O	7334	6782	7:5 %
2D	325	311	4.00%
3D	2320	2302	0:8 %

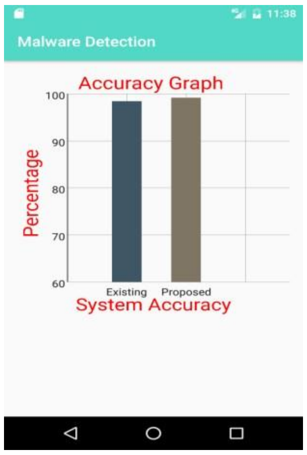
Output set:

$$y = [bn - anb0\,bn - 1 - an - 1b0..\,b1 - a1b0\begin{bmatrix} x1 \\ x2 \\ xn \end{bmatrix}] + b0u$$

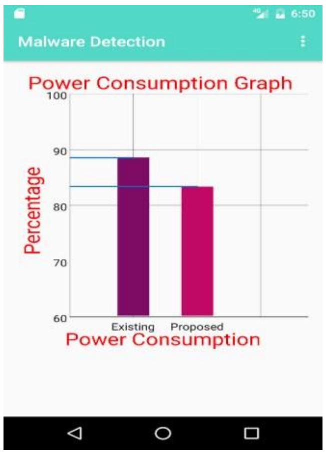
5. RESULT

Following graphs explain the excepted practical results for proposed work C-IBE. The practical work is designed and implemented using Java platform under real time cloud deployment settings.

5.1 Accuracy Graph



5.2. Power Consumption Graph



6. CONCLUSIONS

In this paper, we presented MONET to detect malware variants and to defend against transformation attack. MONET will generate a runtime behavior signature which consists of RBG and SSS to accurately represent the runtime behavior of a malware. Our system include a backend detection server and a client app which is easy to deploy on mobile devices. Our experiments show that MONET can accurately detect malware variants and defend against transformation attacks with only a minimal performance and battery overhead jade a system that enables computation offloading for mobile devices. Jade can effectively reduce energy consumption of mobile devices and dynamically change its. Offloading strategy according to device status. Face detection application and a path finding application. Results showed that jade can effectively reduce energy consumption for both application while better their performance.

REFERENCES

1. H. Qian and D. Andresen. Extending Mobile Device's Battery Life by Offloading Computation to Cloud. In Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2015.
2. H. Qian and D. Andresen. An Energy-saving Task Scheduler for Mobile Devices. In Proceedings of the 14th IEEE/ACIS International Conference on Computer and Information Science (ICIS), 2015.
3. H. Qian and D. Andresen. Emerald: Enhance Scientific Workflow Performance with Computation Offloading to the Cloud. In Proceedings of the 14th IEEE/ACIS International Conference on Computer and Information Science (ICIS), 2015.
4. Q. Chen, H. Qian et al. BAVC: Classifying Benign Atomicity Violations via Machine Learning. In Advanced Materials Research, Vols 765-767, pp. 1576-1580, Sep, 2013.
5. F. Wei, S. Roy and S. Ou. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vetting of Android Apps. In proceedings of the 2014 ACM Conference on Computer and Communications Security. 2014.
6. L. Peng, Y. Yang et al. Highly Accurate Video Object Identification Utilizing Hint Information. In proceedings of the International Conference on Computing, Networking and Communications (ICNC), 2014.
7. S. Zhang, X. Zhang and X. Ou. After We Knew It: Empirical Study and Modeling of Cost-effectiveness of Exploiting Prevalent Known Vulnerabilities Across IaaS Cloud. In proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS), 2014.
8. D. Arp, M. Spreitzenbarth, M. H ¨ ubner, H. Gascon, K. Rieck, andC. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in Prof. of the Network and Distributed System Security Symposium, 2014.
9. S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P.Ranganath, H. Li, and N. Guevara, "Experimental study with realworld data for android app security analysis using machine learning," in ACSAC. ACM, 2015.
10. K. Xu, Y. Li, and R. H. Deng, "Iccdetector: Icc-based malware detection on android," TIFS, 2016.
11. W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security." in USENIX Security, 2011.
12. L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in CCS, 2012.
13. S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," in PLDI, 2014.
14. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "Droidminer: Automated mining and characterization of finegrained malicious behaviors in android applications," in ESORICS, 2014.
15. W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information flow tracking system for realtime privacy monitoring on smartphones," Communications of the ACM, 2014.
16. M. Sun, T. Wei, and J. C. S. Lui, "Taintart: A practical multi-level information-flow tracking system for android runtime," in CCS, 2016.
17. L.-K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis." In USENIX Security, 2012.
18. Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in CCS, 2013.

19. K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, “Copperdroid: Automatic reconstruction of android malware behaviors.” in NDSS, 2015.
20. M. Lindorfer, M. Neugschwandtner, and C. Platzer, “Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis,” in COMPSAC, 2015.
21. S. Bugiel, S. Heuser, and A.-R. Sadeghi, “Flexible and fine-grained mandatory access control on android for diverse security and privacy policies.” in Usenix security, 2013.
22. C. Wu, Y. Zhou, K. Patel, Z. Liang, and X. Jiang, “Airbag: Boosting smartphone resistance to malware infection,” in NDSS, 2014.
23. X. Li, H. Hu, G. Bai, Y. Jia, Z. Liang, and P. Saxena, “Droidvault: A trusted data vault for android devices,” in ICECCS. IEEE, 2014.
24. X. Wang, K. Sun, Y. Wang, and J. Jing, “Deepdroid: Dynamically enforcing enterprise policy on android devices.” in NDSS, 2015.
25. M. Sun, J. C. S. Lui, and Y. Zhou, “Blender: Self-randomizing address space layout for android apps,” in RAID, 2016.