Android Vulnerability Analysis and Approach of Malware Detection

Hemesh Sawakar[1], Prof. Kiran K. Joshi[2]

[1]M. Tech Student, Dept of Computer Engineering and IT, VJTI, Mumbai, Maharashtra, India
[2]Assistant Professor, Dept of Computer Engineering and IT, VJTI, Mumbai, Maharashtra, India

---***---

Abstract – *Android, google's open source operating system of smartphone devices, tablets, TVs is the most popular one and widely used around the globe. Google play store hosting many android applications cannot check for all vulnerabilities in applications which leaves the risk of those being exploited by malwares. Most android users with excitement for sake of getting things done by using app do not bother to correlate the functions of app with app permissions or rather are unaware of how the user's data will be used by an app from given allowed permission. Also, on android devices before version Marshmallow If one wants to download an app then (s)he has to accept all permission even if some look unacceptable. Majority of android devices have newer operating system with enhanced security, and permission control which follows request on access need of component. Still many data leak incidents are reported which is a growing concern. There is no easy way than careful inspection and analysis from security experts and researchers to know if user's privacy sensitive information is being leaked or safe.*

In this paper, we are presenting some literature work happened in this area so far which addresses such problems with their concepts, own approaches and techniques. Also from our understanding we are proposing generic architecture to term an app as data leaking, vulnerable/ malware infected.

***Key Word:* Android, vulnerability, malware, Information flow, repackaging, obfuscation, reverse engineering, sandboxing, static analysis, dynamic analysis**

1. INTRODUCTION

The next generation of open operating systems won't be on desktops or mainframes but on the small mobile devices we carry every day [5]. As said we see it happening. Many new android versions are released every year by google. The openness of these new environments is leading to new applications on play store that has enabled greater integration. Android smartphones come with stock androids which have native apps pre-installed. Apart from that user can download third part apps available on playstore by accepting permissions to use data such as contact information, phone status and identity, pictures, audios, videos, accounts, location. There was no partial permission acceptance concept to download an app, earlier. Either accept all and install; or do not, even If one permission is unacceptable. With enhancements it changed but it is user who are most of the times do not know or fail to realise relation between app functionality and asked permissions. Some of such permissions leak user private data. It can be sold to companies collecting data or advertising sites. It is a growing concern since data in today's time is one of the valuable asset which should not be getting used for spamming, or harming one's reputation.

Leaking user data can be done by changing information flow control by injecting malwares in android applications. While playstore applications considered as malware free, but there can be some apps which are not hosted on playstore provide unique features; or some with same basic functionality as their official playstore counterpart but some additional functionality which attracts user to download them. applications developers of such applications can have malice intents. In some countries playstore is not available due to laws and regulations. There are several vulnerabilities found in android OS which can lead to exploitations. Common vulnerabilities and exposure details can be found on CVE [11] site with CVE ID, type, CVSS score, access type and complexity. As per their data, results are as shown in following charts.

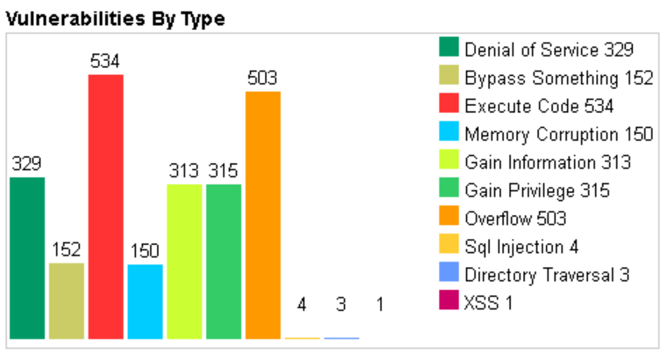

Chart 1: Type and count of vulnerabilities

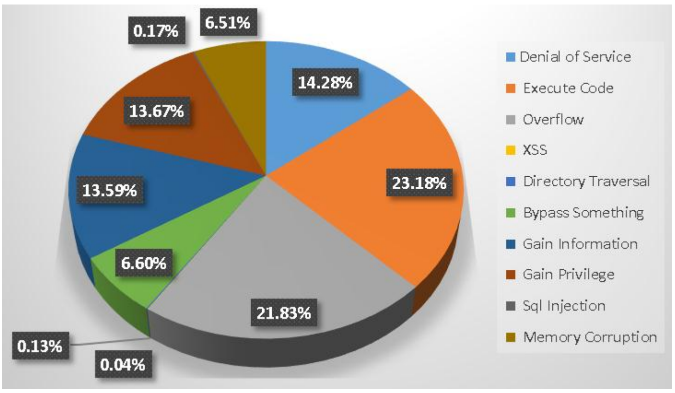

Chart 2: Type and % count of vulnerabilities

Out of these, one or more vulnerabilities can lead to leakage of user sensitive information by changing information flow. Benign applications can be injected with malwares to steal the information by doing modification in source code of application and intelligently hiding malware by making malware code unintelligible by a process called obfuscation. Hence, it is necessary to perform analyses of android applications to prove their benignity.

2. LITERATURE REVIEW

Theoretically, app stores have the potential to screen the published apps which contain unsafe code, bug or malware. However, practically they fail to run thorough checks to detect information disclosure and dangerous misuse of permissions and Android APIs. No app store performs runtime tests, nor do they exclude apps signed with the same key corresponding to different developer [8]. An automated full proof mechanism is necessary which can screen apps awaiting publish. Apps on google playstore are not approved manually, unlike apple store. Google play protect has its malware scanners which to certain extent perform some levels of security screenings. But they are not enough. Harmful apps may bypass it and get published. There is need of constant checking of published apps for hidden malwares and data breach. Currently, it is being done manually by researchers, security experts, organizations and labs working in cyber security areas. various kinds of security analyses techniques are performed in this mainly, static or dynamic code analysis.

2.1 Information Flow Control

Information flow Control is an essential and precise mechanism for tracking of information throughout a system such that information is handled safely and with regard to security triads. There are usually Implicit flows and explicit flows of information. Explicit flow is when there happens direct transfer, copying data or rather in terms of sensitive information we call transfer of secret between variables of different security labels. This is more of a data flow. In implicit flow there are certain conditional statements such as if-else. Condition results in change of control flow which in turn decides data flow. In android, flow analysis greatly differs from conventional control flow and data flow graphs. Flow graphs are fragmented in android ecosystem. The reason behind is component-based nature of android apps. Inter component communication (ICC) between independent components such as activities, services, content providers through intent passing allows arbitrary ordered execution of components depending on user interactions and order of system events taking place [9].

Control flow and data flow analysis are the methods of formal static analysis. Static analysis is analysis performed on still code which is not under execution. It provides the large coverage of flows. But in real setting all of those flows might not occur when code is under execution. In dynamic analysis, apps are executed/emulated in a sandboxed environment, in order to keep monitoring their activities via generated log information and identify anomalous behaviours, that are difficult with static analysis otherwise [1].

2.2 Repackaging

Repackaging is a process of rebuilding an android applications package file (.apk) file, signing it using tool such as keytool ,jarsigner. and optimising it using obfuscation tools. To do this all, one needs to decompile android applications package file (.apk) using tool such as apktool. and then reverse engineer decompiled file to obtain java source code using tools such as dex2jar. At this part, malware code can be inserted in source code using malware injection techniques and a new apk can be obtained using repackaging. The developers of DVmap, the first android malware with code injection; injected malicious Trojan code in system libraries at run-time. They bypassed the playstore security checks by uploading clean app on playstore. Then updated it with malicious version five times over period of month. At the end of the same day of upload, they would take down malicious version and reupload clean one. App was downloaded more than 50,000 times [12].

2.3 Obfuscation

Obfuscation is an intentional process of making source code hard to understand. It is used to protect code/algorithm from reverse engineering. Malware developers use malware obfuscation method to help malicious code go unnoticed. There are many tools such as ProGuard and DexGuard which can obfuscate and optimise code.

2.3 Application Sandboxing

Application Sandboxing is process of separating the application from rest and run it in a custom emulated protected environment. It is also known as containerization. It is an isolation used for protecting benign part of code, application from suspicious or dangerous application or code. [10] SELinux is a sandbox used in android. Potentially harmful, suspicious looking part of code is tested in sandboxed environment.

2.3 Analyses methods

Static analysis tools analyse the decompiled application code structure and they are faster since they analyse the code without running it. It analyses the code structure, instructions, information flow and generates Control Flow Graph (CFG) which can be used to determine valid flow.
One such tool is [3] Amandroid which generates point to point analysis of components and builds highly accurate inter- procedural control flow graphs (ICFG), which are both

flow and context sensitive. Using every ICFG, it is also capable of building each component's data flow graph (DFG) and data dependence graph (DDG) from DFG.

Steven artz designed [4] Flowdroid is another tool. It statically analyses and computes data flow in android apps. It is based on IFDS framework, creates dummy main method and perform static taint analysis based on selected flow algorithms, by identifying sources and sinks. It reduces the code to intermediate graph representation to efficiently track data flows by modelling each android component's life cycles.

Dynamic analysis tools analyse the applications by running them in a sandboxed environment. They are slow in execution compared to static analysis tools. But, they are better at dealing with false positives in which static analysis tools fail. It has a concept of taint analysis. Taint is a label tag given to privacy sensitive information. So such analysis is also known as 'dynamic taint analysis' or 'taint tracking'.

At the taint source Sensitive information is first identified where it is assigned an information type. This is called as marking a taint. In Dynamic taint analysis information flows are tracked for how labelled data is tainting other data in a way called 'taint propagation' tracked at file, message method, variable, message granularity levels that might leak the original sensitive information. Often, at machine level instructions taint tracking is performed. At last, data leaving out of system is identified as leak at taint sink. (more often times, network interface) [2].

Some of the tools are DexMonitor and TaintDroid.

[6] DexMonitor is a proposed prototype, used for dynamically analysing and monitoring obfuscated android applications. It systematically analyses android bytecode.
[2] TaintDroid, an efficient, system-wide dynamic taint tracking and analysis system capable of simultaneously tracking multiple sources of sensitive data with contexts. TaintDroid last release was long ago for android 4.3 version, thereafter no updates have been released. TaintDroid enabled real-time analysis by leveraging Android's virtualized execution environment which was DVM back then. It was replaced completely by Android Runtime (ART) in subsequent release of android versions. TaintDroid successfully found major leaks in top android apps back then with only 32% incurred performance overhead on CPU-bound micro benchmark and negligible overhead on third-party applications.

Other than this there is all in one framework, mobile security framework (MobSF) which is a pen testing, security assessment, malware analysing tool capable of performing both static and dynamic analysis. Apk file when uploaded is scanned for vulnerabilities in code and files then a detailed report is generated along with vulnerability indicators and scores. Dynamic analysis module of mobsf is used for live behaviour analysis using penetration testing scripts such as Frida.

3. PROPOSED SYSTEM

We propose a general system architecture as shown in figure.

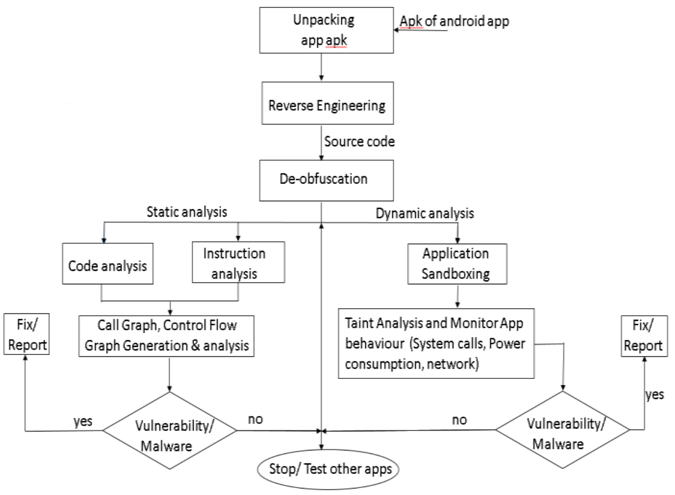

Fig. 1: Analysis process for Vulnerability or Malware detection

Android applications which are to be tested for benignity are unpacked from apk file using decompilation tool. The source code is can be obtained using reverse engineering tools. Once source code is available it is de-obfuscated to obtain redundancy free code. The code undergoes through static and dynamic analysis phases. Static analysis tools will analyse the code for its structure, meta data, instructions and generate control glow graphs (CFG) and analyse information flow. If vulnerabilities or malwares are found they will be reported and fixed. Similar procedures will take place in dynamic analysis but with difference that application code will be executed in sandboxed environment. Taint analysis tools will monitor the information flow from taint source to taint sink. The app behaviour with respect to system calls, power consumption rate, network logs will also be monitored to trace the signs of possible malwares.

few tools are stated in our study however there are many tools which can be used for analyses. Each have their unique characteristic and speciality discussed in [7] very detailed. Malware detection techniques can be performed to declare the type of detected malware. Techniques for anomaly based and signature based malware detection can be applied further.

4. CONCLUSION

In this paper, we discussed on popularity of android OS and growing security concerns about leakage of user privacy sensitive information by vulnerable android apps. Then past survey literature knowledge is presented which consists of important techniques of application decompilation, reverse engineering, execution in protected environment, code obscureness, analysis methods, tools and frameworks.

We proposed a general architecture based on what we understood from study of our reference literature. Using all these studied techniques we will unwrap the application apk files for in detail metadata and code analysis using static and dynamic analysis methods. We are going to generate call and control flow graphs and trace every possible path for data leak and malware detection. It is an approach for securing user data from suspicious data privacy breaching apps by tracking information flow control and bringing evidences in light so that users can be aware while using such applications.

The proposed architecture will get more refined as one starts with implementation. Furthermore, research can be performed on how could the vulnerabilities and malwares be mitigated. The use of machine learning along with hybrid analysis tools is suggested which take up best of every analysis tool to make analyses faster, efficient and precise.

REFERENCES

[1] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, Senior Member, IEEE, and Muttukrishnan Rajarajan "Android Security: A Survey of Issues, Malware Penetration, and Defenses" IEEE communication surveys & tutorials, vol. 17, no. 2, second quarter 2015

[2] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-gon chun, Landon P. Cox, Jaeyeon Jung, Patrick Mcdaniel, Anmol N. Sheth "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones" ACM Transactions on Computer Systems, Vol. 32, No. 2, Article 5, Publication date: June 2014.

[3] Fengguo Fei, Sankardas Roy, Xinming Ou, Robby "Amandroid: A Precise and General Inter-Component Data Flow Analysis Framework for Security Vetting of Android Apps" ACM Transactions on Privacy and Security, Vol. 21, No. 3, Article 14. Publication date: April 2018.

[4] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Damien Octeau, Patrick McDaniel, and Yves Le Traon, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps" Department of Computer Science and Engineering, 2014.

[5] W. Enck, M. Ongtang and P. McDaniel, "Understanding Android Security," in IEEE Security & Privacy, vol. 7, no. 1, pp. 50-57, Jan.-Feb. 2009, doi: 10.1109/MSP.2009.26.

[6] Haehyun cho, Jeong Hyun Yi, (member, ieee) and Gail-joon Ahn(Senior Member, IEEE) "DexMonitor: Dynamically Analyzing and Monitoring Obfuscated Android Applications" volume 6,2018 IEEE Access.

[7] Rashidi, Bahman and Carol J. Fung. "A Survey of Android Security Threats and Defenses." J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl. 6 (2015): 3-35.

[8] Y. Acar, M. Backes, S. Bugiel, S. Fahl, P. McDaniel and M. Smith, "SoK: Lessons Learned from Android Security Research for Appified Software Platforms," 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, 2016, pp. 433-451, doi: 10.1109/SP.2016.33.

[9] kimberly tam, ali feizollah, nor badrul anuar, and rosli salleh, lorenzo cavallaro "The Evolution of Android Malware and Android Analysis Techniques" ACM Computing Surveys, Vol. 49, No. 4, Article 76, Publication date: January 2017.

[10] A. N. Narvekar and K. K. Joshi, "Security sandbox model for modern web environment," 2017 International Conference on Nascent Technologies in Engineering (ICNTE), Navi Mumbai, 2017, pp. 1-6, doi: 10.1109/ICNTE.2017.7947885.

[11] https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

[12] https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/