Android Application Security

[1]Gulista Tabassum, [2]Shikha Kumari, [3]Nupur Ghosh
Dronacharya College of Engineering,Greater Noida,utter Pradesh,India

Abstract : As the smart phone craze spreads through the modern world,it is becoming crucial to reason about the flow of sensitive data on the mobile platform.Some work has already been done on the Android security model,including several analysis of the model and framework aimed at enforcing security standards.In this paper we use this work as a starting point as we synthesize a detailed model of the information flow within the android platform and present two detailed case studies.

Keywords*:* MOS - Mobile operating system, Android-OS, App Security,Application retargeting,Optimization.

1. INTRODUCTION

Android is an MOS based on the Linux kernel with a user interface, mainly focused touch screen mobile devices such as Smartphone and tablet computers, since Android's source code is provided by Google under open source licence so various Mobile manufacturing companies like Samsung, HCL, etc which uses it as their MOS. IOS is a MOS developed by Apple Inc. and it is available only on Apple hardware such as iPhone, iPad, iPod Touch, and Apple TV. There are a lot of applications which are developed for both of these MOS. Current number of Android apps in the market are around:1,235,976[23] and for IOS are around 1,567,359[24]. While using these Apps on our mobile device or tablets or IPad, we should be careful about security of our data and personal information stored on the devices. The most commonly spread malware on Android is one where text messages are sent to some unknown numbers without the awareness of the user, and the sending of personal information to unauthorized third parties. When we talk about security in mobile Apps, we should take care of these facts like Root kit Detectors, Process isolation, file and resource permissions and memory protection.

Root kit are malwares which actually provide administrator rights to the attacker of the Mobile. Using Root kit detectors we can detect these malwares. Process isolation is a technology in MOS by which Apps has their own area of execution they will not interfere each other until some authenticated communication is provided. File and resource permissioncontrol is provided by every MOS so that we can achieve secure Inter process communication. Memory protection is achieved through Address Space Layout Randomisation (ASLR).

In this paper we are comparing security of Android-OS and IOS adopted by developers, App stores and end-users. Apps attacks can include following areas like, Attack on sensitive data of application, weak or no encryption during
transmission of data, unauthorised access to administration interfaces or data stores. Runtime inoculation – This allows an attacker to manipulate and abuse the runtime of an application to bypass security locks, bypass logic checks, access privileged parts of an application, and even steal data stored in memory. Escalated privileges –Exploits a bug, design flaw or configuration oversight in order to gain access to resources normally protected from an application or user[1]

2. SECURITY AT APP DEVELOPMENT TIME

As we know that no application is perfect. There are many bugs in Application, some are caused during design phase of application, some are occurred due to implementation mistakes done by developers. Due to these bugs user data is compromised and sometimes arbitrary binary data is executed by the application injected by the attackers. In Android it is the responsibility of the developer to isolate application from other system resources through application sandboxing. For Android, the application sandboxing is based on the Linux kernel platform. It is a complex and robust sandbox model. Application sandboxing in Android is controlled by each application and required permission and approval to continue accessing what the application needed. This will improve and build the security tighter. Each application has its own sandbox directory and the permission is per App.

3. SECURITY AT PUBLISHING APP IN APP STORE

Publishing means that once our application is completely finished and tested. Now is the time to distribute it to the users to use it. The most common ways to publish an app on Android or IOS is using their corresponding App store like Google Play and Apple store.

Before publishing every App must be signed. Signing is a process in which App is assigned a key. This is done with asymmetric encryption algorithm. There are two types of keys one is Debug and one is Release. The process of publishing an Android App on the Google Play store, the developer needs access to the Developer Console. The developer console is a web based set of tools that allows the developer to publish and monitor their apps. Android has a flaw that Google App store is not a single place where application can be released. User can choose or install an App from a website or email by modifying their mobile settings to install the App from unknown source. When building a debug version of the app, the signing is done automatically with a debug key. The debug key does not provide any security of verification. Developer Console does not allow developers to upload apps that are signed with them. Instead, the developer is required to create their own key pair and certificate.

4. Application Retargeting

The initial stage of decompilation retargets the application.dex file to Java classes. Figure 3 overviews this process: (1) recovering typing information, (2) translating the constant pool, and (3) retargeting the byte code.Type Inference: The first step in retargeting is to identify class and method constants and variables. However, the Dalvik bytecode does not always provide enough information to determine the type of a variable or constant from its register declaration. There are two generalized cases where variable types are ambiguous: 1) constant and variable declaration only specifies the variable width (e.g., 32 or 64 bits), but not whether it is a float, integer, or null reference; and 2) comparison operators do not distinguish between integer and object reference comparison(i.e., null reference checks).Type inference has been widely studied [44]. The seminal Hindley-Milner [33] algorithm provides the basis for type inference algorithms used by many languages such as Haskell and ML. These approaches determine unknown types by observing how variables are used in operations with known type operands. Similar techniques are used by languages with strong type inference, e.g., OCAML, as well weaker inference, e.g., Perl.

5. Optimization and Decompilation

At this stage, the retargeted .class files can be decompiled using existing tools, e.g., Fernflower [1] or Soot [45]. However, ded's bytecode translation process yields unoptimized Java code. For example, Java tools often optimize out unnecessary assignments to the local variable table, e.g., unneeded return values. Without optimization,decompiled code is complex and frustrates analysis. Furthermore, artifacts of the retargeting process can lead to decompilation errors in some decompilers.The need for bytecode optimization is easily demonstrated by considering decompiled loops. Most decompilers convert for loops into infinite loops with break instructions. While the resulting source code is functionally equivalent to the original, it is significantly more difficult to understand and analyze, especially for nested loops. Thus, we use Soot as a post-retargeting optimizer.

While Soot is centrally an optimization tool with the ability to recover source code in most cases, it does not process certain legal program idioms (bytecode structures) generated by ded. In particular, we encountered two central problems involving, 1) interactions between synchronized blocks and exception handling, and 2) complex control flows caused by break statements. While the Java bytecode generated by ded is legal, the source code failure rate reported in the following section is almost entirely due to Soot's inability to extract source code from these two cases. We will consider other decompilers in future work, e.g., Jad [4], JD [3], and Fernflower [1].

6. CONCLUSION

In light of our foray into the realm of Android Security, we are convinced that in order to analyze applications effectively automated analysis is necesssary. This automated analysis needs tackle implicit and explicit flows to provide the user with guarantees about the security of their private data.The smart phone platform presents some unique challenges for information flow tracking. For one, these platforms have less computing power and are much more sensitive to high overhead costs.Additionally, smartphones and their interactions with applications
present different types of sensitive information to consider.These sources of information need to be treated differently and any sort of automated analysis would need to account for this.Further, because applications can share information, information flow analysis cannot be done at the application level. It must be more fine grained, however, the more fine grained it gets the higher the risk of a high overhead cost.

REFFERENCES

[1] Android security overview, . URL http://source.android.com/tech/security/index.html.
[2] Security and permissions, . URL http://developer.android.com/guide/topics/security/security.html.
[3] Almost half of u.s. smartphones running android, . URL http://code.google.com/p/dex2jar/.
[4] android4me, . URL http://code.google.com/p/android4me/.
[5] In-stat: Majority in u.s. to have smartphones,URL http://news.cnet.com/8301-13506_3-20095949-17/in-stat-majority-in-u.s-to-have-smartphones-tablets-by-2015/.
[6] Almost half of u.s. smartphones running android, .URLhttp://www.pcmag.com/article2/0,2817,2401250,00.asp.
[7] 1 in 5 android apps pose potential privacy threat,. URL http://mashable.com/2010/06/23/android-apps-privacy-threat/.
[8] J. Burns. Moblie application security on android. Black Hat USA,2009,URl http://www.blackhat.com/presentations/bh-usa-09/BURNS/BHUSA09-Burns-AndroidSurgery-PAPER.pdf.
[9] 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. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.
URL http://dl.acm.org/citation.cfm?id=1924943. 1924971.
[10] W. Enck, D. Octeau, P. Mcdaniel, and S. Chaudhuri. A study of android application security. USENIX Security, (August):935–936, 2011. URL http://www.usenix.org/event/sec11/tech/slides/enck.pdf.
[11] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster,and T. Millstein. Dr. Android and Mr. Hide: Fine-grained security policies on unmodified Android. Technical Report CS-TR-5006, Department of Computer Science, University of Maryland, College Park,December 2011.
[12] Fernflower - java decompiler. http://www.reversed-java.com/fernflower/.
[13] Fortify 360 Source Code Analyzer (SCA). https://www.fortify.com/products/fortify360/source-code analyzer.html.
[14] Jad. http://www.kpdus.com/jad.html.
[15] Jd java decompiler. http://java.decompiler.free.fr/.
[16] Mocha, the java decompiler. http://www.brouhaha.com/~eric/software/mocha/.
[17] ADMOB. AdMob Android SDK: Installation Instructions.http://www.admob.com/docs/AdMob_Android_SDK_ Instructions.pdf. Accessed November 2010.
[18] ASHCRAFT, K., AND ENGLER, D. Using Programmer-Written Compiler Extensions to Catch Security Holes. In *Proceedings of the IEEE Symposium on Security and Privacy* (2002).
[19] BBC NEWS. New iPhone worm can act like botnet say experts. http://news.bbc.co.uk/2/hi/technology/ 8373739.stm, November 23, 2009.
[20] BORNSTEIN, D. Google i/o 2008 - dalvik virtual machine internals. http://www.youtube.com/watch?v=ptjedOZEXPM.
[21] BURNS, J. Developing Secure Mobile Applications for Android.iSEC Partners, October 2008. http://www.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf.
[22] CHEN, H., DEAN, D., AND WAGNER, D. Model Checking One Million Lines of C Code. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium* (Feb. 2004).
[23] EGELE, M., KRUEGEL, C., KIRDA, E., AND VIGNA, G. PiOS:Detecting Privacy Leaks in iOS Applications. In *Proceedings of the Network and Distributed System Security Symposium* (2011).