Yucheng Liu, MSINFS, GaTech, yliu3003@gmail.com
Eby John, MSINFS, GaTech, ebyjohn@gatech.edu

I. Related Work

The popularity of Android and its open source nature makes it one of the most scrutinized piece of software. Security is of paramount importance in such a widely deployed mobile platform. Android platform is based on Linux and hence its security is largely guaranteed by the robust and publicly vetted Linux kernel. User based protection available on Linux is used to isolate applications by creating a kernel level application sandbox by assigning each application a different user id. Thus to break out of this sandbox would require the application to find a vulnerability in the Linux kernel itself [1]. An example of this is the recent Exynos kernel vulnerability which affected Samsung android phones like Galaxy S2, S3 and Note 2 and let a malicious application complete access to physical memory [2]. Android provides a set of protected API's which support functions related to Camera, Location data, Network connectivity and other sensitive resources which cannot be directly accessed by the application. The application must explicitly request permission to access these resources during install time [1]. Apart from these mechanisms Android provides component encapsulation by allowing application components like Activities and Services to be declared as either public or private. Android also provides facility for application signing [3].

The fluidity of application markets complicates smartphone security. Low barriers of entry for application developers increases the security risk for end user [4]. Applications are developed and deployed very quickly which can have an adverse effect on security [5]. The lack of fine grained security permissions go against the principle of least privilege [5]. A standard attack vector on Android is Inter Component Communication. A lesser privileged application is not prevented from accessing the components of a more privileged application and the lack of such a transitive permission usage check leads to vulnerabilities [3][6][7]. The WebView component of Android is also vulnerable and weakens the trusted computing base at the client side and the browser [8].

In order to build a standard for researchers and testers to certify whether an application is secure, OWASP started its own Mobile Security project. The Top 10 Risks of Mobile Security [9], ordered by their probability and severity, is a fundamental work which attempts to increase security awareness among developers and testers alike.

While there is no systematic work about analyzing popular Android applications to see if they correctly handle the security risks as addressed by the OWASP top 10, some works have partially covered it in their analysis.

- **Insecure Data Storage:** Data stored on on external storage is globally readable and writable [10]. Not encrypting sensitive data stored on the mobile is a bad security practice when we consider the threat model where the mobile is stolen [11].
- **Weak Server Side Controls:** Even though server side controls are not under the scope of this project it is very important that these controls do not trust the client implicitly and validate all incoming data before acting on them [12].
- **Insufficient Transport Layer Protection:** Fahl et al. [13] built a tool (MalloDroid) to detect applications that potentially use SSL/TLS incorrectly. They manually mounted man-in-the-middle (MITM) attacks against some applications containing HTTPS URLs and successfully captured sensitive credentials. At the other end of the spectrum many applications completely disregard SSL and leak sensitive data through the use of HTTP.
- **Client Side Injection:** SQL injection is possible in Content Providers when the application is backed by a SQLite db [14][15][16]. It is also possible to execute malicious javascript code via vulnerabilities in the WebView component [8].
- **Poor Authorization and Authentication:** Some applications rely on potentially immutable values like IMEI, IMSI, UUID or other hardware identifiers which could be compromised to authenticate applications [12][17].
- **Improper Session Handling:** Some applications use SSL only for authentication. Further communication including session tokens and data are sent in the clear. Using device identifiers as session tokens is a bad practice [12][18].
- **Security Decisions Via Untrusted Inputs:** It is possible for malicious applications to spoof intents incase the components are public and do not require the sender to have strong permissions. This causes data injection and it is possible for the malicious application to control the flow of the program [7][14].
- **Side Channel Data Leakage:** Cryptographic processing on a mobile processor was shown to be vulnerable to EM attacks [19].
- **Broken Cryptography:** Applications use unsafe cryptographic keys (including insufficient key size) and algorithms [4].
- **Sensitive Information Disclosure:** Enck et al. [20] described an extension to Android platform named TaintDroid to allow users to control sensitive information and also studied 30 popular applications which revealed that two-thirds exhibit suspicious handling of sensitive data. Similar work has been done by Enck et al. [4] addressed that phone identifiers like IMEI are used by applications to track individual users. Applications which report location might be doing so at a might higher granularity than what the user expects.

There are different techniques to analyze the security of Android applictions. One of the main methods to test for vulnerabilities in an application is static analysis of the code. ITS4 was one of the earliest static code analyzers written in the early 2000s and since then great strides have been made in this front, making this form of analysis a standard part of code vulnerability analysis [21]. The first step in static analysis is decompiling the bytecode file (DEX file in Android) with tools such as dex2jar, Kivlad and etc. Enck et al [4] used ded decompiler to scan 1100 popular free Android apps and checked for dangerous functionality in the context of the application. Another testing tool from the SIIS Laboratory named Dare(Dalvik Retargeting) further improved ded and provided a way for android application certification [22]. Permission creep on Android is increasing as more applications request much more permissions than they need for it's proper functioning[23]. By checking mismatch between required and requested permissions some inference can be made about the malicious nature of the application.

There are several existing tools/methods which will help with the security analysis. Some of them are DED/DARE decompiler [4][22], FindBugs [24]- A static analysis tool to detect common software defects, Privacy Oracle [25]- A system that reports user information leakage via data sinks like networks using differential testing, Comdroid[7]- A tool to detect application communication vulnerabilities, SCanDroid [26]- A tool to extract security permissions from the application's manifest file and check if the data flow for the application is consistent with the security requested.

References

[1] GOOGLE INC. *System and Kernel Level Security*. http://source.android.com/tech/security/system-and-kernel-level-security.

[2] Smith, G. Samsung: Exynos 4 processors are vulnerable to serious attack. http://www.techrepublic.com/blog/smartphones/samsung-exynos-4-processors-are-vulnerable-to-serious-attack/6134.

[3] Davi, L., Dmitrienko, A., Sadeghi, A., Winandy, M. Privilege Escalation Attacks on Android, 2011

[4] Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S. A Study of Android Application Security. In *Networking and Security Research Center Technical Report*, 2011

[5] Raphael, J. Google: Android wallpaper apps were not security threats. http://blogs.computerworld.com/16666/google_android_wallpaper_apps, Aug, 2010.

[6] Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., Sadeghi, A. XManDroid: A New Android Evolution to Mitigate Privilege Escalation Attacks. In *CASED Technical Report*, 2011.

[7] Chin, E., Felt, A., Greenwood, K., Wagner, D. Analyzing Inter-Application Communication in Android. In *MobiSys'11*, Jun-July, 2011.

[8] Luo, T., Hao, H., Du, W., Wang, Y., Yin, H. Attacks on WebView in the Android System. In *ACSAC'11*, Dec, 2011.

[9] OWASP. OWASP Mobile Security Project: Top Ten Mobile Risks. https://www.owasp.org/index.php/Mobile, Sep, 2011.

[10] Google Inc. Security Tips of Android. http://developer.android.com/training/articles/security-tips.html.

[11] jhaddix. Exploring The OWASP Mobile Top 10: M1 Insecure Data Storage. http://h30499.www3.hp.com/t5/Application-Security-Fortify-on/Exploring-The-OWASP-Mobile-Top-10-M1-Insecure-Data-Storage/ba-p/5904609. Dec, 2012.

[12] Mannino, J., Zusman, M., Lanier, z. OWASP Top 10 Mobile Risks. In *Appsec USA*. Sep, 2011. http://2011.appsecusa.org/p/mobiletopten.pptx

[13] Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgartner, L., Freisleben, B. Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security. In *CCS'12*. Oct, 2012.

[14] O'Neil, Y., Chin, E. Seven Ways to Hang Yourself with Google Android. http://www.cs.berkeley.edu/~emc/slides/SevenWaysToHangYourselfWithGoogleAndroid.pdf.

[15] Nils. The Risk You Carry in Your Pocket. In Black Hat, Abu Dhabi 2010. http://media.blackhat.com/bh-ad-10/Nils/Black-Hat-AD-2010-android-sandcastle-slides.pdf.

[16] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S. Google Android: A State-of-the-Art Review of Security Mechanisms. http://arxiv.org/ftp/arxiv/papers/0912/0912.5101.pdf.

[17] Schrittwieser, S., Fruhwirt, P., Kieseberg, P., Leithner, M., Mulazzani, M., Huber, M., Weippl, E. Guess Whos Texting You? Evaluating the Security of Smartphone Messaging Applications. http://www.internetsociety.org/sites/default/files/07_1.pdf

[18] Sawyer, J., Eston, Tom., Johnson, Kevin. Smart Bombs Mobile Vulnerability and Exploitation. In *ASDC'12*.

[19] Kenworthy, G., Rohatgi, P. Mobile Device Security: The case for side channel resistance. In *Mobile Security Technologies(MoST)*, 2012.

[20] Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P., Sheth, A. TaintDroid An Information-Flow Tracking System for Realtime Privacy. In *OSDI' 10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*.

[21] Chess, B., McGraw, G. Static Analysis for Security. In *IEEE Computer Society*, 2004.

[22] Systems and Internet Infrastructure Security Lab, Penn State University. Dare, Dalvik Retargeting. http://siis.cse.psu.edu/dare/index.html#banner.

[23] Vidas, T., Christin, N., Cranor, L. Curbing Android Permission Creep. https://www.andrew.cmu.edu/user/nicolasc/publications/VCC-W2SP11.pdf.

[24] Ayewah, N., Pugh, W., Morgenthaler, J., Penix, John., Zhou, Y. Evaluating Static Analysis Defect Warnings On Production Software. In *PASTE'07*, Jun, 2007.

[25] Jung, J., Sheth, A., Greenstein, B., Wetherall, D., Maganis, G., Kohno, T. Privacy Oracle: a System for Finding Application Leaks with Black Box Differential Testing. In *CCS '08 Proceedings of the 15th ACM conference on Computer and Communication Security*, 2008.

[26] Fuchs, A., Chaudhuri, A., Foster, J. SCanDroid: Automated Security Certi?cation of Android Applications. http://www.cs.umd.edu/~avik/papers/scandroidascaa.pdf.