Secure Copy Protection for Mobile Apps

Nils T. Kannengiesser
Technical University of Munich
Chair for Operating Systems (F13)
Munich, Germany
Nils.Kannengiesser@tum.de

Uwe Baumgarten
Technical University of Munich
Chair for Operating Systems (F13)
Munich, Germany
baumgaru@tum.de

Sejun Song
University of Missouri-Kansas City
Computing and Engineering
Kansas City, USA
sjsong@umkc.edu

***Abstract* — Copy protection for Android apps exists since the early days, but even today the existing solutions like Google's License Verification Library or Amazon's DRM solutions are proven to be insecure according to our recent analyses. In this paper we suggest to use secure elements to improve the overall security and to separate confidential data from the insecurity of the Android OS. The content is taken from our ongoing research.**

***Keywords* — Apps, Android, Copy Protection, Secure Element**

I. INTRODUCTION

“When Google’s first smartphone, the Nexus One, hit the market in early 2010, nobody could have known whether it could effectively compete with existing smartphones”[15]. Nowadays Android’s market share is estimated by 85% [1], and most developers create apps for Android these days, too. “In comparison to the development of apps for other platforms, it’s relatively easy to create and upload an app to Google’s App Market – Google Play.” [15] Instead iOS developers’ apps have to pass severe testing by the AppStore team and publishing an app can take several weeks. In 2010 [2] “Google released the LVL[1] […] to satisfy the needs for basic software protection. Nevertheless this release was immediately cracked [3]. A major issue with the protection was the reengineering possibility to change one line of code only, and get an unlicensed app working [3].” [15] In random app evaluations, we found that developers are starting to use obfuscation tools nowadays (e.g. ProGuard[2]) and many major apps are protected now. Nevertheless these apps can still “be processed by reengineering tools (e.g. APKtool[3]), which produce smali[4] or java output.”[15] The resulting code is much harder to read, but it is still just a matter of time and attackers may look for known patterns. Even the “LVL has been updated since its initial failure,” [15] most implementations are still easy to crack. In a recent master’s thesis one of our students stated “the LVL is very popular yet very much broken”[10].

For instance “Root users can access any part of a phone, decompile apps as they like and remove or disable security features. Some manufacturers (e.g. Google) allow rooting by default. On other phones, there are multiple ways that allow users to gain root access, too. The timeline for root exploits (Figure 1) shows that every Android version is affected after only a few months on the market.” [15] There have been root exploits in recent times again, too. For instance the “Towelroot” method by George Hotz [11]. In summary we found out that none of the existing license verification methods (cf. Google’s LVL nor Amazon’s Appstore DRM) is really secure [10][16]. For increasing the security one option could be to use secure elements to store sensitive data in it. For instance data examples deserving protection are encryption keys or links to generate the corresponding OTLs[5] [15].

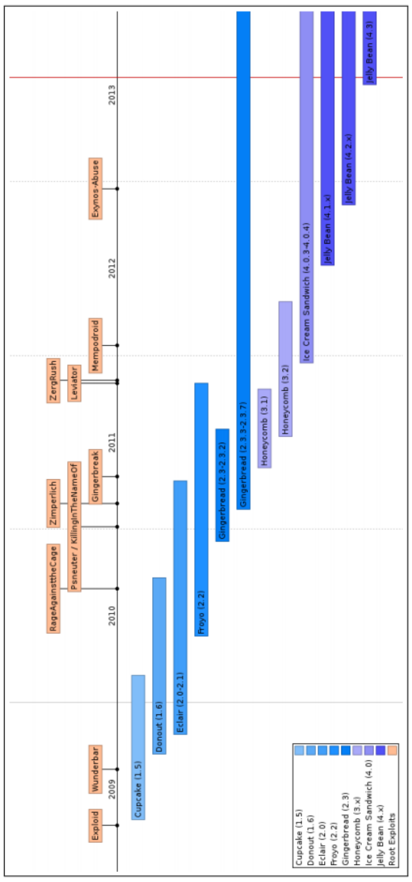

Fig. 1. “Timeline of Root Exploits” (till 2013) [4][15]

[1] License Verification Library
[2] Obfuscation Tool - http://proguard.sourceforge.net/
[3] Reengineering Tool - https://code.google.com/p/android-apktool/
[4] Smali = Assembly language (output of APKtool)

[5] OTL = One Time Link

II. Foundations

“Google began to worry about security early in the development of Android [5]. Android, itself, enforces security by separating apps to run as isolated processes with their unique user- and group-ids. The access to sensitive hardware is controlled by predefined permissions that need to be accepted by a user during the initial installation phase [5].” [15]

“To support the security of commercial apps Google released the License Verification Library (LVL) in 2010 [2]. This library provides developers with the ability to integrate license checks in their apps.” [15] Figure 2 shows its implementation:

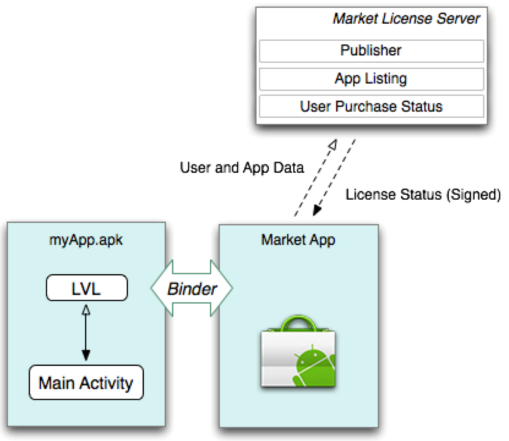

Fig. 2. “Licence Verification Library” [6] [15]

“Most of the later piracy issues arising from using the License Verification Library came from how developers used the library.” [15] Many developers intend to copy unchanged LVL packages into their apps “without using any obfuscation tools”. [15] Even Google tries to encourage developers to modify everything (cf. “the security […] ultimately relies on the design of your implementation” and “actual enforcement and handling of the license [...] are up to you” [6]), many applications include the default, unmodified framework [15]. Even the LVL uses replies that are signed nowadays [6], our latest research shows that there are still loopholes to allow successful in-memory attacks [12].

“A possible way to solve this security issue”[15] is “the usage of a so called *Secure Element*. Such an element provides a secure space that is separated from the smartphone [8] and its vulnerabilities (cf. Android exploits, rooting, malware like trojans etc.). For instance, Google is using it as part of their Google wallet application [7]. Secure elements are available in form of UICCs[6], commonly known as SIM cards, as an external flash memory card or even already embedded in the hardware of the phone itself [7]. One of the manufactures for external secure elements in form of memory cards is Giesecke and Devrient. Their product, the MSC[7], is used in our research.”[15]

Since modern phones often come without a slot for SD cards, it was required to look for a different access option. We were able to identify a possible adapter, which allows to connect the MSC (with its embedded SE) using the provided micro USB port on most devices.

Fortunately “there is a non-standard possibility to access the SE over special reads and writes to the file system […] [It] requires that the card is accessible through the hosts file system or on a block-level without caching in between (O_DIRECT required).” [13]

A major problem on most modern Android versions is that they do not support the unbuffered access (O_DIRECT flag) anymore (see [14]) and the own written commands are read back instead of the actual SE’s response. This has been a major issue for the industry for now and current solutions require a rooted and modified device [13] to mount an external storage device (e.g. the MSC) and to add the unbuffered access (kernel change) again.

III. Proposals

Last year we proposed various ideas at the AmiEs conference, including the identification of users and devices, the exchange of information in a secure manner, extended content protection as well as an obfuscation of the execution [15]. Most of these ideas require a secure element. Therefore we focused on solving the existing issues on secure elements first.

“Because it has been relatively easy for users to gain root access to many different smartphones as well as the history of such root access exploitation for all kinds of Android versions and devices, it can be assumed that Android will very likely be insecure in the future [, too]. Therefore data, which shouldn’t be accessed by a user (c.f. license information), isn’t stored securely” [15] at the moment.

“For this reason we focus on design ideas that combine apps and secure elements. Besides issues involving the storage of critical data, we also face the problem of identifying a certified device or user for purposes of copy protection.”[15]

IV. Current Solutions

Due to the fact that modern devices do not have any SD card slots quite often, one of the requirements was to use a special SD card adapter[8] via the micro USB interface.

Since mounting an external storage device is not possible without root rights and due to the afore mentioned issue, the library “libaums”[9] was developed.

The library allows the unbuffered access to any USB storage device like USB Flash drives/hard drives and SD cards (connected via an adapter).

Early examinations combining the library with the MSC framework of Giesecke & Devrient reveal that the access to the

[6] Universal Integrated Circuit Card
[7] **M**obile **S**ecurity **C**ard by Giesecke & Devrient

[8] e.g. Dash Micro by MeeNova, http://www.meenova.com/
[9] “**Lib**rary to **a**ccess **U**SB **M**ass **S**torage devices”, M. Jahnen, https://github.com/mjdev/libaums

MSC's secure element is possible now. It is a workaround for the O_DIRECT issue (cf. [14]).

V. Related work

In the recent months there have been no major improvements and many solutions are still software- or cloud-based, and therefore vulnerable to software-based attacks or provide less mobility (cf. cloud solutions are not working in situations without reception).

"Most related work uses software-based optimizations (e.g. the kernel modifications in "SEAndroid"[10]), while other vendors try to establish security by proposing new hardware (e.g. TEE[11] by Trustonic)." [15] In June 2014 Trustonic and Thundersoft announced a partnership to bring their technique to the smartphone market soon [20].

"In terms of copy protection, research papers propose software related solutions. For instance, a paper by researchers of the Dankook and KonKuk University suggest using "Class Separation and Dynamic Loading for Android Applications" [9]. " [15] Another paper from 2013 suggests to use "fingerprinting […] for detecting illegal Apps" [17], which is similar to the approach of using "forensic marks" and a "self-checking library" [18]. Other researchers focus on analyzing and attacking currently used methods by "memory hacking" [19].

In summary the following methods can be identified "to prevent illegal execution of Android Apps" [19] for now: Protection based on/using

- licensing libraries (e.g. Google's LVL) [6]
- cryptography [19]
- "forensic marks" [18]
- "mandatory access control" [19]
- "Online Execution Class […] a technique that loads dynamically a part of the class of the entire App code from the server" [19]
- "Hybrid Design of Online Execution Class and Encryption-based Copyright Protection" [19]
- secure elements (our approach) [15]

VI. Conclusion

Our assumption in getting access to the MSC as well as its secure element by using the existing USB framework of Android was successful. It is a major step and strong requirement for the upcoming verification of all ideas, which were presented at AmiEs 2013 (see [15] for details).

"We are [still] assuming that the proposed method of using secure elements (MSC) is going to improve the overall protection against piracy." [15]

[10] Modified Android version by the NSA
[11] Trusted Execution Environment

VII. Future work and problems

The next step is to verify our ideas (see [15]) and analyze their usefulness in terms of copy protection to draw a final conclusion about the increased security.

A rising issue might be the performance of the secure elements. We are assuming that not all ideas can be implemented in the proposed way, since the typical reaction time for a command is about 110ms for even easy tasks.

Furthermore Android L is presumably already more secure, because of the usage of the ART runtime. It might become more difficult for attackers to reengineer apps in general. [10]

References

[1] IDC, "Smartphone OS Market Share, Q2 2014", http://www.idc.com/prodserv/smartphone-os-market-share.jsp, last access: 25th August 2014

[2] Red, "Kopierschutz von Android Market geknackt", http://derstandard.at/1282273487603/App-Piraterie-Kopierschutz-von-Android-Market-geknackt, last access: 26th August 2013

[3] Justin Case, "Google's Android Market License Verification Easily Corcumvented, Will Not Stop Pirates", http://www.androidpolice.com/2010/08/23/exclusive-report-googles-android-market-license-verification-easily-circumvented-will-not-stop-pirates/ , last access: 26th August 2013

[4] Janosch Maier, "Enhanced Android Security to prevent Privilege Escalation", p.16

[5] Google, "Android Security Overview", http://source.android.com/devices/tech/security/index.html, last access: 27th August 2013

[6] Google, "Licensing Overview", http://developer.android.com/google/play/licensing/overview.html , last access: 27th August 2013

[7] Thomas Zefferer, A-SIT, "Secure Elements am Beispiel von Google Wallet", http://www.a-sit.at/pdfs/Technologiebeobachtung/20120428%20Studie_Google_Wallet.pdf, p.1 , last access: 30th August 2013

[8] Josef Langer, Andreas Dyrer, "Secure Element Development", p.6, http://www.nfc-forum.org/events/oulu_spotlight/2009_09_01_Secure_Element_Programming.pdf, last access: 30th August 2013

[9] Youn-Sik Jeong, Jae-Chan Moon, Dongjin Kim, Yeong-Ung Park, Seong-Je Cho, Minkyu Park, "An Anti-Piracy Mechanism based on Class Separation and Dynamic Loading for Android Applications", RACS '12

[10] Marius Muntean, "Improving License Verification in Android", p. 109

[11] MyCE, "Geohot releases universal Android root exploit – just install an APK", http://www.myce.com/news/geohot-releases-universal-android-root-exploit-just-install-an-apk-71833 , last access: August 25th 2014

[12] Marius Muntean, "Improving License Verification in Android", p. 54ff

[13] Daniel A., "Seek-For-Android - Details", https://code.google.com/p/seek-for-android/wiki/MscSmartcardService , last access: September 15th 2014

[14] Google Code, "Issue 67406, O_DIRECT for hardware communication (microSD-Card) on Android 4.4 not implemented", https://code.google.com/p/android/issues/detail?id=67406 , last access: September 15th 2014

[15] Nils T. Kannengiesser et al., "Secure Copy Protection for Mobile Apps", AmiEs 2013 Symposium, Berlin

[16] Marius Muntean, "Improving License Verification in Android", p. 32

[17] Hyunho Ji, Woochur Kim, "Design of a Mobile Inspector for Detecting Illegal Android Applications using Fingerprinting", RACS '13

[18] Sanghoon Choi et al., "Android Application's Copyright Protection Technology based on Forensic Mark", RACS '12

[19] Ho Kwon Lee et al., "Memory Hacking Analysis in Mobile Devices for Hybrid Model of Copyright Protecton for Android Apps", RACS '13

[20] Trustonic, "Thundersoft and Trustonic join forces to bring next-generation security to developing smartphone markets" , https://www.trustonic.com/news/release/thundersoft-and-trustonic-join-forces-to-bring-next-generation-security-to-developing-smartphone-markets/en, last access: September 21st 2014