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

Notice: AmiEs 2013 slide

Secure Copy Protection for Mobile Apps

Content

Introduction

Foundations

Proposals

Conclusion / Outlook / Problems

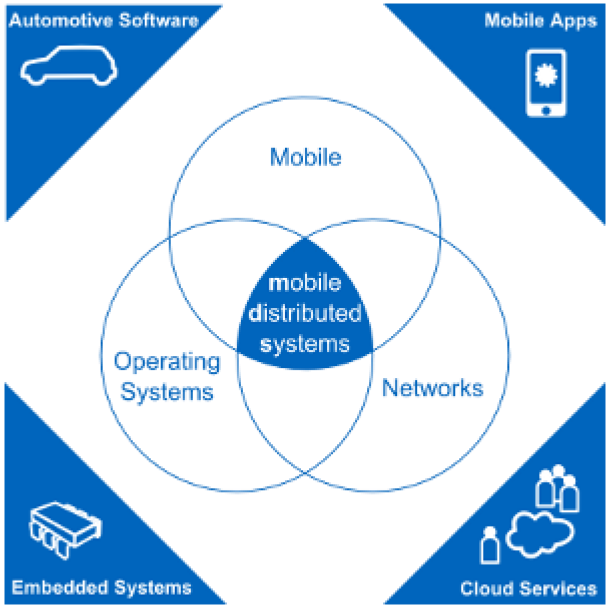
Notice: AmiEs 2013 slide

Secure Copy Protection for Mobile Apps

Introduction

Nils T. Kannengiesser
http://www.os.in.tum.de/personen/kannengiesser/

- Computer Science
- Research/Teaching Associate at TUM/F13
- Research field: Android Security / Copy Protection
- Teaching
  - Android Practical Course
  - “Computersysteme 2” training
  - Advisor of various theses



Chair for Operating Systems (F13)
- Cloud Services
- Automotive Software
- Mobile Apps
- Embedded Systems

Notice: AmiEs 2013 slide (modified)

Foundations

Notice: AmiEs 2013 slide

Secure Copy Protection for Mobile Apps

Foundations

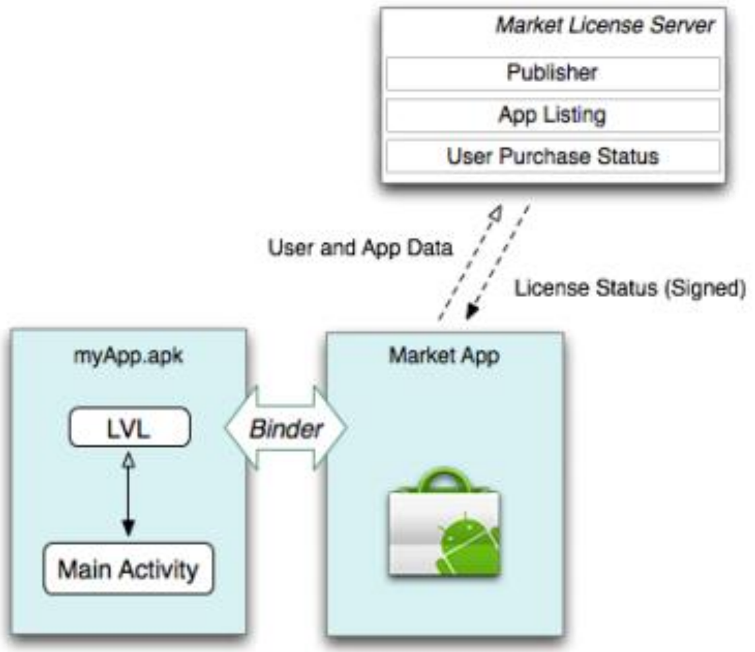
Main Research Question:

How can be achieved that an app can only be used on valid devices?

Notice: AmiEs 2013 slide (modified)

Foundations

- For monetization of apps it is reasonable to use some kind of copy protection (= less piracy)
- Google released the **License Verification Library** in 2010

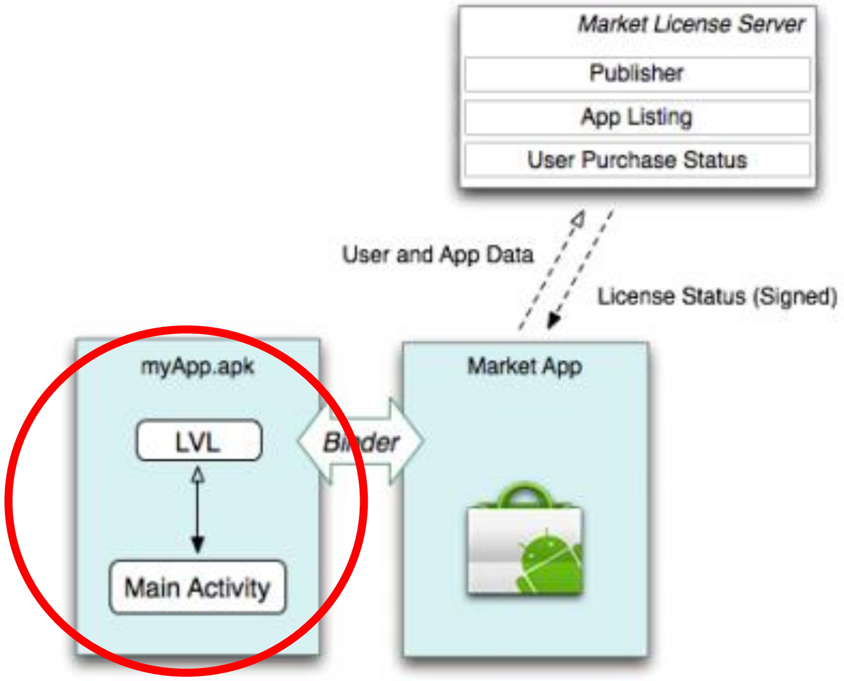


Ref. http://developer.android.com/google/play/licensing/overview.html

Notice: AmiEs 2013 slide

Foundations

- It got cracked the same year by changing a condition within any application



Ref. http://www.androidpolice.com/2010/08/23/exclusive-report-googles-android-market-license-verification-easily-circumvented-will-not-stop-pirates/
http://developer.android.com/google/play/licensing/overview.html

Notice: AmiEs 2013 slide

Foundations

- The problem: Reengineering (usual) Android Apps is quite readily
    - Tools: APKtool (baksmali/smali), dex2jar & JD-GUI

- The resulting code is an assembly dialect (smali) or even Java code containing **all class and variable names**

Example: LicenseValidator.smali

```
[…] .field private static final LICENSED:I = 0x0
.field private static final LICENSED_OLD_KEY:I = 0x2
.field private static final NOT_LICENSED:I = 0x1
[…]
.sparse-switch
0x0 -> :sswitch_d3
0x1 -> :sswitch_de
[…]
```

Ref. Example Source Code - http://www.androidpolice.com/2010/08/23/exclusive-report-googles-android-market-license-verification-easily-circumvented-will-not-stop-pirates/

Notice: AmiEs 2013 slide

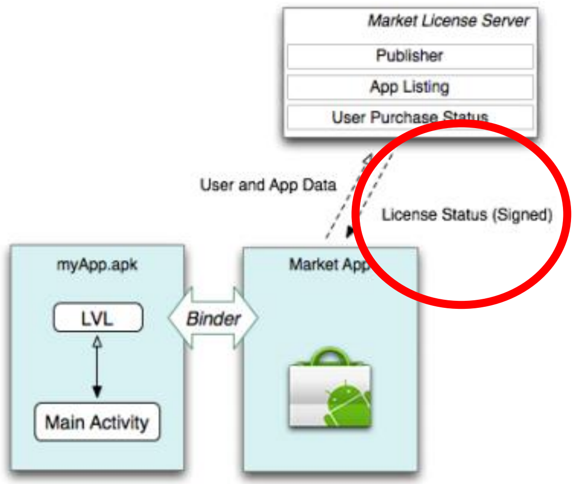
Foundations

- Later they included another tool in their IDE:
  ProGuard (Obfuscator* ; disabled by default)
  ➔ main reason that lots of apps are still easy to reengineer

- Since then Google updated the LVL, of course. In their latest release they added e.g. signed replies.

*** Example:**
aa.smali

```
[…]
.field private static final c:I = 0x1
[…]
.sparse-switch
0x0 -> :sswitch_d3
0x1 -> :sswitch_de
[…]
```



Ref. Graphic http://developer.android.com/google/play/licensing/overview.html

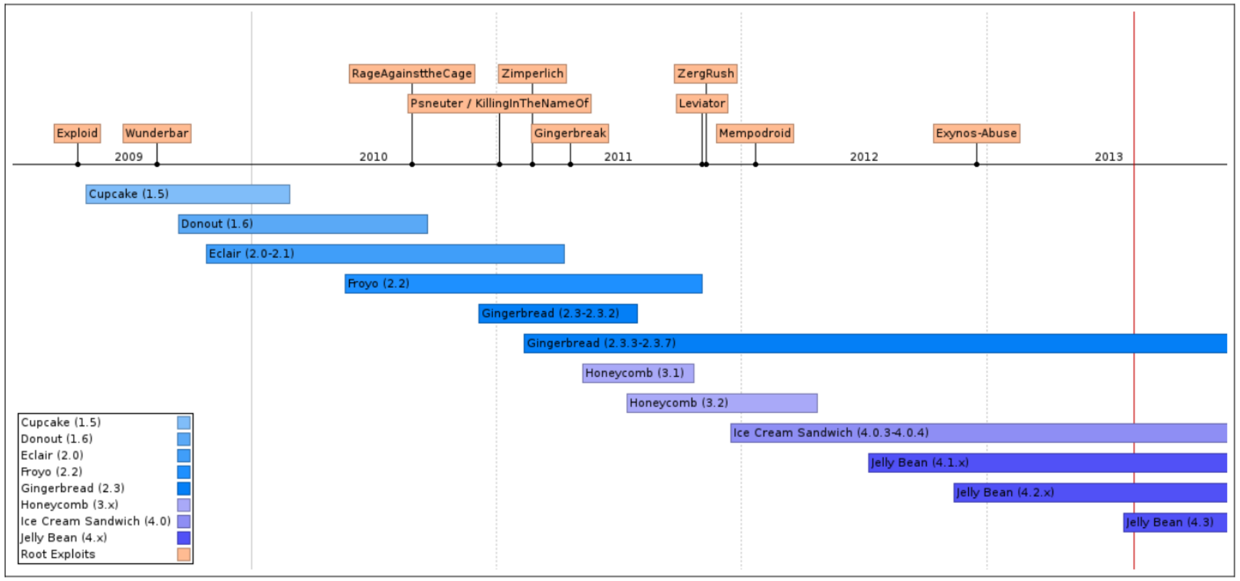
Notice: AmiEs 2013 slide

Foundations

Why is reengineering actually a huge problem on Android?

There are lots of exploits and it can be assumed that every Android version is insecure and it‘s going to be insecure. It’s possible to access APKs* and private files (as root user). Also the reengineered code is often quite easy to understand.

* Android **A**pplication **Pack**age File



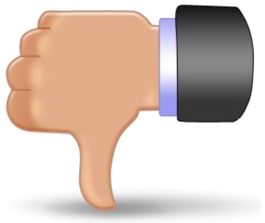
Ref. Graph. Janosch Maier, Bachelor’s thesis

Notice: AmiEs 2013 slide (modified)

Secure Copy Protection for Mobile Apps

Foundations

Google Android's LVL summary:



- According to our latest research the **L**icense **V**erification **L**ibrary as well as other protection libraries **offer minimal protection for apps**
- We were able **to identify severe issues to crack** almost any Android app instantly and notified the Google Android Security Team about it (09/05/14)
  - We **canceled the publication/paper to allow Google to fix it**
  - Therefore we are unable to present the details this time
  - The method might apply to other services (in-app-billing) and is under current investigation by us
- In general Google points out that "actual enforcement and handling of the license [...] are up to you"

Ref. Quote http://developer.android.com/google/play/licensing/overview.html ; Marius Muntean's Master's thesis

Notice: AmiEs 2013 slide (modified)

Foundations

- Possible solutions (for instance) to secure it (related work examples) :

Trusted Execution Environments	**Secure Elements**	**Enhanced Operating Systems**
ARM's TrustZone	Giesecke & Devrient's MSC	NSA's SEAndroid
	Any SIM card	Fraunhofer's TrustDroid
	Embedded (phone's NFC chip)	

Advantages

Highly secure	Secure ; HW attached easily	No additional HW required

Disadvantages

New hardware New OS/drivers	Limited security (cf. root access)	Less secure (exploits?)

Notice: AmiEs 2013 slide (modified)

Proposals

Secure Copy Protection for Mobile Apps

Proposals: Hardware

- We decided to go with ***Secure Elements***, since our solution should be available for a huge variety of phones
- Due to available partnerships the **M**obile **S**ecurity **C**ard (MSC) by Giesecke & Devrient was picked for our research purposes

Ref. Graphic http://www.gi-de.com/de/about_g_d/press/press_releases/global_press_release_7234.jsp

Notice: AmiEs 2013 slide

Proposal: MSC / Secure Element

- Since we want to be able to offer it for current smartphones as well, we plan to make **use of a MicroSD adapter** (via micro USB).

  A possible device is the “Dash Micro” from Meenova, which allows to mount any micro SD card on your Android device.

- It turned out that **mounting** a micro SD card (e.g. MSC) is **not possible** under a stock Android **without root rights** (= many customers won’t be able to use it)

Notice: AmiEs 2013 slide (modified)

Proposal: MSC / Secure Element

- **After mounting it on a rooted device** instead, a **communication problem** was discovered (see below).

The Secure Element of the **MSC** is **accessible** via **two interfaces**:

- Advanced Security SD (ASSD) interface [not available ; no slot*]
- Special file I/O operation [result: written command read back]

Unfortunately **Android** does **not support the O_DIRECT flag** (4.x)
→ The special file I/O interface cannot be used, since the reply by the SE cannot be fetched ; huge problem for the industry

* on many devices

Ref. https://code.google.com/p/seek-for-android/wiki/MscSmartcardService

Proposal: MSC / Secure Element

"If the facts don't fit the theory, change the facts“, Albert Einstein

- Android offers with the USB Device class a way to access device directly.

- Therefore we developed the library “libaums” (**lib**rary to **a**ccess **U**SB **M**ass **S**torage [devices])

  The library accesses USB devices, interprets the raw data, identifies the used file system (FAT supported only) and allows **direct reading/writing of files**. It does **not even require root rights** – a one-time permission by the user to access the USB device is sufficient.

Ref. https://github.com/mjdev/libaums

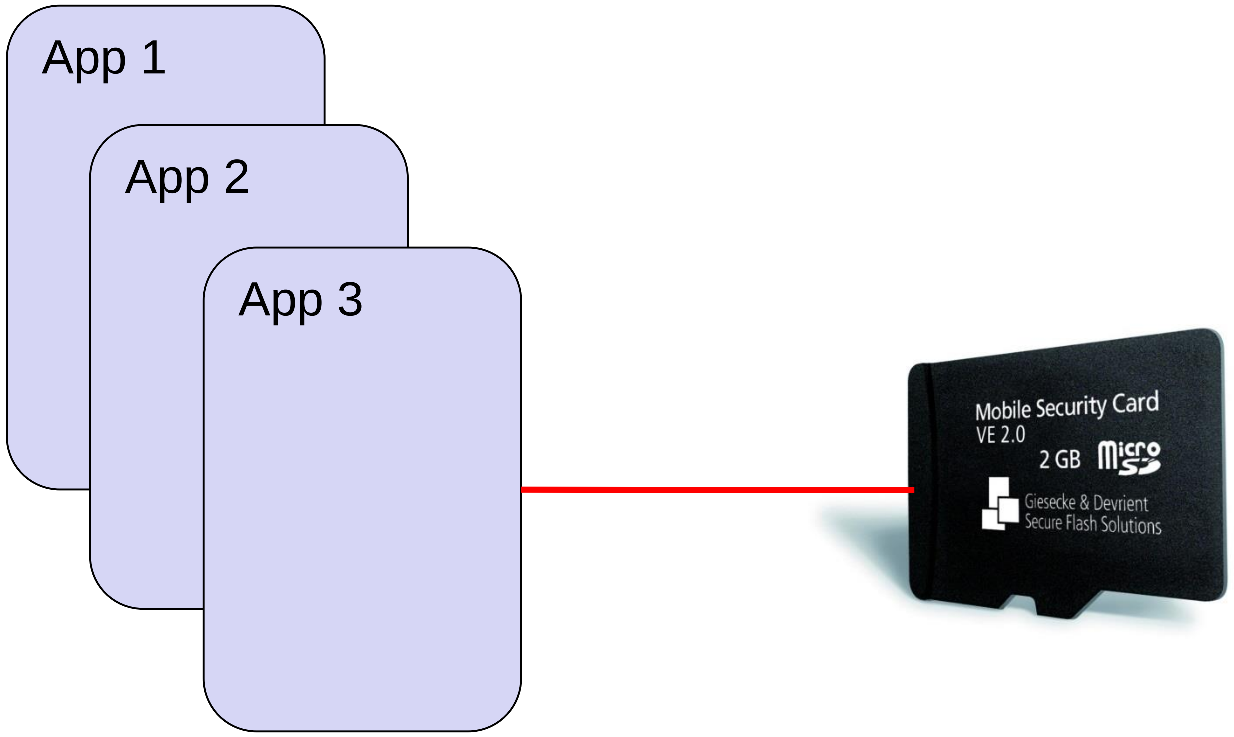
Proposal: MSC / Secure Element

- We combined the existing MSC frameworks with our library
    - **The communication** between Android and the Secure Element (MSC) **is working now**.
- Also it's a **workaround** to the **O_DIRECT issue**, assuming an adapter is used.
- Furthermore it allows the **mounting on non-rooted devices** (within an app)

Secure Copy Protection for Mobile Apps

Proposals: Idea

- The basic idea is to ***bind apps to their owners*** (Apps <> MSC).
- We assume that the source code is available to be able to modify it.



Ref. Graphic http://www.gi-de.com/de/about_g_d/press/press_releases/global_press_release_7234.jsp

Notice: AmiEs 2013 slide

Proposals

The initial goal of every copy protection mechanism is to identify a valid user/device.

- **Device identification**

Current API functions for the device ID might be spoofed.

Therefore we propose to implement our **own identification algorithms** based on "Short Term"- and "Long Term"-information.

MAC 09-11-22-02-24-20
Googleaccount, Contact details, …

Usual phone locations,
available WiFi networks, …

Notice: AmiEs 2013 slide

Proposals

Beside the identification we need *to add new security features* to bind an app and MSC together. Some are based on initial ideas by Google.

- **Content protection (1)**

APK files are usually not encrypted. After reengineering it's possible to extract resources (audio etc.).

We propose to encrypt all files, while the keys are received from the MSC during runtime.

The same applies to received files in future.

Benefit:
Basic protection of files.

Notice: AmiEs 2013 slide

Proposals

Beside the identification we need to add new security features:

- **Content protection (2)**

Strings will be replaced by MSC calls, while the MSC returns the string on (validated) requests.

e.g.
MSCauth(…)
[… other code …]
Print: MSC(“msg1”)

Benefit:
Attackers are no longer able to look for strings in reengineered code nor is it possible to deobfuscate it easily.

Notice: AmiEs 2013 slide

Proposals

Beside the identification we need to add new security features:

- **Content protection (3)**

Most URLs should be removed from the original source code and replaced by MSC calls. The MSC will provide a temporary URL later on, while keeping the original URL secret.

e.g.
1d2e2: www.os.in.tum.de/scripts.php?p1=222
www.os.in.tum.de/MSC?appid=1&l=1d2e2&p1=222
www.os.in.tum.de/MSC?appid=1&l=1d2e3&p1=222

Benefit:
Real URLs hidden. It's not possible to use the app without the MSC.

Notice: AmiEs 2013 slide

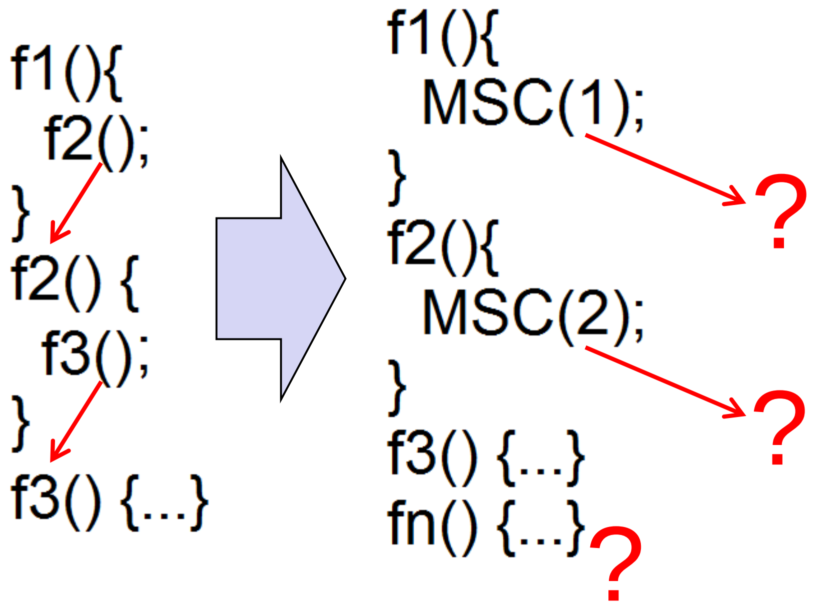
Proposals

Beside the identification we need to add new security features:

- **Obfuscate execution**

Add nonsense functions based on available functions to the code, while replacing function calls with MSC calls. The MSC replies with the required method name.

Attacker's view after reengineering:

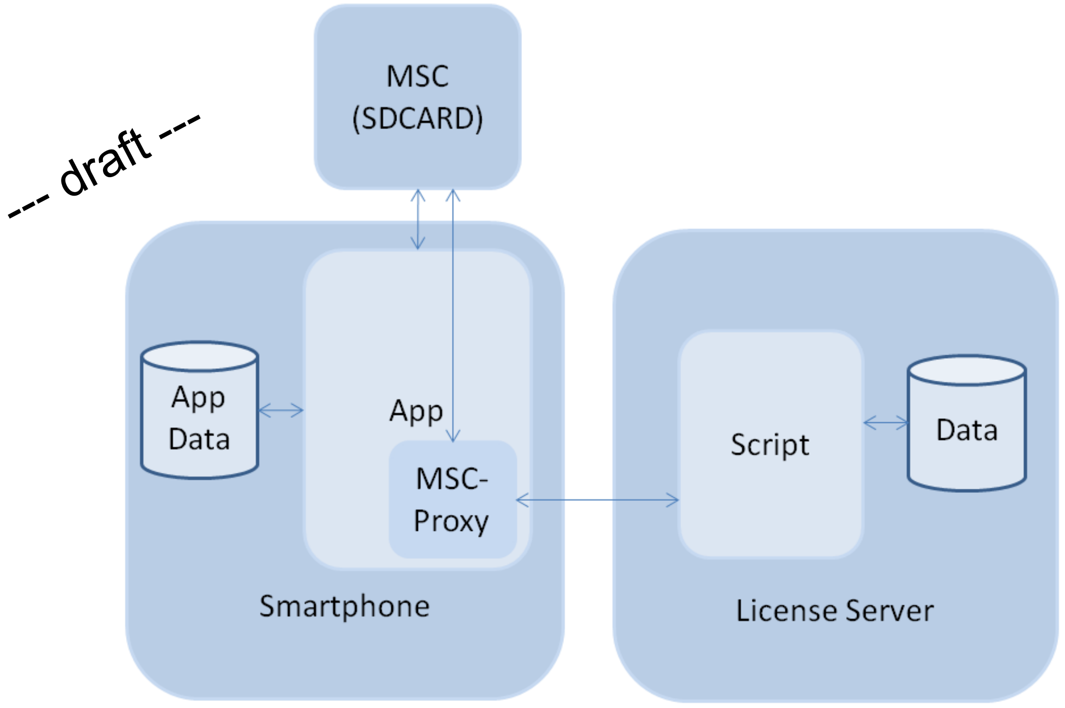


Benefit:
Attackers will be confused by lots of nonsense code.
Reengineering level increased to difficult (live debugging tools?)

Notice: AmiEs 2013 slide

Proposals:

- Also there is the requirement to update the stored information in the MSC according to the used app by using a MSC proxy and a special License server. The connection will be encrypted.



Notice: AmiEs 2013 slide

Proposals

Beside these methods, we can try to integrated known methods:

- Integrity checks
- Use methods to break disassemblers (e.g. wrong OPcode)
  → still works with many disassemblers and they were not updated (August 2014)

Conclusion

Conclusion / Problems / Outlook

We **identified severe issues** in the current **LVL implementation by Google** (and other vendors) finally and can **confirm our early assumptions** from the last year.

We were able to **solve fundamental issues** in getting access to micro SD cards on modern devices by using an adapter and the mentioned library.

Also we **solved the O_DIRECT issue on Android** to provide the **missing interface for accessing secure elements** (of the MSC) using that library.

All other presented theories about increasing the security of Android's copy protection mechanisms **are still under investigation and require further research**.

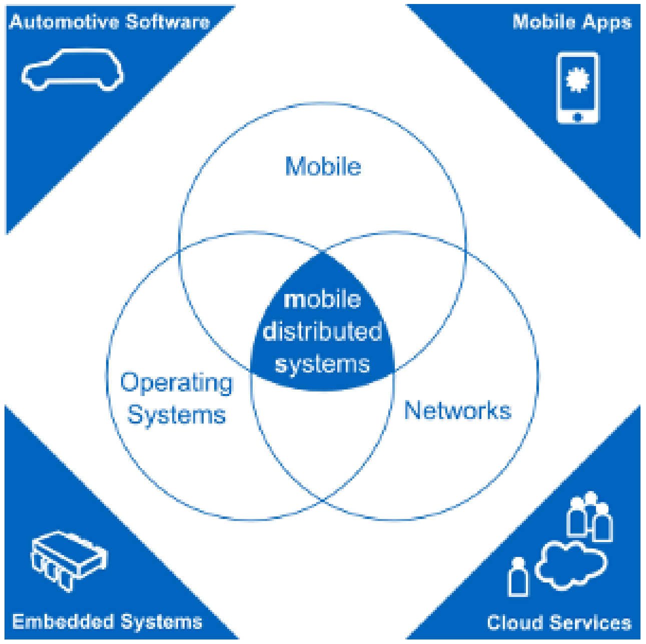
We are still assuming that **secure elements are the key to solve lots of security-related issues** on current (Android) devices.

Secure Copy Protection for Mobile Apps

Last but not least …

We are always looking for new partners and cooperations – for research projects as well as for teaching purposes.

Research Topics



Android Practical Course

Deutsches Herzzentrum München

Thank you for your attention

Nils.Kannengiesser@TUM.de