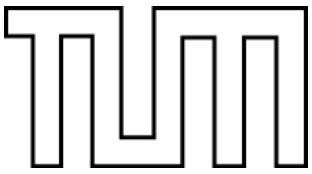


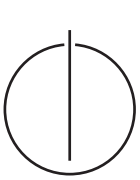
DEPARTMENT OF INFORMATICS

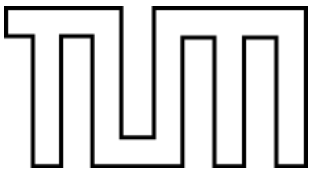
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Analysis of Android Cracking Tools and Investigations in Counter Measurements for Developers

Johannes Neutze, B. Sc.





DEPARTMENT OF INFORMATICS

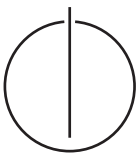
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Analysis of Android Cracking Tools and Investigations in Counter Measurements for Developers

Analyse von Android Crackingtools und Untersuchung geeigneter Gegenmaßnahmen für Entwickler

Author:	Johannes Neutze, B. Sc.
Supervisor:	Prof. Dr. Uwe Baumgarten
Advisor:	Nils Kannengießer, M. Sc.
Submission Date:	March 15, 2015



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, March 15, 2015 Johannes Neutze, B. Sc.

Acknowledgments

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Assumptions

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

http://users.ece.cmu.edu/~koopman/essays/abstract.html Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Contents

Acknowledgments iii

Assumptions iv

Abstract v

Glossary 1

Acronyms 2

1 Introduction 3
- 1.1 Licensing . . . 3
- 1.2 Motivation . . . 3
- 1.3 Related Work . . . 4

2 Foundation 5
- 2.1 Software Piracy . . . 5
  - 2.1.1 Overview . . . 5
  - 2.1.2 Threat to Developers . . . 5
  - 2.1.3 Risks to Users . . . 5
  - 2.1.4 Piracy on Android . . . 5
- 2.2 Android . . . 6
  - 2.2.1 Introduction . . . 6
  - 2.2.2 Evolution of the Android Compiler . . . 6
  - 2.2.3 Build Process of Android Applications . . . 6
  - 2.2.4 Original Copy Protection . . . 6
  - 2.2.5 Root on Android . . . 6
- 2.3 License Verification Libraries . . . 7
  - 2.3.1 Amazon . . . 7
  - 2.3.2 Google . . . 8
  - 2.3.3 Samsung . . . 8
- 2.4 Reengineering Basics . . . 9

3 Cracking Android Applications with LuckyPatcher **13**
3.1 What is LuckyPatcher and what is it used for? . . . 13
3.2 Modus Operandi . . . 13
3.3 What are Patching Modi are there and what do they do? . . . 14
3.4 What patterns are there and what do they do? . . . 14
3.5 Learnings from LuckyPatcher . . . 14

4 Counter Measurements for Developers **15**
4.1 Tampering Protection . . . 15
4.1.1 Prevent Debuggability . . . 16
4.1.2 Root Detection . . . 16
4.1.3 LuckyPatcher Detection . . . 17
4.1.4 Sideload Detection . . . 19
4.1.5 Signature Check . . . 19
4.1.6 Signature . . . 19
4.2 LVL Modifications . . . 21
4.2.1 Modify the Library . . . 21
4.2.2 Checken ob ganzer code abläuft und dann nacheinander elemente aktivieren . . . 21
4.2.3 dynamische Codegeneration . . . 21
4.3 Preventing Reengineering . . . 21
4.3.1 Basic Breaks for Common Tools . . . 21
4.3.2 Optimizors and Obfuscators . . . 23
4.3.3 Protectors and Packers . . . 25
4.4 External Improvements . . . 29
4.4.1 Service-managed Accounts . . . 29
4.4.2 ART . . . 29
4.4.3 Secure Elements . . . 30

5 Evaluation of Counter Measurements **31**
5.1 Tampering Protection . . . 31
5.1.1 Prevent Debuggability . . . 31
5.1.2 Root Detection . . . 31
5.1.3 LuckyPatcher Detection . . . 31
5.1.4 Sideload Detection . . . 31
5.1.5 Signature Check . . . 31
5.1.6 Remote Verification and Code nachladen . . . 32
5.2 LVL Modifications . . . 32

5.3 Prevent Reengineering 32
5.3.1 Packers 32
5.4 External Improvements 32
5.4.1 Service-managed Accounts 32
5.4.2 ART 32
5.4.3 Secure Elements 32

6 Conclusion **33**
6.1 Summary 33
6.2 Discussion 33
6.3 Future Work 33

List of Figures **35**

List of Tables **36**

Bibliography **37**

List of Code Snippets **38**

Glossary

.class Java Byte Code produced by the Java compiler from a .java file.

.dex Dalvik Byte Code file, translated from the Java bytecode. Dalvik Executables are designed to run on system with memory or processor constraints. For example, the .dex file of the Phone application is inside the system/app/Phone.apk.

.odex Optimized Dalvik Byte Code file are Dalvik Executables optimized for the current device the application is running on. For example, the .odex file of the Phone application is system/app/Phone.odex.

ADB The Android Debug Bridge is a command-line application providing different debugging tools.

APK An Android Application Package is the file format used for distributing and installing applications on the Android operating system. It contains the applications assets, code (.dex file), manifest and resources.

Acronyms

.dex Dalvik EXecutable file.

.odex Optimized Dalvik EXecutable file.

ADB Android Debug Bridge.

APK Android Application Package.

SDK Software Development Kit.

TUM Technische Universität München.

1 Introduction

sis is the introduction

1.1 Licensing

Was ist licensing?
Ziele von Licensing
was für möglichkeiten gibt es (lvl, amazon, samsung)

1.2 Motivation

Piracy
lose money from sale/IAP
lose ad revenues
others earn the money - ad ID replacement
no control at all when cracked and in other markets -> no fixes/updates (https://youtu.be/TNnccRimhsI?t=

for user: when downloading pirated apk, no idea what they changed (malware, stealing data,privacy, permissions)
wont notice any difference since in backgound
unpredicted traffic for your server, be prepared to block pirated traffic
cracking can lead to bad user experience, e.g. copied apps, mostly for paid apps

awesome algorithms can be stolen

similar problems with inapp billing

best way to counter: license verification libraries

encryption can be dumped from memory

generell piracy!!!

enthält als Abschluss SCOPE

1.3 Related Work

related work

2 Foundation

Before discussing the counter measurements, necessary basics have to be explained. This includes piracy in general as well as its relationship with Android.

2.1 Software Piracy

asdasdas http://www.xda-developers.com/piracy-testimonies-causes-and-prevention/

2.1.1 Overview

Definition of Software Piracy?

History of Software Piracy

Forms of Software Piracy

Release Groups, blackmarket, app beispiele, foren etc

2.1.2 Threat to Developers

scahden für entwickler (ad id klau,)

2.1.3 Risks to Users

malware, bad user experience

2.1.4 Piracy on Android

Poor model, since self-signed certificates are allowed, 27 http://newandroidbook.com/files/Andevcon-Sec.pdf
http://www.fiercedeveloper.com/story/preventing-android-applications-piracy-possible-requi

2012-08-14 piracy umfrage on android
übergehen zu wie android funktioniert und warum es dort piracy gibt

2.2 Android

flow wie funktioniert android und warum ist es so einfach zu piraten
https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

2.2.1 Introduction

What is Android? Where is it used? When was it founded? Who does it belong to?

platform developed by the "Android Open Source Project"

2.2.2 Evolution of the Android Compiler

2.2.3 Build Process of Android Applications

für 4.1 erklären, bild http://developer.android.com/tools/building/index.html

2.2.4 Original Copy Protection

It replaces the old system copy protection system, wherein your APKs would be put in a folder that you can't access. Unless you root. Oh, and anyone who can copy that APK off can then give it to someone else to put on their device, too. It was so weak, it was almost non-existant.
kann mit root umgangen werden

2.2.5 Root on Android

what is it? how is it achieved? what can i do with it? (good/bad sides)
Android insecure, can be rooted and get apk file http://androidvulnerabilities.org/all search for root

2.3 License Verification Libraries

This chapter contains the LVL which will be looked at
What is a lvl? why are they used? connection to store
http://www.digipom.com/how-the-android-license-verification-library-is-lulling-you-into-a
dependent on store
major players have own stores and thus own lvl

Since the original approach of subsection 2.2.4 was voided, another method had to be introduced. First looks great, puts the copy protection inside the app, a from of DRM
communicate with server, authorize use of application
does not prevent user from copying/transfering app, but copy useless since the app does run without the correct account
google die ersten, andere folgen, anfangs problem, dass dadurch nur durch google store geschützt war, grund dafür dass evtl ein programmierer in meinen store kommt

2.3.1 Amazon

amazon drm kiwi

Implementation

done by amazon packaging tool
release http://www.androidheadlines.com/2010/10/amazon-send-developers-a-welcome-package.html
https://developer.amazon.com/public/support/submitting-your-app/tech-docs/submitting-your-app

Functional Principle

sis is text was sind voraussetzungen? amazon app, account active der die app hat

Example

anhand eigener app

2.3.2 Google

License Verification Library
http://www.digipom.com/how-the-android-license-verification-library-is-lulling-you-into-a
related https://developers.google.com/games/services/android/antipiracy
http://android-developers.blogspot.de/2010/09/securing-android-lvl-applications.html
http://developer.android.com/google/play/licensing/overview.html
problem http://daniel-codes.blogspot.de/2010/10/true-problem-with-googles-license.html
The LVL library only works on apps sold through Google's Android Market Release
http://android-developers.blogspot.de/2010/07/licensing-service-for-android.html

Implementation

sis is text

Functional Principle

how does google license check work http://android.stackexchange.com/questions/22545/how-does-google-plays-market-license-check-work
sis is text
was sind voraussetzungen? googel acc auf dem smartphone welcher die app gekauft hat

bild http://android-developers.blogspot.de/2010/07/licensing-service-for-android.html

Example

anhand eigener app

2.3.3 Samsung

Zirconia http://developer.samsung.com/technical-doc/view.do?v=T000000062L

Implementation

sis is text http://developer.samsung.com/technical-doc/view.do?v=T000000062L

Functional Principle

sis is text
was sind voraussetzungen?

Example

anhand eigener app
http://developer.samsung.com/technical-doc/view.do?v=T000000062L

2.4 Reengineering Basics

The Cracking Tool has to alter an application's behaviour by applying patches only to the Android Application Package (APK) file, since it is the only source of code on the phone. This is the reason for the investigations to start with analysing the APK. This is done using static analysis tools. The aim is to get an accurate overview of how the circumventing of the license verification mechanism is achieved. This knowledge is later used to find counter measurements to prevent the specific Cracking Tool from succeeding.
The reengineering has to be done by using different layers of abstraction. The first reason is because it is very difficult to conclude from the altered bytecode, which is not human-readable, to the new behaviour of the application. The second reason is because the changes in the Java code are interpreted by the decompiler, which might not reflect the exact behaviour of the code or even worse, cannot be translated at all.
These problems are encountered by analysing the different abstraction levels of code as well as different decompilers.
https://mobilesecuritywiki.com/
https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf
main tools

Getting an APK

In the following there will be an example application to generalise the procedure. The application is called L̈icenseTeständ has for our purpose a license verification library included (Amazon, Google or Samsung).
In order to analyse an APK, it has to be pulled from the Android device onto the

computer. First the package name of the app has to be found out. This can be done by using the Android Debug Bridge (ADB). Entering example 1 returned example 2
adb shell ṕm list packages -f́ example 2 enthällt return von 1
adb pull /data/app/me.neutze.licensetest-1/ /Downloads

Dex

mein custom script erklären
jedes tool:
woher kommt es?
wozu wurde es erfunden?
wer hat es erfunden? quelle
blabla von der seite
wozu benutze ich es?
welches abstrahierungslevel
beispiel

wo findet man es?
welches level?
vorteil
blabla aus dem internet

```
a6 8e 15 00 bd 8e 15 00 d5 8e 15 00 f0 8e 15 00 |................|
```

Code Snippet 2.1: Quelle

Smali Code

https://github.com/JesusFreke/smali
Smali code is the generated by disassembling Dalvik bytecode using baksmali. The result is a human-readable, assambler-like code
selbe wie dex, jedoch human readable, no big difference
wo findet man es?
wie ist es erstellt?
informationsverlust?
vorteil
blabla aus dem internet

jedes tool:
woher kommt es?
wozu wurde es erfunden?
wer hat es erfunden? quelle
blabla von der seite
wozu benutze ich es?
welches abstrahierungslevel
beispiel

wo findet man es?
welches level?
vorteil
blabla aus dem internet

Java

Since interpretations sache
deswegen zwei compiler
unterschiedliche interpretation resultiert in flow und auch ob sies können ist unterschiedlich

ectl unterschiede/vor-nachteile

Androguard

https://github.com/androguard/androguard
jedes tool:
woher kommt es?
wozu wurde es erfunden?
wer hat es erfunden? quelle
blabla von der seite
wozu benutze ich es?
welches abstrahierungslevel
beispiel

jadx

https://github.com/skylot/jadx
jedes tool:
woher kommt es?
wozu wurde es erfunden?
wer hat es erfunden? quelle
blabla von der seite
wozu benutze ich es?
welches abstrahierungslevel
beispiel

Comparison of Code

needed to see differences before and after cracking tool

diff is used

https://wiki.ubuntuusers.de/diff
-N: Treat absent files as empty; Allows the patch create and remove files.
-a: Treat all files as text; Allows the patch update non-text (aka: binary) files.
-u: Set the default 3 lines of unified context; This generates useful time stamps and context.
-r: Recursively compare any subdirectories found; Allows the patch to update subdirectories.
script erklären

wo findet man es?
welches level?
vorteil
blabla aus dem internet

3 Cracking Android Applications with LuckyPatcher

There are plenty of applications which can be used to modify Android apps. This thesis focuses on the on on device cracking application LuckyPatcher, especially on its license verification bypassing mechanism.

3.1 What is LuckyPatcher and what is it used for?

wer hat ihn geschrieben?
auf welcher version basiere ich
su nicht vergessen
was kann er alles
was schauen wir uns an?

LuckyPatcher is described as following on the offical webpage: "Lucky Patcher is a great Android tool to remove ads, modify apps permissions, backup and restore apps, bypass premium applications license verification, and more. To use all features, you need a rooted device." [1]

install apk from palystore -> have root -> open lucky -> chose mode

LuckyPatcher is an Android Cracking Tool which can be downloaded at `http://lucky-patcher.netbew.com/`.

similar cracking tools:
or manual: decompile and edit what ever you want

3.2 Modus Operandi

wo arbeitet er?
warum dex und nicht odex anschauen?
patterns und patching modes grob erklären (modi von luckypatcher die verschiedene

operationen (pattern) auf app anwenden) => vorgehensweise zur

Since the code is modified directly a static analysis is sufficient.

UM ES EINFACHER ZU MACHEN, KEINE ODEX (WARUM), APK CREATEN UND AUF EINEM NORMALEN HANDY INSTALLIEREN(dann sieht man dass man die app wem anders gecracked geben kann - ringschluss blackmarket)

3.3 What are Patching Modi are there and what do they do?

kombination von patterns.
welche modes gibt es? welche patterns benutzen sie?
welche apps getestet und welche results?

3.4 What patterns are there and what do they do?

was greift jedes pattern an? wie wird der mechanismus ausgeklingt? was ist das result?

3.5 Learnings from LuckyPatcher

was fällt damit weg?
erklären warum (2) 5.1.2 Opaque predicates zb nicht geht, da auf dex ebene einfach austauschbar
simple obfuscation for strings? x -> string (damit name egal)

4 Counter Measurements for Developers

Now that that the functionality of LuckyPatcher is analyzed, it is time to investigate in possible solutions for developers. Counter measurements preventing the cracking app from circumventing the license check mechanism are addressed in four different ways. The first chapter covers functions to discover preconditions in the environment cracking apps use to discover weaknesses or need to be functional. The second chapter uses the aquired knowledge about LuckyPatcher to modify the code resulting in the patching being unsuccessful. In the third chapter presents methods to prevent the reengineering of the developerś application and thus the creation of custom cracks. Further hardware and external measurements are explained in the fourth chapter.

general suggestions by google `http://android-developers.blogspot.de/2010/09/securing-android-lvl-applications.html`

4.1 Tampering Protection

Environment and Integrity Checks, wenn die umgebung falsch ist, kann die app verändert werden. deswegen von vornherein ausschließen, dass die bedingungen dafür gegeben sind.
siehe masterarbeit 2

mechanisms should work for amazon/lvl/samsung –> beweis! (amazon die signature den die seite vorgibt?)

force close im falle von falschem outcome, entspricht nicht android qualität `http://developer.android.com/distribute/essentials/quality/core.html` aber so wird es dem user klarer dass seine application gecracked ist. harmlosere variante dialog anzeigen oder element nicht laden.

es gibt verschieden punkte um die integrity der application sicherzustellen. dies beinhaltet die umgebung debugg oder rootzugriff, die suche nach feindliche installierte applicationen oder checks nach der rechtmäßigen installation und rechtmäßigen code.

4.1.1 Prevent Debuggability

der debug modus kann dem angreifer informationen/logs über die application geben während diese läuft, aus diesen informationen können erkenntnisse über die funktionsweise geben die für einen angriff/modifikation gewonnen werden können. aus diesen informationen können dann patches für software wie lucky patcher entwickelt werden, da man die anzugreifenden stellen bereits kennt. kann erzwungen werden indem man das debug flag setzt (wo ist es, wie kann es gesetzt werden)
um dies zu verhindern kann gecheckt werden ob dieses flag forciert wird und gegebenenfalls das laufen der application unterbinden

```
14  public static boolean isDebuggable(Context context) {
15      boolean debuggable = (0 != (context.getApplicationInfo().flags & ApplicationInfo.
            FLAG_DEBUGGABLE));
16
17      if (debuggable) {
18          android.os.Process.killProcess(android.os.Process.myPid());
19      }
20
21      return debuggable;
22  }
```

Code Snippet 4.1: asd[1]

Code SNippet /refCode Snippet: luckycode zeigt eine funktion die auf den debug modus prüft. Dazu werden zuerst in zeile 15 die appinfo auf das debug flag überprüft. ist dieses vorhanden, ist die variable debuggable true. in diesem fall wird dann die geschlossen

4.1.2 Root Detection

`http://stackoverflow.com/questions/10585961/way-to-protect-from-lucky-patcher-play-licens`

```
16  public static boolean findBinary(Context context, final String binaryName) {
17      boolean result = false;
18      String[] places = {
19              "/sbin/",
20              "/system/bin/",
21              "/system/xbin/",
22              "/data/local/xbin/",
23              "/data/local/bin/",
24              "/system/sd/xbin/",
```

```
25                  "/system/bin/failsafe/",
26                  "/data/local/"
27          };
28
29          for (final String where : places) {
30              if (new File(where + binaryName).exists()) {
31                  result = true;
32                  android.os.Process.killProcess(android.os.Process.myPid());
33              }
34          }
35
36          return result;
37      }
```

Code Snippet 4.2: Partial Listing

SafetyNet provides services for analyzing the configuration of a particular device, to make sure that apps function properly on a particular device and that users have a great experience. `https://developer.android.com/training/safetynet/index.html` Checking device compatibility with safetynet
Unlocked bootloader doesn't matter. Can't have root installed initially. Has to be a stock / signed ROM. `https://www.reddit.com/r/Android/comments/3kly2z/checking_device_compatibility_with_safetynet/`

4.1.3 LuckyPatcher Detection

As the example shows, this check is not only a solution to prevent the application from running when LuckyPatcher is present on the device. The screening can be expanded to check for the installation of any other application, like black market apps or other cracking tools as the code example Code Example 4.5 shows.
`http://stackoverflow.com/questions/13445598/lucky-patcher-how-can-i-protect-from-it` `http://android-onex.blogspot.de/2015/07/anti-piracy-software-activated-solved.html`

```
9       public static boolean checkInstall(final Context context) {
10          boolean result = false;
11          String[] luckypatcher = new String[]{
12                  // Lucky patcher
13                  "com.dimonvideo.luckypatcher",
14                  // Another lucky patcher
15                  "com.chelpus.lackypatch",
16                  // Black Mart alpha
17                  "com.blackmartalpha",
18                  // Black Mart
```

```
19                  "org.blackmart.market",
20                  // Lucky patcher 5.6.8
21                  "com.android.vending.billing.InAppBillingService.LUCK",
22                  // Freedom
23                  "cc.madkite.freedom",
24                  // All−in−one Downloader
25                  "com.allinone.free",
26                  // Get Apk Market
27                  "com.repodroid.app",
28                  // CreeHack
29                  "org.creeplays.hack",
30                  // Game Hacker
31                  "com.baseappfull.fwd"
32          };
33
34          for (String string : luckypatcher) {
35              if(checkInstallerName(context, string)){
36                  result = true;
37              }
38
39              if (result) {
40                  android.os.Process.killProcess(android.os.Process.myPid());
41              }
42          }
43
44          return result;
45      }
46
47      private static boolean checkInstallerName(Context context, String string) {
48          PackageInfo info;
49          boolean result = false;
50
51          try {
52              info = context.getPackageManager().getPackageInfo(string, 0);
53
54              if (info != null) {
55                  android.os.Process.killProcess(android.os.Process.myPid());
56                  result = true;
57              }
58
59          } catch (final PackageManager.NameNotFoundException ignored) {
60          }
61
62          if (result) {
63              android.os.Process.killProcess(android.os.Process.myPid());
64          }
65          return result;
66      }
```

```
67 }
```

Code Snippet 4.3: Partial Listing

4.1.4 Sideload Detection

`http://stackoverflow.com/questions/10809438/how-to-know-an-application-is-installed-from-`

4.1.5 Signature Check

```
15 public class Sideload {
16     private static final String PLAYSTORE_ID = "com.android.vending";
17     private static final String AMAZON_ID = "com.amazon.venezia";
18     private static final String SAMSUNG_ID = "com.sec.android.app.samsungapps";
19
20     public static boolean verifyInstaller(final Context context) {
21         boolean result = false;
22         final String installer = context.getPackageManager().getInstallerPackageName(context.
               getPackageName());
23
24         if (installer != null) {
25             if (installer.startsWith(PLAYSTORE_ID)) {
26                 result = true;
27             }
28             if (installer.startsWith(AMAZON_ID)) {
29                 result = true;
30             }
31             if (installer.startsWith(SAMSUNG_ID)) {
32                 result = true;
33             }
34         }
35         if(!result){
36             android.os.Process.killProcess(android.os.Process.myPid());
37         }
38
39         return result;
40     }
```

Code Snippet 4.4: Partial Listing

4.1.6 Signature

`http://developer.android.com/tools/publishing/app-signing.html`
`http://forum.xda-developers.com/showthread.php?t=2279813&page=5`

CONTRA

Local Signature Check

local check whether signature is allowed
once in code
save to use signature in code?

```
51      public static boolean checkAppSignature(final Context context) {
52          //Signature used to sign the application
53          static final String mySignature = "...";
54          boolean result = false;
55
56          try {
57              final PackageInfo packageInfo = context.getPackageManager().getPackageInfo(context.
                    getPackageName(), PackageManager.GET_SIGNATURES);
58
59              for (final Signature signature : packageInfo.signatures) {
60                  final String currentSignature = signature.toCharsString();
61                  if (mySignature.equals(currentSignature)) {
62                      result = true;
63                  }
64              }
65          } catch (final Exception e) {
66              android.os.Process.killProcess(android.os.Process.myPid());
67          }
68
69          if (!result) {
70              android.os.Process.killProcess(android.os.Process.myPid());
71          }
72
73          return result;
74      }
```

Code Snippet 4.5: Partial Listing

Remote Signature Verification and Remote COde Loading

certificate an server, get signature and send to server
content direkt von server laden (e.g. all descriptions, not sure if dex possible)
maps checks for signature?
e.g. account auf seite erstellen, ecrypted dex ziehen der von loader stub geladen wird (like packer) kann wiedermal dann gezogen werden und dann als custom patch verteilt

werden

4.2 LVL Modifications

siehe masterarbeit 2 http://www.digipom.com/how-the-android-license-verification-library-is-l
What can I do?
Googleś License Verification Library is modified in a way

4.2.1 Modify the Library

google

4.2.2 Checken ob ganzer code abläuft und dann nacheinander elemente aktivieren

master1 - testen

damit die ganzen blöcke durchlaufen werden müssen

4.2.3 dynamische Codegeneration

4.3 Preventing Reengineering

Reverse engineering of Android applications is much easier than on other architectures
-> high level but simple bytecode language
Obfuscation techniques protect intellectual property of software/license verification
possible code obfuscation methods on the Android platform focus on obfuscating Dalvik bytecode -> limitations of current reverse engineering tools

https://blog.fortinet.com/post/how-android-malware-hides
http://www.hotforsecurity.com/blog/mobile-app-development-company-fights-off-android-malw
html

4.3.1 Basic Breaks for Common Tools

pros and cons sagen?
https://github.com/strazzere/APKfuscator

http://www.strazzere.com/papers/DexEducation-PracticingSafeDex.pdf
https://youtu.be/Rv8DfXNYnOI?t=811

Filesystem

make classname to long
https://youtu.be/Rv8DfXNYnOI?t=985 works except for the class
breaks only baksmali

Inject bad OPcode or Junkbytes

JUNKBYTES
use bad opcode in deadcode
code runs but breaks tools
put it into a class you do not use –> care proguard, it will not use it since it is not included
–> fixed...
https://youtu.be/Rv8DfXNYnOI?t=1163
reference not inited strings
https://youtu.be/Rv8DfXNYnOI?t=1459

Throw exceptions which are different in dalvik than in java

recursive try/catch? –> valid dalvik code
https://youtu.be/Rv8DfXNYnOI?t=1650

Increase headersize

you have to edit every other offset as well
https://youtu.be/Rv8DfXNYnOI?t=1890
dexception, dex within a dex by shifting
this is a packer/encrypter
slowdown automatic tools
https://youtu.be/Rv8DfXNYnOI?t=1950

Endian Tag

reverse endian
breaks tools works on device (odex)

lot work for little gain
https://youtu.be/Rv8DfXNYnOI?t=2149

4.3.2 Optimizors and Obfuscators

Obfuscators/Optimizors definition
remove dead/debug code
potentially encrypt/obfuscate/hide via reflection
https://youtu.be/6vFcEJ2jgOw?t=243

Proguard

https://youtu.be/6vFcEJ2jgOw?t=419

http://developer.android.com/tools/help/proguard.html
optimizes, shrinks, (barely) obfuscates –> free, reduces size, faster
gutes bild https://youtu.be/TNnccRimhsI?t=1360
removes unnecessary/unused code
merges identical code blocks
performs optimiztations
removes debug information
renames objects
restructures code
removes linenumbers –> stacktrace annoying
https://youtu.be/6vFcEJ2jgOw?t=470
–>hacker factor 0
does not really help
googles commentar http://android-developers.blogspot.de/2010/09/proguard-android-and-licens
html
https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_
Protection_in_Android.pdf ProGuard [4] is an open source tool which is also integrated in the Android SDK [5]. It can be easily used within the development process. ProGuard is basically a Java obfuscator but can also be used for Android applications because they are usually written in Java. The feature set includes identifier obfuscation for packages, classes, methods, and fields. Besides these protection mechanisms it can also identify and highlight dead code so it can be removed in a second, manual step. Unused classes can be removed automatically by ProGuard. Without proper naming of classes and methods it is much harder to reverse engineer an application, because

in most cases the identifier enables an analyst to directly guess the purpose of the particular part. The program code itself will not be changed heavily, so the obfuscation by this tool is very limited.
E. Lafortune. Proguard. Visited: May, 2012. [Online]. Available: `http://proguard.sourceforge.net/`

Dexguard

master2
OVERVIEW
son of proguard
the s̈tandardp̈rotection
optimizer
shrinekr
obfuscator/encrypter, does not stop reverse engineering
`https://youtu.be/6vFcEJ2jgOw?t=643`
WHAT DOES IT DO
everything that proguard does
automatic reflection
strign encryption
asset/library encryption
class encryption(packign)
applciation tamper protection
file->automatic reflection->string encryption->file
`https://youtu.be/6vFcEJ2jgOw?t=745`
class encryption= packer, unpackers do it most of the time in few seconds, aber aufwand auf handy, nicht so einfach wie pattern in luckypatcher
CONS
may increase dex size, memory size; decrease speed
removes debug information
string, etc encryption
best feature: automatic reflection with string encryption
reversible with moderate effort
hacker protection factor 1

Allatori

http://www.allatori.com/clients/index.php

WHAT DOES IT
name obfuscation
control flow flattening/obfuscation
debug info obfuscation
string encryption
RESULT
decreases dex size, memory, increases speed
remvoes debug code
not much obfuscation
Proguard+string encryption
easily reversed
hacker protection factor 0.5
https://youtu.be/6vFcEJ2jgOw
https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf
Allatori [6] is a commercial product from Smardec. Besides the same obfuscation techniques like ProGuard, shown in section 2.1, Allatori also provides methods to modify the program code. Loop constructions are dissected in a way that reverse engineering tools cannot recognize them. This is an approach to make algorithms less readable and add length to otherwise compact code fragments. Additionally, strings are obfuscated and decoded at runtime. This includes messages and names that are normally human readable and would give good suggestions to analysts. The obfuscation methods used in Allatori are a superset of ProGuards so it is more powerful but does not prevent an analyst from disassembling an Android application
Allatori. Allatori obfuscator. Visited: May, 2012. [Online]. Available: http://www.allatori.com/doc.html

4.3.3 Protectors and Packers

from malware

APKprotect

stub fixes broken code which is normally not translated by tools, breaks static analysis
https://youtu.be/6vFcEJ2jgOw?t=347
https://youtu.be/6vFcEJ2jgOw

chinese protector
also known as dexcrypt, appears active but site down, clones might be available
anti-debug, anti-decompile, almost like a packer
string encryption
cost ???
tool mangles code original code
-modifies entrypoint to loader stub
-prevents static analysis
during runtime loader stub is executed
-performs anti-emulation
-performs anti-debugging
-fixes broken code in memory
FUNCTION
dalvik optimizes the dex file into momory ignoring bad parts
upon execution dalvik code initiates, calls the native code
native code fixes odex code in memory
execution continues as normal
RESULT
slight file size increase
prevents easily static analysis
hard once, easy afterwards
easily automated to unprotect
still has string encryption (like DexGuard, Allatori) afterwards
not much iteration in the last time, do not knwo if still alive
hacker protection factor 3, no public documentation, but every app is the same

Packers

break static analysis tools, you ahve to do runtime analysis
like UPX, stub application unpacks, decrypts, loads into memory which is normally hidden from static analysis
http://www.fortiguard.com/uploads/general/Area41Public.pdf
https://books.google.de/books?id=ACjUCgAAQBAJ&pg=PA372&lpg=PA372&dq=ijiami+integrity&source=bl&ots=NTf7YaqJiZ&sig=M5GKDCcQB5dcwXR3hjtIv8pMlAA&hl=de&sa=X&ved=0ahUKEwjH3umt1b3JAhXGLA8KHYhwDGsQ6AEIMDAC#v=onepage&q=ijiami%20integrity&f=false
https://www.blackhat.com/docs/asia-15/materials/asia-15-Park-We-Can-Still-Crack-You-Gener
pdf

https://www.virusbtn.com/conference/vb2014/abstracts/Yu.xml
https://www.virusbtn.com/pdf/conference_slides/2014/Yu-VB2014.pdf
https://www.youtube.com/watch?v=6vFcEJ2jgOw
https://books.google.de/books?id=ACjUCgAAQBAJ&pg=PA372&lpg=PA372&dq=ijiami+integrity&source=bl&ots=NTf7YaqJiZ&sig=M5GKDCcQB5dcwXR3hjtIv8pMlAA&hl=de&sa=X&ved=0ahUKEwjH3umt1b3JAhXGLA8KHYhwDGsQ6AEIMDAC#v=onepage&q=ijiami%20integrity&f=false

concept erklären und dann die beispiele nennen, nicht mehr aktiv/gecracked aber prinzip ist gut

examples for packers are

hosedex2jar
https://youtu.be/6vFcEJ2jgOw?t=1776
PoC packer
https://github.com/strazzere/dehoser/
not available for real use
appears defunct
near zero ITW samples
mimics dexception attack from dex education 101
FUNCTION
encrypts and injects dexfile into dex header (deception)
very easy to spot
very easy to decrypt, just use dex2jar
static analysis does not work since it sees the encrypted file
on execution loader stub decrypts in memory and dumps to file system
loader stub acts as proxy and passes events to the dex file on system using a dexClassLoader
RESULT
simple PoC
slight file size increase
attempts to prevent static analysis - kind of works
lots of crashing
easily automated to unpack
easy to reverse, good for learning
hacker protection factor 0.5

Pangxie

https://youtu.be/6vFcEJ2jgOw?t=1982
anti-debug
anti-tamper
appears to be defunct product
little usage/samples ITW
FUNCTION
https://youtu.be/6vFcEJ2jgOw?t=2040
encrypts dex file and bundles as asset in APK
very easy to find, logcat has to much information
dalvik calls JNI layer to verify and decrypt
easy to reverse, both dalvik and native, excellent for beginners to Android and packers
aes used only for digest verification
easily automated, 0x54 always the key
or dynamically grab app_dex folder
slightly increase file size
prevents static analysis - though easy to identify
uses static 1 byte key for encryption
easily automated to unpack
very easy to reverse, good for learning
good example of an unobfuscated packer stub for cloning
hacker protection faktor 1.5
only working till <4.4
simple packer, increase encryption with key, do not just dump on filesystem

BANGCLE
anti-debugging
anti-tamper
anti-decompilation
anti-runtime injection
online only service, apk checked for malware
detected by some anti virus due to malware
cost 10k
no one has done it before...
stopped working on 4.4
FUNCTION
dalvik execution talks launched JNI
JNI launches secondary process
chatter over PTRACE between the two processes
newest process decrypts dex into memory

original dalvik code proxies everything to the decrypted dex
RESULT
well written, lots of anti-* tricks
seems to be well supported and active on development
does a decent job on online screening - no tool released for download (though things clearly to slip through)
not impossible to reverse and re-bundle packages
current weakness (for easy runtime unpacking) is having a predictable unpacked memory location
hacker protect faktor 5
probably best tool out there but lag when updating since online approval

4.4 External Improvements

4.4.1 Service-managed Accounts

`https://youtu.be/TNnccRimhsI?t=1636`
check on server what content should be returned or logic on server

kann man einen lagorithus haben um rauszufinden was man auslagern kann?

if not possible remote code loading

`https://www.youtube.com/watch?v=rSH6dnUTDZo` was ist dann geschützt? content, servers, time constrained urls, obfuscation by using reflection combined with SE -> makes slow but no static analysis

very very slow, e.g 10kHz so no big calculations possible
250bytes, 200ms

`http://amies-2014.international-symposium.org/proceedings_2014/Kannengiesser_Baumgarten_Song_AmiEs_2014_Paper.pdf`

4.4.2 ART

art hat masschinen coed
wenn reengineerbar dann nicht gut

warum jetzt noch keine art apps? `https://en.wikipedia.org/wiki/Android_Runtime` dex2oat

4.4.3 Secure Elements

new section trusted execution environment trusttronic letzte conference samsung knox –>gelten eher sicher

5 Evaluation of Counter Measurements

Evaluation der vorgeschlagenen punkte mit pro cons und umsetzbarkeit

http://forum.xda-developers.com/showthread.php?t=2279813

5.1 Tampering Protection

just as easy to crack as LVL when you know the code
evtl create native versions because harder to crack

5.1.1 Prevent Debuggability

5.1.2 Root Detection

5.1.3 LuckyPatcher Detection

already in some roms
https://www.reddit.com/r/Piracy/comments/3gmxun/new_way_to_disable_the_antipiracy_setting_on/
https://github.com/AlmightyMegadeth00/AntiPiracySupport
https://www.reddit.com/r/Piracy/comments/3eo8sj/antipiracy_measures_on_android_custom_roms/
example roms http://www.htcmania.com/archive/index.php/t-1049040.html

https://forums.oneplus.net/threads/patch-disable-the-antipiracy-feature-on-all-roms-sur
334772/

5.1.4 Sideload Detection

5.1.5 Signature Check

maps checks for signature?
http://stackoverflow.com/questions/13582869/does-lucky-patcher-resign-the-app-it-patches-

https://developers.google.com/android/guides/http-auth http://forum.xda-developers.com/showthread.php?t=2279813&page=5
generell
self signing, kann genauso geskippt werden dann remote
wenn einmal geladen -> skippen und geladene datei als custom patch nachschieben

5.1.6 Remote Verification and Code nachladen

trotzdem doof wenn einmal geladen kann man das file extrahieren etc

5.2 LVL Modifications

reengineering kann aushebeln

5.3 Prevent Reengineering

5.3.1 Packers

already cracked https://www.google.de/search?q=hosedex2jar&oq=hosedex2jar&aqs=chrome..69i57j69i60j69i59j69i60l3.1680j0j7&sourceid=chrome&es_sm=91&ie=UTF-8
http://www.hotforsecurity.com/blog/mobile-app-development-company-fights-off-android-malwa
html

BEISPIELBILDER!!

5.4 External Improvements

sis is text

5.4.1 Service-managed Accounts

5.4.2 ART

art hat oat files aber die haben dex files

5.4.3 Secure Elements

new section trusted execution environment trusttronic letzte conference samsung knox –>gelten eher sicher

6 Conclusion

auch wichtig weil wenn crackable dann upload zu stores und dann malware
http://www.hotforsecurity.com/blog/mobile-app-development-company-fights-off-android-malware-with-obfuscation-tool-3717.html

6.1 Summary

sis is text alles hilft gegen lucky patcher auf den ersten blick, jedoch custom patches können es einfach umgehen -> deswegen hilft nur reengineering schwerer zu machen
every new layer is another complexity

6.2 Discussion

sis is text http://www.digipom.com/how-the-android-license-verification-library-is-lulling-yo
What Google should have really done
http://programmers.stackexchange.com/questions/267981/should-i-spend-time-preventing-pirac
You are asking the wrong question. Technical safeguards such as proguard are a must
but are trying to solve the problem the hard way.
content driven http://stackoverflow.com/questions/10585961/way-to-protect-from-lucky-patcher
google sagt http://android-developers.blogspot.de/2010/09/securing-android-lvl-applications.html

6.3 Future Work

art?
smart cards
google vault
all papers with malware and copyright protection is interesting since they also want to

hide their code

List of Figures

List of Tables

Bibliography

[1] ChelphuS. *crapler*. URL: http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm (visited on 09/30/2010).

List of Code Snippets

2.1 Name, Quelle 10

4.1 asd[1] 16
4.2 Partial Listing 16
4.3 Partial Listing 17
4.4 Partial Listing 19
4.5 Partial Listing 20