



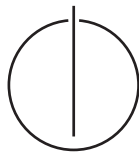
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

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

Johannes Neutze





DEPARTMENT OF INFORMATICS

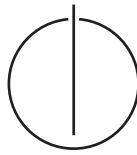
TECHNISCHE UNIVERSITÄT MÜNCHEN

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
Supervisor: TODO: Supervisor
Advisor: TODO: Advisor
Submission Date: TODO: Submission date



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

Munich, TODO: Submission date

Johannes Neutze

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

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	iv
Assumptions	v
Abstract	vi
Glossary	2
Acronyms	3
1 Introduction	4
1.1 Licensing	4
1.2 Motivation	4
1.3 Related Work	5
2 Foundation	6
2.1 Software Piracy	6
2.1.1 Overview	6
2.1.2 Threat to Developers	6
2.1.3 Risks to Users	6
2.1.4 Piracy on Android	6
2.2 Android	7
2.2.1 Introduction	7
2.2.2 Evolution of the Android Compiler	7
2.2.3 Basics of Android	7
2.2.4 Root on Android	7
2.3 License Verification Libraries	8
2.3.1 Amazon	8
2.3.2 Google	8
2.3.3 Samsung	9
2.4 Reengineering Tools	9
2.4.1 Reengineering	9
2.4.2 Dex	9

2.4.3	baksmali	9
2.4.4	Java	10
2.4.5	Diff	10
3	Cracking Android Applications with LuckyPatcher	11
3.1	What is LuckyPatcher and what is it used for?	11
3.2	Operation	11
3.3	What patterns are there and what do they do?	11
3.4	What are Patching Modes are there and what do they do?	11
3.5	Learnings from LuckyPatcher	12
4	Counter Measurements for Developers	13
4.1	Tampering Protection	13
4.1.1	Prevent Debuggability	13
4.1.2	Root Detection	13
4.1.3	LuckyPatcher Detection	13
4.1.4	Sideload Detection	14
4.1.5	Signature Check	14
4.1.6	Remote Verification and Code nachladen	14
4.2	LVL Modifications	15
4.2.1	Modify the Library	15
4.2.2	Junkbyte Injection	15
4.2.3	Checken ob ganzer code abläuft und dann nacheinander elemente aktivieren	15
4.2.4	dynamische Codegeneration	15
4.3	Prevent Reengineering	15
4.3.1	Basic Breaks for Common Tools	15
4.3.2	Optimizors and Obfuscators	17
4.3.3	Protectors	19
4.3.4	Packers	20
4.3.5	BANGCLE	22
4.4	External Improvements	22
4.4.1	Service-managed Accounts	22
4.4.2	ART	23
4.4.3	Secure Elements	23
5	Evaluation	24
5.1	Tampering Protection	24
5.1.1	Prevent Debuggability	24

Contents

5.1.2	Root Detection	24
5.1.3	LuckyPatcher Detection	24
5.1.4	Sideload Detection	24
5.1.5	Signature Check	24
5.1.6	Remote Verification and Code nachladen	24
5.2	Prevent Reengineering	25
5.2.1	Basic Breaks for Common Tools	25
5.2.2	Optimizors and Obfuscators	25
5.2.3	Protectors	25
5.2.4	Packers	25
5.2.5	BANGCLE	25
5.3	External Improvements	25
5.3.1	Service-managed Accounts	26
5.3.2	ART	26
5.3.3	Secure Elements	26
6	Conclusion	27
6.1	Summary	27
6.2	Discussion	27
6.3	Future Work	27
	List of Figures	29
	List of Tables	30

Contents

Glossary

API An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API .

computer is a machine that...

valid device a device which is allowed to run software specified by the license.

Acronyms

API Application Programming Interface.

TUM Technische Universität München.

1 Introduction

sis is a text

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 background

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

sis is a text

2.1 Software Piracy

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

2.1.1 Overview

What is 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-requ>

2012-08-14 piracy umfrage on android

2.2 Android

sis is text

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?

2.2.2 Evolution of the Android Compiler

sis is text

Java Virtual Machine

sis is text

Dalvik Virtual Machine

sis is text

Android Runtime

im Moment abwärtskompatibilität dex in oat (tools zum extrahieren nennen)

2.2.3 Basics of Android

sis is text

2.2.4 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>

2.3.1 Amazon

Amazon DRM

Implementation

sis is text

Functional Principle

sis is text

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>

Implementation

sis is text

Functional Principle

how does google license check work [http://android.stackexchange.com/questions/](http://android.stackexchange.com/questions/22545/how-does-google-plays-market-license-check-work)

22545/how-does-google-plays-market-license-check-work

sis is text

Example

anhand eigener app

2.3.3 Samsung

Zirconium

Implementation

sis is text

Functional Principle

sis is text

Example

anhand eigener app

2.4 Reengineering Tools

main tools

https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

2.4.1 Reengineering

https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

2.4.2 Dex

mein custom script erklären

2.4.3 baksmali

<https://github.com/JesusFreke/smali>

2.4.4 Java

Androguard

<https://github.com/androguard/androguard>

jadx

<https://github.com/skylot/jadx>

2.4.5 Diff

<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

can also be used to learn the code -> was nacher das allgemeine problem ist

3 Cracking Android Applications with LuckyPatcher

<http://lucky-patcher.netbew.com/>

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?

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

similar cracking tools:

or manual: decompile and edit what ever you want

3.2 Operation

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

3.3 What patterns are there and what do they do?

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

3.4 What are Patching Modes are there and what do they do?

kombination von patterns.

welche modes gibt es? welche patterns benutzen sie?

welche apps getestet und welche results?

3.5 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

am besten mit example

4.1 Tampering Protection

Environment and Integrity Checks

siehe masterarbeit 2

just as easy to crack as LVL when you know the code

evtl create native versions because harder to crack

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

4.1.1 Prevent Debuggability

sis is text

4.1.2 Root Detection

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

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

4.1.3 LuckyPatcher Detection

<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>

com.dimonvideo.luckypatcher, // Lucky patcher

com.chelpus.lackypatch, // Another lucky patcher

com.blackmartalpha, // Black Mart alpha
org.blackmart.market, // Black Mart
com.android.vending.billing.InAppBillingService.LUCK, // Lucky patcher 5.6.8
cc.madkite.freedom, // Freedom
com.allinone.free, // All-in-one Downloader
com.repodroid.app, // Get Apk Market
org.creeplays.hack, // CreeHack
com.baseappfull.fwd // Game Hacker
-> can be also done for black markets

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>

4.1.4 Sideload Detection

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

4.1.5 Signature Check

once in code
save to use signature in code?
<http://forum.xda-developers.com/showthread.php?t=2279813&page=5>

4.1.6 Remote Verification and Code nachladen

certificate an server, get signature and send to server
content direkt von server laden (e.g. all descriptions, not sure if dex possible)
e.g. account auf seite erstellen, encrypted 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?

4.2.1 Modify the Library

google

4.2.2 Junkbyte Injection

master1

4.2.3 Checken ob ganzer code abläuft und dann nacheinander elemente aktivieren

master1 - testen

damit die ganzen blöcke durchlaufen werden müssen

4.2.4 dynamische Codegeneration

4.3 Prevent Reengineering

sis is text <https://blog.fortinet.com/post/how-android-malware-hides>
<http://www.hotforsecurity.com/blog/mobile-app-development-company-fights-off-android-malware>
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/Rv8DfXNYn0I?t=985> works except for the class

breaks only baksmali

Inject bad OPCODE

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/Rv8DfXNYn0I?t=1163>

reference not inited strings

<https://youtu.be/Rv8DfXNYn0I?t=1459>

Throw exceptions which are different in dalvik than in java

recursive try/catch? -> valid dalvik code

<https://youtu.be/Rv8DfXNYn0I?t=1650>

Increase headersize

you have to edit every other offset as well

<https://youtu.be/Rv8DfXNYn0I?t=1890>

dexception, dex within a dex by shifting

this is a packer/encrypter

slowdown automatic tools

<https://youtu.be/Rv8DfXNYn0I?t=1950>

Endian Tag?

reverse endian

breaks tools works on device (odex)

lot work for little gain

<https://youtu.be/Rv8DfXNYn0I?t=2149>

4.3.2 Optimizers and Obfuscators

Obfuscators/Optimizers definition

remove dead/debug code

potentially encrypt/obfuscate/hide via reflection

<https://youtu.be/6vFcEJ2jg0w?t=243>

Relfection

<https://www.youtube.com/watch?v=Rv8DfXNYnOI>

irgendwo erklären

Proguard

<https://youtu.be/6vFcEJ2jg0w?t=419>

https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

<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 optimizations

removes debug information

renames objects

restructures code

removes line numbers -> stacktrace annoying

<https://youtu.be/6vFcEJ2jg0w?t=470>

-> hacker factor 0

does not really help

Dexguard

master2

OVERVIEW

son of proguard

the standard protection

optimizer

shrinekr

obfuscator/encrypter, does not stop reverse engineering

<https://youtu.be/6vFcEJ2jg0w?t=643>

WHAT DOES IT DO

everything that proguard does

automatic reflection

string encryption

asset/library encryption

class encryption(packign)

application tamper protection

file->automatic reflection->string encryption->file

<https://youtu.be/6vFcEJ2jg0w?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>

https://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf

WHAT DOES IT

name obfuscation

control flow flattening/obfuscation

debug info obfuscation

string encryption

RESULT

decreases dex size, memory, increases speed

removes debug code

not much obfuscation

Proguard+string encryption

easily reversed

hacker protection factor 0.5

<https://youtu.be/6vFcEJ2jg0w>

Dexprotector

master2

4.3.3 Protectors

stub fixes broken code which is normally not translated by tools, breaks static analysis

<https://youtu.be/6vFcEJ2jg0w?t=347>

APKprotect

<https://youtu.be/6vFcEJ2jg0w>

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 memory 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 know if still alive

hacker protection factor 3, no public documentation, but every app is the same

4.3.4 Packers

break static analysis tools, you have 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=M5GKDCcQB5dcwXR3hjtIv8pM1AA&hl=de&sa=X&ved=0ahUKEwjH3umt1b3JAhXGLA8KHYYhwDGsQ6AEIMDAC#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=6vFcEJ2jg0w>

<https://books.google.de/books?id=ACjUCgAAQBAJ&pg=PA372&lpg=PA372&dq=ijiami+integrity&source=bl&ots=NTf7YaqJiZ&sig=M5GKDCcQB5dcwXR3hjtIv8pM1AA&hl=de&sa=X&ved=0ahUKEwjH3umt1b3JAhXGLA8KHYYhwDGsQ6AEIMDAC#v=onepage&q=ijiami%20integrity&f=false>

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

hosedex2jar

<https://youtu.be/6vFcEJ2jg0w?t=1776>

PoC packer

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 dexClass-Loader

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/6vFcEJ2jg0w?t=1982>

anti-debug
anti-tamper

appears to be defunct product
little usage/samples ITW

FUNCTION

<https://youtu.be/6vFcEJ2jg0w?t=2040>

encrypts dex file and bundles as asset in APK
very easy to find, logcat has too 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 factor 1.5
only working till <4.4
simple packer, increase encryption with key, do not just dump on filesystem

4.3.5 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

sis is text

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

4.4.3 Secure Elements

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

5 Evaluation

Evaluation der vorgeschlagenen punkte mit pro cons und umsetzbarkeit

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

5.1 Tampering Protection

5.1.1 Prevent Debuggability

5.1.2 Root Detection

5.1.3 LuckyPatcher Detection

<https://forums.oneplus.net/threads/patch-disable-the-antipiracy-feature-on-all-roms-survi-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>

5.1.6 Remote Verification and Code nachladen

trotzdem doof wenn einmal geladen kann man das file extrahieren etc

5.2 Prevent Reengineering

5.2.1 Basic Breaks for Common Tools

Filesystem

Inject bad OPcode

Throw exceptions which are different in dalvik than in java

Increase headersize

Endian Tag?

5.2.2 Optimizors and Obfuscators

Relfection

Proguard

Dexguard

Allatori

Dexprotector

5.2.3 Protectors

APKprotect

5.2.4 Packers

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

hosedex2jar

Pangxie

5.2.5 BANGCLE

5.3 External Improvements

sis is text

5.3.1 Service-managed Accounts

5.3.2 ART

art hat oat files aber die haben dex files

5.3.3 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-you>

What Google should have really done

<http://programmers.stackexchange.com/questions/267981/should-i-spend-time-preventing-piracy>

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