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 General 6
  - 2.1.2 Threat to Developers 6
  - 2.1.3 Risks to Users 6
- 2.2 Android 6
  - 2.2.1 Introduction 6
  - 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 7
  - 2.3.1 Amazon 7
  - 2.3.2 Google 8
  - 2.3.3 Samsung 8
- 2.4 Reengineering Tools 8
  - 2.4.1 Dex 9
  - 2.4.2 baksmali 9
  - 2.4.3 Java 9

2.4.4 Diff 9

3 Cracking Android Applications with LuckyPatcher 10
3.1 What is LuckyPatcher and what is it used for? 10
3.2 Operation 10
3.3 What patterns are there and what do they do? 10
3.4 What are Patching Modes are there and what do they do? 10
3.5 Learnings from LuckyPatcher 11

4 Counter Measurements for Developers 12
4.1 Tampering Protection 12
4.1.1 Prevent Debuggability 12
4.1.2 Root Detection 12
4.1.3 LuckyPatcher Detection 12
4.1.4 Sideload Detection 12
4.1.5 Signature Check 13
4.1.6 Remote Verification and Code nachladen 13
4.2 LVL Modifications 13
4.2.1 Modify the Library 13
4.2.2 Junkbyte Injection 13
4.2.3 Checken ob ganzer code abläuft und dann nacheinander elemente aktivieren 13
4.2.4 dynamische Codegeneration 14
4.3 Prevent Reengineering 14
4.3.1 Basic Breaks for Common Tools 14
4.3.2 Optimizors and Obfuscators 15
4.3.3 Protectors 17
4.3.4 Packers 18
4.3.5 BANGCLE 20
4.4 External Improvements 21
4.4.1 Service-managed Accounts 21
4.4.2 ART 21
4.4.3 Secure Elements 21

5 Evaluation 22
5.1 Tampering Protection 22
5.1.1 Prevent Debuggability 22
5.1.2 Root Detection 22
5.1.3 LuckyPatcher Detection 22

5.1.4 Sideload Detection 22
5.1.5 Signature Check 22
5.1.6 Remote Verification and Code nachladen 22

6 Conclusion **23**
6.1 Summary 23
6.2 Discussion 23
6.3 Future Work 23

List of Figures **24**

List of Tables **25**

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. . . .

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 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

sis is a text

2.1 Software Piracy

2.1.1 General

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

2.2 Android

sis is text

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)

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

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

Implementation

sis is text

Functional Principle

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

2.4.1 Dex

mein custom script erklären

2.4.2 baksmali

https://github.com/JesusFreke/smali

2.4.3 Java

Androguard

https://github.com/androguard/androguard

jadx

https://github.com/skylot/jadx

2.4.4 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-licensi`

4.1.3 LuckyPatcher Detection

`http://stackoverflow.com/questions/13445598/lucky-patcher-how-can-i-protect-from-it`
–> can be also done for black markets

4.1.4 Sideload Detection

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

4.1.5 Signature Check

once in code
save to use signature in code?

SECURE NET VON GOOGLE

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, 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

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-malwa`
`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

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`

Relfection

`https://www.youtube.com/watch?v=Rv8DfXNYnOI`
irgendwo erklären

Proguard

`https://youtu.be/6vFcEJ2jgOw?t=419`
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/6vFcEJ2jg0w?t=470`
–>hacker factor 0
does not really help

Dexguard

master2
OVERVIEW
son of proguard
the s̈tandardp̈rotection
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
strign encryption
asset/library encryption
class encryption(packign)
applciation 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

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`

Dexprotector

master2

4.3.3 Protectors

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

APKprotect

`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

4.3.4 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/pdf/conference_slides/2014/Yu-VB2014.pdf`
`https://www.youtube.com/watch?v=6vFcEJ2jgOw`

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

hosedex2jar

`https://youtu.be/6vFcEJ2jgOw?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/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

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

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

5.1 Tampering Protection

5.1.1 Prevent Debuggability

5.1.2 Root Detection

5.1.3 LuckyPatcher Detection

5.1.4 Sideload Detection

5.1.5 Signature Check

5.1.6 Remote Verification and Code nachladen

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

6.2 Discussion

sis is text

6.3 Future Work

art?

List of Figures

List of Tables