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

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.

API 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.

assembler Ein Assembler (auch Assemblierer[1]) ist ein Computerprogramm, das Assemblersprache in Maschinensprache übersetzt, beispielsweise den Assemblersprachentext „CLI“ in den Maschinensprachentext „11111010“..

disassembler Ein Disassembler ist ein Computerprogramm, das die binär kodierte Maschinensprache eines ausführbaren Programmes in eine für Menschen lesbarere Assemblersprache umwandelt. Seine Funktionalität ist der eines Assemblers entgegengesetzt..

Acronyms

.dex Dalvik EXecutable file.

.odex Optimized Dalvik EXecutable file.

ADB Android Debug Bridge.

ADT Android Developer Tools.

AOT Ahead-Of-Time.

APK Application Programming Interface.

APK Android Application Package.

DVM Dalvik Virtual Machine.

GC Garbage Collection.

JIT Just-In-Time.

JNI Java Native Interface.

JVM Java Virtual Machine.

NDK Native Development Kit.

SDK Software Development Kit.

TUM Technische Universität München.

1 Introduction

This is my real text! Rest might be copied or not be checked!
[4]

1.1 Licensing

This is my real text! Rest might be copied or not be checked!
Was ist licensing? nutzungsrecht für dritte an Intelectual property oder ähnliches unter bestimmten definierten bedingungen an authorisation by one party licensor to another party lincensee, may require paying a fee

Ziele von Licensing eigene IP schützen, vermarkten etc
wikipedia
Software licensing grants right to purchase, install and use software according to vendor's license agreement protect both vendor, against piracy by stealing IP, user, fine for stealing or misuse enforce legal agreement by mechanism in software, disable or limit for users outside of bought software license different forms of license protection (account, abgescahltete features, watermark) can be circumvented, must protect by code obfuscation, license check and dialog [6]
was für möglichkeiten gibt es (lvl, amazon, samsung)

andere wehren sich schon `http://www.sueddeutsche.de/digital/illegale-kopien-von-computerspi`
2810482 bloß android noch nciht so weit

1.2 Motivation

This is my real text! Rest might be copied or not be checked!
android grows thus becomes lucrative, increased interest from software pirates developers know from plagiarism and piracy -see- QUELLE studies for plagiarism in Play Store-see- QUELLE [6]
spreading causes giant market (zahlen) -see- google play with big financial value (zahlen)

je größer der markt desto attraktiver für cracker da auch mehr leute die app ggfs gecracked runterladen würden (genauere argumente)
deswegen auch gesteigertes interesse bei developern ihre app und IP zu schützen

enthält als Abschluss SCOPE

1.3 Related Work

This is my real text! Rest might be copied or not be checked!
many papers for better obfuscation (examples, was machen die so) in general not on how to stop an cracking application
related work

2 Foundation

Before understanding the attack mechanisms and discussing counter measurements, necessary background knowledge has to be provided. Motivation and risks of software piracy and the basics of Android will be explained as well as existing licensing solutions and reengineering tools and methodics.

2.1 Software Piracy

This is my real text! Rest might be copied or not be checked!

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

2.1.1 Overview

This is my real text! Rest might be copied or not be checked!

Definition of Software Piracy?

This is my real text! Rest might be copied or not be checked!

History of Software Piracy

This is my real text! Rest might be copied or not be checked!

ewiges hin und her wer besser ist, siehe computer spiele, streaming drm, generell alles wo DRM ist

Forms of Software Piracy

This is my real text! Rest might be copied or not be checked!

Release Groups, blackmarket, app beispiele, foren etc

2.1.2 Threat to Developers

This is my real text! Rest might be copied or not be checked!
scahden für entwickler (ad id klau,)

2.1.3 Risks to Users

This is my real text! Rest might be copied or not be checked!
malware, bad user experience
It is not unlikely for a malware developer to abuse existing applications by injection of malicious functionalities and consequent redistribution of the trojanized versions `C.A. CastilloandMobileSecurityWorkingGroupMcAfee,\T1\textquotedblleftAndroidmalwarepast, present,andfuture,\T1\textquotedblright2011.`

2.1.4 Piracy on Android

This is my real text! Rest might be copied or not be checked!
bytecode for other architectures as well as the little protection applied in practice, Dalvik bytecode is currently an easy target for the reverse engineer
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

2.2.1 Introduction

This is my real text! Rest might be copied or not be checked!
Android is an open source Linux-based operating system running on a large set of touchscreen devices, wearables and many more
Launched in 2007 by Google, it is designed to meet the limited computational capacity of a mobile device's hardware. The principal processor of Android devices is the ARM platform for which the operating system is optimized

AUFBAU ANDROID The underlying entity of the system is its kernel which bridges

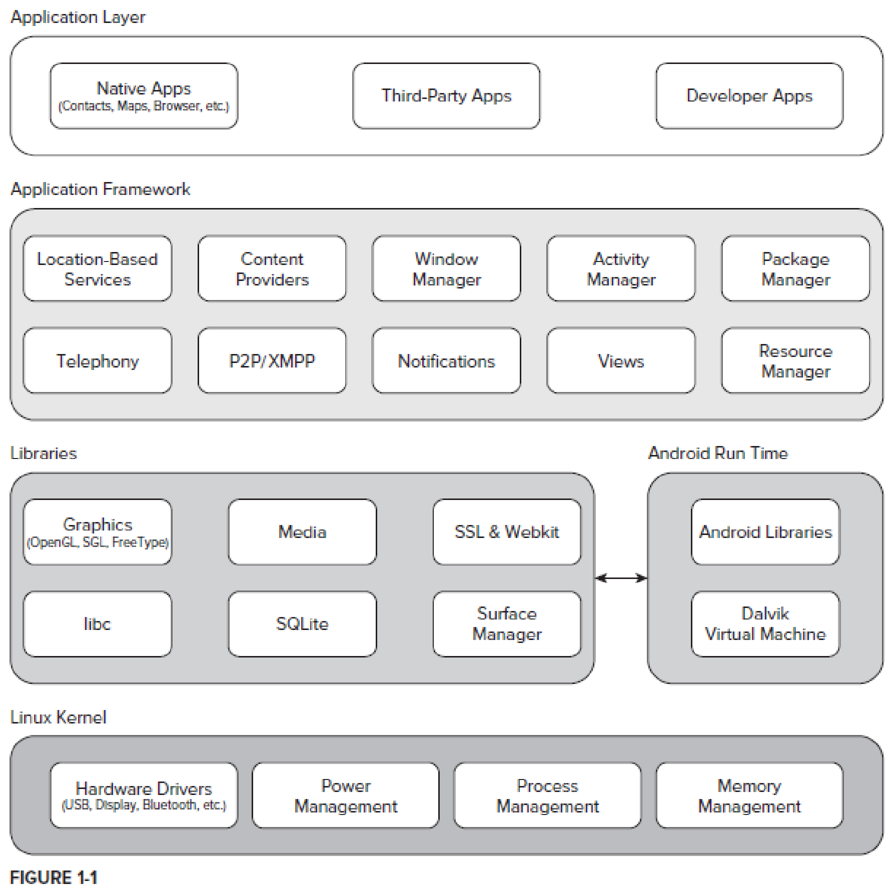


Figure 2.1: stack

the hardware of the device and the remaining software components. Being a Linux-based kernel, it allows remote access to the device via a Linux shell as well as the execution of standard Unix commands.
A level above is the Android Runtime, which will be explained closer
At the same abstraction level as the virtual machine are the native libraries of the system. Written in C/C++, they permit low level interaction between the applications and the kernel through Java Native Interface (JNI)
The next layer is the application framework which provides generic functionality to mobile software through Android's API. The top layer of the Android OS stack is where custom applications are compiled, installed and executed.
[4]

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

platform developed by the "Android Open Source Project" -see- official website?
currently one of the main operating systems for mobile devices -see- quelle
focussed on simplicity for users, install of apps etc -see- quelle
riesiger markt

2.2.2 Dalvik Virtual Machine

This is my real text! Rest might be copied or not be checked!

Flow what is dalvik, what is different to java, what is dex (build process), VGL `http://newandroidbook.com/files/ArtOfDalvik.pdf`

The applications for Android are written using the Java programming language.

stack abstraction is the Dalvik Virtual Machine (DVM)
DVM is highly tailored to work according to the specifications of the Android platform optimized for a slower CPU in comparison with a stationary machine andworks with relatively little RAM memory (• limited processor speed • limited RAM • no swap space • battery powered • diverse set of devices • sandboxed application runtime)[2]
DVM is register-based, differing from the standard Java Virtual Machine (JVM) which is stack-based, register-based architectures require fewer executed instructions than stack-based architectures, register-based code is approximately 25 percent larger than the stack-based, the increase in the instructions fetching time is negligible: 1.07 percent extra real machine loads[2]
the Android OS has no swap space imposing that the virtual machine works without swap. Finally, mobile devices are powered by a battery thus the DVM is optimized to be as energy preserving as possible, Except being highly efficient, the DVM is also designed to be replicated quickly because each application runs within a “sandbox”: a context containing its own instance of the virtual machine assigned a unique Unix user ID

wie der build process funktioniert wird später gesondert beschrieben, hier sagen wir einfach das ergebnis ist die dex datei

[4] [2]

DVM is - Customized optimized JVM based on Apache Harmony, - Not fully J2SE or J2ME compatible -see- Java compiles into DEX code -see- 16-bit opcodes and Register, rather than stack-based
History, Dalvík was introduced along with Android, created by Created by Dan Born-

stein, Named after an Icelandic town, 2.2 brought current just in time compilation (ERKLÄREN)
Dalvik vs Java - Dalvík is a virtual machine implementation, Based on Apache Harmony, Borrows heavily from Java - Brings significant improvements over Java, in particular J2ME, Virtual Machine architecture is optimized for memory sharing, Reference counts/bitmaps stored separately from objects, Dalvik VM startup is optimized through Zygote - Java .class files are further compiled into DEX.
Overview creating APK unterschied zu java und dann auf dex?

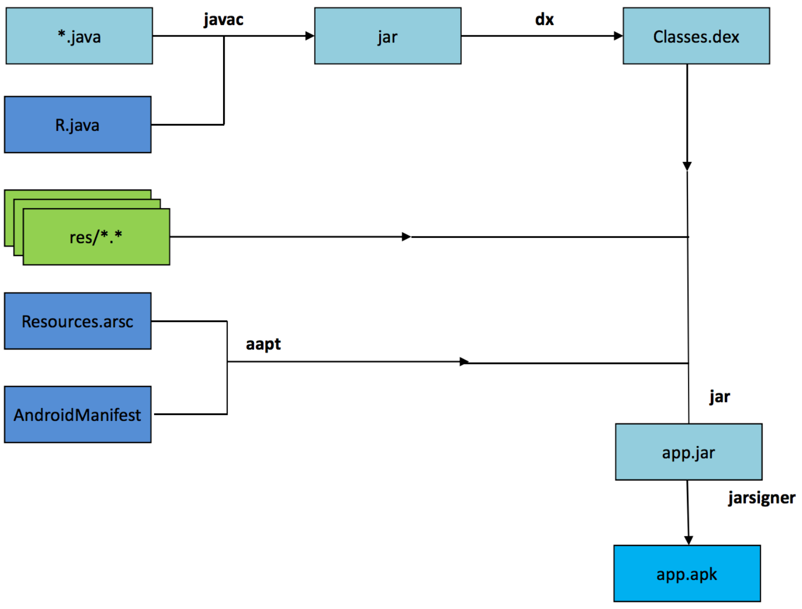


Figure 2.2: apk

[5]

2.2.3 Dalvik EXecutionable File

This is my real text! Rest might be copied or not be checked!

AUFBAU DEX DVM is register based. Registers are considered 32 bits wide to store values such as integers or floating point numbers. Adjacent register pairs are used to store 64-bit values
dest-then-source ordering for its arguments

there are 218 used valid opcodes in Dalvik bytecode -see- QUELLE

Due to its simplicity over bytecode for other architectures as well as the little protection applied in practice, Dalvik bytecode is currently an easy target for the reverse engineer.

VERGLEICHSBILD JAVA—DEX[2]
Each .class file has its own heterogeneous constant pool which may contain duplicating data, BEISPIEL, memory efficiency of a .dex file comes primarily from the type-specific constant pools used to store the data, BEISPIEL,
significantly more references within a .dex file compared to a .class file[2]
compression as efficiently as up to 44 percent of the size of an equivalent .jar archive[2]
hier noch einfach dex, später erst opcode nennen, the exact dex format will be explained in 2.4.2

[4] [2]

dx utility converts multilple class files to classes.dex, java bytecode is converted to dex bytecode, dex instructions 16bit mltiples, java 8bit constant, string,type and method pools are merged, significant savings for strings, types and methods in multiple calsses overall memory footprint diminished by about 50dex file format specified in android dcomentation -see- SOURCE
dex instructions refer to indexes (in pools)
de bytecode is striktly similar to java bytecode, allows for ease de/re compilation back and forth to/from java

dex vs java
java vm is stack based, dex is register based java bytecode is actually more compact than dex dex bytecode is more suited to arm architectures, mapping from dex registers to arm registers dex supports bytecode optimizations, java no, apk's calsses.dex are optimized before install, on device
[5]

AUFBAU DALVIK BYTECODE
The program code of an Android application is delivered in form of Dalvik bytecode It will be executed by the Dalvik Virtual Machine and is comparable to Java bytecode. So there is a separate optimizing step while installing an application, where the bytecode gets optimized for the underlaying architecture. The optimized form is also called "odex". The optimization is done by a program called "dexopt" which is part of the Android platform. The DVM can execute optimized as well as not optimized Dalvik bytecode

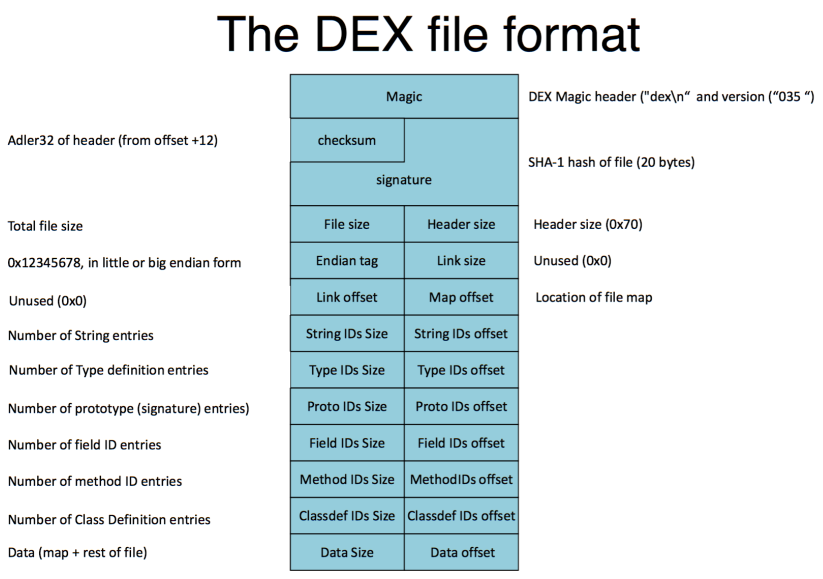


Figure 2.3: dex1

unterschied `http://newandroidbook.com/files/Andevcon-DEX.pdf`

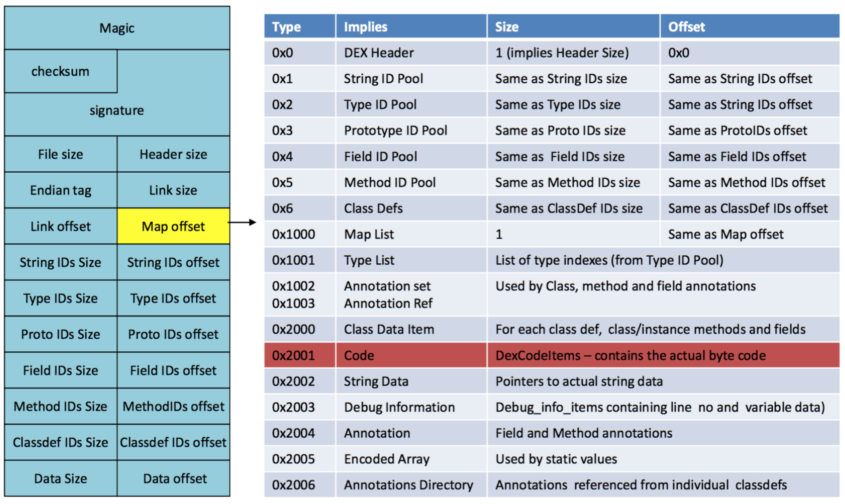
.dex file The Dalvik bytecode consists of opcodes and is thus hard to read for humans. The Cracking Tool has to modify the opcodes in order to alter the behaviour of the application. Since it is directly read by the Dalvik virtual machine, it is the single point of truth.

.odex file Dalvik bytecode of an application is normally not optimized, because it is executed by a DVM which can run under different architectures

2.2.4 Build Process of Android Applications

This is my real text! Rest might be copied or not be checked!

APK The file format of the install ready application is called Android Package (APK) and all the mobile software is distributed over Google Play in this format. The APK format is a package management system based on the ZIP file archive format. CONTENT: META-INF directory: MANIFEST.MF: the Manifest file CERT.RSA: The certificate of the application. CERT.SF: The list of resources and SHA-1 digest of the corresponding lines in the MANIFEST.MF file



Magic	
checksum	
signature	
File size	Header size
Endian tag	Link size
Link offset	Map offset
String IDs Size	String IDs offset
Type IDs Size	Type IDs offset
Proto IDs Size	Proto IDs offset
Field IDs Size	Field IDs offset
Method IDs Size	MethodIDs offset
Classdef IDs Size	Classdef IDs offset
Data Size	Data offset

Type	Implies	Size	Offset
0x0	DEX Header	1 (implies Header Size)	0x0
0x1	String ID Pool	Same as String IDs size	Same as String IDs offset
0x2	Type ID Pool	Same as Type IDs size	Same as String IDs offset
0x3	Prototype ID Pool	Same as Proto IDs size	Same as ProtoIDs offset
0x4	Field ID Pool	Same as Field IDs size	Same as Field IDs offset
0x5	Method ID Pool	Same as Method IDs size	Same as Method IDs offset
0x6	Class Defs	Same as ClassDef IDs size	Same as ClassDef IDs offset
0x1000	Map List	1	Same as Map offset
0x1001	Type List	List of type indexes (from Type ID Pool)	
0x1002 0x1003	Annotation set Annotation Ref	Used by Class, method and field annotations	
0x2000	Class Data Item	For each class def, class/instance methods and fields	
0x2001	Code	DexCodeItems – contains the actual byte code	
0x2002	String Data	Pointers to actual string data	
0x2003	Debug Information	Debug_info_items containing line no and variable data)	
0x2004	Annotation	Field and Method annotations	
0x2005	Encoded Array	Used by static values	
0x2006	Annotations Directory	Annotations referenced from individual classdefs	

Figure 2.4: dex2

lib: the directory containing the compiled code that is specific to a software layer of a processor, the directory is split into the different processor types

res: the directory containing resources not compiled into resources.arsc (see below).

assets: a directory containing applications assets, which can be retrieved by AssetManager.

AndroidManifest.xml: An additional Android manifest file, describing the name, version, access rights, referenced library files for the application. This file may be in Android binary XML that can be converted into human-readable plaintext XML with tools such as AXMLPrinter2, android-apktool, or Androguard. classes.dex: The classes compiled in the dex file format understandable by the Dalvik virtual machine

resources.arsc: a file containing precompiled resources, such as binary XML for example.

BUILD PROCESS written using the Java programming language.
Standard Java environment compiles each separate class in the .java source code file into a corresponding Java bytecode .class file. Each class will be compiled into a single .class file. These are later packed together in a single .jar archive file. The JVM unpacks the .class files, parses and executes their code at runtime.
On the Android platform, the build process differs after the point when the .class files have been generated. Once having the latter, they are forwarded to the "dx" tool which is part of the standard Android SDK. This tool compresses all .class files into a single classes.dex file i.e. the .dex file is the sole bytecode container for all the application's

classes. After it has been created, the classes.dex is forwarded to the ApkBuilder tool altogether with the application resources and shared object (.so) files which, if present, contain native code. As a result, the APK archive is created and the final compulsory step is its signing. Figure 1.2 shows the APK build process and the possible obfuscation manipulations which are optional during the build stages

[4] [2]
für 4.1 erklären, bild `http://developer.android.com/tools/building/index.html` `http://newandroidbook.com/files/ArtOfDalvik.pdf` seite 10

Aufbau APK erklären

many steps and tools until the APK is build and ready to be deployed
applications are written in the Java programming language by the developer, code is available in .class files
step 1: Java files which will be compiled into .class files by a Java Compiler, similar to a Java program build process, class files contain Java bytecode representing the compiled application, optional step apply a Java Obfuscator
step 2: transformation from Java bytecode into Dalvik bytecode, see oben, dx programm from android sdk (due to it is necessity for building an application for the Android platform), output saved in singel .dex file, included in an APK in next step, possible to apply a further obfuscator operating on Dalvik bytecode(ERKLÄRUNG)
step 3: packing and signing the APK, ApkBuilder constructs an apk file out of the "classes.dex" file and adds further resources like images and ".so" files, ".so" files are shared objects which contains native functions that can be called from within the DVM, jarsigner adds developers signature to APK(ERKLÄREN WARUM SIGNATURE NÖTIG UND WO GEPRÜFT), now the app can be installed

Android applications are written in the Java [11] programming language and deployed as files with an ".apk" suffix, later called APK. It is basically a ZIP-compressed file and contains resources of the application like pictures and layouts as well as a dex file
This dex file, saved as "classes.dex", contains the program code in form of Dalvik bytecode. Further explanations on this bytecode format are given in section 3.2. The content of the APK is also cryptographically signed, which yields no security improvement but helps to distinguish and confirm authenticity of different developers of Android applications.
DIe apk kann dann per adb, market oder direkt installiert werden
Within the installation process, every installed application gets its own unique user ID (UID) by default. This means that every application will be executed as a separate

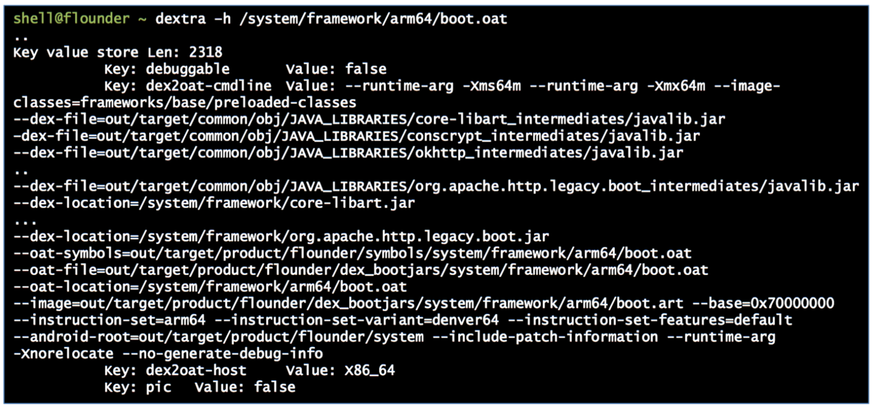
system user. -see- QUELLE/SINN?

so why is dalvik deprecated? JIT is slow, consuming both cycles and battery power gabage collection causes hangs/jitters dalvik is 32bit, cannot beneift from 64bit architecture kitkat first introduce art, lollipop adopts it

2.2.5 ART

art introduced in kitkat 4.4, available only through developer options, declared to be preview release, own risk, very little documentation, if any in lollipop art becomes runtime of choice, supersedes dalvik, breaks compatibility with older dex, as well as itself, very little docu constantly evolving through marshmallow, major caveat : oftenc changes in between android minro versions, android optimizes apps everytime you update
art was designed to address shortcomings of dalvik: virutal machine maintenance is expensie, interpreter/jit simply arent efficient as native code, doing jit all over again on every execution is wasteful, maintenance threads require significantly mroe cpu cycles, cpu cycles translate to slower performance and shorter battery life dalvik garbage collection frequently causes hangs/jitter virtual machine architecture is 32bit only, android i sfollowing ios into 64bit space
advantages of art art mvoes compilation from Just-In-Time (JIT) to Ahead-Of-Time (AOT) VM maintenance not as expensive as dalvik, art compiles to native AOT not JIT, less maintenance threads and overhead cycles than dalvik garbage collection parallizable (foreground/background), non blocking -see- QUELLE 64bit bus some issues still exist -see- quelle
main idea of art/aot: actually compilation can be to one of two types, quick (native code), portable(llvm code) in practice preference is to compile to naitve code, portable implies another layer of IR (LLVM's bit code)
Art itself: art uses not one but two file formats art - only one file, boot.art, in /syste/framework/architecture (arm,...) oat - master file, boot.oat, in /syste/framework/architecture (arm,...) - odex files no longer optimized dex but oat, alongeside apk for system apps/frameworks, /data/dalvik-cache for 3rd party apps, still uses odex extension, now file format is elf/oat
art files is a proprietary format, poorly documented, changed format internally repeatedly art file maps in memory right before oat, which links with it contains pre-initilited classes, objects and support structures
ART/OAT files are created (on device or on host) by dex2oat art still optimizes dex but uses dex2oat instead, odex files are actually oat files (elf shared objects WAS IST DAS), actual dex payload buried deep inside command line saved inside oat file's key value

store



```
shell@flounder ~ dextra -h /system/framework/arm64/boot.oat
..
Key value store Len: 2318
           Key: debuggable       Value: false
           Key: dex2oat-cmdline  Value: --runtime-arg -Xms64m --runtime-arg -Xmx64m --image-
classes=frameworks/base/preloaded-classes
--dex-file=out/target/common/obj/JAVA_LIBRARIES/core-libart_intermediates/javalib.jar
-dex-file=out/target/common/obj/JAVA_LIBRARIES/conscrypt_intermediates/javalib.jar
--dex-file=out/target/common/obj/JAVA_LIBRARIES/okhttp_intermediates/javalib.jar
..
--dex-file=out/target/common/obj/JAVA_LIBRARIES/org.apache.http.legacy.boot_intermediates/javalib.jar
--dex-location=/system/framework/core-libart.jar
...
--dex-location=/system/framework/org.apache.http.legacy.boot.jar
--oat-symbols=out/target/product/flounder/symbols/system/framework/arm64/boot.oat
--oat-file=out/target/product/flounder/dex_bootjars/system/framework/arm64/boot.oat
--oat-location=/system/framework/arm64/boot.oat
--image=out/target/product/flounder/dex_bootjars/system/framework/arm64/boot.art --base=0x70000000
--instruction-set=arm64 --instruction-set-variant=denver64 --instruction-set-features=default
--android-root=out/target/product/flounder/system --include-patch-information --runtime-arg
-Xnorelocate --no-generate-debug-info
           Key: dex2oat-host     Value: X86_64
           Key: pic   Value: false
```

Figure 2.5: oat

art file format

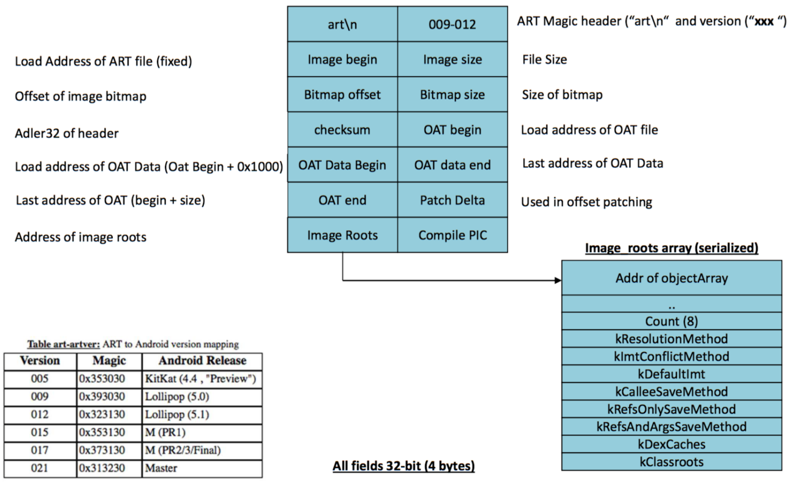


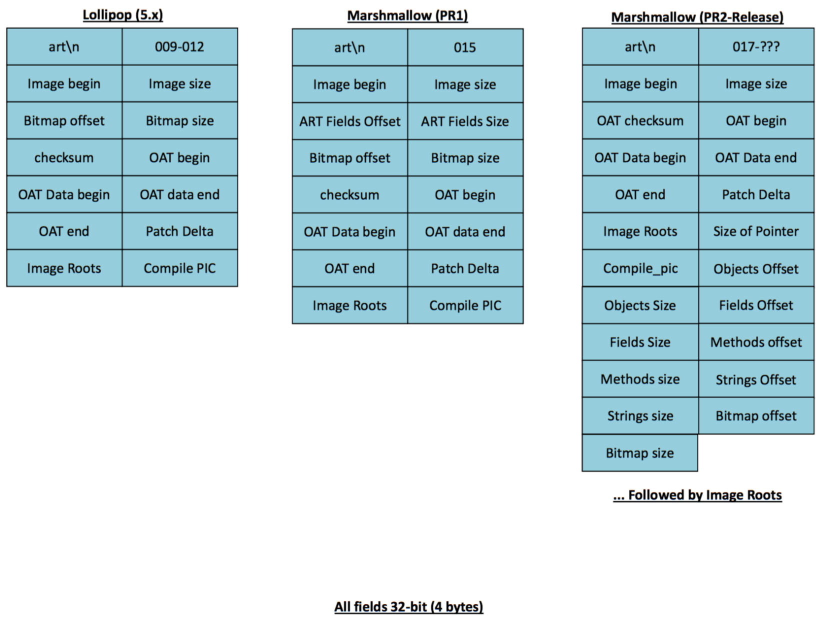
Table art-artver: ART to Android version mapping

Version	Magic	Android Release
005	0x353030	KitKat (4.4 , "Preview")
009	0x393030	Lollipop (5.0)
012	0x323130	Lollipop (5.1)
015	0x353130	M (PR1)
017	0x373130	M (PR2/3/Final)
021	0x313230	Master

Figure 2.6: art

the oat dexfile header oat headers are 1...n dex files, actual value given by dexfilecount field in header

finding dex in oat odex files will usually have only one (=original) dex embedded booat.oat is something else entirely, some 14 dex files the best of the android framework jars, each dex contains potentilly hundres of classes



Lollipop (5.x)

art\n	009-012
Image begin	Image size
Bitmap offset	Bitmap size
checksum	OAT begin
OAT Data begin	OAT data end
OAT end	Patch Delta
Image Roots	Compile PIC

Marshmallow (PR1)

art\n	015
Image begin	Image size
ART Fields Offset	ART Fields Size
Bitmap offset	Bitmap size
checksum	OAT begin
OAT Data begin	OAT data end
OAT end	Patch Delta
Image Roots	Compile PIC

Marshmallow (PR2-Release)

art\n	017-???
Image begin	Image size
OAT checksum	OAT begin
OAT Data begin	OAT Data end
OAT end	Patch Delta
Image Roots	Size of Pointer
Compile_pic	Objects Offset
Objects Size	Fields Offset
Fields Size	Methods offset
Methods size	Strings Offset
Strings size	Bitmap offset
Bitmap size	

... Followed by Image Roots

All fields 32-bit (4 bytes)

Figure 2.7: art2

art code generation oat method headers point to offset of native code each method has a quick or portable method header, contains mapping from virtual register to underlying machine registers each method has a quick or protable frame info, provides frame size in bytes, core register spill mask, fp register spill mask generated code uses unusual registers, especially fond of using lr as call register, still saves/restores registers so as not to violate arm conventions

art supports mutliple architectures(x86,arm/64,mips) compiler is layered architecture, using portable (llvm) adds another lvl with llvm bitcode (not in this scope)
vergleich java/dex/odex(art) code

lessons base code is dex so vm is still 32bit, no 64bit registers or operands so mapping to underlying arch inst always 64bit, there are actually a frw 64bit instructions but most dex code doesnt use them generated code isnt always that efficient, not on same par as an optimizing antive code compiler, likely to improve with llvm optimizations overall code (determined by Mir optimizations) flow is same garbage collection, register maps, likewase same cavears: not all methods guaranteed to be compiled, reversing can be quite a pain

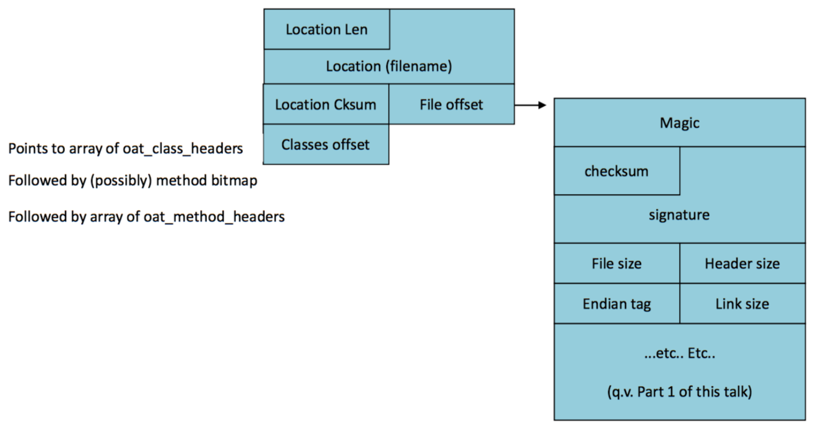


Figure 2.8: oatdex

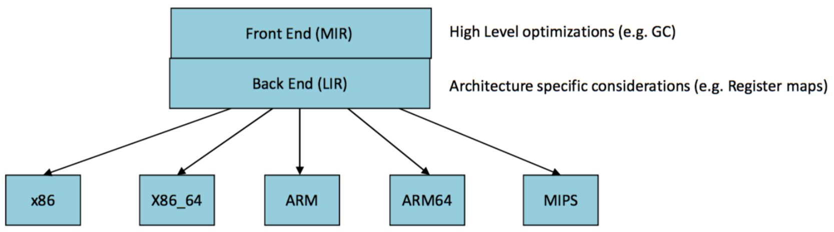

Front End (MIR)
Back End (LIR)
High Level optimizations (e.g. GC)
Architecture specific considerations (e.g. Register maps)
x86
X86_64
ARM
ARM64
MIPS

Figure 2.9: artarch

ART runtime threads the daemon threads are started in java by libcore, daemon class wraps thread class and provides singleton INSTANCE, do same basic operations as they did in "classic" dalvikVM, libart subree in libcore implementation slightly different

isn't android all dalvik now? art is runtime but application compile into dex, art is compiled on device during install, art binaries has dalvik embedded, some methods may be left as dex to be interpreted, dalvik is miuch easier to debug than art –see-evaluation
[5]

2.2.6 Installation, Running and Original Copy Protection

This is my real text! Rest might be copied or not be checked!

Installation two steps: primary is the APK verification and secondary is the bytecode optimization
legitimate signature as well as correct classes.dex structure cannot be verified are rejected for installation by the OS
Once verified, the .dex file is forwarded for optimization: a necessary step due to the high diversity of Android running hardware (dex)-see- Dalvik executable is a generic file format which needs additional processing to achieve best performance for the concrete device architecture (odex)
optimization
step removes the classes.dex from the original APK archive and loads in memory the .odex file upon execution, occurs only once, during the initial run of the application which explains the usually slower first application launch comparing to the subsequent ones

ablauf starten von app
When an Android application is executed, the process consists of the following four parts: • Dalvik bytecode, which is located in the dex file • Dalvik Virtual Machine [13], which executes the Dalvik bytecode • Native Code, like shared objects, which is executed by the processor • Android Application Framework, which provides services for the application
[4]

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

Im original vom Markt direkt rutnergeladen und dann wird sie an den ort geschoben und kann nicht mehr zugegriffen werden -see- rechte etc, QUELLE

2.2.7 Root on Android

This is my real text! Rest might be copied or not be checked!

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 is my real text! Rest might be copied or not be checked!

Since the original approach of subsection 2.2.6 was voided, another method had to be introduced. auch für den markt leiter ist es interessant -see- gründe warum es für ihn gut ist große player sind google (android ökosystem, largest market place, by users, apps, money), amazon (eigenes ökosystem, pushen mit underground und billigen tablets die als einstieg dienen und app store soll generieren), samsung (wollen was besonderes sein, wie apple, spezielle apps für galaxy/note devices, haben auch eigene services), gibt auch chinesische aber die nicht genau betrachtet da nur in china relevanz und westliche eher auf westlichen market (vllt besonderheiten) each is

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

This is my real text! Rest might be copied or not be checked!

amazon drm kiwi unique, da wrapper

Implementation

This is my real text! Rest might be copied or not be checked!

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

This is my real text! Rest might be copied or not be checked!

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

Example

This is my real text! Rest might be copied or not be checked!
anhand eigener app

2.3.2 Google

gradThis is my real text! Rest might be copied or not be checked!
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

This is my real text! Rest might be copied or not be checked!
sis is text

Functional Principle

This is my real text! Rest might be copied or not be checked!
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

This is my real text! Rest might be copied or not be checked!

anhand eigener app

2.3.3 Samsung

This is my real text! Rest might be copied or not be checked!
Zirconia `http://developer.samsung.com/technical-doc/view.do?v=T000000062L`

Implementation

This is my real text! Rest might be copied or not be checked!
sis is text `http://developer.samsung.com/technical-doc/view.do?v=T000000062L`

Functional Principle

This is my real text! Rest might be copied or not be checked!
sis is text
was sind voraussetzungen?

Example

This is my real text! Rest might be copied or not be checked!
anhand eigener app
`http://developer.samsung.com/technical-doc/view.do?v=T000000062L`

2.4 Code Analysis

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.

recover the original code of an application bytecode analysis is most often used. By applying both dynamic and static techniques to detect how behavior is altered dynamic analysis during runtime, static raw code, done by automatic tools using reverse engineering algorithms, best case whole code recovered, worst case none

When speaking of reverse engineering an Android application we mostly mean to reverse engineer the bytecode located in the dex file of this application.

The classes.dex file is a crucial component regarding the application's code security because a reverse engineering attempt is considered successful when the targeted source code has been recovered from the bytecode analysis. Hence studying the DEX file format together with the Dalvik opcode structure is tightly related to both designing a powerful obfuscation technique or an efficient bytecode analysis tool.

[4]

dex to java .dex and .class are isomorphic dex debug items map dex offsets to java line numbers tools like dex2jar can easily decomile from dex to a jar extremly useful for reverse engineering, even more so useful for malice
flow from dex to java is bidirectional, decompile code back to java, remove annoyances

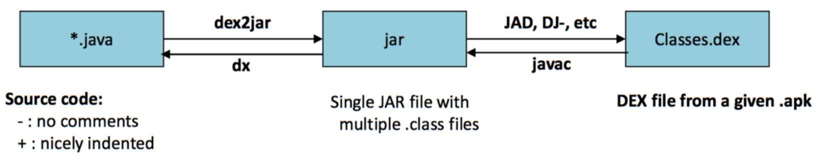


Figure 2.10: re1

like ads, registration, uncover sensitive data (app logic, secrets), replace certain classes with others (malicuous ones), recompile back to jar, then dex, put cloned/trojaned version of your app on play or other market
[5]

gaining information about a program and its implementation details, process aims at enabling an analyst to understand the concrete relation between implementation and functionality, optimal output of such a process would be the original source code of the application, not possible in general
Therefore, it is necessary for such a process to provide on the one hand abstract information about structure and inter-dependencies and on the other hand result in very detailed information like bytecode and mnemonics that allow interpretation of implementation

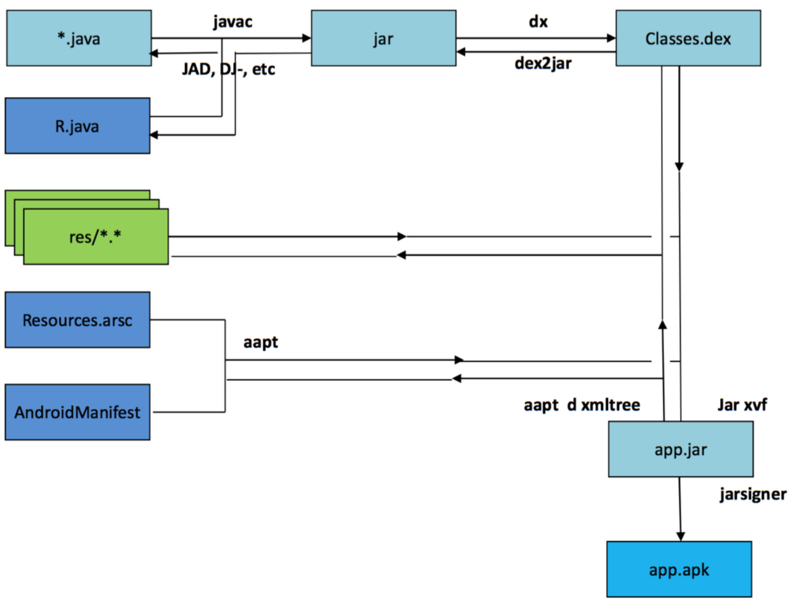


Figure 2.11: re2

hoffentlich starting points für investigations
java, e.g. read the program code faster

was ist reengineering? wie funktioniert es? was ist das ziel?
reverse engineering process makes use of a whole range of different analysis methodologies and tools.
only consider static analysis tools

IN ORDER TO GET FULL OVERVIEW DEX/SMALI/JAVA -see- WARUM?

WAS MACHEN DIE TOOLS IM ALLGEMEINEN? WOZU BENUTZEN WIR SIE?

`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 ÏicenseTeständ 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 -f1́ example 2 enthällt return von 1
adb pull /data/app/me.neutze.licensetest-1/ /Downloads
dann auf verschiedenen leveln anschauen mit den folgenden tools

Dex

This is my real text! Rest might be copied or not be checked!
nur dex weil die apps im moment so vorliegen
aosp-supplied dexdumo to disassemble dex
[5]
code wie er vorliegt, wenn was geändert wird wird es hier geändert
RESULT OUTPUT

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

Code Snippet 2.1: Quelle

SCRIPT (LISTING BENUTZEN UND RICHTIGE APP)
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
additional features?
WARUM SCHAUEN WIR ES UNS AN?
wo findet man es?
welches level?
vorteil
blabla aus dem internet

Smali Code

This is my real text! Rest might be copied or not be checked!

smali, most popular Dalvik bytecode decompilers (used by multiple reverse engineering tools as a base disassembler, amongst which is the also well-known apktool) [4]

stichwort mnemonics, eine seite dex und auf der anderen seite smali, dex bytecode vs smali, Only a few pieces of information are usually not included like the addresses of instructions
unintuitive representation, deswegen smali mit corresponding mnemonics
mnemonics and vice versa is available due to the bijective mapping
correct startaddress and offset can be challenging. There are two major approaches: linear sweep disassembling and recursive traversal disassembling, The linear sweep algorithm is prone to producing wrong mnemonics e.g. when a assembler inlines data so that instructions and data are interleaved. The recursive traversal algorithm is not prone to this but can be attacked by obfuscation techniques like junkbyte insertion as discussed in section 4.4. So for obfuscation, a valuable attack vector on disassembling is to attack the address finding step of these algorithms

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

The smali [7] program is an assemblerhas own disassembler called "baksmali"
can be used to unpack, modify, and repack Android applications
interesting part for obfuscation and reverse engineering is baksmali. baksmali is similar to dexdump but uses a recursive traversal approach to find instructions
vorteil? -see- So in this approach the next instruction will be expected at the address where the current instruction can jump to, e.g. for the "goto" instruction. This minimizes some problems connected to the linear sweep approach. baksmali is also used by other reverse engineering tools as a basic disassembler

RESULT OUTPUT: selbe wie dex, jedoch human readable, no big difference, nebeneinanderstellung dex/smali

SCRIPT (LISTING BENUTZEN UND RICHTIGE APP)

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
additional features?
WARUM SCHAUEN WIR ES UNS AN?
wo findet man es?
welches level?
vorteil
blabla aus dem internet

Java

This is my real text! Rest might be copied or not be checked!

probleme des disassemlbers erklären
interpretations sache
deswegen zwei compiler
unterschiedliche interpretation resultiert in flow und auch ob sies können ist unterschiedlich

ectl unterschiede/vor-nachteile
ggf bezug zu DALVIK/buildprocess (Java wird disassembled und dann assembler)

Androguard

This is my real text! Rest might be copied or not be checked!

An analysis and disassembling tool processing both Dalvik bytecode and optimized bytecode
DAD which is also the fastest due to the fact it is a native decompiler, WAS ist dad? ERKLÄREN? .dex files was performed with DAD, the default disassembler in the androguard analysis tool, largest successful disassembly ratio
underlying algorithm is recursive traversal
androguard has a large online open-source database with known malware patterns [4]

`https://github.com/androguard/androguard`

powerful analysis tool is Androguard
includes a disassembler and other analysis methods to gather information about a program
Androguard helps an analyst to get a good overview by providing call graphs and an

interactive interface -see- habe nur CLI benutzt
The integrated disassembler also uses the recursive traversal approach for finding instructions like baksmali, see section 2.2
one most popular analysis toolkits for Android applications due to its big code base and offered analysis methods -see- quelle, warum

RESULT OUTPUT code Listing
SCRIPT (LISTING BENUTZEN UND RICHTIGE APP)
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
additional features?
WARUM SCHAUEN WIR ES UNS AN?
wo findet man es?
welches level?
vorteil
blabla aus dem internet

jadx

This is my real text! Rest might be copied or not be checked!
RESULT OUTPUT code Listing
SCRIPT (LISTING BENUTZEN UND RICHTIGE APP)
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
additional features?
WARUM SCHAUEN WIR ES UNS AN?

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

Es gibt noch mehr tools, wurden angewendet und verglichen, aber diese waren die haupttools und haben ihren dienst erfüllt

Comparison of Code

This is my real text! Rest might be copied or not be checked!
vergleich gibts guten einblick was geändert wurde und wie es auf dem gegebenem lvl funtkioniert

vergleich von original und modifizierten code einer apk auf einer code ebene
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

This is my real text! Rest might be copied or not be checked!

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?

This is my real text! Rest might be copied or not be checked!

wer hat ihn geschrieben?
auf welcher version basiere ich
su nicht vergessen
was kann er alles
was schauen wir uns an?
erklären wie man ihn gestestet hat, woher die apps, nachgefragt ob ok etc

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 -see- have root -see- open lucky -see- 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

This is my real text! Rest might be copied or not be checked!

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?

This is my real text! Rest might be copied or not be checked!
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?

This is my real text! Rest might be copied or not be checked!
was greift jedes pattern an? wie wird der mechanismus ausgeklingt? was ist das result?

3.5 Learnings from LuckyPatcher

This is my real text! Rest might be copied or not be checked!
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 -see- 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

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN DASS IM PROGRAMMIER PROZESS IMPLEMENTIERT

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 –see- 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

This is my real text! Rest might be copied or not be checked!
WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

This is my real text! Rest might be copied or not be checked!

WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

```
public static boolean findBinary(Context context, final String binaryName) {
    boolean result = false;
    String[] places = {
            "/sbin/",
            "/system/bin/",
            "/system/xbin/",
            "/data/local/xbin/",
            "/data/local/bin/",
            "/system/sd/xbin/",
            "/system/bin/failsafe/",
            "/data/local/"
    };

    for (final String where : places) {
        if (new File(where + binaryName).exists()) {
            result = true;
            android.os.Process.killProcess(android.os.Process.myPid());
        }
    }

    return result;
}
```

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

This is my real text! Rest might be copied or not be checked!
WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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`

```
    public static boolean checkInstall(final Context context) {
        boolean result = false;
        String[] luckypatcher = new String[]{
                // Lucky patcher
                "com.dimonvideo.luckypatcher",
                // Another lucky patcher
                "com.chelpus.lackypatch",
                // Black Mart alpha
                "com.blackmartalpha",
                // Black Mart
                "org.blackmart.market",
                // Lucky patcher 5.6.8
                "com.android.vending.billing.InAppBillingService.LUCK",
                // Freedom
                "cc.madkite.freedom",
                // All-in-one Downloader
                "com.allinone.free",
                // Get Apk Market
                "com.repodroid.app",
                // CreeHack
                "org.creeplays.hack",
                // Game Hacker
                "com.baseappfull.fwd"
        };

        for (String string : luckypatcher) {
            if(checkInstallerName(context, string)){
                result = true;
            }

            if (result) {
                android.os.Process.killProcess(android.os.Process.myPid());
            }
        }

        return result;
    }

    private static boolean checkInstallerName(Context context, String string) {
        PackageInfo info;
        boolean result = false;

        try {
            info = context.getPackageManager().getPackageInfo(string, 0);

            if (info != null) {
                android.os.Process.killProcess(android.os.Process.myPid());
                result = true;
            }

        } catch (final PackageManager.NameNotFoundException ignored) {
        }

        if (result) {
            android.os.Process.killProcess(android.os.Process.myPid());
        }
        return result;
    }
}
```

Code Snippet 4.3: Partial Listing

4.1.4 Sideload Detection

This is my real text! Rest might be copied or not be checked!
WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

```
public class Sideload {
    private static final String PLAYSTORE_ID = "com.android.vending";
    private static final String AMAZON_ID = "com.amazon.venezia";
    private static final String SAMSUNG_ID = "com.sec.android.app.samsungapps";

    public static boolean verifyInstaller(final Context context) {
        boolean result = false;
        final String installer = context.getPackageManager().getInstallerPackageName(context.
            getPackageName());

        if (installer != null) {
            if (installer.startsWith(PLAYSTORE_ID)) {
                result = true;
            }
            if (installer.startsWith(AMAZON_ID)) {
                result = true;
            }
            if (installer.startsWith(SAMSUNG_ID)) {
                result = true;
            }
        }
        if(!result){
            android.os.Process.killProcess(android.os.Process.myPid());
        }

        return result;
    }
```

Code Snippet 4.4: Partial Listing

4.1.5 Signature

This is my real text! Rest might be copied or not be checked!
`http://developer.android.com/tools/publishing/app-signing.html`
`http://forum.xda-developers.com/showthread.php?t=2279813&page=5`

CONTRA

Local Signature Check

This is my real text! Rest might be copied or not be checked!
WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

```
    public static boolean checkAppSignature(final Context context) {
        //Signature used to sign the application
        static final String mySignature = "...";
        boolean result = false;

        try {
            final PackageInfo packageInfo = context.getPackageManager().getPackageInfo(context.
                getPackageName(), PackageManager.GET_SIGNATURES);

            for (final Signature signature : packageInfo.signatures) {
                final String currentSignature = signature.toCharsString();
                if (mySignature.equals(currentSignature)) {
                    result = true;
                }
            }
        } catch (final Exception e) {
            android.os.Process.killProcess(android.os.Process.myPid());
        }

        if (!result) {
            android.os.Process.killProcess(android.os.Process.myPid());
        }

        return result;
    }
```

Code Snippet 4.5: Partial Listing

Remote Signature Verification and Remote Code Loading

This is my real text! Rest might be copied or not be checked!
WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN DASS IM PROGRAMMIER PROZESS IMPLEMENTIERT

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

4.2.1 Modify the Library

This is my real text! Rest might be copied or not be checked!

google

WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

4.2.2 Checken ob ganzer code abläuft und dann nacheinander elemente aktivieren

This is my real text! Rest might be copied or not be checked!

WAS IST DIE IDEE DAHINTER? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

master1 - testen

damit die ganzen blöcke durchlaufen werden müssen

4.2.3 Native Implementierung

This is my real text! Rest might be copied or not be checked!

Library native
native ist nicht in dex -see- besser?
so nur schnittstelle? dann einfach sozusagen die schnittstelle faken

luckypatcher tauscht so ganz aus bzw modifizeirt dex
ggf einfach .so library austauschen, aber großer aufwand zu kompilieren davor

4.3 Preventing Reengineering

This is my real text! Rest might be copied or not be checked!

Reverse engineering and code protection are processes which are opposing each other, neither classified as good nor bad
"good" developer: malware detection and IP protection
"bad" developer: analysis for attack and analysis resistance

[4]

Application developers are interested in protecting their applications. Protection in this case means that it should be hard to understand what an application is doing and how its functionalities are implemented.

Reverse engineering of Android applications is much easier than on other architectures -see- 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 -see- 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-malwa`
`html`

4.3.1 Basic Breaks for Common Tools

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

pros and cons sagen?
`https://github.com/strazzere/APKfuscator`
`http://www.strazzere.com/papers/DexEducation-PracticingSafeDex.pdf`
`https://youtu.be/Rv8DfXNYnOI?t=811`

Filesystem

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

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

Inject bad OPcode or Junkbytes

This is my real text! Rest might be copied or not be checked!

viele schlagen vor: junkbyte injection well known technique in x86, confuse disassemblers in a way that they produce disassembly errors and disallow correct interpretations, inserting junkbytes in selected locations within the bytecode where a disassembler expects an instruction, junkbyte must take the disassembling strategy into account in order to reach a maximum of obfuscated code, break the two disassebly stragien from 2.4.0, condition for the location is that the junkbyte must not be executed, because an execution would result in an undesired behavior of the application, junkbyte must be located in a basic block which will never be executed [7]

funktioniert nicht mehr This is not expected to work. The bytecode verifier explicitly checks all branches for validity. The question of whether or not an address is an instruction or data is determined by a linear walk through the method. Data chunks are essentially very large instructions, so they get stepped over.

You can make this work if you modify the .odex output, and set the "pre-verified" flag on the class so the verifier doesn't examine it again – but you can't distribute an APK that way.

This "obfuscation" technique worked due to an issue in dalvik. This issue was fixed somewhere around the 4.3 timeframe, although I'm not sure the first released version that contained the fix. And lollipop uses ART, which never had this issue.

Here is the change that fixed this issue: `https://android-review.googlesource.com/#/c/57985/` `http://stackoverflow.com/questions/33110538/junk-byte-injection-in-android`
ERWÄHNEN WO IM PROZESS ANGEWENDET

JUNKBYTES
use bad opcode in deadcode
code runs but breaks tools
put it into a class you do not use –see- care proguard, it will not use it since it is not included
–see- 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

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

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

Increase headersize

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

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

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

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

This is my real text! Rest might be copied or not be checked!
(a) at source code and (b) bytecode level, Most existing open-source and commercial tools work on source code level
Java code is architecture-independent giving freedom to design generic code transformations. Lowering the obfuscation level to bytecode requires the algorithms applied to be tuned accordingly to the underlying architecture
[4]
a few dex obfuscators exist, with different approaches proguard or sdex, rename methods, field and calss names – break down string operations so as to chop hard coded strings or encrypt – can use dynamic class loading (dexloader classes to impede static analysis) can add dead code and dummy lopps (minor impact of performance) can also use goto into other onstructions EVAULATIOn in practice quite limited due to: reliance on android framework apis (remain unobfuscated) jdwp and application debuggability at the java lvl if dalvik can execute it, so can a proper analysis tool (e.g.

IDA, dextra) popular enough obfuscators (dexguard) have deobfuscators this is why JNI is so popular

[5]

ERWÄHNEN WO IM PROZESS ANGEWENDET

Obfuscators/Optimizors definition
Obfuscation techniques are used to protect software and the implemented algorithms designed to make reverse engineering harder and more time consuming, hin und her zwischen obfuscation und reverse engineering techniken
obfuscation techniques must not alter the behavior of programs, often only target specific reverse engineering steps, few general protection schemes, possible slower execution, not topic here, just examples for obfuscation applications

remove dead/debug code
potentially encrypt/obfuscate/hide via reflection
`https://youtu.be/6vFcEJ2jg0w?t=243`

definition obfuscation, was macht es, wie funktioniert es, wer hat es erfunden, wie wendet man es an

"hard to reverse engineer" but without changing the behavior of this application, was heißt hardto reverse

parallele zu disassembler ziehen

Obfuscation cannot prevent reverse engineering but can make it harder and more time consuming. We will discuss which obfuscation and code protection methods are applicable under Android and show limitations of current reverse engineering tools

The following optimizers/obfuscators are common tools. (dadrin dann verbreitung preis etc erklären)

Proguard

This is my real text! Rest might be copied or not be checked!

A Java source code obfuscator [30]. ProGuard performs variable identifiers name scrambling for packages, classes, methods and fields. It shrinks the code size by automatically removing unused classes, detects and highlights dead code, but leaves the developer to remove it manually [4] `https://youtu.be/6vFcEJ2jgOw?t=419`

WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

identifier mangling, ProGuard uses a similar approach. It uses minimal lexical-sorted strings like a, b, c, ..., aa, ab, original identifiers give information about interesting parts of a program, Reverse engineering methods can use these information to reduce the amount of program code that has to be manually analyzed -see- neutralizing these information in order to prevent this reduction, remove any meta information about the behavior, meaningless string representation holdin respect to consistence means identifiers for the same object must be replaced by the same string, advantage of minimizing the memory usage, e development process in step "a" or step "b"
string obfuscationa, string must be available at runtime because a user cannot understand an obfuscated or encrypted message dialog, information is context, other is information itself, e.g. key, url, injective function and deobfuscation stub which constructs original at runtime so no behaviour is changed, does not make understanding harder since only stub is added but reduces usable meta information
[7]
EVALUATION: dynamic analysis beats this and read directly from memory

ProGuard is an open source tool which is also integrated in the Android SDK `http://proguard.sourceforge.net/` `http://developer.android.com/sdk/index.html`
was ist proguard? was macht er? -see- ProGuard is basically a Java obfuscator but can also be used for Android applications because they are usually written in Java // feature set includes identifier obfuscation for packages, classes, methods, and fields
was kann er noch? -see- 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.
easy integration -see- how

`http://developer.android.com/tools/help/proguard.html`
optimizes, shrinks, (barely) obfuscates –see- 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 –see- stacktrace annoying
`https://youtu.be/6vFcEJ2jgOw?t=470`
–see-hacker factor 0
does not really help
googles commentar `http://android-developers.blogspot.de/2010/09/proguard-android-and-licens`
`html`

eine art result bzw zusammenfassung -see- 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.

Dexguard

This is my real text! Rest might be copied or not be checked!

A commercial Android obfuscator [37] working both on bytecode and source code level (should not be mistaken with dexguard analysis tool). Performs various techniques including strings encryprion, encrypting app resources, tamper detection, removing logging code. [4]

ERWÄHNEN WO IM PROZESS ANGEWENDET

WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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-see-automatic reflection-see-string encryption-see-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

ESULT -see- UNTERSCHIED ZU DEN VORHERIGEN -see- The obfuscation methods used in Allatori(dexguard) are a superset of ProGuards so it is more powerful but does not prevent an analyst from disassembling an Android application.

Allatori

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

Allatori 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.

cannot recognize them. 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`

RESULT -see- UNTERSCHIED ZU DEN VORHERIGEN -see- 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.

4.3.3 Protectors and Packers

This is my real text! Rest might be copied or not be checked!
from malware

APKprotect

This is my real text! Rest might be copied or not be checked!

WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

dynamic code loading best would be an application is transformed by obfuscator that it does not contain any meta information or directly interpretable bytecode, not possible because DVM would not be able to read -see- packing, often used in malware [3]
packer transforms code that a reverse engineer cannot directly extract information, e.g. by encrypting programm data no information can be extracted without decrypting, would be bad for programm as well that is why packer uses loader stub to decrypt in runtime, soultion decrypt by hand or dynamic analysis
in general two components are created, loader stub and encrypted app, on android the encrypted dex file, easier to create as the loader stub
BILD WIE ER FUNKTIONIERT step 1 load encrypted app into memory (download from server, extract from data structure, plain file available) step 2 app file is decrypted -see- original dex, can be any encryption from simple to hard, speed may slow down step 3 load decrypted dex into DVM from a bytearray, see [7] step 4 execute

[7] stellt hier basic version vor, bessere versionen ist das im folgenden

result: protection makes it hard to analyze the target application, because its bytecode is only available encrypted, decrypted version the unpacking stub has to be analyzed, great slow down, other obfuscations can be applied on stub

EVALUTAION
soultion decrypt by hand or dynamic analysis
generell: angriff nur nach 2 wenn decrypted, einfach aus memory lesen da es ja iwo hingelegt werden muss um es widerum zu laden, man muss herausfinden wo stub es herläd

ggf auch nur die eine library so laden, aber dazu muss man sie so schreiben, aber auch der cracker muss aufwand betreiben

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
This is my real text! Rest might be copied or not be checked!
WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

`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 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

This is my real text! Rest might be copied or not be checked!

WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

`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

This is my real text! Rest might be copied or not be checked!

WER HAT ES HERGESTELLT? WAS IST ES? WAS SIND DIE FEATURES? WIE FUNKTIONIERT ES? WIE WIRD ES IMPLEMENTIERT? WIE SIEHT DAS RESULT AUS (EXAMPLE BILD)

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

This is my real text! Rest might be copied or not be checked!

4.4.1 Service-managed Accounts

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

`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 -see-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 endlich durchsetzen

This is my real text! Rest might be copied or not be checked!
Evaluation Why is Android not all ART now? Your applications still compile into Dalvik (DEX) code, Final compilation to ART occurs on the device, during install, Even ART binaries have Dalvik embedded in them, Some methods may be left as DEX, to be interpreted, Dalvik is much easier to debug than ART
[5]
ERWÄHNEN WO IM PROZESS ANGEWENDET

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

This is my real text! Rest might be copied or not be checked!
ERWÄHNEN WO IM PROZESS ANGEWENDET

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

5 Evaluation of Counter Measurements

This is my real text! Rest might be copied or not be checked!

Testmethode erklären -see- LuckyPatcher anwenden -see- erfolgreich? Result in die folgenden Auswertungen einfliessen lassen

Evaluation der vorgeschlagenen punkte mit pro cons und umsetzbarkeit

`http://forum.xda-developers.com/showthread.php?t=2279813`
This is my real text! Rest might be copied or not be checked!

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 -see- 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

This is my real text! Rest might be copied or not be checked!
reengineering kann aushebeln

5.3 Prevent Reengineering

This is my real text! Rest might be copied or not be checked!

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-malw
html

BEISPIELBILDER!! This is my real text! Rest might be copied or not be checked!
take away: dex is dangerous executable format risks to app developers are significant with no clear solutions
zu ART. dex isnt dead yet, even with art still buried deep inside those oat files far easier to reverse engineer embedded dex than do so for oat
art is a far more advanced runtime architecture, brings android closer to ios native level performance vestiges of dex still remain to haunt performance, dex code is still

32bit very much still a shifting landscape, internal structures keep on changing, google isnt afraid to break compatibility, llvm integration likely to only increas eand improve for most users the change is smooth, better performance and power consumption, negligible cost binary size increase, minor limitations on dex obfuscation remain, for optimal performance and obfuscation nothing beats JNI [5]

5.4 External Improvements

sis is text

5.4.1 Service-managed Accounts

5.4.2 ART

5.4.3 Secure Elements

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

6 Conclusion

This is my real text! Rest might be copied or not be checked!

research and also a valuable market for companies
Because source code can be easier recovered from an application in comparison to x86, there is a strong need for code protection and adoption of existing reverse engineering methods. Main parts of Android application functionalities are realized in Dalvik bytecode. So Dalvik bytecode is of main interest for this topic
Also, the Android system does not prevent modification of this bytecode during runtime, This ability of modifying the code can be used to construct powerful code protection schemata and so make it hard to analyze a given application.
[7]

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

This is my real text! Rest might be copied or not be checked!

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

6.2 Discussion

This is my real text! Rest might be copied or not be checked!

sis is text `http://www.digipom.com/how-the-android-license-verification-library-is-lulling-y`
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

This is my real text! Rest might be copied or not be checked!
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

List of Code Snippets

Bibliography

[1] ChelpuS. *Lucky Patcher*. URL: `http://lucky-patcher.netbew.com/` (visited on 01/09/2016).

[2] D. Ehringer. *The Dalvik Virtual Machine Architecture*. Mar. 2010.

[3] F. Guo, P. Ferrie, and T.-c. Chiueh. "A Study of the Packer Problem and Its Solutions." English. In: *Recent Advances in Intrusion Detection*. Ed. by R. Lippmann, E. Kirda, and A. Trachtenberg. Vol. 5230. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 98–115. ISBN: 978-3-540-87402-7. DOI: `10.1007/978-3-540-87403-4_6`.

[4] A. Kovacheva. "Efficient Code Obfuscation for Android." Master's Thesis. Université de Luxembourg, Faculty of Science, Technology and Communication, Aug. 2013.

[5] J. Levin. *Dalvik and ART*. Dec. 2015.

[6] M.-N. Muntean. "Improving License Verification in Android." Master's Thesis. Technische Universität München, Fakultät für Informatik, May 2014.

[7] P. Schulz. "Code Protection in Android." Lab Course. Friedrich-Wilhelms-Universität Bonn, Institute of Computer Science, July 2012.