

Universität Bremen

Fachbereich 3 - Informatik

Erstgutachter: Dr. Karsten Sohr
Zweitgutachter: Prof. Dr. Martin Gogolla

18. April 2011

Diplomarbeit

Sicherheitsaspekte der Google Android Plattform

Erstellt durch
Stefan Klement
sklement@informatik.uni-bremen.de
Matrikelnummer: 1958299

Erklärung

Ich versichere, die Diplomarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Bremen, den 18. April 2011

Inhaltsverzeichnis

1	Einleitung	1
1.1	Vom Mobiltelefon zum Smartphone	1
1.2	Google Android	3
1.3	Ziel der Diplomarbeit	3
1.4	Kapitelübersicht	4
2	Grundlagen	5
2.1	Informationssicherheit	5
2.2	Kernel	7
2.3	Linux	8
2.4	Android	10
3	Android-Architektur	13
3.1	Linux-Kernel	14
3.2	Android Laufzeitumgebung	14
3.3	Bibliotheken	16
3.4	Anwendungsrahmen	17
3.5	Anwendungen	18
4	Sicherheitsanalyse	23
4.1	Android Kernel	24
4.2	Android Sandboxing	28
4.3	Anwendungs-Rechte	33
4.4	Dalvik Virtual Machine	39
4.5	Bibliotheken	40
4.6	Android Anwendungen	44
4.7	Android Market	46
4.8	Betriebssystem-Updates	66
5	Themennahe Arbeiten	70
6	Zusammenfassung der Angriffsvektoren	72
7	Abschlussbetrachtung und Ausblick	77
8	Literaturverzeichnis	80

Abbildungsverzeichnis

1	Android-Architektur	13
2	Rechte-Bestätigung	34
3	Android Market Startbildschirm	47
4	Anwendung-Detailansicht	48
5	Kommentare im Market	50
6	Details einer installierten Anwendung	51
7	Anwendungs-Rechte	52
8	Deinstallation	56

1 Einleitung

1.1 Vom Mobiltelefon zum Smartphone

Mobile Kommunikation hat die Welt verändert. Nach einigen ersten Schritten in den 1980er Jahren verbreiteten sich im Laufe der 90er Jahre kleine, hierzu-landes unter der Bezeichnung „Handy“ bekannt gewordene Geräte unter der Bevölkerung der Industrienationen. Aus den ersten mit skeptischen Blicken oder Kopfschütteln bedachten öffentlich durchgeführten Telefonaten wurde eine Selbstverständlichkeit. Mobiltelefone erlaubten es einem, unterwegs Termine auszumachen oder zu ändern, ohne auf die Verfügbarkeit einer Telefonzelle angewiesen zu sein, Verspätungen anzukündigen oder einfach die Wartezeit auf den Bus mit einem Gespräch zu verkürzen. Wie man dazu persönlich auch stehen mag: Jederzeit erreichbar zu sein ist eine Eigenschaft geworden, die viele Menschen von einem erwarten. Die Sprachkommunikation wurde nach einer Weile ergänzt um rudimentäre Textnachrichten in Form von SMS (Short Message Service), die den Informationsaustausch auf eine für die Umgebung weniger offenbare und beim Empfänger keiner sofortigen Reaktion bedürftigen Art und Weise regelte. Mit steigender Leistung und Speicherkapazität wurden den Geräten immer neue Funktionen hinzugefügt, die teilweise nichts mehr mit Telefon-Kommunikation zu tun hatten. Die Termine konnten nicht nur ausgehandelt, sondern danach auch gleich in einen Minikalender eingetragen werden. Simple Spiele halfen dabei, die Langeweile zu vertreiben. Sogar minimalistischer Internetzugang ließ sich bewerkstelligen, auch wenn die Benutzbarkeit sehr eingeschränkt war.

Die Möglichkeiten üblicher Mobiltelefone waren allerdings sehr beschränkt im Vergleich zum Computer zuhause. Wer wirklich den Bedarf hatte, seine Termine, Aufgaben und sein Adressbuch unterwegs zu verwalten, legte sich ein zweites, darauf spezialisiertes Gerät zu, einen PDA (Personal Digital Assistant). Zwar war es nicht besonders praktisch, gleich zwei Geräte mit sich zu führen, jedoch nahmen dies eine Reihe von Menschen auf sich.

Es war nur eine Frage der Zeit und des technischen Fortschritts, bis diese zwei Geräteklassen zu einer verschmolzen. Ende der 90er Jahre war es dann soweit: Das erste Smartphone kam auf den Markt. Klassische Sprach- und Textkommunikation vereint mit großem Display, leistungsfähiger Adress- und Kalenderverwaltung, E-Mail-Funktion, Webzugriff und Spielen, um sich die Zeit zu vertreiben, sowie die Möglichkeit, wie bei einem PC Software nachzuinstallieren.

Die gesprächigen Taschencomputer hatten anfangs nur eine begrenzte Zielgruppe. Die größere Funktionsvielfalt und Flexibilität der Smartphones konnten die meisten Menschen nicht von ihren einfacheren und günstigeren Mobiltelefonen abbringen. Smartphones beschränkten sich größtenteils auf Geschäftskunden und besonders technikverliebte Benutzer. Dennoch stiegen die Verkaufszahlen mit der Zeit an. Mit der sich verbessernden Technik erhöhten sich auch die Ansprüche der Käufer.

2007 veröffentlichte Apple das iPhone. Zwar war dies im Grunde nur ein neues Gerät unter vielen, die mediale Aufmerksamkeit und Apples Marketing sorgten allerdings dafür, dass auch Menschen, die bisher nicht groß über Smartphones nachgedacht hatten, einen Blick in diese Welt warfen. Und was sie sahen, gefiel ihnen. Mit glänzendem Äußeren, stilvollen Benutzungsoberflächen, drehenden und zoomenden Multimedia-Inhalten, einem großen Bildschirm und dem Verzicht auf eine klobige Tastatur, wurde in vielen Leuten das „Will-ich“-Gefühl geweckt. Ein Smartphone wurde zum Kultobjekt.

Die Wende vollzog sich Ende des Jahrzehnts. Der Markt der klassischen Mobiltelefone war gedeckt, erstmals gingen die Verkaufszahlen zurück. Dies galt jedoch nicht für Smartphones. Hier stiegen die Zahlen weiter unaufhörlich an, spätestens seit 2009 kann man von einem regelrechten Boom sprechen. [1]

1.2 Google Android

Im aufstrebenden Markt der Smartphones versucht nun auch Google Fuß zu fassen. Das Konzept des iPhone hat gut funktioniert, ein stark wachsender Markt von Dritthersteller-Programmen und die ungebrochen hohe Medienpräsenz bezeugen dies. Der Nachteil des iPhones ist jedoch die fehlende Auswahl und Freiheit. Es gibt nur ein Gerät, zu einem Preis, mit einer Ausstattung und zum Teil Bindung an nur einen Netzbetreiber. Hier setzt Google an. Statt selber der einzige Hardware-Hersteller zu sein, stellt Google nur ein Betriebssystem zur Verfügung, das von anderen Herstellern auf passende Geräte gespielt wird. Auf diese Weise steht eine Vielzahl an Smartphones in unterschiedlichsten Leistungs-, Ausstattungs- und Preiskategorien zur Verfügung. Das Betriebssystem, das Google dafür entwickelt hat, heißt Android.

Mit Android setzt Google seine Strategie fort, freie bzw. Open-Source-Software als Basis für die Entwicklung von Produkten zu nutzen. Schon für ihren Browser Chrome verwendeten sie die freie Browser-Engine Webkit¹. Des Weiteren unterstützt Google die Entwicklung mehrerer Open-Source-Projekte, etwa Mozilla Firefox², mit Geldmitteln oder im Rahmen ihres Summer of Code³. So setzt sich Android aus einer Reihe Open-Source-Bestandteilen zusammen, darunter der Linux-Kernel, SQLite, das erwähnte Webkit und andere. Das Ergebnis ist ein freies Betriebssystem, das von Geräteherstellern mit zusätzlicher Software erweitert und angepasst und dann in ihren Smartphones verwendet werden kann.

1.3 Ziel der Diplomarbeit

Inzwischen sind Android-betriebene Telefone seit einer Weile in verschiedensten Preisklassen käuflich erwerbbar und am Markt etabliert. Mit der Zahl der Nutzer steigt aber auch die Attraktivität der Plattform für Datendiebe und

¹<http://webkit.org/>

²<http://www.mozilla.com/>

³<http://code.google.com/soc/>

andere Menschen mit zweifelhaften Absichten. Ihr Ziel sind die Informationen, die in den Speichern der Geräte lagern, das Geld der Benutzer oder einfach der Spaß daran, diese mit Fehlfunktionen zu behindern.

Um auf solche Gefahren vorbereitet zu sein, wurden in Android eine Reihe von Sicherheitskonzepten verwirklicht. Allerdings sind diese in der offiziellen Dokumentation⁴ nur unvollständig und ungenau beschrieben, ebenso wie die Funktionsweise und das Zusammenspiel der Komponenten, aus denen Android besteht. Informationen dazu lassen sich mehreren veröffentlichten Arbeiten entnehmen, diese decken allerdings immer nur einen Teilbereich ab. Der Quellcode von Android liegt zwar offen, Erkenntnisse daraus zu gewinnen ist jedoch ohne das Wissen der Entwickler und Softwarearchitekten schwierig und zeitaufwändig.

Das Ziel der Diplomarbeit ist es, einen Überblick über die Sicherheitskonzepte von Android zu geben, sie zu analysieren, Schwächen darin aufzudecken und mögliche Angriffsvektoren herauszuarbeiten. Hierzu werden die fragmentierten Informationen zu dem Thema zusammengeführt und können als Ergänzung zur offiziellen Dokumentation dienen.

1.4 Kapitelübersicht

In Kapitel 2 werden für das Verständnis der Arbeit wichtige Informationen vermittelt und Begriffe erklärt, in Kapitel 3 wird die Architektur des Android-Systems näher beschrieben, in Kapitel 4 wird die Android-Plattform einer Sicherheitsanalyse unterzogen, in Kapitel 5 werden einige andere Arbeiten zu dem Thema vorgestellt, in Kapitel 6 werden die gefundenen Angriffsmöglichkeiten zusammengefasst und in Kapitel 7 wird ein Fazit gezogen und ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

⁴<http://developer.android.com/guide/index.html>

2 Grundlagen

In diesem Kapitel werden die für das Verständnis der Diplomarbeit wichtigen Konzepte und Begriffe eingeführt und erläutert. Kapitel 2.1 behandelt Ziele und Begriffe der Informationssicherheit, Kapitel 2.2 erklärt den Begriff des Kernels, Kapitel 2.3 stellt den Betriebssystem-Kernel Linux vor und Kapitel 2.4 gibt einen Überblick über Google Android.

2.1 Informationssicherheit

Die Informationssicherheit ist ein Bereich der Informatik. Sie beschäftigt sich mit dem Thema Sicherheit informationstechnischer Systeme im Sinne des englischen Wortes „security“. Als „sicher“ gilt dabei ein System, in dem die Manipulation oder Erlangung von Informationen nur nach vorheriger Autorisierung geschehen kann. Um ein System in einen sicheren Zustand zu bringen, ist es nötig, eine Reihe von Schutzzielen zu erfüllen. Diese lauten wie folgt:

- **Authentizität** – Authentizität bedeutet, dass ein Objekt oder Subjekt eine eindeutige, nachprüfbare Identität besitzt, anhand derer es seine Glaubwürdigkeit nachweisen kann. Im Falle eines Benutzers (Subjekt) kann das ein Passwort oder ein biometrisches Merkmal (etwa ein Fingerabdruck) sein. Ein Datenpaket oder Programm (Objekt) dagegen kann digital signiert sein, um seine Authentizität nachzuweisen. Das bedeutet, das Objekt verfügt über eine digitale Unterschrift, die mittels eines kryptographischen Verfahrens unter Verwendung eines Signaturschlüssels erstellt wurde. Nach einer Veränderung des Objekts würde die Überprüfung der Signatur fehlschlagen.
- **Datenintegrität** – Die Integrität von Daten ist dann gewährleistet, wenn eine Veränderung an ihnen nur nach erfolgter Autorisierung erfolgen kann. Zu diesem Zweck werden oft Zugriffsrechte auf die Daten definiert. Subjekte oder Objekte, die über die entsprechenden Rechte verfügen, können Änderungen vornehmen, alle anderen nicht. Weiterhin

gilt: Falls doch Änderungen unbefugt vorgenommen wurden, darf dies nicht unbemerkt bleiben, um die Weiterverarbeitung der manipulierten Daten zu verhindern.

- **Informationsvertraulichkeit** – Informationsvertraulichkeit ist gegeben, wenn nur dazu berechnigte Instanzen lesenden Zugriff auf Informationen haben. Maßnahmen, diese sicherzustellen, sind auch hier wieder die Vergabe von Zugriffsrechten sowie Verschlüsselung. Transitive Informationsweitergabe, bei der ein zugriffsberechnigtes Subjekt oder Objekt Informationen an eine dritte, unbefugte Stelle weitergibt, kann so allerdings nicht verhindert werden. Dazu sind weitergehende Informationsflusskontrollen notwendig. Die Informationsvertraulichkeit ist ein wichtiges Ziel zum Schutze der Privatsphäre der Benutzer eines Systems.
- **Verfügbarkeit** – Ein Informationssystem gewährleistet Verfügbarkeit, wenn alle Subjekte oder Objekte die ihnen erlaubten Aktionen auf dem System durchführen können. Die Verfügbarkeit kann etwa dadurch verletzt sein, dass der Speicher des Systems voll ist und keine weiteren Daten annimmt, oder dass das System unerwartet abgeschaltet wurde. Gegen die Inbeschlagnahme aller Systemressourcen können etwa Quoten eingeführt werden.
- **Verbindlichkeit** – Verbindlichkeit bedeutet, dass alle auf einem System durchgeführten Aktionen nachträglich einem Subjekt zugeordnet werden können. Dies kann zur Auffindung der Quelle einer unbefugten Informationsweitergabe wichtig sein. Die Protokollierung aller Aktionen ist eine Maßnahme, dies zu gewährleisten.

[2]

Als Angriff auf ein System wird im Verlauf der Arbeit ein Versuch bezeichnet, eines oder mehrere dieser Schutzziele zu behindern oder zu vereiteln. Ein Beispiel ist die unbefugte Erlangung privater Benutzerinformationen durch Ausnutzung einer Sicherheitslücke im System. Der Einfall ins System über die Sicherheitslücke wird als Angriffsvektor bezeichnet.

Eine besondere Art des Angriffs, die im Rahmen dieser Diplomarbeit wichtig ist, soll hier noch erwähnt werden, das „Social Engineering“. Dabei versucht der Angreifer nicht, über eine Sicherheitslücke ins System zu gelangen, sondern er versucht, einen Benutzer des Systems durch Ausnutzung menschlicher Eigenschaften, etwa mittels Täuschung oder Verwirrung, dazu zu bringen, ihm freiwillig Zugriff zu gewähren oder Informationen zukommen zu lassen. [3]

2.2 Kernel

Der Kernel ist das Herz eines Betriebssystems. Ohne ihn kann das System nicht verwendet werden. Welche Aufgaben dem Kernel selbst und welche ausgelagerten Prozessen des Betriebssystems zugeschrieben werden, hängt davon ab, ob es sich um einen Monolith-, Mikro- oder Makrokernel handelt. Monolithische Kernel integrieren besonders viele Funktionen. Beispielsweise können die Treiber für sämtliche Geräte direkt im Kernel laufen und haben damit direkten Hardwarezugriff, was Vorteile bei der Geschwindigkeit bringen kann. Nachteile dieser Art Kernel sind die hohe Komplexität und damit die höhere Anfälligkeit gegenüber Fehlern in den enthaltenen Bestandteilen. Mikrokern dagegen lagern so viele Funktionen wie möglich in separate Prozesse aus, was jedoch zusätzlichen Kommunikations-Overhead mit sich bringt. Makro- bzw. Hybridkernel sind ein Versuch, die Vorteile dieser beiden Architekturen zu vereinen, indem einige leistungskritische Teile in den Kernel verlagert werden, andere aber in externen Prozessen laufen. [4]

Unabhängig von der Art des Kernels, die grundlegenden zu erledigenden Aufgaben sind ([5]):

- Prozessverwaltung – Der Kernel startet und beendet Prozesse. Er verwaltet sie in der Weise, dass mehrere Prozesse auf einer CPU arbeiten können, indem er ihnen Rechenzeit auf der CPU zuweist (Scheduling). Er steuert außerdem die Interprozesskommunikation (IPC).

- Speicherverwaltung – Der Kernel weist Prozessen einen Speicherbereich zu, mit dem er arbeiten kann und der ausschließlich ihm gehört. Überschneidungen mit anderen Prozessen sind damit verhindert. Außerdem steuert der Kernel Lese- und Schreibvorgänge vom und in den Speicher.
- Geräteverwaltung – Um mit angeschlossenen Peripheriegeräten arbeiten zu können, stellt der Kernel Kommunikationskanäle in Form von Gerätetreibern von und zu diesen bereit. Diese verarbeiten beispielsweise Tastatureingaben oder stellen Informationen auf einem Bildschirm dar.
- Dateisystemverwaltung – Damit Daten auf einer Festplatte abgelegt oder davon gelesen werden können, muss der Speicherplatz mittels eines Dateisystems verwaltet werden. Der Kernel abstrahiert die physikalische Beschaffenheit des Mediums zu einer logischen Organisation, die dann vom Rest des Systems verwendet wird.
- Netzverwaltung – Informationen müssen oft nicht nur zwischen Prozessen, sondern auch zwischen verschiedenen Rechnern ausgetauscht werden. Die Adressierung, Weiterleitung und der Empfang entsprechender Datenströme oder -pakete über geeignete Kommunikationsgeräte wird vom Kernel übernommen.

2.3 Linux

Linux ist die Bezeichnung eines Betriebssystem-Kernels. Es wurde ursprünglich vom Finnen Linus Torvalds im Jahr 1991 entwickelt. Der Grund für die Schaffung von Linux war die Unzufriedenheit Torvalds mit den Lizenzierungsbedingungen anderer Betriebssysteme. Um ein funktionsfähiges Betriebssystem mit Linux als Kernel zu schaffen, tat Torvalds sich mit den Entwicklern des GNU-Projekts zusammen, die diverse benötigte Bibliotheken und Programme beitrugen. Da Linux nur zusammen mit der GNU-Software ein Betriebssystem bildet, wird bis heute vielfach von GNU/Linux gesprochen. Im allgemei-

nen Sprachgebrauch hat sich allerdings größtenteils Linux als Bezeichnung des ganzen Betriebssystems etabliert.

Linux steht unter der General Public License⁵ des GNU-Projekts⁶ und ist damit freie bzw. Open-Source-Software. Als solche ist es Teil vieler Betriebssystem-Paketzusammenstellungen, so genannten Distributionen, die von einer Reihe von Herstellern, etwa Red Hat, Novell oder Canonical, oder Gruppen, beispielsweise dem Debian-Projekt, herausgebracht werden. Aufgrund der zentralen Bedeutung von Linux für diese, spricht man allgemein von Linux-Distributionen. In diesen sind der Linux-Kernel, die GNU-Software und eine Reihe von anderen Programmen zu einem Paket zusammengeschnürt, das jeder Anwender auf seinem Computer benutzen kann, um eine Vielzahl an Aufgaben zu erledigen. Zu den zusätzlichen Programmen zählen oftmals eine Desktopumgebung, Office-Software, Webbrowser, E-Mail-Client oder auch Programmierertools oder Server-Software, je nachdem, auf welchen Zweck die Distribution ausgerichtet ist.

Technisch gesehen ist Linux ein monolithischer Kernel. Es sind nicht nur die grundsätzlichen Bestandteile eines Kernels wie etwa die Prozessverwaltung darin enthalten, sondern auch die Gerätetreiber. Allerdings weicht Linux ein wenig vom rein monolithischen Design ab, indem es Kernelmodule verwendet. Dies sind Teile des Kernels, die zur Laufzeit geladen und entfernt werden können. So brauchen zum Beispiel Treiber, die von verwendetem System nicht benötigt werden, nicht geladen zu werden. Dies verringert den benötigten Arbeitsspeicher.

Linux ist heutzutage weit verbreitet. Der Anteil entsprechender Distributionen am Desktop-Markt ist zwar gering, dafür ist ihre Bedeutung im Server-Bereich um so größer. Seiner Flexibilität und Skalierbarkeit hat Linux es zu verdanken, dass es außerdem sowohl im Bereich der Supercomputer als auch der eingebetteten Systeme zu finden ist. So wird es nicht nur in Android verwendet, sondern

⁵<http://www.gnu.org/licenses/gpl.html>

⁶<http://www.gnu.org/>

auch in mehreren anderen Smartphone-Betriebssystemen, etwa MeeGo⁷ und WebOS⁸. Eine große Gruppe von Entwicklern kümmert sich inzwischen um die Entwicklung des Kernels, die aber weiterhin von Linus Torvalds geleitet wird.

2.4 Android

Android ist ein Betriebssystem für mobile Endgeräte, entwickelt von Google und der Open Handset Alliance⁹ im Rahmen des Android Open Source Project (AOSP)¹⁰. Es wird im Quelltext bereitgestellt und seine Bestandteile stehen größtenteils unter verschiedenen Open-Source-Lizenzen. Verschiedenste Hersteller benutzen Android als Betriebssystem für ihre Smartphone-Produktpalette. Auch Tablet-Rechner mit Android existieren schon. Die Verbreitung auf noch weitere Geräteklassen wie Fernseher oder Netbooks ist möglich.

Android ist für die Bedienung mittels Touch-Displays ausgelegt, Eingaben erfolgen also durch Berührung des Bildschirms mit dem Finger. Texteingabe erfolgt entweder über eine Bildschirmtastatur oder eine im Gerät integrierte Hardwaretastatur. Eine Eingabe mittels Stift ist möglich, aber bei den meisten Geräten nicht vorgesehen.

Android verwendet einen auf Linux basierenden Kernel und eine Reihe von freien, quelloffenen Bibliotheken für die Wiedergabe von Bild und Ton, die Anzeige von Text und HTML-Seiten und andere Grundfunktionen. Die Anwendungen werden in der Programmiersprache Java¹¹ geschrieben und auf einer von Google entwickelten virtuellen Maschine ausgeführt, einem in Software ausgeführten Rechner, der in einem eigenen Speicherbereich auf einem echten Rechner läuft. Mehr Informationen zur Architektur finden sich in Kapitel 3

⁷<http://meego.com/>

⁸<http://developer.palm.com/>

⁹<http://www.openhandsetalliance.com/>

¹⁰<http://source.android.com/>

¹¹<http://java.com/>

auf Seite 13, ihre Sicherheit wird im Rahmen der Sicherheitsanalyse in Kapitel 4 auf Seite 23 behandelt.

Neben den vorinstallierten Anwendungen können auf einem Android-System weitere Programme nachinstalliert werden, die dessen Funktionen erweitern. Der vorgesehene Weg dazu ist die Installation über den Android Market. Dieser ist per Internet erreichbar, der Zugriff erfolgt mittels einer vorinstallierten Anwendung. Auf dem Market-Server liegen viele tausend Anwendungen, die von einer Vielzahl Unternehmen oder Einzelentwickler dort eingestellt wurden. Eine Anwendung kann mit wenigen Tastendrücken heruntergeladen und installiert werden. Der Market wird in Kapitel 4.7 auf Seite 46 näher beschrieben und untersucht. Der Begriff „Android-Plattform“ bezieht sich in dieser Arbeit auf das Betriebssystem, den Market und die Update-Infrastruktur der Geräte-Hersteller.

Android verfügt über einige grundlegende Sicherheitskonzepte zum Schutz der Benutzer, ihrer Daten und Geräte. Zum einen ist dies die Trennung der Anwendungen voneinander. Im Grundzustand kann eine Anwendung keinerlei Zugriff auf die Daten anderer Anwendungen (z.B. E-Mails), auf geschützte Systemfunktionen (z.B. Internetzugriff) oder geschützte Gerätekomponenten (z.B. eine Kamera) nehmen. Schädigende Funktionen sind somit auf den Rahmen der Anwendung begrenzt, die sie enthält. Diese Abgrenzung nennt sich „Sandboxing“ (siehe dazu Kapitel 4.2 auf Seite 28).

So auf sich beschränkt wären viele Anwendungen allerdings unbrauchbar, daher gibt es Ausnahmeregeln. Hier kommt ein weiteres Sicherheitskonzept von Android zum Tragen, die Anwendungsrechte. Jede Anwendung kann eine Reihe von Rechten anfordern, die sie zum Funktionieren braucht. Ein Webbrowser braucht beispielsweise Internetzugriff. Einige dieser Rechte werden automatisch gewährt, andere benötigen die Zustimmung des Benutzers. Welche Rechte eine Anwendung einfordert, wird von ihrem Entwickler in ihrer Manifest-Datei definiert (siehe Kapitel 3.5 auf Seite 18). Wie dieser Mechanismus abgesichert ist, wird in Kapitel 4.3 auf Seite 33 untersucht.

Ein wichtiger Teil des Sicherheitskonzept eines jeden Betriebssystems ist sein Update-Mechanismus. Im Falle von Android erhalten die Geräte (mit Ausnahme von Googles eigenen Geräten Nexus One und Nexus S) ihre Aktualisierungen nicht von Google selbst, sondern sie werden von den Geräteherstellern damit versorgt. Welche Auswirkungen das auf ihre Sicherheit hat, wird in Kapitel 4.8 auf Seite 66 näher betrachtet.

3 Android-Architektur

In diesem Kapitel wird die Architektur des Android-Betriebssystems beschrieben. Dabei wird sich an Googles eigener Architekturdarstellung orientiert, siehe Abbildung 1. Die Beschreibung wird von unten nach oben durchgeführt, vom Grundsystem bis zu den Anwendungen. In Kapitel 3.1 wird der in Android verwendete Kernel vorgestellt, in Kapitel 3.2 wird auf die Laufzeitumgebung eingegangen, in Kapitel 3.3 werden einige wichtige Bibliotheken beschrieben, in Kapitel 3.4 wird kurz auf den Anwendungsrahmen eingegangen und in Kapitel 3.5 wird dargestellt, wie Anwendungen in Android aufgebaut sind.



Abbildung 1 – Die Architektur von Android ([16])

3.1 Linux-Kernel

Google verwendet für Android den Linux-Kernel. Genauer gesagt haben sie diesen für ihre Zwecke modifiziert. Zum einen wurde er an die Bedürfnisse eines Smartphones angepasst. Treiber, die für ein solches unwichtig sind, wurden entfernt, andere, etwa für spezielle Tastaturen oder eingebaute Kameras, hinzugefügt. Solche Art Anpassungen sind beim Erstellen eines Betriebssystems auf Linuxbasis üblich, denn der Kernel ist standardmäßig mit einer breiten, aber nicht allumfassenden Auswahl an Treibern ausgestattet. Allerdings hat Google auch einige andere Teile des Kernels verändert. Auf diese wird im Rahmen der Sicherheitsanalyse in Kapitel 4.1 auf Seite 24 noch eingegangen.

Die Kernel-Versionen, mit denen Android ausgeliefert wurde, waren wie folgt ([7] [8]):

- Android 1.0 und 1.1: Linux Kernel 2.6.25
- Android 1.5: Linux Kernel 2.6.27
- Android 1.6, 2.0 und 2.1: Linux Kernel 2.6.29
- Android 2.2: Linux Kernel 2.6.32
- Android 2.3: Linux Kernel 2.6.35.7

3.2 Android Laufzeitumgebung

Die Dalvik Virtual Machine (Dalvik VM) bildet zusammen mit ihren Kernbibliotheken Androids Laufzeitumgebung. Anwendungen für Android werden in Java geschrieben. Zur Ausführung dieser Sprache wird eine virtuelle Maschine benötigt. Für mobile Geräte steht bereits die von Sun entwickelte J2ME zur Verfügung. Allerdings ist diese in ihren Möglichkeiten beschränkt. Es kann beispielsweise nicht auf die volle Anzahl an Java-Bibliotheken zugegriffen werden. Google entschied sich für einen anderen Weg. Sie implementierten eine eigene virtuelle Maschine, genannt Dalvik, und fügten ihr Klassenbibliotheken

aus dem Apache Harmony Projekt¹² hinzu. Der Grund für die Verwendung der Harmony-Bibliotheken ist die Lizenz, unter der sie veröffentlicht werden, der Apache Software License 2.0. Diese ist die von Google bevorzugte Open-Source-Lizenz, da sie zulässt, damit lizenzierte Software auch in Closed-Source-Programmen zu verwenden, anders als es bei der verbreiteten GPL der Fall wäre, die in Linux-Systemen oft zum Einsatz kommt ([10]). Dalvik VM ist keine Java-VM. Dalvik kann keinen Java-Bytecode ausführen. Die entsprechenden .class-Dateien müssen zuerst in Dalvik-Bytecode umgewandelt werden, der in .dex-Dateien (Dalvik Executable) gespeichert wird. Diese können von der VM ausgeführt werden. Die Anwendungen werden in .apk-Dateien (Android Package) ausgeliefert. Dies sind komprimierte Archive mit den benötigten .dex-Dateien und einigen anderen benötigten Bestandteilen als Inhalt. Damit sind sie die Gegenstücke zu Javas .jar-Dateien. Android verwendet aufgrund dieser Unterschiede nicht den Namen Java im Zusammenhang mit seiner Laufzeitumgebung. Auf diese Weise wollte Google Patente und Lizenzzahlungen an Sun – jetzt Oracle – umgehen. Aktuell hat Oracle dennoch eine Klage gegen Google wegen einer Patentverletzung eingereicht. Der Ausgang des Rechtsstreits ist ungewiss und liegt außerhalb des Erfassungsrahmens dieser Diplomarbeit.

Dalvik wurde in mehrerer Hinsicht auf die Bedürfnisse leistungs- und speicherbeschränkter Geräte zugeschnitten. Der Speicherverbrauch wurde etwa dadurch verringert, dass Redundanzen im Bytecode entfernt wurden. Beispielsweise steht eine Methodensignatur, die einmal in einem Interface, in der Implementation des Interfaces und bei der Benutzung der Implementation auftritt, nur einmal im Dalvik-Bytecode, statt mehrmals. Letztlich sind dadurch die unkomprimierten .dex-Dateien häufig kleiner als komprimierte .jar-Dateien mit klassischem Java-Bytecode. Auch findet „memory sharing“, also das gemeinsame Verwenden von im Speicher befindlichem Code, in hohem Maße statt. Vor Version 2.2 wurde zudem auf einen die Codeausführung beschleunigenden Just-In-Time-Compiler (JIT-Compiler) verzichtet, weil er als nicht notwendig angesehen wurde. Diese Ansicht revidierte Google allerdings offenbar und seit

¹²<http://harmony.apache.org/>

2.2 verfügt Android über einen solchen.

Der Grund, warum ein JIT-Compiler überflüssig sein sollte, ist gleichzeitig Dalviks Maßnahme zur Schonung der CPU: Native Code. Viele der aufwendigeren Berechnungen innerhalb von Android werden nicht in Form interpretierbarer Dalvik-Bibliotheken ausgeführt, sondern als Native Code, sehr schnellen C- oder C++-Bibliotheken. Unter Verwendung des Native Development Kits (NDK) können Anwendungsentwickler auch eigenen Native Code für ihre Programme erstellen und so geschwindigkeitskritische Berechnungen auslagern.

[11]

3.3 Bibliotheken

Wie in der Beschreibung von Dalvik erwähnt, werden in Android eine ganze Reihe von Bibliotheken eingesetzt, die nicht auf der virtuellen Maschine laufen, sondern Native Code darstellen, der direkt für die verwendete Hardwarearchitektur kompiliert wurde, zumeist in C oder C++. Auf diese Weise werden einige besonders rechenintensive Funktionen ausgelagert und können schneller ausgeführt werden. Dazu gehören die Oberflächenbeschleunigung des Android-Benutzungsinterfaces und die Bibliotheken zur Medienwiedergabe. Außerdem finden sich unter diesen Bibliotheken auch solche, die von separaten Open-Source-Projekten entwickelt wurden und in Android genutzt werden. Im Rahmen dieser Diplomarbeit werden nun relevante Bibliotheken, die in der spätere Sicherheitsanalyse noch Erwähnung finden, kurz vorgestellt:

- **libc**: Dies ist die Standard-C-Bibliothek von Android, in der grundlegende Funktionen und Systemaufrufe zu finden sind. Google hat Androids libc Bionic genannt. Es handelt sich nicht um die glibc, die in den meisten Linux-Distributionen zu finden ist, sondern um eine Eigenentwicklung von Google, die unter der BSD-Lizenz steht. Google versucht nach Möglichkeit, GPL-lizenzierte Software aus Android herauszuhalten, um die Verwendbarkeit mit Closed-Source-Software zu gewährleisten. Bionic

wurde mit Blick auf geringe Größe und hohe Geschwindigkeit entwickelt. Sie ist nicht völlig kompatibel mit der glibc. [12]

- Freetype: Dies ist eine auf Schlankheit und Portabilität ausgelegte Font-Rendering-Engine zur Darstellung von Schriften [13]
- SQLite: Es handelt sich um ein schlankes und schnelles eingebettetes relationales Datenbankmanagementsystem (DBMS), das von Anwendungen zur Speicherung von Daten genutzt werden kann. SQLite ist besonders für leistungsschwache Systeme ohne große Anforderungen an ein DBMS geeignet. [14]
- WebKit: Dies ist eine freie Browserengine, die ursprünglich der KHTML-Engine des KDE-Projekts entstammt und dann von Apple unter dem Namen WebKit adoptiert wurde. Google hat WebKit bereits in ihrem Browser Chrome in Verwendung. [15]

3.4 Anwendungsrahmen

Der Anwendungsrahmen stellt eine Reihe von System-Komponenten dar, welche von Anwendungen genutzt werden können. Die meisten davon sind „Manager“-Komponenten, die den Zugriff auf diverse Systemfunktionen verwalten. Der „Telephony Manager“ etwa stellt Funktionen bereit, um aus der eigenen Anwendung heraus Anrufe tätigen zu können, der „Package Manager“ ist ein zentraler Bestandteil bei der Installation und Verwaltung von Anwendungen. Die Manager sind dabei selber in Form von Anwendungen auf dem System vorhanden. Im späteren Verlauf dieser Diplomarbeit wird nur auf einzelne Bestandteile des Anwendungsrahmens eingegangen, die Content Provider und der Package Manager. Die Betrachtung aller aufgeführten Teile würde zu weit gehen.

3.5 Anwendungen

Anwendungen sind das Hauptunterscheidungsmerkmal zwischen einem normalen Mobiltelefon und einem Smartphone. Android bietet die Möglichkeit, Anwendungen nachzuinstallieren. Dadurch ist man nicht auf die vorgegebenen Möglichkeiten des jeweiligen Geräts beschränkt.

Anwendungen in Android setzen sich aus bis zu vier verschiedenen Komponenten zusammen, Activities, Services, Content Provider, Broadcast Receiver. Diese kommunizieren über Nachrichtenobjekte, genannt Intents. Die Komponenten können unabhängig voneinander aufgerufen werden. Beispielsweise kann eine Anwendung einen Service einer anderen Anwendung starten und verwenden, falls sie dazu berechtigt ist.

Activities sind die sichtbaren Teile einer Anwendung. Mit ihnen interagiert der Benutzer. Es befindet sich immer eine Activity im Fokus, während die anderen inaktiv im Hintergrund sind. Oftmals wird eine Activity nur dazu benötigt, Dateneingaben vom Benutzer entgegen zu nehmen, eine andere Activity zu starten und die Daten an diese zur Bearbeitung weiter zu reichen.

Services können ähnlich wie Activities Daten verarbeiten. Allerdings haben sie keine Benutzungsoberfläche, ein Benutzer kann mit ihnen nur über Activities kommunizieren. Ein Service hat gegenüber einer Activity den Vorteil, dass er im Hintergrund arbeiten kann. Es können mehrere Services gleichzeitig aktiv sein und verschiedene Aufgaben erledigen, während der Benutzer mit einer davon unabhängigen Activity beschäftigt ist. Beispielsweise kann ein Service Musik abspielen, während der Benutzer eine E-Mail schreibt. Ein Service kann in einem eigenen Prozess gestartet werden und kann somit weiterlaufen, wenn die zugehörige Anwendung schon beendet wurde. Zur Kommunikation mit einem solchen Service wird die IPC-Schnittstelle (Inter Process Communication-Schnittstelle) OpenBinder verwendet (siehe Kapitel 4.1 auf Seite 27).

Content Provider sind die einzige in Android vorgesehene Möglichkeit, auf dem Gerät gespeicherte Daten mehreren Anwendungen zur Verfügung zu stel-

len. Sollen etwa die Einträge in einem Adressbuch oder Multimediadateien für andere als die Ersteller-Anwendung zugreifbar sein, wird dies über eine Content Provider-Schnittstelle realisiert. Die Daten sind dabei in einer Datenbank-Struktur organisiert. Der Zugriff erfolgt mittels eines URI (Uniform Resource Identifier), ähnlich der Adressierung von Websites. Die Manipulation der Daten wird per SQL-Anweisungen vorgenommen.

Broadcast Receiver lauschen in das System und warten auf bestimmte Meldungen, die sie abonniert haben. Hat zum Beispiel das Gerät einen niedrigen Akku-Ladestatus erreicht, kann es eine ungezielte Nachricht aussenden. Ein Broadcast Receiver kann auf dieses spezielle Signal warten und es empfangen. Dann löst er eine entsprechende Aktion aus, etwa die Verringerung der Display-Helligkeit. Ein anderes Beispiel ist das Öffnen einer Datei. Hat der Benutzer etwa als Anhang einer E-Mail ein Bild geschickt bekommen und will dieses öffnen, so kann das E-Mail-Programm, statt ein bestimmtes Bildbetrachtungsprogramm zu starten, das Senden eines Broadcasts veranlassen. Dieser enthält die Aufforderung, eine Datei vom Typ „Bild“ zu öffnen. Ein Bildbetrachtungsprogramm kann einen Broadcast Receiver enthalten, der auf diese Meldung wartet. Ist es das einzige installierte Programm, das diese Datei öffnen kann, so geschieht dies. Ansonsten wird dem Benutzer eine Auswahl an Programmen präsentiert.

Diese Architektur orientiert sich grob am „Model View Controller“-Muster (MVC-Muster), wobei Model die Datenspeicherung, View die Datenpräsentation und -entgegennahme und Controller die Datenverarbeitung darstellen. Eine Activity ist eine View, kann aber zusätzlich auch als Controller fungieren, ein Service ist ein Controller, ein Content Provider ist ein Model. Broadcast Receiver lassen sich nicht in dieses Muster einordnen. [52]

Intents sind Objekte, mit denen sich Komponenten gegenseitig aufrufen können, nicht nur Anwendungsintern sondern auch zwischen verschiedenen Anwendungen. Ein Intent enthält immer eine auszuführende Aktion und eine URI, die auf die Daten weist, auf denen die Aktion ausgeführt werden soll. Optional kann es noch einen Komponentennamen, eine Kategorie, einen Typ und

Extrainformationen enthalten. Es kann explizit oder implizit sein. Ein explizites Intent wenden sich an eine mittels des Komponentennamens aufgeführte Komponente. Ein implizites Intent richtet sich an alle Komponenten, die die im Intent aufgeführten Aktion auf den genannten Daten ausführen kann. Ob eine Komponente dazu fähig ist, legt sie mittels der Angabe eines Intent-Filters in ihrer Manifest-Datei (siehe unten) fest. Empfangen kann eine Anwendung diese Art Intents mittels des erwähnten Broadcast Receivers. Alle Möglichkeiten aufzuführen, die Intents bieten, würde den Rahmen dieser Diplomarbeit überschreiten. Der offizielle Android Developer's Guide enthält eine sehr umfassende Dokumentation hierzu. [18]

Jede Anwendung enthält eine **Manifest-Datei** namens AndroidManifest.xml. Diese stellt sozusagen ihren „Ausweis“ dar. In ihr sind unter anderem der Paketname der Anwendung, ihre Version, alle Komponenten, aus denen die Anwendung besteht, die Intent-Filter, die für diese Komponenten festgelegt wurden und die Rechte, die die Anwendung verlangt, aufgeführt.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Copyright (C) 2009 The Android Open Source Project

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.wiktionary"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/app_icon" android:label="@string/app_name"
        android:description="@string/app_descrip">

        <!-- Browser-like Activity to navigate dictionary definitions -->
        <activity
```

```

        android:name=".LookupActivity"
        android:theme="@style/LookupTheme"
        android:launchMode="singleTop"
        android:configChanges="orientation|keyboardHidden">

        <intent-filter >
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter >

        <intent-filter >
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="wiktionary" android:host="lookup" />
        </intent-filter >

        <intent-filter >
            <action android:name="android.intent.action.SEARCH" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter >

        <meta-data android:name=
            "android.app.searchable" android:resource="@xml/searchable" />
    </activity>

    <!-- Broadcast Receiver that will process AppWidget updates -->
    <receiver android:name=".WordWidget" android:label="@string/widget_name">
        <intent-filter >
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter >
        <meta-data android:name="android.appwidget.provider"
            android:resource="@xml/widget_word" />
    </receiver>

    <!-- Service to perform web API queries -->
    <service android:name=".WordWidget\$UpdateService" />

</application>

<meta-data android:name=
    "android.app.default_searchable" android:value=".LookupActivity" />

<uses-permission android:name="android.permission.INTERNET" />
<uses-sdk android:minSdkVersion="3" android:targetSdkVersion="4" />

</manifest>

```

Listing 1 – Beispiel-Manifest-Datei [17]

Im Beispiel zu sehen ist eine Anwendung namens „Wiktionary“
(`package="com.example.android.wiktionary"`),
welche in Version 1.0 vorliegt
(`android:versionName="1.0"`),
eine Activity, einen Broadcast Receiver und einen Service besitzt und das Recht
auf eine Internetverbindung anfordert
(`<uses-permission android:name="android.permission.INTERNET" />`).

William Enck, Machigar Ongtang und Patrick McDaniel haben in ihrem Paper
„Understanding Android Security“ ein Werkzeug namens „Kirin“ vorgestellt,
mit welchem sich die Manifest-Dateien installierter Anwendungen untersuchen
lassen. So kann festgestellt werden, ob die Anwendungen gegen festgelegt Si-
cherheitsrichtlinien verstoßen (siehe Kapitel 5 auf Seite 70).

[53] [19]

4 Sicherheitsanalyse

In diesem Kapitel wird die Android-Plattform näher auf ihre Sicherheit hin untersucht. In Kapitel 4.1 wird der Kernel betrachtet, in Kapitel 4.2 wird Androids Sandboxing-Mechanismus analysiert, in Kapitel 4.3 wird das Rechtssystem für Anwendungen untersucht, Kapitel 4.4 beschäftigt sich mit der Dalvik Virtual Machine, in Kapitel 4.5 werden einige in Android verwendete Bibliotheken angeschaut, Kapitel 4.6 gibt Einblick in einige Mechanismen von Android-Anwendungen, in Kapitel 4.7 wird der Android Market beschrieben und untersucht, sowie mit alternativen Lösungen verglichen, Kapitel 4.8 schließlich nimmt näheren Einblick ins Updateverhalten von Google und den Geräteherstellern.

Die Untersuchung wird „bottom up“ durchgeführt, also vom Betriebssystem-Kernel aufwärts bis zu den Anwendungen und darüber hinaus zum Market und den Geräteherstellern. Bei der Analyse kamen Informationen aus einer Reihe von anderen Arbeiten zum Tragen. Ein Problem von Android stellt die Verstreuung von Informationen auf diverse Dokumente dar. Keine der zitierten Arbeiten hat alle der hier aufgeführten Aspekte untersucht.

Bei der Untersuchung wird zunächst die Funktionsweise und Bedeutung der jeweiligen Komponente erläutert, soweit noch nicht im Grundlagen- oder Architekturkapitel geschehen. Dann werden die Implikationen dieser Funktionen für die Sicherheit angeführt und mögliche Angriffsvektoren herausgestellt, soweit sie entdeckt wurden. Dabei finden auch Angriffe Erwähnung, die in der Literatur schon beschrieben oder in der Praxis bereits durchgeführt worden sind.

4.1 Android Kernel

Google verwendet Linux als Kernel für Android. Allerdings haben sie Linux in mehrerer Hinsicht an ihre Zwecke angepasst.

Auf seiner Website schreibt Linux-Kernelentwickler Greg Kroah-Hartman zu Googles Änderungen folgendes:

„The Android kernel code is more than just the few weird drivers that were in the drivers/staging/android subdirectory in the kernel. In order to get a working Android system, you need the new lock type they have created, as well as hooks in the core system for their security model.

In order to write a driver for hardware to work on Android, you need to properly integrate into this new lock, as well as sometimes the bizarre security model. Oh, and then there's the totally-different framebuffer driver infrastructure as well.“

[6]

Um einen Treiber für die spezifische Hardware eines Mobilfunkgerätes, das mit Android laufen soll, zu erstellen, müssen also Googles Anpassungen berücksichtigt werden. Solche Veränderungen werden normalerweise den Linux-Entwicklern vorgelegt und dann, wenn sie den Konventionen entsprechen, in den so genannten „Linux Staging Tree“ eingefügt, einer Art Wartezimmer für Code, der noch nicht bereit ist für die feste Aufnahme in den Hauptteil des Kernels. Der Code ist damit schon Teil des Kernels und kann benutzt werden, es erscheint allerdings eine Warnmeldung, wenn er geladen wird. Dies geschah anfangs auch mit Googles Code. Damit solcher Code jedoch im Staging Tree verbleibt, wird vorausgesetzt, dass auch an ihm gearbeitet wird, um ihn für die endgültige Aufnahme bereit zu machen. Dies hat Google versäumt. Die Konsequenz war, dass der Code mit Kernel 2.6.33 entfernt wurde. Das heißt, Google verwendet für Android nun eine in Teilen inkompatible Version von Linux. Das hat zur Folge, dass Treiber für Androidgeräte nun speziell für Googles

Variante des Kernels geschrieben werden müssen und nicht in den offiziellen Linux-Kernel gelangen können, da sie dort nicht funktionieren. Damit profitieren sie auch nicht automatisch von sicherheitsrelevanten Fehlerkorrekturen, wie es ansonsten der Fall wäre.

Diese Angelegenheit hat in der Linux-Community für einige Verstimmung gesorgt. Die Ernsthaftigkeit von Googles Unterstützung freier Software wird in Frage gestellt. Entwickler Ryan Paul behauptet sogar, Android sei kein Linux. [9]

Auf jeden Fall richtig ist, dass es nicht ohne Weiteres möglich ist, den original Linuxkernel in einer Android-Umgebung zum Laufen zu bekommen. Für herkömmliche Linux-Distributionen existieren Anleitungen, wie ein Benutzer diesen für sein System kompilieren und verwenden kann. Unter Android ist der Benutzer auf die offiziellen Kernel-Updates von Google angewiesen oder muss sich tiefer mit der Materie beschäftigen. Es werden bereits spezielle Android-Kernel auf diversen Websites angeboten. Diese zielen allerdings meist nicht darauf ab, Sicherheitslücken zu schließen, sondern die Leistung zu verbessern oder den Energiebedarf zu senken.

Trotz der Unterschiede zwischen beiden lassen sich einige Aussagen zu Linux auch auf den Android-Kernel beziehen, da ein großer Teil der Mechanismen übereinstimmt. So ist Linux eine seit Jahren stetig weiterentwickelte Software mit einer riesigen Entwicklergemeinde. Angestellte vieler großer Unternehmen wie IBM und Intel entwickeln Code für Linux. Linux wird in einer Vielzahl an Geräten eingesetzt, vom Router über Desktop-Rechner bis zu Supercomputern. Linux ist also praxiserprobt, wird sehr aktiv entwickelt und da der Quellcode jedem offen steht, haben eine Menge Menschen einen Blick darauf und suchen nach Fehlern und Sicherheitslücken. Linux scheint daher eine gute Basis zu sein, um darauf sichere Produkte zu entwickeln.

Dafür spricht die Analyse, die das Unternehmen Coverity durchgeführt hat und deren Ergebnisse sie im November 2010 veröffentlicht haben. Coverity verwendet ein Werkzeug zur statischen Analyse von Quelltext und testet damit seit

Jahren eine große Anzahl populärer Open-Source-Projekte auf Fehler, etwa Pufferüberläufe. Dazu gehörte diesmal auch der Kernel der Android-Version 2.2. Coverity stellte fest, dass die Fehlerrate im Schnitt bei 0,47 Fehlern pro 1000 Zeilen Code lag. Die höchste Fehlerrate lag dabei in den Android-eigenen Erweiterungen vor, 0,78 Fehler pro 1000 Zeilen. Die Fehlerrate in den Code-Abschnitten, die von Linux übernommen worden waren, ist also noch ein Stück besser. Dies ist ein sehr gutes Ergebnis, der Industrie-Durchschnitt liegt bei etwa einem Fehler pro 1000 Zeilen. Linux ist also, zumindest nach der Analyse von Coverity zu urteilen, eine sehr gut gepflegte Software mit geringem Fehleranteil. Allerdings lag die Rate der kritischen Fehler bei etwa 25%. Um Google die Möglichkeit und Zeit zu geben, diese zu schließen, wurden die gefunden Fehler an die Entwickler weitergeleitet und nicht öffentlich gemacht. [47] [48] [49]

Auf der anderen Seite steht die Aussage des Sicherheitsexperten Jon Oberheide, der Linux in seinem Vortrag „Android Hax“ als „Schweizer Käse“ bezeichnet. In diesem Vortrag geht er auf die Sicherheitsfunktionen von Android und mögliche Angriffsmöglichkeiten ein. Er vertritt dort die Ansicht, dass Linux eine „hervorragende Angriffsfläche“ bietet. Die Praxis scheint ihm Recht zu geben: Bisher ist noch keine Android-Version erschienen, auf der nicht nach kurzer Zeit über Sicherheitslücken im Kernel Root-Rechte (System-Administrations-Rechte) für den Benutzer freigeschaltet werden konnten. Das so genannte „Rooten“ dient bisher vor allem dazu, den Nutzern mehr Möglichkeiten zu geben, ihr Gerät an ihre Bedürfnisse anzupassen, etwa, um eigene, angepasste System-Images („Custom-ROMs“, von der Community erstellte Android-Versionen) aufzuspielen oder vorinstallierte Systemanwendungen zu deinstallieren. Allerdings ist dies auch ein Zeichen dafür, dass in jeder Android-Version mindestens eine Sicherheitslücke offen steht, über die eine Anwendung mehr Rechte erlangen kann, als der Benutzer ihr eigentlich zugestanden hatte. Eine Anwendung mit Root-Rechten kann beispielsweise die Daten, die eine andere Anwendung gespeichert hat, auslesen, etwa gesendete SMS.

Um zu zeigen, dass dies auch in der Praxis funktioniert, hat Oberheide eine

harmlos aussehende Anwendung geschrieben und im Google Market zur Installation freigestellt. Diese prüft regelmäßig im Internet, ob Code zum Ausnutzen einer Sicherheitslücke im Kernel zur Verfügung steht, lädt diesen herunter und erlangt damit Rootrechte. Die Anwendung führt keinen Schadcode aus, sie dient nur zur Demonstration. In kurzer Zeit wurde sie mehrere hundert mal heruntergeladen. Inzwischen wurde sie aus dem Market entfernt. Die Zeit hat jedoch gereicht, um zu zeigen, dass Angriffe auf Android durch Ausnutzung einer Lücke im Kernel erfolgreich durchgeführt werden können.

[43]

Ein Bestandteil, den Google für Android in den Kernel eingefügt hat, ist das Binder-System. Binder ist ein Gerätetreiber im Kernel, der für IPC (Inter-Process Communication) genutzt wird. Dabei werden Nachrichten mittels ioctl-Kommandos (Input-Output Control) von einem Prozess in den Gerätetreiber hineingeschrieben und von einem anderen wieder herausgelesen. Der Gerätetreiber ist daher von jedem Prozess beschreibbar und lesbar. Theoretisch kann die Kommunikation zwischen zwei Prozessen daher durch andere manipuliert werden. Wie der Gerätetreiber dagegen abgesichert ist, ist nicht bekannt und kann wohl nur durch ein genaues Studium des Quellcodes herausgefunden werden.

Der Binder-IPC-Mechanismus wird zum einen genutzt, um Komponenten an Services zu binden. Auf die Weise können beispielsweise regelmäßige Status-Updates von dem Service zu der Komponente, die die Verbindung initiiert hat, gesendet werden. Außerdem kommt Binder noch bei zahlreichen anderen Kommunikationen zum Einsatz, etwa bei der Abfrage von Anwendungsrechten mittels `checkPermission`.

Das Thema Binder ist sehr komplex und die Dokumentation dazu wenig hilfreich, was Einschätzungen hinsichtlich der Sicherheit angeht. Auch andere Arbeiten widmen sich dem Thema meist nur am Rande. Eine tiefer gehende Behandlung des Themas findet im Rahmen dieser Diplomarbeit nicht statt.

4.2 Android Sandboxing

Einer der zentralen Sicherheitsmechanismen, die Google in Android implementiert hat, ist die Trennung der Anwendungen voneinander, das Sandboxing.

Das Prinzip dahinter ist folgendes: Keine Anwendung hat Zugriff auf die Ressourcen einer anderen Anwendung oder des Systems, außer es wird explizit erlaubt. Alle Aktionen, die nicht im Rahmen der Anwendung selbst stattfinden, sind im Grundzustand verboten, bis sie ausdrücklich genehmigt werden. Dadurch wird theoretisch erreicht, dass eine sehr genaue Kontrolle des Informationsflusses innerhalb von Android möglich ist.

Zur Umsetzung des Sandboxing werden vorhandene Mechanismen des zugrunde liegenden Linux-Systems verwendet. Da dieses ein Mehrbenutzer-System ist, existiert eine Trennung zwischen verschiedenen Benutzern. Jeder Benutzer bekommt eine eigene POSIX-User-ID zugewiesen und kann nur auf Ressourcen zugreifen, die über entsprechende Berechtigungen mit dieser ID verknüpft sind. Ein Benutzer kann zum Beispiel nicht auf die von einem anderen Benutzer angelegten Dateien zugreifen. Diese Art Trennung verwendet Google nun auch für die Anwendungen in Android. Jede Anwendung bekommt bei der Installation eine eigene User-ID, als wäre sie ein eigener Benutzer. Der Zugriff auf Ressourcen wird über die gleichen Berechtigungen geregelt, wie bei Linux-Benutzern. Allerdings wird nicht für jede Android-Anwendung ein eigenes Home-Verzeichnis angelegt. Die Überprüfung der Zugriffsrechte findet im Kernel statt. Der Mechanismus ist praxiserprobt und in dieser Form schon seit Jahren im Einsatz. Er wurde nicht für Android neu entwickelt. Daher kann er als ausreichend sicher angesehen werden.

Anwendungstrennung auf Dateisystem-Ebene

Eine Komponente, in der die Nutzertrennung anhand der Zugriffsrechte umgesetzt wird, ist das Dateisystem. In Android kommt dazu bis zur Version 2.2 standardmäßig das auf Flashspeicher optimierte YAFFS¹³ zum Einsatz, auch

¹³<http://www.yaffs.net/>

wenn einige Hersteller für ihre Geräte eigene Dateisysteme verwenden. Ab Android 2.3 wird YAFFS aus Leistungsgründen durch Ext4 ersetzt. ([21]) Gemeinsamkeit dieser Dateisysteme ist die Unterstützung der Linux-Berechtigungen auf User-ID-Basis. Andernfalls könnten alle Anwendungen auf beliebige auf dem Gerät gespeicherte Dateien zugreifen.

Während die im Geräte-eigenen Speicher verwendeten Dateisysteme mit dem Berechtigungssystem zusammenarbeiten, gibt es an anderer Stelle ein Problem: Aus Gründen der Kompatibilität wird auf den Flash-Speicherkarten (auch SD-Karte), die zur Erweiterung der Kapazität in die Geräte gesteckt werden können, das alte FAT-Dateisystem verwendet, welches von Microsoft entwickelt wurde. Vorteil dieses Dateisystems ist, dass es von praktisch allen Betriebssystemen gelesen und beschrieben werden kann. Einer der Nachteile ist die fehlende Unterstützung von Benutzerrechten. Eine Datei, die auf der Karte gespeichert ist, kann von allen Benutzern bzw. installierten Anwendungen gelesen und geschrieben werden, die Zugriff auf das Speichermedium haben. Viele Anwendungen speichern Dateien, die von ihnen erstellt wurden, auf diesen externen Datenträger, um den oft stark begrenzten geräteinternen Speicher nicht unnötig zu füllen. Fotos und Videos etwa, die mit dem Gerät aufgenommen wurden, liegen auf der SD-Karte. Nicht alle Anwendungen haben Zugriff auf den externen Speicher, sie benötigen dazu eine spezielle Android-Berechtigung (siehe Kapitel 4.3), aber es gibt viele, die dieses Recht anfordern. Eine Anwendung, die Zugriffsrecht auf die Speicherkarte und aufs Internet hat, könnte theoretisch alle mit der Kamera aufgenommenen Bilder und Videos auslesen und an einen Server im Internet schicken und somit die Privatsphäre der Benutzer gefährden.

Seit Android 2.2 gibt es zudem die Möglichkeit, Anwendungen, die dies zulassen (durch Setzen des entsprechenden Install-Flags), direkt auf die SD-Karte zu installieren. Ohne weitere Maßnahmen würde dies das Sandboxing für die Anwendung zunichte machen, da alle anderen Anwendungen mit Speicherkarten-Zugriff sie beeinflussen könnten. Um dies zu verhindern, setzt Google Verschlüsselung ein. Anwendungen, die auf der SD-Karte gespeichert sind, werden

in einem verschlüsselten Container abgelegt. Der Schlüssel liegt in dem Gerät, auf welchem die Installation bzw. der Verschiebevorgang auf die Karte stattfand. Die Anwendung kann daher nur auf diesem Gerät genutzt werden, in einem anderem Gerät könnte zwar auf die Karte, jedoch nicht auf den Inhalt des Containers zugegriffen werden.

Die Funktion ist im PackageManagerService implementiert, hier der dazugehörige Code:

```
int copyApk(IMediaContainerService imcs, boolean temp)
throws RemoteException {
    if (temp) {
        createCopyFile();
    }

    final String newCachePath;
    try {
        mContext.grantUriPermission(DEFAULT_CONTAINER_PACKAGE,
            packageURI, Intent.FLAG_GRANT_READ_URI_PERMISSION);
        newCachePath = imcs.copyResourceToContainer(packageURI, cid,
            getEncryptKey(), RES_FILE_NAME);
    } finally {
        mContext.revokeUriPermission(packageURI,
            Intent.FLAG_GRANT_READ_URI_PERMISSION);
    }

    if (newCachePath != null) {
        setCachePath(newCachePath);
        return PackageManager.INSTALL_SUCCEEDED;
    } else {
        return PackageManager.INSTALL_FAILED_CONTAINER_ERROR;
    }
}

[...]

// ————— apps on sdcard specific code —————
static final boolean DEBUG_SD_INSTALL = false;
```

```

private static final String SD_ENCRYPTION_KEYSTORE_NAME = "AppsOnSD";
private static final String SD_ENCRYPTION_ALGORITHM = "AES";
static final int MAX_CONTAINERS = 250;
private boolean mMediaMounted = false;

private String getEncryptKey() {
    try {
        String sdEncKey = SystemKeyStore.getInstance().retrieveKeyHexString(
            SD_ENCRYPTION_KEYSTORE_NAME);
        if (sdEncKey == null) {
            sdEncKey = SystemKeyStore.getInstance().generateNewKeyHexString(128,
                SD_ENCRYPTION_ALGORITHM, SD_ENCRYPTION_KEYSTORE_NAME);
            if (sdEncKey == null) {
                Slog.e(TAG, "Failed to create encryption keys");
                return null;
            }
        }
        return sdEncKey;
    } catch (NoSuchAlgorithmException nsae) {
        Slog.e(TAG, "Failed to create encryption
            keys with exception: " + nsae);
        return null;
    } catch (IOException ioe) {
        Slog.e(TAG, "Failed to retrieve encryption keys with exception: "
            + ioe);
        return null;
    }
}
}

```

Listing 2 – AppsOnSD-Code aus PackageManagerService.java

Die Methode `copyApk` kopiert das Anwendungspaket in eine Containerdatei auf der SD-Karte und ruft `getEncryptKey` auf, um sich einen Schlüssel erstellen zu lassen. Evident ist, dass zur Verschlüsselung der Algorithmus AES (Advanced Encryption Standard [22]) verwendet wird, der nach heutigem Stand der Technik als sicher gilt.

Die Verschlüsselung verhindert den Zugriff auf die Anwendung von einer an-

deren aus. Zum Starten der Anwendung wird der Container ins Dateisystem eingebunden („gemountet“) und von Android entschlüsselt. Es handelt sich bei dieser Methode zwar nicht um Sandboxing, die Integrität der Anwendung ist dennoch geschützt.

Shared User IDs

Zwei Anwendungen können durch ihre unterschiedlichen IDs nicht im selben Prozess gestartet werden. Android bietet Entwicklern jedoch eine Möglichkeit, diese Einschränkung zu umgehen. Die Anwendungen können eine „Shared User ID“ beim Betriebssystem beantragen, indem sie ein entsprechendes Attribut in ihren Manifest-Dateien benutzen. Die Anwendungen müssen beide mit demselben Schlüssel signiert worden sein, um diese Funktion nutzen zu können. Sie erhalten dann beide dieselbe User-ID und gelten für Android als eine einzige Anwendung, was ihre Zugriffsrechte angeht. Genutzt wird diese Eigenschaft beispielsweise für Erweiterungen vorhandener Anwendungen. Für den Browser Dolphin etwa gibt es eine Reihe von Zusätzen, die dessen Funktionalität erweitern. Sie werden wie normale Anwendungen installiert und laufen dann mit der gleichen ID wie der Browser selbst. Nur so können sie ihre Funktionen in den Browser integrieren. Das Konzept der Shared User ID kann im Zusammenhang mit Androids eigenem Rechte-System für einen Angriff genutzt werden, wie in Kapitel 4.3 auf Seite 37 gezeigt wird.

Bewertung

Durch das in Android umgesetzte Sandboxing wird den installierten Anwendungen ein enger Rahmen zur Einflussnahme auf ihre Umgebung gesetzt. Auf diese Weise findet die Interaktion mit anderen Anwendungen und dem Betriebssystem auf eine gut kontrollierbare Art und Weise statt. Problematisch ist die Aufweichung dieser Grenzen in einigen Bereichen. Die Anwendung der Benutzertrennung auf Daten, die auf der SD-Karte liegen, ist aus Kompatibilitätsgründen nicht möglich. Das Fehlen eines modernen, durch alle Betriebssysteme unterstützten Dateisystems ist ein Problem, welches nicht nur Android betrifft. Hier ist die Sicherheit der Daten auf der SD-Karte dadurch gefährdet.

Die Benutzer müssen sich darüber im Klaren sein und sich genau überlegen, welchen Anwendungen sie Zugriff auf das externe Speichermedium gewähren. Welche Folgen die Umgehung des Sandboxing mittels Shared User ID haben kann, wird im nächsten Kapitel dargestellt. Prinzipiell ist die Abgrenzung der Anwendungen voneinander jedoch eine gute, wirksame Sicherheitsfunktion.

4.3 Anwendungs-Rechte

Jede Anwendung in Android ist mit einer Reihe von Rechten ausgestattet, die definieren, auf welche Funktionen des Systems sie Zugriff hat. Mittels dieser Rechte wird die strenge Abschottung der Anwendungen voneinander gelockert. Die meisten Anwendungen benötigen mehrere Rechte, um ihre Funktion erfüllen zu können. Rechte werden bei der Installation der Anwendung erteilt. Eine Anwendung, die einmal ein Recht zugewiesen bekommen hat, kann dieses von nun an jederzeit ausüben. Eine weitere Bestätigung durch den Benutzer, sobald während des Betriebs der Anwendung dieses Recht benötigt wird, wird nicht eingeholt. Eine selektive Bestätigung einzelner Rechte ist nicht möglich. Es können nur alle Rechte erteilt oder die Installation abgebrochen werden (siehe Abb. 2).

Die Rechte sind in vier Kategorien eingeteilt:

1. **„normal“** – Dies sind harmlose Rechte, mit denen eine Anwendung keinen Schaden im System anrichten, keine Kosten verursachen und keine Daten des Benutzers auslesen kann. Sie werden jeder Anwendung, die sie anfordert, automatisch gewährt. Es findet keine Bestätigung durch den Benutzer statt. Ein Beispiel ist „android.permission.VIBRATE“, welches Zugriff auf die Vibrationsfunktion des Geräts gewährt.
2. **„dangerous“** – Hier handelt es sich um Rechte, mit denen eine Anwendung potentiell gefährliche Aktionen durchführen kann, wie das Verursachen von Kosten oder das Ausspionieren von Benutzerdaten. Fordert eine

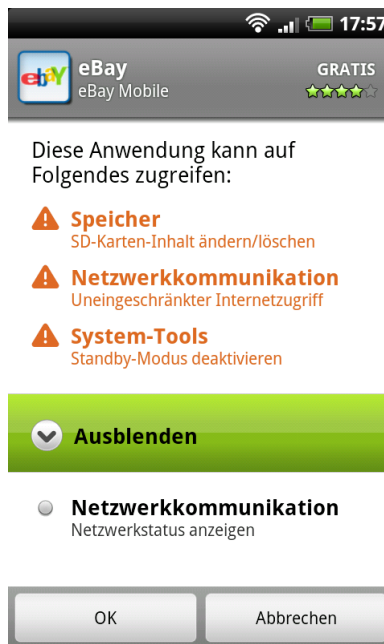


Abbildung 2 – Bestätigung der geforderten Rechte

Anwendung derartige Rechte an, werden sie vor der Installation dem Benutzer aufgelistet, und dieser muss bestätigen, dass er den Vorgang fortsetzen möchte. Ein Beispiel ist „android.permission.SEND_SMS“, das Recht, SMS zu verschicken.

3. **„signature“** – Ein solches Recht wird erteilt, wenn die anfragende Anwendung mit demselben Schlüssel signiert wurde, wie die Anwendung, die das Recht definiert hat. Diese Kategorie ist bedeutsam für Rechte, die von einer nachträglich installierten Anwendung definiert wurden. Ein Entwickler kann in seiner Anwendung eigene Rechte festlegen zum Zugriff auf die Ressourcen der Anwendung. Andere von ihm selbst entwickelte Anwendungen sind mit dem gleichen Schlüssel signiert und erhalten dadurch diese Rechte automatisch erteilt. Eine Anfrage an den Benutzer findet nicht statt.
4. **„signatureOrSystem“** – Diese Rechte werden nur Anwendungen erteilt, die Teil des Basis-Systems sind, welches auf dem Gerät vorinstal-

liert ist, oder die sich mit diesem die Signatur teilen. Diese Kategorie ist nur für einige Ausnahmefälle vorgesehen. Im Allgemeinen wird stattdessen die Einstufung „signature“ verwendet. Ein Beispiel ist „android.permission.CALL_PRIVILEGED“, welches der Anwendung die Fähigkeit verleiht, beliebige Telefonnummern anzurufen.

[50] [51]

Die Anzeige der angeforderten „dangerous“-Rechte vor der Installation einer Anwendung soll dazu dienen, dem Benutzer die möglichen Konsequenzen aufzuzeigen, die er damit auf sich nimmt. Der Benutzer soll aufmerksam werden, wenn ein Recht angefordert wird, das die Anwendung auf den ersten Blick nicht für die Erfüllung ihres Zwecks benötigt. Ein Schachspiel sollte beispielsweise keinen Bedarf daran haben, die gespeicherten SMS auszulesen. Benutzer, die sich der Risiken bewusst sind, die bei der Verwendung des Android Markets zur Installation von Software bestehen, werden sich die geforderten Rechte sicherlich durchlesen und von der Installation des genannten Schachspiels absehen.

Das Rechte-System von Android hat einige Probleme, die unvorsichtige und teilweise auch erfahrene Benutzer betreffen können:

Ein Problem liegt in den vor der Installation angezeigten Rechten. Wie bereits gesagt, können nur alle Rechte auf einmal gewährt oder abgelehnt werden. Entsprechend gibt es auch nur einen Bestätigungs-Knopf, nicht einen für jedes Recht. Die Erfahrung mit Betriebssystemen wie Windows hat gezeigt, dass viele Benutzer nicht dazu bereit sind, sich lange Meldungen durchzulesen, insbesondere, wenn sie nicht gleich verstehen, was diese bedeuten sollen. Sie bestätigen eine Aktion daher oftmals, ohne sich über die Konsequenzen genau im Klaren zu sein. Dieses Verhalten könnte für einen Social Engineering-Angriff ausgenutzt werden: Der Angreifer erstellt ein Programm, welches eine große Zahl an Berechtigungen anfordert, von denen viele dem Benutzer auf den ersten Blick auch plausibel für die Funktion des Programms erscheinen, etwa, weil

es viele Funktionen bietet. In der Bestätigungsansicht sind nun so viele angeforderte Rechte aufgeführt, dass es dem Nutzer nicht auffallen dürfte, dass sich auch einige darunter befinden, die die Anwendung normalerweise nicht brauchen würde und die für bösartige Zwecke ausgenutzt werden könnten. Der Benutzer hat aufgrund der Menge keine Lust, sich alle Rechte genau anzusehen und bestätigt sie nach kurzem Überfliegen, in der Annahme, dass alles seine Richtigkeit hat. Auf diese Weise könnte die Unbedarftheit vieler Anwender ausgenutzt werden, um Schadsoftware auf ihre Geräte zu bringen.

Ein weiteres Problem ist die Grobheit einiger Rechte. Sie sind teilweise zu global gehalten. Beispielsweise kann das Recht „android.permission.INTERNET“ von einer Anwendung angefordert werden, welches ihr erlaubt, beliebig auf das Internet zuzugreifen. Es gibt keine Möglichkeit, dieses Recht nur in eingeschränkter Form zu gewähren. So wäre es etwa sinnvoll, einer E-Mail-Anwendung nur Zugriff auf die von ihr benötigten Kommunikationsprotokolle (POP3, IMAP, SMTP) zu erteilen. In der jetzigen Form erlaubt das Recht ihr aber, beliebige Daten an beliebige Server zu senden und von ihnen zu empfangen. Damit stehen Anwendungen Kommunikationswege offen, die von den Benutzern möglicherweise nicht gewünscht sind und die zum unerlaubten Sammeln von Daten genutzt werden können.

Benutzer können dieses Risiko nur schwer umgehen, da eine Vielzahl der für Android im Market verfügbaren Anwendungen sich über Werbung finanziert, welche regelmäßig aus dem Internet nachgeladen wird. Daher fordern selbst Anwendungen ohne eine mit Internetkommunikation verknüpfte Funktion oftmals dieses Recht an. Als Benutzer weiß man nicht, ob über die Verbindung wirklich nur Werbung nachgeladen wird oder ob Daten auch in die Richtung des Entwicklers gesendet werden. Zum Teil gibt es von den werbefinanzierten Programmen auch Versionen, die gekauft werden müssen und dann ohne Werbeeinblendungen auskommen. Sollten diese ebenfalls Internetzugriff verlangen, obwohl die Werbung der einzige Grund zu sein schien, ist zur Vorsicht zu raten. Fällt die Forderung nach dem Internet-Recht bei der kostenpflichtigen Variante weg, muss das allerdings nicht bedeuten, dass die werbefinanzierte Version

nicht mehr überträgt, als gewünscht.

„`android.permission.WRITE_EXTERNAL_STORAGE`“ ist ein anderes Recht, welches zu umfassend gestaltet ist. Es erlaubt, externe Speichermedien zu beschreiben. Gemeint ist die in den Geräten enthaltene SD-Karte. Hier wird nicht unterschieden zwischen dem Anlegen neuer Dateien und dem Löschen vorhandener. Dies ist allerdings auch eine Folge des auf den Speicherkarten verwendeten Dateisystems (siehe Kapitel 4.2, Seite 29), welches keine feineren Dateiberechtigungen erlaubt. Wünschenswert wäre die Verwendung eines anderen Dateisystems in Verbindung mit einer Aufteilung des genannten Anwendungsrechts in mehrere Rechte wie „Datei erstellen“, „Datei löschen“ und „Datei verändern“.

Die im vorangegangenen Kapitel erwähnte Funktion der Shared User ID sorgt für eine weitere Angriffsmöglichkeit. Diese erlaubt es, mehrere Anwendungen mit der gleichen User-ID zu installieren. Im Rahmen des Rechtessystems gelten diese Anwendungen als eine. Die Rechte werden einer User-ID zugeteilt, nicht den einzelnen Anwendungen. Daher teilen sich die Anwendungen auch die zugewiesenen Rechte. Der Benutzer bekommt darüber keine Information. Ein Entwickler könnte dies Ausnutzen, um seinen Anwendungen mehr Rechte zu verschaffen, als der Benutzer ihnen zugestanden hat: Er erstellt mehrere unabhängige Anwendungen, die verschiedenen Zwecken dienen, etwa einen Bildbetrachter und ein Spiel, welches Werbeeinblendungen enthält, die aus dem Internet heruntergeladen werden. Beide lässt er eine gemeinsame ID anfordern. Der Benutzer sieht anhand der geforderten Rechte, dass der Bildbetrachter nur Zugriff auf die SD-Karte haben möchte, aber keine Verbindung nach außerhalb aufbauen kann und vertraut ihm daher. Das Spiel lädt zwar Werbung aus dem Internet nach und hat daher Kommunikationsmöglichkeit, aber dies ist durchaus gängig und für sich nicht sehr gefährlich, da es keine gespeicherten Daten lesen kann. Wenn beide Programme installiert sind, haben beide sowohl das Recht, Daten zu lesen, als auch, mit Rechnern im Internet zu kommunizieren. Der Bildbetrachter könnte nun jedes Bild, welches er anzeigt oder auf welches er Zugriff hat, per Internet verschicken und somit die Privatsphäre

des Benutzers schädigen. Tatsächlich könnte der Bildbetrachter, aufgrund der erwähnten Problematik mit den Zugriffsrechten auf das externe Speichermedium, jede dort gespeicherte Datei auslesen und versenden. Diese Methode lässt sich auf diverse scheinbar vertrauenswürdige Kombinationen von Programmen anwenden, um etwa an E-Mails oder SMS zu gelangen. Der Benutzer kann nur bemerken, dass ein Programm über mehr Rechte verfügt, als es angefordert hat, indem er dessen Rechte explizit nach der Installation auflistet. Dies wird ein durchschnittlicher Benutzer allerdings nicht tun.

Noch eine Möglichkeit zum Angriff kann über die Definition eigener Rechte erreicht werden. Ein Entwickler kann eigene Rechte definieren, die anderen Anwendungen Zugriff auf Funktionen seiner Anwendung geben. Der Angriff könnte folgendermaßen ablaufen: Der Entwickler stellt zuerst eine Anwendung in den Android Market (siehe Kapitel 4.7), die einige nützliche Funktionen besitzt und dazu sicherheitskritische Rechte anfordert, etwa ein SMS-Programm, mit dem SMS empfangen, gelesen und verschickt werden können und das auch SMS auf die SD-Karte sichern kann. Diese Anwendung verhält sich unauffällig und erfüllt genau die Funktion, die es soll. Sie wird dadurch beliebt, erhält positive Kommentare und Bewertungen. Die Anwendung enthält jedoch ein selbst definiertes Recht, welches anderen Anwendungen Zugriff auf ihre Funktionen gibt. Das Recht ist als „normal“ gekennzeichnet, also als harmlos. Ein anderer Entwickler, der mit dem Entwickler der SMS-Anwendung zusammenarbeitet, stellt später ebenfalls eine Anwendung in den Market, ein simples Spiel etwa, ohne ungewöhnliche Rechtforderungen, außer der bei vielen Spielen üblichen Internetverbindung für Werbung. Der Entwickler weiß allerdings, dass er ohne Sicherheitsabfrage über das von der SMS-Anwendung definierte Recht an dessen Funktionen kommen kann. Das Spiel enthält versteckten Code, der diese Funktionen ausnutzt, wenn auch die SMS-Anwendung installiert ist. So kann es beispielsweise Daten von der SD-Karte oder SMS auslesen und per Internetverbindung verschicken, was der SMS-Anwendung mangels Internetzugriff nicht möglich gewesen wäre. Fiele einem Benutzer ungewöhnliche Aktivität auf, so würde er zuerst das Spiel verdächtigen, da es erst nach dessen Installation dazu käme und er würde es womöglich deinstallieren. Die Sicherheitslücke

in Form der SMS-Anwendung wäre aber weiterhin installiert und könnte durch eine andere Anwendung wieder ausgenutzt werden.

Eine ähnliche Art Angriff wäre möglich, indem in der Manifest-Datei der SMS-Anwendung das „exported“-Attribut der Komponenten auf „true“ gesetzt würde. Dieses macht die Komponenten der Anwendung für alle anderen Anwendungen auf dem System zugreifbar, „public“ statt „private“. Das Attribut ist dafür gedacht, eine Komponente einer Anwendung durch die einer anderen Anwendung aufrufbar zu machen. Ansonsten könnte nur der Benutzer diese manuell starten. Die SMS-Anwendung könnte Komponenten enthalten, die im normalen Betrieb nicht verwendet werden, aber als „exported“ markiert sind und somit von der Spieleanwendung verwendet werden können. Sie könnte dadurch beispielsweise den Inhalt von Kurznachrichten dem Spiel zukommen lassen, durch Speicherung in einem Content Provider mit „exported“-Attribut auf „true“, auf welchen das Spiel dann zugreift, die Nachrichten ausliest und an einen Server im Internet verschickt. [53]

Die genannten Probleme führen dazu, dass einige Anwendungen mehr Rechte erhalten, als die Benutzer ihnen zugestehen würden, wenn sie dazu die Möglichkeit hätten. So kann es passieren, dass sich Schadsoftware Zugriff zum Gerät verschafft und dessen Sicherheit kompromittiert. Die Gefahren gehen dabei vom Datenklau bis zur direkten Verursachung von Kosten. Insgesamt bietet Android den Benutzern zu wenig Informationen und Kontrolle darüber, was die Anwendungen auf ihrem System tun dürfen.

4.4 Dalvik Virtual Machine

Die Dalvik Virtual Machine (DVM) ist wurde von Google zur Ausführung ihres eigens entwickelten Bytecodes in Android implementiert. Der Bytecode wird dabei aus Java-Programmcode erzeugt. Er ist nicht kompatibel mit dem Bytecode, der von Suns bzw. Oracles Java Virtual Machine (JVM) verarbeitet wird. Existierende Java-Programme laufen daher nicht ohne weiteres

auf Android. Shabtai et al. ([28]) stellten fest, dass damit auch existierende Java-Schadsoftware keine Gefahr für Android darstellt.

Ein Vorteil, den die Ausführung von Programmen in der JVM hat, ist das automatische Sandboxing, das diese bietet. Damit erhöht die JVM die Sicherheit eines Systems. Die DVM bietet theoretisch den gleichen Vorteil, jedoch spielt dieser laut Google für Android keine Rolle. Rich Cannings, seines Zeichens Android Security Lead, schrieb in einer E-Mail vom 26. Juli 2010: „Note that unlike the JVM, Dalvik is not a security barrier.“. Auch in der Android Dokumentation ist diese Aussage inzwischen zu finden. [23]

Tatsächlich würde ein Ausbrechen aus der virtuellen Maschine der Anwendung keine zusätzlichen Privilegien einbringen, da sie sich weiterhin in der abgegrenzten Umgebung von Androids eigenem Sandboxing-Mechanismus befände. Die Verwendung von Native Code hätte den gleichen Effekt.

Während der Installation einer Anwendung, während sie in den Speicher geladen wird und in geringerem Maße, während sie ausgeführt wird, findet eine Verifikation des Programmcodes statt ([28]). Diese scheint unter den gegebenen Umständen, dass die DVM keine Sicherheitsfunktion innehat, ungewöhnlich. Worauf im Verifikationsprozess geprüft wird, ist nicht bekannt, der zugehörige Code ist äußerst komplex.

Zum jetzigen Zeitpunkt lautet die Schlussfolgerung, dass die Verwendung der DVM keine Sicherheits-spezifischen Implikationen zulässt. Eine eingehendere Analyse Dalviks mag jedoch anderes zu Tage fördern. Diese findet jedoch nicht im Rahmen dieser Diplomarbeit statt.

4.5 Bibliotheken

In Android finden eine Reihe von freien, quelloffenen Bibliotheken Verwendung, die aus anderen Projekten stammen, also nicht im Rahmen der Android-Entwicklung entstanden. Einige davon sind für die Sicherheitsbetrachtung bedeutsam, da sie schon mit Sicherheitslücken auf sich aufmerksam gemacht ha-

ben oder eine Lücke in ihnen Auswirkungen auf die Sicherheit des Systems hätten.

libc (Bionic)

Die libc-Implementierung von Android, von Google „Bionic“ genannt. Sie stellt die C-Bibliotheken für Anwendungen zur Verfügung, die Gebrauch von Androids Native Code-Schnittstelle, dem NDK, machen. Vorgesehen ist dies, wenn eine Anwendung eine besonders leistungskritische Verarbeitung durchführen muss, da diese in Native Code schneller ablaufen kann. Bionic ist zwar eine Eigenentwicklung von Google, basiert aber auf vorhandenen libc-Implementierungen aus dem Bereich der BSD-Betriebssysteme (OpenBSD, NetBSD, FreeBSD). [24]

Diese sind schon seit Jahren im Einsatz und erprobt, jedoch nicht immun gegen Sicherheitslücken. Ein Beispiel für eine Lücke, die dem Benutzer das Ausführen beliebigen Codes erlaubt ist ein Buffer Overflow, den SecurityTracker für Anfang 2008 in der FreeBSD-libc gelistet hat. [25]

Wie viel Code sich Bionic mit schon vorhandenen libc-Implementierungen teilt, ist nicht bekannt, daher kann auch die Anwendbarkeit von Sicherheitslücken in diesen auf Bionic nicht beurteilt werden. Allerdings wäre die schwere einer solchen Sicherheitslücke im Falle von Android eingeschränkt, da sie sich nur innerhalb der Sandbox der Anwendung auswirken könnte. Hier greift also das Sicherheitskonzept von Android. Die Gefährdung für das System hängt von den Rechten ab, über die die Anwendung verfügt.

FreeType

FreeType ist eine freie Font-Rendering-Engine. Sie kommt unter anderem in vielen Linux-Distributionen zum Einsatz und sorgt auch in Android für die Darstellung von Schrift.

FreeType ist schon mehrfach durch Sicherheitslücken aufgefallen, die zu Abstürzen oder Ausführung beliebigen Codes führen konnten, siehe [26] oder [27]. Es ist zu erwarten, dass sich weitere derartige Lücken in dieser Bibliothek

befinden. Auch hier wäre die Auswirkung auf den Rahmen der Anwendungssandbox beschränkt. Die zwei genannten Sicherheitslücken weisen klar darauf hin, dass der Code, der ausgeführt werden könnte, mit den Rechten der Anwendung laufen würde.

SQLite

SQLite ist das DBMS, welches in Android zur Verwaltung von Daten benutzt wird, etwa solche, auf die mittels eines Content Providers zugegriffen werden soll.

SQL-Datenbanken sind oftmals anfällig für einen Angriff mittels SQL-Injection. Dabei wird der Datenbank eine Anfrage übergeben, welche wiederum einen SQL-Befehl enthält. Ist die Datenbank nicht ausreichend dagegen geschützt, könnte ein Angreifer so möglicherweise Zugriff auf Daten bekommen, die ihm normalerweise verwehrt sein sollten.

Asaf Shabtai et al. haben in ihrem Paper „Google Android: A State-of-the-Art Review of Security Mechanisms“ die Möglichkeit eines solchen Angriffs untersucht. Dabei stellten sie fest, Android sei grundsätzlich gut dagegen abgesichert, da entsprechend manipulierte Anfragen abgefangen würden und die Anwendungsentwickler durch das API zur Verwendung sicherer Parameterverarbeitung ermutigt würden. Nur schlecht geschriebene Anwendungen würden für einen solchen Angriff anfällig sein. [28]

WebKit

WebKit ist eine verbreitete Browser-Engine, die unter anderem in Googles eigenem Browser Chrome und in Apples Safari eingesetzt wird. Auch der Browser des iPhone setzt sie ein.

Ein Browser ist eine der wichtigsten Schnittstellen zwischen dem Benutzer und dem Internet und eines der meistgenutzten Programme auf heutigen Rechnern und Smartphones. Gleichzeitig ist er der schnellen Weiterentwicklung des World Wide Webs und immer mehr webbasierter Dienste ausgesetzt und wird entsprechend ständig komplexer und funktionsreicher. Er ist daher für Fehler leicht anfällig und eines der vorrangigen Ziele für Angriffe aus dem Netz, etwa

mittels Cross-Site Scripting. Ein großer Teil der als gefährlich einzustufenden Sicherheitslücken in Betriebssystemen betreffen in der einen oder anderen Form den Browser, und die Geschwindigkeit, mit der sie geschlossen werden, ist oftmals entscheidend für die Sicherheit des ganzen Systems.

WebKit hat einen guten Ruf als schlanke, gut programmierte und inzwischen auch praxiserprobte Browser-Engine. Mit Apple, Nokia und Google hat sie starke Unterstützer, die in der Lage sind, eine konstante Weiterentwicklung und schnelle Fehlerupdates sicherzustellen. Sie ist daher für die Entwicklung eines darauf basierenden Browsers gut geeignet. Wie bei allen Browsern, gibt es auch bei Chrome oder Safari immer wieder Berichte von Sicherheitslücken, dies ist bei einer so sehr unter Beobachtung stehenden und kritischen Komponente nicht zu verhindern.

Im Falle des Android-Browsers ist die Situation zwiespältig. Die Browser-Engine ist von der unzureichenden Versorgung mit Updates (siehe Kapitel 4.8) besonders betroffen, da oft monatelang keine neue Version dafür erscheint. Inzwischen ist auch eine Sicherheitslücke in Androids Browser entdeckt worden, die das Auslesen privater Daten des Benutzers ermöglichen kann. [38]

Andererseits sind auch hier die Möglichkeiten, dem System Schaden zuzufügen, durch das Sandboxing begrenzt. Es können nur Daten gelesen oder manipuliert werden, auf die der Browser auch Zugriff hat. Allerdings gehören dazu auch Daten, die Informationen über den Benutzer und sein Verhalten beinhalten können, etwa Cookies und Browser-Cache-Daten. Am kritischsten hierbei sind die Zugangsdaten (Passwörter, Benutzernamen) zu Webangeboten zu beurteilen. Diese könnten einen Angreifer befähigen, Kommentare in Foren zu verfassen, die scheinbar vom Benutzer stammen oder in seinem Namen in einem Webshop einzukaufen.

WebKit an sich ist eine gute Basis, um darauf einen möglichst sicheren Browser aufzubauen. Die Sicherheit wird in Android durch die strenge Abgrenzung der Anwendungen voneinander noch erhöht. Allerdings wird sie auf der anderen Seite wieder kompromittiert, durch den Mangel an regelmäßigen Upda-

tes. Sollte dieser Nachteil in Zukunft behoben werden, kann das Browsen auf Android-Smartphones potentiell sicherer sein, als auf den meisten PCs oder Laptops.

4.6 Android Anwendungen

Wie die Anwendungen in Android gegeneinander abgesichert sind, wurde in Kapitel 4.2 schon erläutert. Innerhalb der Anwendungen gibt es aber ebenfalls einige Mechanismen, die Einfluss auf die Sicherheit haben. Diese werden nun untersucht.

Anwendungen in Android setzen sich aus Komponenten zusammen (Activities, Services, Broadcast Receiver, Content Provider), die miteinander über Intents kommunizieren. In diesem Zusammenhang gibt es zwei Mechanismen, die interessant sind.

Zum einen sind dies die „Pending Intents“. Diese stellen einen Mechanismus dar, mit welchem Aktionen delegiert und verzögert ausgeführt werden können. Ein Pending Intent enthält einen normalen Intent. Im Gegensatz zu diesem, kann der Pending Intent vom System zwischengespeichert werden und zu einem späteren Zeitpunkt den darin enthaltenen Intent auslösen. Die ursprüngliche Anwendung, die diesen erstellt hat, braucht zu diesem Zeitpunkt nicht mehr geladen sein. Der Intent wird mit den Rechten der Ursprungsanwendung ausgeführt. Ein Pending Intent kann einer anderen Anwendung übergeben werden, damit diese den Intent dann später ausführt. Die Anwendung, der der Pending Intent übergeben wurde, kann unspezifizierte Werte in dem enthaltenen Intent mit eigenen Daten füllen und so die Bedeutung des Intents modifizieren. Die Dokumentation von Android weist daher darauf hin, dass Pending Intents mit Vorsicht zu verwenden sind. [29]

Der andere Mechanismus sind die URI-Permissions. Diese stellen eine Möglichkeit für eine Anwendungs-Komponente dar, auf Daten in Content Providern

zuzugreifen, auf die sie normalerweise keine Zugriffsrechte haben. Eine E-Mail-Anwendung kann beispielsweise einen Bildbetrachter anweisen, ein Bild anzuzeigen, welches über einen Content Provider erreichbar ist, auf welchen die E-Mail-Anwendung Zugriff hat, der Bildbetrachter jedoch nicht. Der Intent, mit dem das E-Mail-Programm den Bildbetrachter aufruft, enthält dazu eine Flag, mit der der Bildbetrachter autorisiert wird, auf die URI mit dem Leserecht des E-Mail-Programms zuzugreifen. Da das System überprüft, welche Anwendung der Ursprung des Intents war, kann es feststellen, dass diese das Recht hat, den Zugriff durchzuführen und erlaubt daher die Weitergabe des Rechts an den Bildbetrachter. Dabei muss beachtet werden, dass ein Content Provider explizit erlauben muss, dass per URI-Permission auf ihn zugegriffen werden darf.

Es ist zwar nicht möglich, dass mittels URI-Permission ein Recht gewährt wird, welches die Ursprungsanwendung nicht besitzt, jedoch ist denkbar, dass die im Beispiel genannte E-Mail-Anwendung über diesen Mechanismus den Inhalt einer E-Mail oder ihres Anhangs an eine Anwendung weitergibt, die darauf keinen Zugriff haben soll. Dieser Angriff ähnelt stark denen, die in Kapitel 4.3 ab Seite 37 beschrieben sind. Er soll daher hier nicht weiter ausgeführt werden.

[23] [53]

Diese beiden Mechanismen erweitern die Möglichkeiten von Android-Anwendungen um Rechte-Delegation in einem gewissen Maße. Für zukünftige Angriffsversuche könnten sie daher eine Rolle spielen, wenn ein Angreifer sie geschickt einsetzt.

4.7 Android Market

In diesem Abschnitt wird untersucht, wie die Verbreitung von Anwendungen über Googles Android Market stattfindet, welche Sicherheitsfunktionen dieser bietet, wie die Installation einer Anwendung abläuft, und es wird mit Hinblick auf die Sicherheit ein Vergleich gezogen zu drei anderen existierenden Verfahren, dem Apple App Store, einem Linux Software-Repository und der dezentralen Verteilung.

Die Idee eines zentralen Softwarearchivs zur Installation von Programmen auf ein Betriebssystem ist nicht neu. Einige Linux-Distributionen wie Debian beispielsweise nutzen ein solches schon seit über zehn Jahren. Auch Apple und Google nutzen derartige Verfahren, um Software für ihre Mobil-Betriebssysteme iOS und Android zur Verfügung zu stellen. Im Vergleich zu Microsofts Ansatz, die Benutzer die von ihnen gewünschten Programme selber besorgen zu lassen, bieten diese Zentralarchive einige Vorteile, sowohl für die Nutzer als auch für die sie jeweils zur Verfügung stellende Firma oder Organisation.

Die Benutzer profitieren vor allem von der höheren Bequemlichkeit bei der Softwarebeschaffung. Statt mit Suchmaschinen das Internet nach der Herstellerseite eines Programms oder einer anderen Quelle für dieses zu durchsuchen oder in den nächsten Laden mit einer Computerabteilung zu gehen, muss nur ein einziger Anlaufpunkt besucht werden. Hier erfolgt der Zugriff auf die Anwendungen einheitlich, geordnet, oftmals auch nach Kategorien sortiert und damit stressfreier und schneller.

Für die bereitstellenden Firmen oder Organisationen liegt der Hauptvorteil in der Kontrolle. Sie haben einen Überblick über den Softwarebestand, der auf den von ihnen hergestellten Systemen installiert werden kann, zumindest, solange die Benutzer sich nicht an andere Quellen wenden. Theoretisch könnten sie von ihnen unerwünschte Anwendungen abweisen oder entfernen, fast beliebige Regeln festsetzen, die zur Aufnahme in ihr Archiv einzuhalten sind, oder ein einheitliches Bezahlssystem einführen, an dem sie selber mitverdienen könnten.

Aus diesen Gesichtspunkten ist es also sinnvoll, ein solches System zu betreiben und zu verwenden. Ob es auch einer erhöhten Sicherheit dient oder dieser sogar abträglich ist, hängt von der konkreten Implementierung ab. Googles Android Market wird hierzu nun genauer betrachtet.

Android Market – Suche und Anzeige von Anwendungen

Um von einem mit Android betriebenen Gerät auf den Market zuzugreifen, wird eine vorinstallierte Anwendung benutzt. Diese stellt das Angebot an online abrufbaren Programmen nach Kategorien gegliedert in einer Liste dar und bietet auch eine Suchfunktion. Des Weiteren kann Einsicht genommen werden in die Liste der bisher aus dem Market installierten Anwendungen.

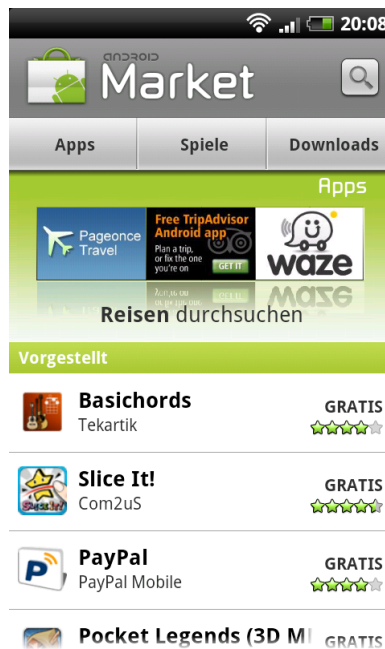


Abbildung 3 – Android Market Startbildschirm

In der Listenansicht erfährt man über die dargestellten Anwendungen neben Namen, Entwickler und Preis auch eine Bewertung in Form von Sternen. Diese stammt von den Benutzern der Anwendung, nicht von einer Prüfstelle oder Google selbst. Jeder Benutzer kann einer auf seinem Gerät installierten Anwendung eine Bewertung geben. Sie bietet einen ersten Hinweis auf die Qualität

des Programms. Allerdings werden nur die wenigsten Benutzer einen tieferen Blick in dessen interne Abläufe geworfen haben und einschätzen können, wie es um die Sicherheit der Software bestellt ist. Die Bewertung bezieht sich daher hauptsächlich auf die Benutzbarkeit und Funktionalität.

Wählt man eine Anwendung aus, die einen interessiert, so erscheint eine Detailansicht mit zusätzlichen Informationen (siehe Abbildung 4).



Abbildung 4 – Detailansicht einer Anwendung im Market

Der Entwickler der Anwendung kann einen eigenen Beschreibungstext formulieren, der hier angezeigt wird. Bei einigen Anwendungen findet sich in diesem Text sogar eine Begründung, warum sie ein bestimmtes Recht benötigt, etwa vollen Internetzugang, um Werbung anzuzeigen. Dies ist allerdings weder verpflichtend noch die Regel und muss natürlich auch nicht der Wahrheit entsprechen.

Interessanter ist die Information, wie viele Bewertungen bisher für die Anwendung abgegeben wurden und wie oft sie heruntergeladen wurde. Bei einer Anwendung, die viele Downloads und Bewertungen vorweisen kann, ist zumindest die Chance höher, dass ein unerwünschtes Verhalten jemandem aufgefallen

wäre. Auch hier gilt jedoch, dass dies nicht die Sicherheit der Anwendung garantiert. Eine Funktion, die unerwünschte Aktivitäten ausführt, könnte so gut versteckt sein, dass sie nur mit sehr großem Aufwand zu entdecken wäre. Andererseits würde eine versteckte, Kosten verursachende Funktion (beispielsweise durch heimliches Senden von SMS-Nachrichten) den Benutzern spätestens mit Erhalt der Rechnung auffallen. Die Verbindung zu einer kürzlich installierten Anwendung zu ziehen, ist dann nur noch ein kleiner Schritt, vorausgesetzt, es wurden nicht zu große Mengen an Software ausprobiert.

Eine Möglichkeit, eine potentiell schädliche Anwendung über die Detailansicht zu melden, ist die Schaltfläche „Als unangemessen Kennzeichnen“. Über diese kann unter Angabe eines Grundes eine Beanstandung an Google gesendet werden. Voreingestellte Gründe für eine Beschwerde sind sexuelle Inhalte, Gewalt oder Beleidigung. Für Sicherheitsmeldungen ist diese Funktion offenbar nicht primär gedacht. Ob Google eine solche ernst nehmen würde und eine Anwendung daraufhin einer Sicherheitsprüfung unterzieht, ist unbekannt. Hat man tatsächlich eine schädliche Anwendung entdeckt, wäre es vermutlich erfolgversprechender, sich nicht auf dieses Werkzeug zu verlassen, sondern direkt in Kontakt zu Google zu treten oder die nach der Deinstallation verfügbare Reporting-Funktion zu verwenden, siehe Seite 56.

Eine andere Möglichkeit, vor einer Anwendung zu warnen, sind die Kommentare (siehe Abbildung 5). Benutzer können nicht nur Bewertungen abgeben, sondern auch diese auch kommentieren. Fällt ihnen bei der Verwendung des Programms etwas auf, können sie so andere Nutzer darüber informieren. Treten hier vermehrt Berichte über überhöhte Rechnungen auf, könnte dies auf ein Fehlverhalten der Anwendung hinweisen. Ein einzelner Bericht über eine Auffälligkeit im Zusammenhang mit der Anwendung könnte dagegen schnell wieder durch neuere Kommentare verdrängt werden, falls er keine Diskussion auslöst.

Zu den Detailinformationen gehören auch Kontaktdaten der Entwickler, üblicherweise eine Website und eine E-Mail-Adresse. Auf diese Weise kann ein Benutzer weitere Informationen zu der Anwendung einholen. Von einer se-



Abbildung 5 – Kommentare zu einer Anwendung im Market

riös aussehenden Homepage seitens des Entwicklers können allerdings keine Rückschlüsse auf die Qualität der Software gezogen werden. Zu einfach ist es heutzutage, gut aussehende Onlineauftritte zu erstellen. Eine schlecht gepflegte Seite, etwa mit vielen Rechtschreibfehlern, könnte andererseits ein Grund für Misstrauen sein. Direkter E-Mail-Kontakt zum Entwickler könnte zwar unter Umständen Einblicke in dessen Charakter gewähren und eine Einschätzung seiner Seriosität erlauben, jedoch ist diese Vorgehensweise sehr aufwändig, insbesondere, wenn man vorhat, mehrere Dutzend Anwendungen zu installieren. Einen derartigen Aufwand würde der allergrößte Teil der Benutzer nicht auf sich nehmen.

Ist eine Anwendung bereits installiert, sind in der Detailansicht die Bewertungs- und Kommentarfunktionen zu finden. Seit Android 2.2 gibt es weiterhin die Möglichkeit, der Anwendung durch Setzen eines Häkchens zu gestatten, sich automatisch auf den aktuellsten Stand zu bringen (siehe Abbildung 6). Auf die Updates von Anwendungen wird auf Seite 57 noch näher eingegangen.

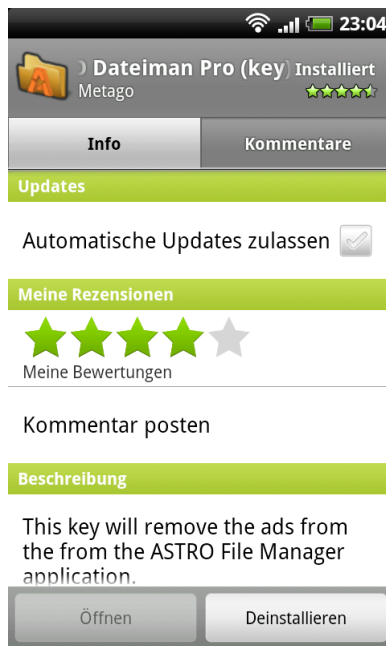


Abbildung 6 – Die Detailansicht einer installierten Anwendung

Insgesamt können die Informationen, die die Detailansicht bietet, zwar Hinweise auf sicherheitsrelevantes Fehlverhalten einer Anwendung geben, ein verlässliches Instrument zur Beurteilung der Sicherheit sind sie jedoch aus den genannten Gründen nicht. Eine explizit für Sicherheitsmeldungen gedachte Funktion gibt es an dieser Stelle nicht.

Android Market – Installation und Deinstallation von Anwendungen

Hat man sich entschieden, eine Anwendung zu installieren, signalisiert man dies durch Betätigung der entsprechenden Schaltfläche. Die Installation wird daraufhin entweder gestartet oder, falls die Anwendung zustimmungsbedürftige Rechte einfordert (zu den Rechten siehe Abschnitt 4.3), eine neue Seite geöffnet. Auf dieser werden die geforderten Rechte aufgelistet (siehe Abbildung 7).

Der Benutzer kann nun entscheiden, ob er der Anwendung alle diese Rechte

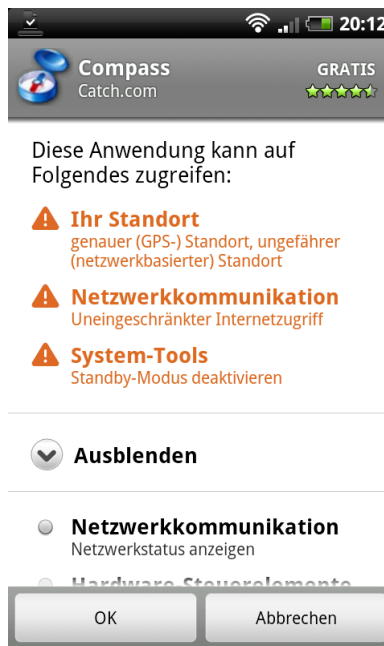


Abbildung 7 – Von einer Anwendung geforderte Rechte

gewähren oder die Installation abbrechen will. Gezieltes Gewähren einzelner Rechte ist nicht möglich, die Anwendung bekommt alle geforderten Rechte, oder sie kann nicht installiert werden (siehe auch Kapitel 4.3). Eine Bestätigung mit „OK“ startet die Installation.

Der Ablauf ist nun folgender: Mit Absenden des Installationsbefehls durch den Benutzer an den Market, wird dem Market-Server mitgeteilt, welche Anwendung installiert werden soll und von wem die Aufforderung stammt. Dabei spielt der bei jedem Android vorinstallierte Dienst „Google Talk“ (GTalkService) eine große Rolle. Dieser Dienst, der normalerweise für den Austausch von Instant Messaging-Nachrichten zwischen Benutzern über das XMPP-Protokoll¹⁴ verwendet wird, hält eine permanente, SSL-verschlüsselte Verbindung zu einem Google-Server offen. Anhand des Google-Accounts des Benutzers wird dieser in Google Talk identifiziert und weiß der Market, von wem die Installationsaufforderung geschickt wurde. Der Market-Server schickt nun ein „IN-

¹⁴<http://xmpp.org/>

STALL_ASSET“ über die offene Verbindung an den Dienst (Push-Verfahren, Nachrichten werden vom Server zum Client geschickt, ohne dass dieser erst explizit eine Verbindung aufbauen muss), welche den Namen und Typ der Anwendung und eine Adresse zum Herunterladen dieser enthält. Diese Nachricht signalisiert Google Talk, dass es eine Anwendung installieren soll. Anhand der Adresse wird die Anwendung dann heruntergeladen. Die nachfolgende Installation ist von der Menge des beteiligten Codes ein komplexer Prozess. GTalk gibt die Anwendung an das Systemprogramm PackageManagerService weiter, welches die Installation durchführt und die Liste der installierten Anwendungen aktualisiert. Während der Installation werden von PackageManagerService.java ausgehend noch weitere Klassen aufgerufen. Den kompletten Vorgang abzubilden würde zu weit führen. Der zentrale Kern der Anwendungsverwaltung ist der PackageManagerService.

Bedeutsam für die Sicherheit von Android ist die genannte Verbindung von Google Talk zu den Servern. Sie erlaubt es, einem Android-Gerät per Push-Verfahren Nachrichten und Befehle zu übermitteln. Dies ist nützlich etwa für ein E-Mail-Programm, welches nicht in regelmäßigen Intervallen den Server kontaktieren muss, um E-Mails abzufragen, sondern von diesem über jede eingehende E-Mail sofort informiert wird. Wie am Beispiel des INSTALL_ASSET zu erkennen ist, wird die Verbindung auch für wichtige Systemverwaltungsaufgaben verwendet. Auch ein REMOVE_ASSET existiert, mit dem die Deinstallation von Anwendungen angestoßen werden kann. Das bedeutet, dass Installations- und Deinstallations-Aufforderungen beliebig von Google an jedes Gerät gesendet werden können, ohne Benutzerinteraktion. Dass diese Möglichkeit schon in der Praxis angewendet wurde, zeigt die Fernlöschung zweier von Jon Oberheide zur Demonstration eines möglichen Angriffsvektors auf Android in den Market gestellter Anwendungen durch Google. [44]

Die Möglichkeit, Anwendungen von allen Geräten zu entfernen, ist ein Sicherheitsgewinn für die Benutzer, da Google sie so vor entdeckter Schadsoftware schützen kann. Es stellt jedoch auch einen Kontrollverlust für die Benutzer dar, die der Willkür Googles ausgesetzt sind. Zensur und das Aufzwingen von

Anwendungen wären theoretisch möglich. Allerdings sollte ein Benutzer, der Misstrauen gegenüber Google hat, sich überlegen, ob er mit Google Android das für ihn richtige Betriebssystem einsetzt.

Ein sehr großes Problem würde allerdings bestehen, wenn jemand anders als Google Zugang zu der GTalk-Verbindung hätte. Wieder Jon Oberheide ist es, der ein Szenario entworfen hat, in welchem eine man-in-the-middle-Attacke auf die SSL-Verbindung durchgeführt wird ([45]). Gelingt dies, könnten auf dem Gerät beliebig Anwendungen installiert oder deinstalliert werden. Geradezu verheerend wäre es, würde ein Angreifer die GTalk-Server von Google zeitweise unter seine Kontrolle bringen. Er könnte allen Android-Geräten auf der Welt gleichzeitig entsprechende Befehle zukommen lassen. Dies ist eine Gefahr, die allen zentralisierten Softwareverteilungs-Verfahren gemein ist. Erfolgreiche Angriffe auf die Server einiger Linux-Distributionen zeigen die Realität dieser Gefahr ([46]). Welche Sicherheitsmaßnahmen Google getroffen hat, um ein solches Szenario zu verhindern, ist nicht bekannt.

Eine weitere Art, Anwendungen auf Android-Geräte zu installieren, präsentierte Google Anfang Februar 2011. Der Market kann von diesem Zeitpunkt an über eine Website erreicht werden. Ein Benutzer kann von einem beliebigen Rechner aus per Browser darauf zugreifen, die Anwendungen durchsuchen und auch die Installation auf seinem Gerät anstoßen. Voraussetzung dafür ist, dass er mit seinem Google-Account eingeloggt ist. In seinem Account ist gespeichert, welches Gerät ihm gehört. Zu diesem wird dann, sofern es online ist, wie gewohnt ein `INSTALL_ASSET` geschickt und die Installation startet. Ist das Gerät zu diesem Zeitpunkt nicht online, wird die Installation gestartet, sobald es wieder Verbindung aufnimmt. Die Anzeige der geforderten Rechte findet dabei im Webbrowser statt, von dem aus die Installation gestartet wurde. [35]

Diese neue Installationsmethode bietet die Möglichkeit für einen Angriff: Ein Angreifer verwendet ein trojanisches Pferd, also eine Spionagesoftware, welches er auf die Desktoprechner vieler Computerbenutzer schleust, um an eine möglichst große Zahl von Google-Accountdaten zu gelangen. Hat er genug da-

von gesammelt, so stellt er eine Anwendung in den Market, die eine beliebige Zahl an Rechten verlangt und damit Zugriff auf alle auf dem Gerät gespeicherten Informationen hätte. Er nennt sie „Belohnung von Google“. Nun loggt er sich mit den gesammelten Accountdaten bei den jeweiligen Benutzern ein und versucht, über den Market die Installation dieser Anwendung auszuführen. Ob zu dem Account ein Android-Gerät gehört, wird ihm bei diesem Vorgang angezeigt. Hat er im Vorfeld genügend Daten gesammelt, so braucht nur ein Bruchteil der Accounts mit einem Android-Gerät verknüpft sein, um an eine Menge Informationen zu kommen. Da die Rechte beim Angreifer abgefragt und bestätigt wurden, startet die Installation bei den Benutzern ohne Nachfrage. Die Benutzer haben danach eine Benachrichtigung in ihrer Informationsleiste, die meldet: „Belohnung von Google erfolgreich installiert“. Der Entwickler ist in dieser Meldung nicht mit aufgeführt. Viele Benutzer werden, wenn sie die Meldung sehen, denken, es wäre tatsächlich eine Anwendung von Google. Mit einem Tipp mit dem Finger auf die Meldung wird die Anwendung sogleich gestartet und hat nun praktisch vollen Zugriff auf alle Daten des Benutzers und kann sie an den Angreifer schicken, oder sie versendet eine große Zahl kostenpflichtiger SMS. Die Entscheidung liegt beim Entwickler der Anwendung. Bis die Anwendung von Market entfernt worden ist, kann dieser schon eine Menge Schaden angerichtet haben.

Möchte man eine Anwendung von seinem Gerät entfernen, stellt Android dafür eine Deinstallationsfunktion bereit. Im Anschluss an die Deinstallation wird nach dem Grund für diese gefragt (siehe Abbildung 8).

Hier kann man zum ersten Mal explizit einen Hinweis darauf geben, dass die Anwendung schädlich ist. Ob Google auf einen einzigen solchen Report schon reagiert und die Anwendung prüft, ist nicht bekannt. Ein Nachteil dieser Funktion ist, dass sie nur aufgerufen werden kann, indem man die Anwendung deinstalliert. Hat man vor, ihre Schadfunktion näher zu untersuchen, möchte man dies vielleicht nicht. In diesem Fall wäre es sinnvoller, sich direkt an Google zu wenden.

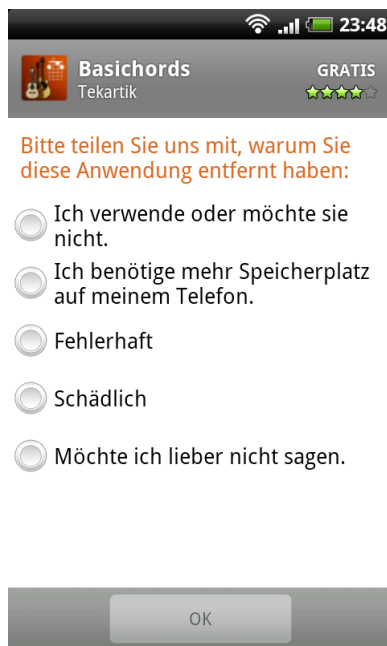


Abbildung 8 – Abfrage nach der Deinstallation

Android Market – Anwendungsupdates

Bei der Entwicklung von Anwendungen ist es meistens notwendig, nachträglich Verbesserungen hinzuzufügen. Dies können neue Funktionen, Fehlerberegnerungen oder das Schließen einer Sicherheitslücke sein. Daher ist es wichtig, die auf einem Gerät installierten Programme aktualisieren zu können. Android bietet diese Funktion für Anwendungen an, die der Benutzer selbst aus dem Market installiert hat. Bibliotheken und viele der vorinstallierten Systemanwendungen wie der Browser können auf diese Weise nicht aktualisiert werden. Für einige dieser Anwendungen sind aber inzwischen Updates erschienen, etwa für den Mail-Client und Google Maps. Das könnte bedeuten, dass Google ihr Updatesystem nach und nach umstellt. Insbesondere für den Browser wäre dies ein wichtiger Schritt, um immer rechtzeitig Sicherheitsaktualisierungen zu erhalten. Wie sich die Situation weiter entwickeln wird, ist zum jetzigen Zeitpunkt jedoch noch nicht abzusehen. Bisher ist die meiste Systemsoftware noch auf vollständige Betriebssystem-Updates angewiesen, um auf den neuesten Stand zu kommen. Mehr dazu findet sich im Abschnitt 4.8 auf Seite 66.

Wie schon auf Seite 51 gesehen, kann es einer Anwendung erlaubt werden, automatisch Updates zu beziehen. Diese werden dann ohne Zutun des Benutzers im Hintergrund heruntergeladen und installiert. Lässt man dieses nicht zu, so können Updates auch manuell abgerufen werden. Hierzu muss die Auflistung der bereits installierten Anwendungen im Market aufgerufen werden. Anwendungen, für die eine Aktualisierung zur Verfügung steht, sind entsprechend gekennzeichnet. Das Update unterscheidet sich insofern von einer Neuinstallation der Anwendung, dass bereits gewährte Rechte nicht noch einmal bestätigt werden müssen. Die neue Version der Anwendung kann jedoch zusätzliche zustimmungsbedürftige Rechte verlangen. Diese müssen vom Benutzer gewährt werden, wie es bei einer erstmaligen Installation auch der Fall wäre. Auf diese Weise wird verhindert, dass zusätzliche Berechtigungen ohne Wissen des Benutzers an die Anwendung vergeben werden. Allerdings kann ein Update auch eine zuvor nicht vorhandene Schadfunktion in das Programm einschleusen, welches die vorhandenen Rechte ausnutzt. So könnte aus einer harmlosen

Anwendung eine gefährliche werden und alle zuvor damit gesammelten Erfahrungswerte wären nichtig. Damit verliert auch die Zahl der Downloads und Bewertungen einer Anwendung seine Bedeutung als Vertrauensindiz, denn der Entwickler könnte gerade auf eine hohe Verbreitung seiner Anwendung gewartet haben, bevor er die Schadfunktion aktiviert bzw. einschleust. Auf diese Weise erschwert er auch die Identifizierung seiner Anwendung als Schadensquelle, denn die Benutzer werden erst einmal die von ihnen erst kürzlich neu installierten Anwendungen im Verdacht haben. In Verbindung mit der automatischen Update-Funktion wird die Verschleierung noch effektiver, da die Benutzer unter Umständen nicht gemerkt haben, dass ein Update stattgefunden hat.

Insgesamt ist die Updatefunktion dennoch ein Sicherheitsgewinn, da davon auszugehen ist, dass durch sie mehr Sicherheitslücken geschlossen als geöffnet werden. Automatische Updates verkürzen die Zeit zwischen Veröffentlichung und Installation der Aktualisierungen, manuelle Updates sorgen dagegen für mehr Kontrolle durch den Benutzer und machen es ihm leichter, den Überblick über die Programmversionen zu behalten, die er installiert hat.

Android Market – Einstellen von Anwendungen

Möchte ein Anwendungsentwickler Programme über den Android Market zur Verfügung stellen, so muss er sich dafür als erstes ein Google-Konto zulegen. Dadurch erlangt er Zugriff auf die diversen Google-Dienste, unter anderem Google Checkout, Googles Bezahldienst für Online-Käufe und -Verkäufe. Über diesen muss eine einmalige Registrierungsgebühr von 25 Dollar entrichtet werden. Die Bezahlung findet per Kreditkarte statt. Auf diese Weise wird auch die Identität des Entwicklers festgestellt. Nun muss der Entwickler noch ein Zertifikat erstellen, mit dem er alle seine Anwendungen signiert. Dieses Zertifikat kann selbstsigniert sein, es ist also nicht nötig, eine Zertifizierungsstelle aufzusuchen. Anhand des Zertifikats kann nicht festgestellt werden, ob die vom Entwickler angegebene Identität echt ist. Das Zertifikat dient lediglich dazu eine Anwendung einem Entwickler zuzuordnen. Die Zertifizierung stellt nur insofern eine Sicherheitsfunktion dar, dass festgestellt werden kann, dass eine

Anwendung tatsächlich vom angegebenen Entwickler stammt. Um wen es sich aber wirklich handelt, ist damit nicht geklärt.

Hat der Entwickler diese Schritte vollzogen, kann er beliebige Anwendungen in den Market einstellen. Eine weitere Überprüfung findet nicht statt, die Anwendungen können also schon kurz darauf von Benutzern installiert werden.

Da die Identität des Entwicklers nur über seine Kreditkartendaten überprüft wird, wäre es einem Angreifer mit Hilfe gefälschter Daten möglich, unerkannt Schadsoftware in den Market einzustellen. Wird diese entdeckt, entfernt und der Account des Entwicklers gelöscht, so kann er mit anderen Daten wieder einen neuen Account anlegen und seine Software in anderer Form wieder einstellen. Wenn die Schadfunktion der Anwendungen gut genug versteckt ist, könnte zwischen den nötigen Accountwechseln eine lange Zeit vergehen und dem Entwickler so viel Geld oder für ihn interessante Informationen liefern, dass ihn die jedes mal nötigen 25 Dollar Gebühr nicht stören. Einen Hobby-Entwickler, der nur aus Spaß oder Zerstörungswut schädliche Software einstellt, würde die Anmeldegebühr und der Aufwand des Identitätsklau wahrscheinlich abhalten, einen professionellen Datendieb oder Betrüger aber eher nicht, vor allem falls Android-Geräte in Zukunft verstärkt im Geschäftsbereich eingesetzt werden.

Alternative Installationsquellen

Neben dem Market kann ein Benutzer auch die Installation aus anderen Quellen zulassen. Diese Funktion muss er allerdings erst im Optionsmenü von Android freischalten. Das Setzen des Häkchens lässt eine Sicherheitswarnung erscheinen, die den Benutzer über die Risiken der Benutzung von unbekanntem Quellen informiert und ihn darauf hinweist, dass er die Verantwortung für dadurch eventuell entstehende Schäden übernimmt. Erst nach Einwilligung in diese Bedingung wird diese Funktion aktiviert. Von nun an ist es möglich, Anwendungen aus anderen Quellen als dem Market auf dem Gerät zu installieren, beispielsweise, indem sie von einer Website heruntergeladen, auf die SD-Karte des Geräts kopiert und von da aus mit einem dazu fähigen Dateimanager in-

stalliert wird. Auch bei dieser Installationsmethode werden dem Benutzer die Rechte angezeigt, die die Anwendung verlangt. Nutzer sind bei der Auswahl der Anwendungen aus Fremdquellen auf sich gestellt, sie haben keine Unterstützung durch Google zu erwarten, wenn es zu Problemen kommt.

Dass sich außerhalb des Markets versehentlich Schadsoftware auf das Gerät geholt werden kann, die im Market nicht zu finden ist, hat sich bereits bestätigt, etwa das trojanische Pferd „Trojan-SMS.AndroidOS.FakePlayer.a“ ([30]), das sich als Media-Player tarnt und das selbstständig SMS versenden kann. Da es hierzu die entsprechenden Rechte anfordert und nur aus einer Fremdquelle installiert werden kann, stellt es nur für sehr unbedarfte und gleichzeitig unvorsichtige Benutzer eine Gefahr dar. Solche Benutzer hätte Google schützen können, indem eine Installation aus fremder Quelle immer verboten wird. Allerdings hat Google ausreichend auf die Risiken hingewiesen, ein gewisses Maß Eigenverantwortung kann von den Nutzern verlangt werden. Google ist daher hier kein Vorwurf zu machen.

Vergleich: Android Market – Apple App Store

Der Apple App Store ist dem Android Market sehr ähnlich. Beide Ansätze zielen auf die Verbreitung von Anwendungen für mobile Endgeräte ab. Es werden im App Store wie bei Google nur Programme und ihre Updates auf diese Weise verteilt, Systemkomponenten werden nicht aktualisiert. Es ist weiterhin wie bei Google eine Gebühr zu entrichten, um seine Anwendungen einstellen möchte. Bei Apple sind es 99 Dollar im Jahr. [31]

Die Unterschiede offenbaren sich erst bei genauerer Betrachtung. Google hat einen offenen Ansatz, der es prinzipiell jedem erlaubt, Anwendungen ohne lange Wartezeit in den Market einzustellen. Ein wichtiger Grund dafür dürfte sein, für möglichst viele Entwickler attraktiv zu sein und schnell zum Konkurrenten Apple aufzuschließen, was die Zahl der Anwendungen im Market angeht. Außerdem betont Google immer wieder, wie wichtig ihnen die Freiheit ist („Android is about freedom and choice.“ [10]), was sich in den Market-Konditionen widerspiegelt. Apple dagegen ist hier restriktiver. Die Anwendun-

gen müssen zunächst bei Apple eingereicht werden und werden einer Prüfung unterzogen. Aufgrund der Menge an eingereichten Anwendungen kann diese mehrere Wochen dauern. Die Prüfung endet entweder mit der Aufnahme in den App Store oder einer Ablehnung, wobei die Kriterien, nach denen geprüft wird, nicht transparent offen gelegt sind. Daher kann die Sorgfalt, mit der die Anwendungen insbesondere auf Sicherheitsprobleme oder Schadfunktionen untersucht werden, nicht beurteilt werden. Offensichtliche Probleme werden wahrscheinlich erkannt werden, eine tiefer gehende Analyse ist aus Zeitgründen jedoch unwahrscheinlich. Dennoch ist Apple mit dieser Politik Google in Sachen Sicherheit einen Schritt voraus, da problematische Anwendungen potentiell schon erkannt werden können, bevor ein Benutzer sie bei sich installiert hat. Google setzt dagegen darauf, dass diese Anwendungen von der Benutzergemeinschaft des Markets entdeckt werden. Der Schaden, der dadurch bei einigen der Benutzern entstehen könnte, wird in Kauf genommen.

Auch in einer anderen Hinsicht verhält sich Apple restriktiver als Google. Während ein Benutzer nach der Freischaltung der Option und Bestätigung der zugehörigen Warnung Anwendungen auch aus anderen Quellen als dem Market installieren kann, ist dies bei Apple untersagt. Das Softwareangebot für Apples mobile Geräte beschränkt sich also auf den App Store und wird damit von Apple vollständig kontrolliert. Es ist allerdings möglich, mittels eines so genannten „Jailbreaks“ an erweiterte Rechte auf dem Gerät zu gelangen, ähnlich wie beim „Rooten“ eines Android-Geräts. Auf diese Weise lassen sich Anwendungen auch ohne den App Store installieren. Diese Methode ist aber für den Durchschnittsbenutzer keine übliche Vorgehensweise. Wer darauf zurückgreift, weiß im Allgemeinen, was er tut, und ist bereit, das Risiko zu tragen. Durch die schwerer zu umgehende Beschränkung auf den App Store bietet dieser dem Benutzer vor allem einen besseren Schutz vor seiner eigenen Fahrlässigkeit.

Die Verifikation der Identität des Entwicklers wird bei Apple wie bei Google über die Kreditkarteninformationen durchgeführt, wenn es sich um eine Einzelperson handelt. Handelt es sich um ein Unternehmen, werden weiterführende Informationen zur Verifikation gefordert. [32]

Der App Store hat also gewisse Vorteile bezüglich der Sicherheit vorzuweisen. Gegenüber einem professionell agierenden Entwickler mit bösen Absichten, der seinen Schadcode gut zu verstecken weiß, sind diese allerdings nur gering.

Vergleich: Android Market – Linux Repository

Die Verteilung von Software über ein Repository wird von den meisten Linux-Distributionen eingesetzt, etwa von Debian, Ubuntu oder Fedora. Einige gehen schon seit über zehn Jahren so vor. Auf den ersten Blick scheinen die Ansätze der Linux-Distributionen denen von Google und Apple zu ähneln, es gibt aber einige bedeutende Unterschiede.

Im Gegensatz zum Android Market werden über ein Linux-Repository nicht nur einzelne Programme installiert und aktualisiert, sondern sämtliche auf dem Gerät befindliche Softwarebestandteile, die nicht vom Benutzer manuell erstellt wurden. Darunter fallen der Kernel, die System-Bibliotheken, Protokoll-Stacks und Laufzeitumgebungen sowie die Anwendungen. Theoretisch kann während eines Update-Vorgangs das gesamte System ausgewechselt werden. Dies hat einige Vorteile für die Sicherheit. Eine Sicherheitslücke beispielsweise im Kernel kann zeitnah geschlossen werden, ohne dass auf eine neue Version des Betriebssystems gewartet werden muss.

Ein weiterer Unterschied besteht in der Art und Weise, wie Software in das Repository gelangt. Die Entwickler laden diese nicht selber auf einen Server. Dies übernehmen so genannte „Package Maintainer“, Personen, die der jeweiligen Organisation angehören, von der die Distribution herausgebracht wird. Ein Package Maintainer kann für ein oder mehrere „Pakete“ – Einzelteile, aus denen die Software besteht – verantwortlich sein, für eine einzelne Bibliothek, für ein Programm, das aus mehreren Bibliotheken und anderen Teilen besteht oder für mehrere Programme. Er holt sich die Software von den Entwicklern, testet die Lauffähigkeit, nimmt Veränderungen vor, um sie den Konventionen der Distribution anzupassen (beispielsweise das verwendete Installationsverzeichnis), baut dann ein Paket daraus und lädt dieses in das entsprechende Repository. Die Kontrolle darüber, welche Software in das Repository aufge-

nommen wird, welche Veränderungen daran vorgenommen werden und wann sie aktualisiert wird, liegt also bei den Entwicklern der Distribution, nicht bei denen der Software.

Vor der Aufnahme einer Software in ein Repository können durch die Distributions-Entwickler diese in beliebigen Ausmaße testen, sowohl auf funktionelle Fehler als auch auf Sicherheitslücken oder Schadcode. Hierbei besteht theoretisch keine zeitliche Beschränkung, außer es handelt sich um eine Software, die wichtig für den Betrieb des Systems ist und die daher bis zu einem bestimmten Termin eingefügt werden muss. In welchem Umfang die Software getestet wird, hängt von den Entwicklern der Distribution ab und von deren zeitlichen, personellen und technischen Möglichkeiten.

Sicherheitsaktualisierungen müssen diesen Prozess ebenfalls durchlaufen, daher kann es bei ihrer Verteilung zu Verzögerungen kommen im Vergleich zu einem direkten Update durch den Entwickler der Software, je nach Zeit und Fähigkeit des Maintainers.

Die Verantwortung für die Stabilität und Sicherheit einer Linux-Distribution liegt in den Händen der Package Maintainer. Daher ist es beispielsweise bei Debian Voraussetzung, dass eine Person, die Maintainer werden möchte, vorher schon längere Zeit aktiv an der Entwicklung der Distribution mitgewirkt hat. Weiterhin muss er zeigen, dass er die Abläufe und Philosophie innerhalb der Organisation versteht, muss einen Befürworter unter den Entwicklern haben und seine Identität mittels Ausweis und nach Möglichkeit persönlichen Kontakts mit einem Entwickler bestätigen. Dies sind sehr hohe Voraussetzungen im Vergleich zu denen, die Google an einen Entwickler stellt. [33]

Durch die stärkere Kontrolle, die umfangreichere Updatemöglichkeit und die höheren Ansprüche an die verantwortlichen Personen ist der Repository-Ansatz der Linux-Distributionen in Sachen Sicherheit dem Googles überlegen, bis auf die Verzögerung bei den Anwendungsupdates. Allerdings hätte ein solch aufwendiges Vorgehen dazu geführt, dass neue Anwendungen wesentlich langsamer und in geringerer Zahl in den Android Market gelangt wären und die

Plattform für Entwickler weniger attraktiv gewesen wäre. Google entschied sich stattdessen für Quantität vor Qualität.

[34]

Vergleich: Android Market – dezentraler Ansatz

Eine ganz andere Art der Softwareverteilung im Vergleich zu den bisher genannten Möglichkeiten ist der dezentrale Ansatz. Hierbei wird die Software nicht an einem Ort gesammelt gespeichert, kann dort durchsucht werden und Aktualisierungen beziehen, sondern die Entwickler stellen ihre Erzeugnisse an einem beliebigen Platz zur Verfügung, meistens die eigene Website. Dies bezieht sich üblicherweise auf die Anwendungssoftware, nicht auf Systemkomponenten. Ein Beispiel für ein Betriebssystem mit einer derartigen Softwareverteilung ist Microsoft Windows. Hier werden Updates für das System an sich zentral bereitgestellt, Anwendungssoftware muss dezentral beschafft werden.

Die Benutzer eines solchen Betriebssystems haben selbst die Verantwortung, sich ihre Anwendungen zu besorgen, von einem Ort ihrer Wahl. Die Anwendungen müssen dann entweder von Hand aktualisiert werden, wenn eine neue Version erscheint oder haben ein eingebautes Aktualisierungswerkzeug, mit dem sie sich automatisch oder per Knopfdruck auf den neusten Stand bringen. Eine allgemeine Aussage darüber, wie schnell und ob überhaupt Sicherheitslücken geschlossen werden, kann nicht getroffen werden, da dies ganz von den Benutzern und den Anwendungen abhängt.

Da Software üblicherweise direkt von den Entwicklern bezogen wird, findet keinerlei Vorab-Kontrolle auf eventuelle Schadfunktionen statt. Die Benutzer haben nur die Wahl, den Entwicklern zu vertrauen oder die Anwendung nicht zu benutzen. Erfolgt die Beschaffung von einem anderen Ort als dem vom Entwickler bereitgestellten, muss auch derjenigen Instanz vertraut werden, die die Software zur Verfügung stellen, da sie diese verändert haben könnte. Dies hat unter Windows zur Verbreitung von Virenschadern geführt, die den installierten Softwarebestand und gespeicherte Installationsdateien auf Schadcode

untersuchen. Allerdings kann im Normalfall nur eine Veränderung der Software durch einen dem Scanner bekannten Computervirus erkannt werden, eine vom Entwickler eingebaute Schadfunktion dagegen nicht.

Die dezentrale Softwareverteilung lebt von der Vorsicht der Benutzer und überlässt ihnen die größte Verantwortung im Vergleich zu den anderen vorgestellten Verfahren. Dennoch ist Googles Ansatz mit der dezentralen Verteilung eher zu vergleichen als mit dem vorgenannten Linux-Repository. Googles Konzept ist, den Anwendungen einen zentralen Ort zu geben, von dem sie bezogen werden können, einen gemeinsamen Update-Mechanismus, Suchfunktion und Kommentare. Doch durch die sehr liberale Kontrollregelung ist der Market grundsätzlich nur wenig vor böswilligen Schadsoftware-Entwicklern abgesichert. Den durchschnittlichen Benutzern wird durch den zentralen Bezugsort ein falsches Gefühl der Sicherheit vermittelt. Sie gehen davon aus, dass Anwendungen, die aus dem Market geladen werden, keine Gefahr darstellen, denn sonst hätte Google sie ja nicht dort platziert. Ihnen ist nicht bewusst, dass sie ein nahezu ebenso hohes Maß an Vorsicht walten lassen müssen, als würden sie die Programme dezentral beziehen. Zwar ist der Market durch Maßnahmen wie die Identifikation der Entwickler und die Möglichkeit, Anwendungen wieder daraus zu entfernen, durchaus sicherer als der dezentrale Ansatz. Dies wird aber dadurch ausgeglichen, dass die Benutzer durch das erzeugte Sicherheitsgefühl unbesorgter eine Menge Anwendungen ausprobieren und sie installieren, ohne sich vorher über sie zu informieren. Dadurch erhöht sich die Chance, auf eine Anwendung mit Schadcode zu stoßen. Bei Benutzern von Microsofts Windows hat sich größtenteils inzwischen herumgesprochen, dass Software nicht einfach beliebig heruntergeladen und installiert werden sollte. Diese Erkenntnis muss in abgeschwächter Weise auch bei den Benutzern von Android auftreten.

Android Market – Zusammenfassung

Insgesamt stellt Googles Android Market einen der größten Risikofaktoren für die Sicherheit von Android dar. Zwar schützt er die Benutzer gut vor unprofessionellen Störenfriedern, die Schadsoftware nur aus Spaß erstellen, aber die deutlich gefährlicheren Absichten eines Datendiebes oder Entwicklers von

Kosten erzeugenden Programmen vermag er nur eingeschränkt zu unterbinden. Der deutlich sicherere Ansatz, den viele Linux-Distributionen mit ihren Software-Repositories verfolgen, verbietet sich als Alternative für Google, da der Erfolg von Android zu einem bedeutenden Teil auf die Verfügbarkeit von einer großen Menge nützlicher oder spaßiger Anwendungen abhängt. Besser geeignet scheint der Mittelweg, den Apple mit ihrem App Store verfolgt: Weniger Hürden für Entwickler, aber dennoch Kontrolle jeder Anwendung vor der Zulassung, vorausgesetzt, diese Kontrolle kann mit vertretbarem Aufwand schnell und genau genug erfolgen. Dem Freiheitsbedürfnis einiger Benutzer wäre mit der Option, Anwendungen aus Fremdquellen zu installieren, möglicherweise ausreichend gedient. Das Problem, dass viele Systemanwendungen nur Updates erfahren, wenn das ganze Betriebssystem in einer neuen Version erscheint, wäre damit allerdings nicht gelöst. Auch hier besteht Handlungsbedarf.

In seiner jetzigen Form können die Softwareverteilungs-Mechanismen von Android nicht als sicher bezeichnet werden. Es bestehen einige Ansätze, die Sicherheit der Benutzer zu gewährleisten, diese sind allerdings angesichts der immer unüberschaubareren Zahl verfügbarer Anwendungen und der steigenden Attraktivität der mobilen Plattformen für Schadsoftware-Entwickler nicht ausreichend für die Zukunft.

4.8 Betriebssystem-Updates

Bei jedem Betriebssystem ist die Frage nach der Update-Vorgehensweise des Herstellers kritisch. Viele Angriffe auf Computersysteme zielen nicht auf eine darauf installierte Anwendungssoftware ab, sondern auf Lücken im zugrunde liegenden Betriebssystem selbst. Diese Angriffe hätten teilweise verhindert werden können, wären rechtzeitig Sicherheitsaktualisierungen durchgeführt worden. Ein Beispiel ist der Wurm Conficker, der Ende 2008 eine große Menge an Windows-Rechnern über eine Lücke im RPC-System infizierte und einigen finanziellen Schaden anrichtete. Ein entsprechendes Sicherheitsupdate lag Wochen vorher auf Microsofts Updateservern bereit, wurde aber teilweise nicht

schnell genug von den Administratoren der betroffenen Systeme eingespielt. [36]

Viele grundlegende Komponenten des Android-Systems können nicht über den Market aktualisiert werden. Dazu gehören der Kernel, die Dalvik-VM und die Webkit-Browserengine. Aber auch einige vorinstallierte Anwendungen sind davon betroffen. Sicherheitslücken in diesen Komponenten zu beheben ist nur über ein Update des Betriebssystems möglich. Daher ist es für die Sicherheit von Android wichtig, dass derartige Updates schnell zu den Benutzern gelangen.

Die Geschwindigkeit, mit der Google neue Versionen von Android veröffentlicht, hat in diesem Jahr deutlich abgenommen. In der Anfangsphase der Entwicklung war es laut Google noch nötig, schnell neue Versionen zu erstellen, da das Betriebssystem noch nicht ganz ausgereift war. Zwischen den Versionen 1.6, 2.0, 2.1 und 2.2 lagen jeweils nur wenige Monate. Die Hersteller der mobilen Endgeräte, die mit Android betrieben wurden, haben sich darüber beschwert, dass sie mit diesem Tempo nicht mithalten könnten. Seit Android Version 2.2 erreicht hat, ist laut Google nun eine Entspannungsphase eingetreten. Android sei nun ausgereift genug, dass man sich auf einen Update-Zyklus von einer Version im Jahr festlegen könne. Für die Entwickler und Gerätehersteller ist dies eine gute Nachricht, da sie mehr Zeit haben, sich auf eine neue Version vorzubereiten. [37]

Angesichts der oben angesprochenen Problematik, dass Systemkomponenten nur mittels Betriebssystemupdate aktualisiert werden können, ist dies jedoch eine riskante Strategie. Eine Sicherheitslücke in einer derartigen Komponente würde im schlimmsten Fall erst nach einem Jahr geschlossen werden können. Wie real diese Bedrohung ist, zeigt der Fall einer vor kurzem entdeckten Lücke. Diese betrifft den Browser von Android. Sie ermöglicht es Angreifern, vom Benutzer erstellte Dateien auszulesen, wenn dieser einen Link im Browser anklickt, mit der Einschränkung, dass der Dateisystem-Pfad zu diesen Dateien bekannt sein muss. Google hat das Problem nachvollzogen und einen Patch entwickelt, der aber für Android Version 2.3 zu spät kam. Die Lücke ist also

auch in der gerade neu veröffentlichten Version noch offen. Wann sie geschlossen wird, ist unbekannt. [38] Falls dies erst mit einer angenommenen Version 2.4 geschehen sollte, wäre die Lücke über ein Jahr nach Bekanntwerden noch offen.

Noch schlimmer ist die Lage, wenn man sich das Updateverhalten der Gerätehersteller ansieht. Android-Geräte erhalten Betriebssystem-Updates nicht von Google direkt, sondern von den Herstellern der Geräte. Diese müssen, wenn eine neue Version von Android erscheint, das Update selber an ihre Kunden weitergeben. Ausnahmen von dieser Regel stellen die Google-eigenen Smartphones Nexus One und Nexus S dar, die ihre Updates von Google selbst beziehen. Die Hersteller passen die auf ihren Geräten laufenden Android-Versionen meist noch zusätzlich an. Sie verändern beispielsweise oft die Benutzungsoberfläche und fügen eigene Anwendungen hinzu. Diese Anpassungen brauchen Zeit und machen Arbeit. Dadurch verzögert sich die Veröffentlichung der Updates noch einmal deutlich. Selbst bei den Top-Modellen einiger Hersteller kann dies der Fall sein, wie das Beispiel Samsung zeigt: Das Update auf die Android Version 2.2 für das Samsung Galaxy S i9000 erschien im November 2010 und damit fünf Monate später, als Google diese veröffentlicht hatte. [39] Noch weiter verzögert wird ein solches Update oftmals, wenn zusätzlich noch die Netzbetreiber, die die Geräte verkaufen, Veränderungen vornehmen wollen, etwa einen Startbildschirm mit ihrem Logo oder einige eigene Anwendungen. Im Falle des Samsung-Geräts und des Netzbetreibers Vodafone betrug diese Verzögerung noch einmal einige Tage. [40]

In einigen Fällen sind für Geräte, die erst wenige Monate auf dem Markt sind, schon keine Updates mehr vorgesehen. So geschah es beim Motorola Milestone XT720, welches zudem nach der Veröffentlichung von Android 2.2 noch mit Version 2.1 ausgeliefert wurde. [41]

Benutzern solcher Geräte bleibt nur übrig, zu hoffen, dass sie nicht von schwerwiegenden Sicherheitslücken betroffen sind, bis sie sich ein neues Smartphone zulegen, oder dass entsprechende Lücken nicht im großen Maße ausgenutzt werden.

Die Update-Politik sowohl von Google als auch von den Geräteherstellern kann in Bezug auf die Sicherheit der Android-Smartphones angesichts der genannten Probleme nur als mangelhaft eingestuft werden. Aktualisierungen kommen zu spät, zu selten, teilweise auch gar nicht. Wie die zukünftige Entwicklung in dieser Hinsicht sein wird, kann nur schwer beurteilt werden. Zwar veröffentlichte Google inzwischen unerwartet ein kleines Update für Android 2.2 mit der Versionsnummer 2.2.1 und zeigt damit, dass es ihnen durchaus möglich ist, Veränderungen nicht nur in Form großer neuer Versionen zu veröffentlichen, allerdings existiert für 2.2.1 keine Liste der vorgenommenen Änderungen. [42] Ob Sicherheitslücken geschlossen wurden, ist nicht bekannt. Auch bei ihren anderen Android-Versionen kommunizierte Google bisher nur die neuen Funktionen, nicht aber behobene Fehler. Googles Vorgehensweise ist sehr intransparent. Dies ist allerdings kaum verwunderlich, wenn man das Gesamtbild betrachtet. Würde Google zu jeder neuen Version auflisten, welche Sicherheitslücken geschlossen wurden, würde man Angreifern damit eine Blaupause für Attacken auf eine Vielzahl von Geräten liefern, die noch mit veralteten Versionen laufen. Nur, wenn für alle Geräte gleichzeitig eine Aktualisierung vorliegen würde, könnte eine solche Liste ohne schwerwiegende Konsequenzen veröffentlicht werden. Google setzt hier also auf das alte Prinzip „Security by Obscurity“, Sicherheit durch Verheimlichung der Lücken, im Gegensatz zum Open-Source-Ansatz der Offenlegung aller Informationen. Wie lange dies funktionieren wird, ist ungewiss. Vielleicht muss erst eine schwere Sicherheitslücke gefunden und in großem Maßstab ausgenutzt werden, bevor der Updateprozess überdacht wird.

5 Themennahe Arbeiten

Es existieren mehrere andere Arbeiten, die sich mit der Sicherheit von Android in der einen oder anderen Weise beschäftigen. Einige davon werden im Folgenden vorgestellt.

William Enck et al. haben mehrere Paper zur Android-Sicherheit vorgestellt. In „Understanding Android Security“ erläutern sie anhand einer Beispielanwendung Androids Komponentensystem, die Funktionsweise von Intents und Androids Rechtesystem. Zum Schluss stellen sie noch ein Werkzeug namens „Kirin“ vor, mit welchem sich die in den Manifest-Dateien befindlichen Rechtforderungen der Anwendungen auslesen und analysieren lassen. Anhand festgelegter Regeln lassen sich Zustände definieren, die als unsicher eingestuft werden. Kirin überprüft dann, ob Anwendungen mit den von ihnen geforderten Rechten einen solchen Zustand herstellen würden. [53]

In „TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones“ stellen sie ein Werkzeug namens „Taintdroid“ vor, mit dem der Informationsfluss innerhalb Androids nachvollzogen werden kann. Mittels des Werkzeugs kann untersucht werden, wie Anwendungen persönliche Daten des Benutzers manipulieren oder auslesen. Es kann verfolgt werden, welchen Weg diese Informationen im System nehmen und ob sie es verlassen. So haben die Autoren bei der Untersuchung von 30 Anwendungen bei 20 von ihnen potentiell unerwünschte Behandlung von Benutzerdaten festgestellt. [54]

In „Semantically Rich Application-Centric Security in Android“ stellen sie ein Infrastruktur-Framework namens „Secure Application INTeraction“ (Saint) vor, welches das normale Rechtesystem von Android erweitert. Saint gibt Anwendungen die Möglichkeit, selber genauer kontrollieren zu können, welche anderen Anwendungen auf welche Weise auf ihre Funktionen und Schnittstellen zugreifen dürfen. Dazu lassen sich in Saint Richtlinien festlegen, mit denen sowohl die zum Installationszeitpunkt erteilten Rechte kontrolliert, als auch die Ertei-

lung von Rechten zur Laufzeit abhängig vom jeweiligen Zustand (etwa Ort, Zeit oder Netzverbindung) gesteuert werden können. [55]

Asaf Shabtai et al. analysieren in ihrem Paper „Google Android: A State-of-the-Art Review of Security Mechanisms“ eine Reihe von Komponenten des Android-Systems auf ihre Sicherheit hin, untersuchen die Wirksamkeit von Androids Sicherheitsmechanismen und die Anwendbarkeit existierender Linux-Schadsoftware. Sie kommen zu dem Schluss, dass Android grundsätzlich gut abgesichert ist. Angriffe würden meist das Entdecken einer Sicherheitslücke im Kernel voraussetzen, über die Root-Zugriff erlangt werden kann. Andere Komponenten seien meist nur innerhalb ihres eigenen Anwendungsrahmens gefährdet. Kritik üben sie an dem unzureichend abgesicherten Rechtesystem. Des Weiteren stellen sie einige Gegenmaßnahmen vor, mit denen Angriffe abgewehrt werden könnten und schätzen ihre Anwendbarkeit in Android ein. [28]

Himanshu Dwivedi et al. stellen in ihrem Buch „Mobile Application Security“ die Sicherheitsmechanismen und den Systemaufbau zahlreicher Betriebssysteme für mobile Endgeräte vor, unter anderem Android. Sie beschreiben Androids IPC-Mechanismus, das Sandboxing, das Rechtesystem, den Komponentenaufbau von Android-Anwendungen und geben Hinweise zur Vermeidung von SQL-Injections. Sie bewerten Android als sehr gute Plattform für die Entwicklung sicherer Anwendungen. Kritisch sehen sie die in der Vergangenheit entdeckte Zahl an Sicherheitslücken in einigen der in Android verwendeten Systembestandteile, etwa im Linux-Kernel und WebKit. [19]

6 Zusammenfassung der Angriffsvektoren

In diesem Kapitel werden die im Rahmen dieser Arbeit gefundenen Angriffsvektoren auf die Sicherheit Androids und seiner Benutzer tabellarisch zusammengefasst. Da die Erfolgsaussichten mehrerer der Angriffe von dem Kenntnisstand und der Wachsamkeit der Benutzer abhängt, werden dafür zwei Beispielbenutzer definiert:

Benutzer A: Benutzer A verwendet sein Smartphone wie ein Telefon mit erweiterten Funktionen. Seine Computerkenntnisse beschränken sich auf die Benutzung einiger Programme, etwa ein Schreibprogramm, ein Browser, ein E-Mail-Programm und einen Multimedia-Player. Er ist nicht an technischen Hintergründen interessiert und liest sich lange Meldungen nicht durch, sondern bestätigt diese einfach, um schnell ans Ziel zu kommen. Er vertraut darauf, dass der Gerätehersteller ihn vor Gefahren schützt und er jede Funktion seines Geräts bedenkenlos verwenden kann.

Benutzer B: Benutzer B verwendet sein Smartphone wie einen Minicomputer. Er besitzt Kenntnisse über die Abläufe in Computersystemen und weiß um die Wichtigkeit von Sicherheitsupdates für seine Programme. Er ist misstrauisch gegenüber zusätzlicher Software für sein Gerät und liest sich Meldungen durch, bevor er sie bestätigt. Er weiß, dass er bei der Verwendung seines Geräts soviel Vorsicht walten lassen muss, wie bei einem PC.

In der Tabelle wird aufgeführt, wie gefährlich ein solcher Angriff für den jeweiligen Benutzer wäre. Die Einstufung lautet wie folgt: niedrige Gefahr, mittlere Gefahr, hohe Gefahr

Außerdem wird eine Einschätzung der Wahrscheinlichkeit eines derartigen Angriffs abgegeben, aufgrund der Komplexität und der Erfolgsaussichten. Die Wahrscheinlichkeit wird dafür in eine der folgenden Stufen eingeteilt: sehr unwahrscheinlich, unwahrscheinlich, wahrscheinlich, sehr wahrscheinlich.

Angriff	Gefahr für Benutzer A	Gefahr für Benutzer B	Wahrscheinlichkeit
Anwendung nutzt Sicherheitslücke im Kernel, um an Rootrechte zu gelangen und Schaden anzurichten	hohe Gefahr, eventuelle Hinweise auf Fehlverhalten der Anwendung in den Kommentaren werden von ihm nicht gelesen	hohe Gefahr; er liest zwar die Kommentare, aber diese enthalten oft keine nützlichen Informationen	wahrscheinlich, da entdeckende Lücken regelmäßig gefunden werden
Anwendung gibt vor, Internetzugang für Werbung zu benötigen, sendet aber tatsächlich unbefugte Daten ins Netz	hohe Gefahr, da Recht zu global ausgelegt ist, um tatsächlichen Zweck zu erkennen	hohe Gefahr, da Recht zu global ausgelegt ist, um tatsächlichen Zweck zu erkennen	sehr wahrscheinlich, da einfach durchzuführen
Zwei Anwendungen nutzen Shared User ID, um an mehr Rechte zu kommen und Schaden anzurichten	hohe Gefahr, da keinerlei Kenntnis über diese Art Angriff	hohe Gefahr, da zwar in der Lage, mittels entsprechenden Werkzeugen den Angriff zu erkennen, jedoch ist dies selbst für diese Art Benutzer eine unübliche Vorgehensweise	wahrscheinlich; einfach durchzuführen, setzt jedoch Installation zweier unabhängiger Anwendungen voraus

Angriff	Gefahr für Benutzer A	Gefahr für Benutzer B	Wahrscheinlichkeit
Eine Anwendung gibt zweiter Anwendung mittels selbst definierten Rechten Zugriff auf seine Funktionen und Ressourcen; zweite Anwendung benutzt diese, um Informationen auszuspähen	hohe Gefahr, da keine Möglichkeit zur Erkennung, bevor Schaden entsteht	hohe Gefahr, da keine Möglichkeit zur Erkennung, bevor Schaden entsteht	unwahrscheinlich, da Angriff über Shared User ID leichter durchzuführen
Eine Anwendung gibt zweiter Anwendung mittels „exported“-Attribut Zugriff auf seine Funktionen und Ressourcen; zweite Anwendung benutzt diese, um Informationen auszuspähen	hohe Gefahr, da keine Möglichkeit zur Erkennung, bevor Schaden entsteht	hohe Gefahr, da keine Möglichkeit zur Erkennung, bevor Schaden entsteht	unwahrscheinlich, da Angriff über Shared User ID leichter durchzuführen
Eine Anwendung gibt zweiter Anwendung mittels URL-Permission Zugriff auf einige seiner Ressourcen; zweite Anwendung benutzt diese, um Informationen auszuspähen	hohe Gefahr, da keine Möglichkeit zur Erkennung, bevor Schaden entsteht	hohe Gefahr, da keine Möglichkeit zur Erkennung, bevor Schaden entsteht	unwahrscheinlich, da Angriff über Shared User ID leichter durchzuführen

Angriff	Gefahr für Benutzer A	Gefahr für Benutzer B	Wahrscheinlichkeit
Schadsoftware aus dem Android Market fordert viele Rechte an, um Benutzerdaten auszuspähen	hohe Gefahr, da er sich die geforderten Rechte nicht gründlich ansieht	mittlere Gefahr; Rechteforderungen machen ihn misstrauisch, aber die Beurteilung fällt ihm durch die Grobheit einiger Rechte schwer	sehr wahrscheinlich, da leicht durchzuführen
Angreifer erlangt Kontrolle über Googles GTalk-Server und installiert allen Android-Geräten auf dem Planeten Schadsoftware	hohe Gefahr, keinerlei Einflussmöglichkeit	hohe Gefahr, keinerlei Einflussmöglichkeit	sehr unwahrscheinlich, da äußerst Aufwendig
Angreifer besorgt sich mittels trojanischem Pferd Accountdaten von Google-Benutzern und verwendet sie, um per Web-Market Schadsoftware auf den Geräten der Benutzer zu installieren	mittlere Gefahr, da nur ein Bruchteil aller Benutzer betroffen und Erlangen der Benutzerdaten von verbreiteten Virencannern unterbunden werden könnten	niedrige Gefahr, da trojanisches Pferd bei dieser Art Benutzer sowohl auf Virencanner als auch weitere Sicherheitsmaßnahmen und höhere Vorsicht stoßen	unwahrscheinlich, da hoher Aufwand und zweifelhafte Erfolgsaussichten

Angriff	Gefahr für Benutzer A	Gefahr für Benutzer B	Wahrscheinlichkeit
Anwendung nutzt Sicherheitslücke in veralteter, aber noch verbreiteter Android-Version, um unbefugt an Informationen zu gelangen	hohe Gefahr, da Wichtigkeit von Updates ihm nicht bekannt ist	mittlere Gefahr; Wichtigkeit von Updates ist ihm bekannt, daher steigt er eher auf ein neueres Gerät um oder installiert neuere Version per Custom-ROM	wahrscheinlich, da Anzahl entdeckter Sicherheitslücken mit Alter der Software steigt

7 Abschlussbetrachtung und Ausblick

In dieser Diplomarbeit wurde die Architektur Androids näher betrachtet und beschrieben, sowie viele ihrer Komponenten einer Untersuchung im Bezug auf die Sicherheit unterzogen. Des Weiteren wurden Android nicht nur als Betriebssystem betrachtet, sondern als Plattform, zu der auch außerhalb der Geräte liegende Mechanismen und Infrastruktur gehören. Der Android Market spielt, wie sich gezeigt hat, für die Sicherheit der Benutzer, ihrer Daten und ihres Besitzes eine ebenso große Rolle, wie der Betriebssystem-Kernel, vielleicht eine größere. Die unterschiedlichen Update-Strategien der Gerätehersteller sind ebenfalls ein wichtiger Aspekt bei der Betrachtung des Gesamtbilds.

Die Untersuchung hat gezeigt, dass einige der größten Bedrohungen für die Sicherheit nicht von fehlerhaften Betriebssystem-Komponenten ausgehen, sondern dass das Mittel des Social Engineering einen Angreifer viel leichter zum Erfolg führen kann. Dieser Faktor wurde in vielen bisherigen Arbeiten nur am Rande behandelt.

Die Beurteilung Androids muss daher aus zweierlei Blickwinkeln betrachtet werden. Zum einen die Sicherheit des Betriebssystems selbst. Android bietet eine solide Basis, vor allem wegen seiner grundsätzlichen Anwendungstrennung. Die Tatsache, dass jede Anwendung sich in einer Sandbox befindet, aus der sie nur durch explizit gewährte Rechte heraus kann, genügt bereits, um eine Reihe von Angriffsvektoren auszuschließen.

Die Verwendung von Bibliotheken und Komponenten, die sich bereits in anderen Betriebssystemen bewährt haben, unterstützt ebenfalls den Eindruck, dass Android einen sicheren Grundzustand besitzt, auch wenn beispielsweise der Linux-Kernel des Öfteren von Sicherheitslücken betroffen ist. Ein eigens entwickelter, quelloffener Kernel hätte wahrscheinlich mit mehr Problemen zu kämpfen. Der Ansatz, die Entwicklung Androids größtenteils mit offen gelegtem Code durchzuführen, kann zu einer Erhöhung der Sicherheit beitragen, wenn es mehr Menschen dazu bringt, die Quellen auf Fehler zu untersuchen und diese Google mitzuteilen. Allerdings besteht an dieser Stelle auch Nachholbedarf.

Eine bessere Zusammenarbeit mit den Linux-Kernelentwicklern und die Aufnahme der Android-spezifischen Änderungen in den Original-Kernel würden der Sicherheit zugute kommen. Das Rechte-System von Android ist in seiner jetzigen Form außerdem noch zu grob und zu sehr auf Benutzungsfreundlichkeit statt Sicherheit ausgelegt. Insgesamt kann dem Android-Betriebssystem jedoch eine gute Sicherheit bescheinigt werden.

Anders sieht es aus, wenn man Android als ganze Plattform betrachtet. Hier spielen die Faktoren Market, Benutzerverhalten und Updates mit hinein. Das offene, kaum kontrollierte System des Markets bietet eine Vielzahl an Möglichkeiten für Social Engineering-Attacken. Gleichzeitig vermittelt das Konzept des Markets, alle Anwendungen aus einer extra dafür geschaffenen Quelle zu beziehen, den Benutzern ein falsches Sicherheitsgefühl. Sie mögen glauben, sie bekämen ihre Programme alle aus einer Hand, der von Google, welche dafür sorgt, dass sie vor schädlicher Software geschützt sind. Dass sie bei der Auswahl ihrer Anwendungen im Market fast ebenso viel Aufmerksamkeit und Vorsicht walten lassen müssen, dürfte vielen nicht klar sein.

Die Updatepolitik vieler Gerätehersteller ist ein weiteres Problem. Viele Benutzer sind zur Zeit noch gezwungen, sich mit veralteten Version zufrieden zu geben, wenn sie keine neuere Gerät kaufen oder auf Custom-ROMs zurückgreifen wollen. Viele Sicherheitslücken, die Google in Android längst geschlossen hat, sind in einer großen Zahl auf dem Markt befindlicher Geräte noch existent.

Aus diesen Gründen besteht für die Android-Plattform als Ganzes Bedarf für Verbesserungen. Andere mobile Plattformen machen es vor: Updates können von einer zentralen Stelle bezogen werden. Eine bessere Kontrolle des Markets auf Kosten der Offenheit wäre eine Möglichkeit. Des Weiteren existieren bereits Werkzeuge wie Kirin und Taintdroid sowie Frameworks wie Saint, mit denen die vorhandenen Probleme angegangen werden könnten.

Wie die Entwicklung von Android weitergehen wird, muss die Zukunft zeigen. Es steht zu hoffen, dass Google die genannten Kritikpunkte erkennt und an-

geht. Die Verbreitung von Android wird jedenfalls höchstwahrscheinlich weiter zunehmen. Dadurch wird die Plattform für Angreifer immer interessanter.

Zukünftige Arbeiten könnten die Weiterentwicklung Androids nachverfolgen. Außerdem wäre eine nähere Betrachtung des Binder-Frameworks und der Dalvik Virtual Machine möglich. Diese konnten im Rahmen dieser Diplomarbeit nicht umfassend behandelt werden.

8 Literaturverzeichnis

- [1]
Gartner.com: Gartner Says Worldwide Mobile Phone Sales Declined 8.6 Per Cent and Smartphones Grew 12.7 Per Cent in First Quarter of 2009,
<http://www.gartner.com/it/page.jsp?id=985912>
(Abruf am 02.01.2011)
- [2]
Claudia Eckert: IT-Sicherheit, 6. überarbeitete und erweiterte Auflage,
Oldenbourg Verlag, München, 2009
- [3]
Philipp Schaumann: Schutz gegen Social Engineering - neue psychologische Ansätze,
http://sicherheitskultur.at/social_engineering.htm
(Abruf am 05.02.2011)
- [4]
Lowlevel: Kernel,
<http://www.lowlevel.eu/w/index.php?title=Kernel&oldid=8651>
(Abruf am 02.01.2011)
- [5]
Alessandro Rubini, Jonathan Corbet: Linux-Gerätetreiber, 2. Auflage,
April 2002, O'Reilly Verlag, Organisation des Kernels,
<http://www.oreilly.de/german/freebooks/linuxdrive2ger/x217.html>
(Abruf am 02.01.2011)
- [6]
Linux Kernel Monkey Log: Android and the Linux kernel community,

- <http://www.kroah.com/log/linux/android-kernel-problems.html>
(Abruf am 02.01.2011)
- [7]
Android Developers,
<http://developer.android.com/sdk/index.html>
(Abruf am 02.01.2011)
- [8]
Android Open Source Project: Release features - Android 1.0,
[https://sites.google.com/a/android.com/opensource/
release-features---android-1-0](https://sites.google.com/a/android.com/opensource/release-features---android-1-0)
(Abruf am 02.01.2011)
- [9]
Ars Technica: Dream(sheep++): A developer's introduction to Google
Android,
[http://arstechnica.com/open-source/reviews/2009/02/
an-introduction-to-google-android-for-developers.ars](http://arstechnica.com/open-source/reviews/2009/02/an-introduction-to-google-android-for-developers.ars)
(Abruf am 02.01.2011)
- [10]
Android Open Source Project: Licensing Information,
<http://source.android.com/source/licenses.html>
(Abruf am 04.02.2011)
- [11]
Google IO: 2008 Google I/O Session Videos and Slides – Dalvik VM In-
ternals,
<http://sites.google.com/site/io/dalvik-vm-internals>
(Abruf am 02.01.2011)
- [12]
Let's talk about Google Android: Google Android Native libc Bionic
library,
<http://discuz-android.blogspot.com/2008/10/>

`google-android-native-libc-bionic.html`

(Abruf am 02.01.2011)

[13]

The FreeType Project: What is FreeType 2?,

`http://www.freetype.org/freetype2/index.html`

(Abruf am 01.02.2011)

[14]

SQLite.org,

`http://www.sqlite.org/`

(Abruf am 02.01.2011)

[15]

WebKit.org: The WebKit Open Source Project,

`http://webkit.org/`

(Abruf am 02.01.2011)

[16]

Android Developers: What is Android?,

`http://developer.android.com/guide/basics/what-is-android.html`

(Abruf am 02.01.2011)

[17]

Android Developers: Sample Code,

`http://developer.android.com/resources/samples/Wiktionary/AndroidManifest.html`

(Abruf am 05.02.2011)

[18]

Android Developer's Guide: Intents and Intent Filters,

`http://developer.android.com/guide/topics/intents/intents-filters.html`

(Abruf am 05.02.2011)

- [19]
Himanshu Dwivedi et al.: Mobile Application Security, McGraw-Hill, 2010
- [20]
Engadget.com: Exclusive: Android Froyo to take a serious shot at stemming platform fragmentation,
<http://www.engadget.com/2010/03/29/exclusive-android-froyo-to-take-a-serious-shot-at-stemming-plat/>
(Abruf am 02.01.2011)
- [21]
Heise Open Source: Android 2.3 mit Ext4-Dateisystem,
<http://www.heise.de/open/meldung/Android-2-3-mit-Ext4-Dateisystem-1152964.html>
(Abruf am 18.01.2011)
- [22]
National Institute of Standards and Technology (NIST): Announcing the ADVANCED ENCRYPTION STANDARD (AES), Federal Information Processing Standards Publication 197, 2001
<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
(Abruf am 04.02.2011)
- [23]
Android Developer's Guide: Security and Permissions,
<http://developer.android.com/guide/topics/security/security.html>
(Abruf am 05.02.2011)
- [24]
Coding Relic: The Six Million Dollar LibC,
<http://codingrelic.geekhold.com/2008/11/six-million-dollar-libc.html>
(Abruf am 01.02.2011)

- [25]
SecurityTracker: FreeBSD libc Buffer Overflow in inet_network() May
Let Users Deny Service or Execute Arbitrary Code,
<http://securitytracker.com/id?1019189>
(Abruf am 01.02.2011)
- [26]
Red Hat: Important: freetype security update,
<https://rhn.redhat.com/errata/RHSA-2009-1061.html>
(Abruf am 01.02.2011)
- [27]
MandrivaUser.de: [security] Schwachstelle in der Freetype Bibliothek,
<http://www.mandrivauser.de/forum/viewtopic.php?f=40&t=29982>
(Abruf am 01.02.2011)
- [28]
Asaf Shabtai et al.: Google Android: A State-of-the-Art Review of Security
Mechanisms, 2009
- [29]
Android Developers: PendingIntent,
[http://developer.android.com/reference/android/app/
PendingIntent.html](http://developer.android.com/reference/android/app/PendingIntent.html)
(Abruf am 05.02.2011)
- [30]
Heise Online: Erster SMS-Trojaner für Android gesichtet,
[http://www.heise.de/newsticker/meldung/
Erster-SMS-Trojaner-fuer-Android-gesichtet-1053377.html](http://www.heise.de/newsticker/meldung/Erster-SMS-Trojaner-fuer-Android-gesichtet-1053377.html)
(Abruf am 02.01.2011)
- [31]
Apple: iOS Developer Program,
<http://developer.apple.com/programs/ios/>
(Abruf am 02.01.2011)

- [32]
Apple: Support Center – Identity Verification,
<http://developer.apple.com/support/ios/identity-verification.html>
(Abruf am 02.01.2011)
- [33]
Debian: Debians New-Maintainer-Ecke,
<http://www.debian.org/devel/join/newmaint>
(Abruf am 02.01.2011)
- [34]
Debian: Anleitung für zukünftige Debian-Betreuer,
<http://www.debian.org/doc/maint-guide/index.de.html>
(Abruf am 02.01.2011)
- [35]
Heise Online: Google bringt den Android Market ins Web,
<http://www.heise.de/newsticker/meldung/Google-bringt-den-Android-Market-ins-Web-1182608.html>
(Abruf am 04.02.2011)
- [36]
Heise Security: Windows-Wurm nimmt an Fahrt auf,
<http://www.heise.de/security/meldung/Windows-Wurm-nimmt-an-Fahrt-auf-218407.html>
(Abruf am 02.01.2011)
- [37]
Stereopoly: Entspannung für Entwickler und Hersteller: Android Updates nur noch einmal im Jahr,
<http://www.stereopoly.de/entspannung-fuer-entwickler-und-hersteller-android-updates-nur-noch-einmal-im-jahr/>
(Abruf am 02.01.2011)

- [38]
Heise Security: Android-Lücke ermöglicht Datenklau,
[http://www.heise.de/security/meldung/
Android-Luecke-ermoeglicht-Datenklau-1140932.html](http://www.heise.de/security/meldung/Android-Luecke-ermoeglicht-Datenklau-1140932.html)
(Abruf am 02.01.2011)
- [39]
Golem.de: Galaxy S I9000 – Update auf Android 2.2 Froyo verfügbar,
<http://www.golem.de/1011/79173.html>
(Abruf am 02.01.2011)
- [40]
Vodafone-Forum: Der Galaxy S I9000 2.2 Update Thread,
<https://www.vodafone.de/forum/posts/list/45/3200.page>
(Abruf am 02.01.2011)
- [41]
Heise Online: Keine Android-Updates für Milestone XT720,
[http://www.heise.de/newsticker/meldung/
Keine-Android-Updates-fuer-Milestone-XT720-1141659.html](http://www.heise.de/newsticker/meldung/Keine-Android-Updates-fuer-Milestone-XT720-1141659.html)
(Abruf am 02.01.2011)
- [42]
Teltarif.de: Google veröffentlicht neue Android-Version 2.2.1,
[http://www.teltarif.de/google-android-software-update/news/
40174.html](http://www.teltarif.de/google-android-software-update/news/40174.html)
(Abruf am 02.01.2011)
- [43]
Jon Oberheide: Android Hax,
[http://jon.oberheide.org/files/summercon10-androidhax-
jonoberheide.pdf](http://jon.oberheide.org/files/summercon10-androidhax-jonoberheide.pdf)
(Abruf am 04.01.2011)
- [44]
Jon Oberheide: Remote Kill and Install on Google Android,

<http://jon.oberheide.org/blog/2010/06/25/remote-kill-and-install-on-google-android/>
(Abruf am 28.01.2011)

[45]

Jon Oberheide: A Peek Inside the GTalkService Connection,
<http://jon.oberheide.org/blog/2010/06/28/a-peek-inside-the-gtalkservice-connection/>
(Abruf am 28.01.2011)

[46]

Heise Open Source: Die Woche: Open Source im Fadenkreuz,
<http://www.heise.de/open/artikel/Die-Woche-Open-Source-im-Fadenkreuz-1179052.html>
(Abruf am 28.01.2011)

[47]

Coverity: Coverity Scan 2010 Open Source Integrity Report Reveals High Risk Software Flaws in Android,
<http://www.coverity.com/html/press/coverity-scan-2010-report-reveals-high-risk-software-flaws-in-android.html>
(Abruf am 04.01.2011)

[48]

Silicon.de: Android mit 359 Sicherheitslücken,
http://www.silicon.de/technologie/mobile/0,39044013,41540156,00/android_mit_359_sicherheitsluecken.htm
(Abruf am 04.01.2011)

[49]

Fast Company: Android's Kernel Security Flaws Highlight Its Strengths,
<http://www.fastcompany.com/1699578/android-kernel-security-flaws-highlight-both-its-weakness-and-strengths>
(Abruf am 04.01.2011)

- [50]
Android Developer's Guide: The AndroidManifest.xml File,
[http://developer.android.com/guide/topics/manifest/
permission-element.html#plevel](http://developer.android.com/guide/topics/manifest/permission-element.html#plevel)
(Abruf am 09.01.2011)
- [51]
Android Reference: Manifest.permission,
[http://developer.android.com/reference/android/Manifest.
permission.html](http://developer.android.com/reference/android/Manifest.permission.html)
(Abruf am 09.01.2011)
- [52]
Chris Hadley: COSC346 - User Interface Labs – Developing G-Phone Ap-
plications,
<http://www.cs.otago.ac.nz/cosc346/labs/COSC346Labs-10.pdf>
(Abruf am 11.01.2011)
- [53]
William Enck et al.: Understanding Android Security, IEEE Security &
Privacy Magazine, 2009
- [54]
William Enck et al.: TaintDroid: An Information-Flow Tracking System
for Realtime Privacy Monitoring on Smartphones, In Proc. of the USENIX
Symposium on Operating Systems Design and Implementation (OSDI),
Vancouver, 2010
- [55]
William Enck et al.: Semantically Rich Application-Centric Security in
Android, Proceedings of the 25th Annual Computer Security Applications
Conference (ACSAC), Honolulu, 2009