Android Security
Sicherheit für mobile Endgeräte

Markus Schlichting
Hochschule der Medien Stuttgart
e-mail: markus.schlichting@hdm-stuttgart.de

Hausarbeit im Seminar
Spezielle Themen der Software-Technologie
bei Prof. Walter Kriha

7. Januar 2010

Inhaltsverzeichnis

Abstract

Android ist ein Betriebssystem für Mobiltelefone und wird von einem durch Google angeführten Konsortium aus 33 namhaften Firmen aus den Bereichen der Mobilfunk-, Software- und Internetbranche entwickelt. Erste Geräte sind seit gut einem Jahr verfügbar. Was sich hinter dem System und der Plattform mit Blick auf Sicherheitsaspekte verbirgt zeigt diese Arbeit.

1 Einleitung

Die Bedeutung mobiler Endgeräte nimmt ständig zu. Von einer kritisch beäugten technischen Spielerei wurden sie kurzzeitig zu einer Art Statussymbol, um sich nun immer mehr in unseren Alltag zu integrieren und zu unseren ständigen Begleitern zu werden. Ärmeren Regionen und Ländern ermöglichen sie sogar, den teuren Ausbau fester Kommunikationsnetze zu überspringen und direkt in das moderne Kommunikationszeitalter einzusteigen (China, Afrika)[1].

Neben der Verbreitung schreitet auch die Entwicklung der Geräte immer weiter voran, so dass moderne Geräte als „Smartphones“ bezeichnet werden und sie so von bisherigen Mobiltelefonen oder der Zwischenstufe „Featurephones“ hervorgehoben werden. Das *Smart* Attribut erhalten diese Geräte vor allem deshalb, weil sie viele Merkmale aus der Welt der PC integrieren: Sie sind über die normalen Mobilfunk-Sprachverbindungen hinaus netzwerkfähig und können relativ große Mengen Datenverkehr verarbeiten. Sie verfügen über verhältnismäßig viel Speicherplatz und erlauben es dem Benutzer, ihren Funktionsumfang mit zusätzlichen Anwendungen zu erweitern und den eigenen Bedürfnissen anzupassen. Sie integrieren ein GPS Empfänger zur Navigation und Positionsbestimmung und verfügen über ausreichend große Displays, so dass man bereits erste Benutzer beim anschauen ihrer Lieblings-TV-Serie in der S-Bahn beobachten kann.

Ein derartiger Funktionsumfang benötigt natürlich auch ein passendes Betriebssystem. Google ist hier durch den Zukauf des *Android*-Systems aus einem kleinen Startup Unternehmen und die Weiterentwicklung innerhalb eines Konsortiums aus 33 namhaften Firmen angetreten, ein solches bereitzustellen und so eine zentrale Rolle in dem zukunftsträchtigen Mobilmarkt einzunehmen.

Android wird als quelloffenes System entwickelt und steht jedem - also auch den Geräteherstellern - praktisch kostenlos zur Verfügung. Dies ist ein Novum innerhalb der Mobilfunkbranche gewesen, in der bis dato fast ausschliesslich von den Herstellern selbstentwickelte oder teuer lizensierte Systeme eine nennenswerte Rolle gespielt haben.[2]

Die nicht anfallenden Lizenzkosten und geringen Entwicklungskosten für das Betriebssystem geben auch eine gute Erklärung dafür, dass Android nach nur gut 1,5 Jahren am Markt

[1] Vgl. z.B. "...Erfindung, die dazu beiträgt, die Kluft zwischen Industrie- und Schwellenländern zu schließenïn *Die mobile Revolution* http://www.welt.de/welt_print/article3306483/Die-mobile-Revolution.html

[2] Eine Ausnahme - zumindest in Bezug auf die Entwicklung - ist Symbian OS, das von Beginn an herstellerübergreifend entwickelt wurde (Nokia, Siemens und Sony Ericsson machten den Anfang), jedoch trotzdem mit Kompatibilitätsproblemen zu kämpfen hatte und durch verschiedene Plattformvarianten für einen zersplitterten Markt sorgte. Zuletzt wurde es als Open Source Software freigegeben, um die Entwicklung voranzutreiben und dem Konkurrenzdruck am Markt - nicht zuletzt durch Android erzeugt - zu begegnen. Vgl. [12]

bereits auf gut 20 Smartphones verschiedener Hersteller verfügbar ist. (Vgl. [13], Abschnitt *Telefone mit Android als Betriebssystem*)

1.1 Motivation

Durch diese recht große Aufmerksamkeit von Herstellerseite, die aktuell noch an Fahrt gewinnt, steigt natürlich auch die Relevanz der innerhalb von Android eingesetzten Sicherheitsmechanismen.

Die Geräte enthalten viele sensible Daten (Kontakte, Terminkalender, eMails, SMS, Browserhistory, evtl gespeicherte Passwörter, Notizen ...), die der Benutzer seinem persönlichen digitalem Begleiter anvertraut. In dieser Arbeit werden diese aus verschiedenen Blickwinkeln betrachtet und in wesentlichen Punkten vorgestellt.

1.2 Grundbegriffe

Da Android eine eigene Architektur mit zum Teil neuen Konzepten mitbringt, müssen auch einige Begrifflichkeiten geklärt werden, bevor auf das eigentliche Thema eingegangen werden kann. Im Detail wurden die Konzepte zum Beispiel in [9] beschrieben und sind in fast jeder Arbeit zu Android erneut dargestellt. Um Redundanz zu vermeiden, sollen hier nur die Bedeutungen der wichtigsten Begriffe aufgefrischt werden:

Intent Ein Nachrichtenobjekt, das die Adresse einer Zielkomponente (ähnlich einer URL) und Daten enthält. *Intents* sind der zentrale Bestandteil der Kommunikation von Komponenten innerhalb von Android.

Action Ein Vorgang innerhalb der Inter-Component-Communication (ICC).

Intent Filters Intents müssen nicht an eine bestimmt Komponente adressiert sein, sie können auch an den *Android Activity Manager* weitergegeben werden, der den Intent dann an eine Anwendung weitergibt, die ihn verarbeiten kann. Ob und welche externe Intents eine Anwendung verarbeiten kann, wird durch *Intent Filter* im System bekannt gemacht.

Activity Eine Bildschirmmaske, mit der der Benutzer interagieren kann, wird als *Activity* bezeichnet. Im Kern handelt es sich bei der Activity jedoch - mit den Vokabeln des MVC Pattern[3] - um den Controller zu einer Maske, der erst zusammen mit einer *View* für den Benutzer als Activity sichtbar wird.

[3] *Model-View-Controller* gesprochen, ein Software-Entwurfsmuster mit dem Benutzeroberfläche, Anwendungslogik und Daten voneinander getrennt werden.

Service Ein Service ist eine Komponente einer Anwendung, die im Hintergrund ausgeführt wird und nicht über eine eigene GUI verfügt. Services dienen sowohl dazu, Dienste für den Benutzer im Hintergrund auszuführen (Download einer Datei, Musik wiedergeben) als auch dazu, anderen Anwendungen Dienste anzubieten und diese auszuführen.

ContentProvider Ein Interface, das standardisierten Zugriff auf Daten, die durch eine Anwendung bereitgestellt werden, bietet.

2 Application Security

2.1 Überblick

Android ist ein Betriebssystem für mobile Endgeräte, das neben dem Linux-Kern zu einem großen Teil auf Java basiert. Dabei wurden zu den durch diese beiden Komponenten bereits mitgebrachten Sicherheitsmechanismen weitere hinzugefügt, die speziell auf mobile Anwendungen zugeschnitten sind.

So kombiniert Android Leistungsmerkmale des Betriebssystems Linux, wie z.B. effiziente Speicherverwaltung, preemptives Multitasking, Unix User Identifiers (UIDs) und Dateizugriffsrechte mit der Typsicherheit der Java Programmiersprache und derer bekannten Klassenbibliothek. Das Ergebnis ist ein Sicherheitsmodell, dass mehr an ein Mehrbenutzersystem als an (Sandkasten) Umgebungen, wie sie zum Beispiel auf der BlackBerry- oder J2ME-Plattform vorkommen, angelehnt ist.

Im Gegensatz zu einem Desktop-PC, auf dem praktisch alle Anwendungen unter demselben Benutzer (und dessen Rechten) ausgeführt werden, sind Anwendungen unter Android getrennt voneinander. Sie werden in getrennten Prozessen, unter jeweils getrennten UIDs mit individuellen Berechtigungen ausgeführt. In der Regel können Anwendungen weder lesend noch schreibend auf Daten anderer Anwendungen oder des Betriebssystems zugreifen. Das Nutzen von Daten durch verschiedene Anwendungen (z.B. im Fall eines Adressbuchs, dass Einträge anderen Anwendungen zur Verfügung stellen soll) muss explizit über spezielle Mechanismen gestattet bzw. ermöglicht werden.

Durch diese Isolation der verschiedenen Anwendungen werden die möglichen Konsequenzen von schad- oder fehlerhafter Software minimiert. Die eingesetzte Prozess- bzw. Komponentenisolation macht komplizierte Konfigurationen, wie sie auf anderen Sandbox-Systemen, wie zum Beispiel SELinux[4], nötig sind, überflüssig.

Zugriffsberechtigungen für Anwendungen unter Android bezeichnen die Erlaubnis, Aktionen wie *Bild mit der Kamera aufzeichnen*, *GPS benutzen* oder *Anruf starten* auszuführen. Während der Installation wird einer Anwendung eine eindeutige UID zugewiesen, unter der diese Anwendung auf dem jeweiligen Gerät dann immer ausgeführt wird. Da die Dateizugriffsrechte der Anwendung an der UID hängen, kann die Anwendung nur auf eigene Dateien zugreifen. So wird von den Entwicklern der Software verlangt, dass sie explizit festlegen müssen welche Daten wie mit anderen Anwendungen ausgetauscht werden können.

[4] SELinux steht für *Security Enhanced Linux* und ist eine Erweiterung des Linux-Kernels, die Zugriffskontrollen auf Ressourcen implementiert. Die Zugriffsrechte müssen dabei über sog. *Policies* vergeben werden, wodurch ein nicht unerheblicher Administrationsaufwand entsteht. Homepage des Projekts unter http://www.nsa.gov/research/selinux/ und http://fedoraproject.org/wiki/SELinux.

Für grundlegende Operationen, wie das Abspielen von Ton und Video oder das Ausführen anderer Anwendungen, benötigen Anwendungen keine speziellen Zugriffsrechte.

Fehlerhafte oder bösartige Software ist leider auf gängigen Plattformen aller Art Realität[5]. Android versucht, dieser Problematik zu begegnen und den möglichen Schaden durch solche Software, zu minimieren. Doch auch wenn die Daten geschützt sind, kann Schadsoftware die Benutzbarkeit eines Gerätes sehr einschränken[6]. Der Benutzer ist in einem solchen Fall gezwungen, die entsprechende Anwendung zu identifizieren und zu deinstallieren. Sollte eine Anwendungen, z.B durch intensive Nutzung der Rechenkapazitäten das Gerät praktisch unbenutzbar machen, bleibt dem Benutzer als letzte Möglichkeit die Option,im sogenannten *safe mode*[7] zu starten und dort die entsprechende Anwendung zu deinstallieren.[2]

Um die Möglichkeiten für Anwendungen, Schaden anzurichten, einzuschränken, muss der Benutzer bei der Installation von Anwendungen für kritische Aktionen / Zugriffe bestätigen, dass die Anwendung darauf zugreifen darf. Die sind zum Beispiel

- direkt Anrufe zu initiieren
- Zugriff auf sensitive Daten, wie z.B. dem Adressbuch
- Zugriff auf die Location API zur Positionsbestimmung

Entwickler, die Software für Android schreiben, müssen, wie in mobilen Umgebungen üblich, mit Einschränkungen bezüglich des verfügbaren Speichers, Rechenkapazitäten und der Stromzufuhr rechnen. Vor allem ist aber auch Sorge dafür zu tragen, dass die Daten des Benutzers geschützt sind. Benutzereingaben und die der eigenen Anwendung zugeteilten Berechtigungen müssen vor dem Missbrauch durch andere Programme geschützt werden.

Durch das zuvor vorgestellte Kernkonzept von Android, jeder Anwendung eine individuelle UID zuzuweisen, können auch Zugriffsberechtigungen auf andere Ressourcen und Funktionen eingeschränkt werden. Auf einem gewöhnlichen Desktop-PC hat jeder Benutzer eine UID, unter der dann alle Anwendungen des Benutzers, praktisch mit den Rechten des Benutzers, ausgeführt werden. Die Trennung unter Android ermöglicht es, dass jeder Anwendung explizit Berechtigungen zugewiesen werden können, die festlegen, was erlaubt ist und was nicht.

[5] Vgl. [8]

[6] z.B. durch Abspielen eines lauten, unangenehmen Tons, intensives Nutzen und somit "leer saugen"der Batterie, etc.

[7] Im *safe mode* wird keine Anwendung, die nicht zum Kernsystem gehört, gestartet. Wie man diesen aktiviert, ist je nach Gerät unterschiedlich. Auf dem G1 muss man z.B. die Menü-Taste beim Einschalten gedrückt halten.

Eine Ausnahme, diese Rechte zu erweitern ist, dass jede Anwendung über den zentralen ActivityManager eine Anfrage stellen kann, eine andere Anwendung (bzw. eine Activity) zu starten. Wird diese Anfrage ausgeführt, kann die gestartete Anwendung dann mit den Rechten der ursprünglichen Anwendung ausgeführt werden.

Die Einstiegspunkte, über die eine Anwendung gestartet werden kann, werden durch den Entwickler der jeweiligen Anwendung bestimmt und können mit Hilfe des Frameworks konsistent abgesichert werden. Anwendungen für Android haben nicht, wie bei klassischen Anwendungen üblich, eine zentrale Hauptfunktion, die bei jedem Start ausgeführt wird. Anstelle dieser Funktion treten Einstiegspunkte, die durch die Registriegung von *Activities*, *Services*, *BroadcastReceivers* oder *ContentProvider* im Hauptsystem definiert werden.

Android verlangt, das Anwendungen durch den Entwickler signiert sein müssen. In der Regel geschieht dies mit selbst signierten Zertifikaten, die jeder Entwickler ohne Erlaubnis oder Unterstützung von anderen erstellen kann. Der Grund für den Einsatz von Zertifikaten liegt also nicht darin, den Zugang zum Veröffentlichen und Anbieten von Anwendungen zu erschweren oder die Rechtevergabe zu zentralisieren.[8]

Vielmehr wird dem Entwickler so die Möglichkeit gegeben, Aktualisierungen für seine Software anbieten zu können, ohne auf zusätzliche Updatemechanismen über eigene Schnittstellen zurückgreifen zu müssen. Anwendungen, die mit dem gleichen Zertifikat unterschrieben sind können darüber hinaus auf Anfrage unter der selben UID ausgeführt werden. Dies erlaubt es dem Entwickler, seine Software möglichst unkompliziert zu aktualisieren und bei Bedarf auch Daten aus der alten Installation in die Neue zu übernehmen.

Ein weiterer Punkt für die Zertifikate ist die Reputation, die sich ein Entwickler mit Hilfe seiner Arbeit und dem Zertifikat erarbeiten kann. Hat er sich mit einer Anwendung einen Namen gemacht, hilft dieser Bekanntheitsgrad, unterstrichen und beglaubigt durch die Signatur mit seinem Zertifikat, seinen weiteren Anwendungen. Dieser Ansatz erscheint deutlich glaubwürdiger, als der Ansatz anderer Plattformen (wie zum Beispiel dem bereits erwähntem Symbian OS), bei dem man sich Zertifikate kaufen muss und allein durch einen solchen Kauf vertrauenswürdig wird.

2.2 Android Manifest Permissions

Anwendungen müssen, wie bereits erwähnt, mit den nötigen Berechtigungen ausgestattet werden, um bestimmte Aktionen wie z.B. ermitteln der aktuellen Position, Photo mit

[8] Dies war zum Beispiel beim Erstellen von Anwendungen für Symbian OS lange Zeit ein großes Manko - die Vergabestellen für solche Zertifikate für Symbian OS waren so stark überlastet, dass es für Entwickler kaum oder nur unter erheblichem finanziellen Aufwand möglich war, ein benötigtes Zertifikat zu erhalten.

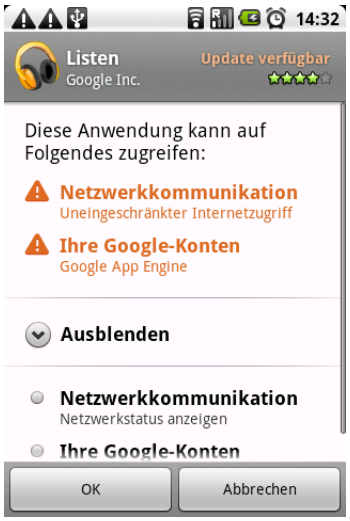


Abbildung 1: Bestätigung von Berechtigungen während der Installation.

der Kamera aufnehmen, etc., ausführen zu dürfen. Android nutzt *manifest permissions* um festzulegen, was der Anwender der Anwendung erlaubt hat, zu tun. Die Rechte, die eine Anwendung benötigt, werden durch den Entwickler in der `AndroidManifest.xml` Datei festgelegt und dem Benutzer bei der Installation zur Bestätigung vorgelegt bzw. angezeigt, wie zum Beispiel im Screenshot in Abblildung 1 zu sehen ist.[9]

So hat der Benutzer beim Installieren neuer Software die Möglichkeit, sich zu entscheiden, ob dem Entwickler auf Grund seiner Reputation oder auf Grund von Reviews anderer Nutzer[10] vertraut und die angeforderte Erlaubnis für die angezeigten Aktionen erteilt werden soll.

Die Beschreibung der Aktionen bzw. benötigten Berechtigungen wird dabei in der Regel so formuliert, dass der Benutzer sich für seine Entscheidung auf die damit erreichbaren Ziele konzentrieren kann und nicht durch Fachbegriffe aus dem Security Jargon abgeschreckt wird. Zum Beispiel braucht eine Anwendung, um auf das Adressbuch zuzugreifen die Berechtigung READ_CONTACTS - was für eine Kontaktverwaltung oder ein E-Mailprogramm plausibel ist, für eine Variante des Spiels Tetris jedoch kaum. Durch dieses einfache Modell ist es möglich, die gesamte Kommunikation zwischen verschiedenen Anwendungen (unter Android als Inter-Component-Communication (ICC) bezeichnet)[11] zu

[9] Dabei ist es kein Unterschied, ob die Anwendung lokal von der SD Karte oder über den Android Market installiert wird. - die angezeigte Warnung ist gleich.

[10] Im Market gibt es eine Kommentar- und Bewertungsfunktion, in der man installierte Anwendungen bewerten und kommentieren kann. Dieses System hilft dabei, sich bei der Suche nach Anwendungen zwischen mehreren Möglichkeiten zu entscheiden und spart Zeit, wenn eine Anwendung bereits von vielen Benutzern als instabil und nicht-funktionstüchtig bewertet bzw. kommentiert wurde.

[11] Da es in der Regel nicht möglich ist, über das Dateisystem auf die Daten anderer Anwendungen zuzugreifen, muss sämtliche Interaktion zwischen Anwendungen über ICC laufen.

kontrollieren und abzusichern. Das Starten von *Activities*, Verwenden von *Services*, der Zugriff auf *ContentProvider*, Senden und Empfangen von *Intents* und das Aufrufen von *Binder* Schnittstellen kann insgesamt die gleiche Berechtigung benötigen. Daher braucht der Anwender bei der Installation auch nicht mehr zu verstehen als *die neue Applikation greift auf mein Adressbuch zu* und muss nicht mit den oben aufgezählten Fachbegriffen belästigt werden.

Nachdem eine Anwendung installiert ist, können die ihr zugeteilten Berechtigungen jedoch nicht mehr verändert werden. Dieser Punkt erhöht die Verantwortung für den Benutzer und das Risiko, wenn eine Anwendung „nur mal kurz“ ausprobiert werden soll und der Bestätigungsdialog für die Berechtigungen schnell bestätigt („weg-geklickt“) wird.

Aus Perspektive des Entwicklers sind Berechtigungen zunächst nur Strings, die mit der Anwendung und der ihr zugeteilten UID verknüpft werden. Über die Methode `checkPermisson(String permisson, int pid, int uid)` der `Context` Klasse kann man prüfen, ob ein Prozess und die zugehörige UID über eine bestimmte Berechtigung, wie zum Beispiel READ_CONTACTS[12], verfügt.[13]
Das folgende Beispiel zeigt eine typische *Permisson*-Definition:

```
<permission
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:name="de.example.android.ACCESS_FRIENDS_LIST"
  android:descrption="@string/access_perm_desc"
  android:protectionLevel="normal"
  android:label="@string/access_perm_label">
</permission>
```

In der Permission-Definition werden die wesentlichen Schlüsselattribute gezeigt. Die beiden Textattribute `label` und `description` verweisen in diesem Fall auf ausgelagerte Ressourcen, um die Lokalisierung der Anwendung zu vereinfachen. Jede Permission braucht einen global eindeutigen Namen, der auch in der zuvor erwähnten `checkPermission`-Methode verwendet wird. Das vierte wichtige Attribut ist `protectionLevel`, welches die folgenden vier Werte annehmen kann:[14]

normal Permission für Funktionen, die einen geringen Effekt haben, wie z.B. `VIBRATE` zum Einschlaten des Vibrationsalarms. Wird verwendet für Rechte, die generell für den Anwender nicht von gesteigerter Bedeutung sind. Die Berechtigungen werden dem Benutzer angezeigt, es erfolgt jedoch keine explizite Warnung.

[12] Korrekterweise sollte hier der voll qualifizierte Name der Berechtigung, android.permission.READ_CONTACTS verwendet werden. Jede Anwendung kann zu den durch Android bereits definierten Berechtigungen weitere hinzufügen.
[13] Dies ist ein exemplarisches Beispiel, wie Zugriffsrechte durch die Laufzeitumgebung preigegeben werden, es existieren noch einige andere Wege.
[14] Vgl. [1], Android Platform Documentation, Permission ProtectionLevel

Ein Beispiel ist in Abbildung 1 zu sehen: Im unteren Abschnitt wird die Berechtigung „Netzwerkkommunikation: Netzwerkstatus anzeigen“als normaler Aufzählungspunkt angezeigt. Die Liste mit diesen Hinweisen kann auch ausgeblendet werden.

dangerous Permissions wie WRITE_SETTINGS oder SEND_SMS werden als „gefährlich“ eingestuft, da sie Einstellungen am Gerät verändern oder Kosten verursachen können. Dieses protectionLevel sollte für Permissions genutzt werden, das für den Benutzer von Interesse sein könnte. Android warnt den Benutzer bei der Installation der Anwendung bezüglich dieser Berechtigungen.

Als Beispiel dient auch hier Abbildung 1: Die beiden oberen Berechtigungen werden mit einem Ausrufezeichen-Symbol hervorgehoben und so den Anwender als kritische Berechtigungen präsentiert.

signature Diese Berechtigungen können nur an Anwendungen vergeben werde, die mit dem gleichen Zertifikat wie das Ziel der Permission unterschrieben sind. Dies ermöglicht eine sichere Verbindung von verschiedenen Anwendungen des gleichen Herausgebers ohne die Spezifikation eines öffentlichen Interfaces.

signatureOrSystem Ähnlich wie „signature“, mit dem Unterschied, dass im System enthaltene Programme ebenfalls Zugriff erhalten können. Dies ermöglicht Anwendungen auf angepassten Androidsystemen[15] diese Permission zu erhalten. Dieses protectionLevel ist für die Integration des Systems (dem Erstellen angepasster Firmware Versionen, etc.) interessant und in der Regel nicht für reine Applikationsentwickler.

Android Manifest Permissions erstellen

Entwickler können für ihre Anwendungen eigene Permissions definieren, wenn sie anderen Anwendungen gestatten wollen, auf ihre Anwendung zuzugreifen. Über *Manifest Permissions* erhält der Benutzer die Möglichkeit, zu entscheiden, welche Anwendungen Zugriff auf die persönlichen Daten innerhalb der Anwendung bekommen, als diese Entscheidung den Entwicklern zu überlassen.

Zum Beispiel kann die Anwendung zur Planung der nächsten Party eine Permission mit dem Namen „de.beispiel.partyplanner.ACCESS_GUEST_LIST“ definieren. Hält die Anwendung nun ein Objekt *GuestList* vor, wäre diese Berechtigung nötig, um darauf zuzugreifen. Sie wäre für aufrufende Anwendung nötig, um die Gästeliste einzusehen oder zu aktualisieren. Nur Anwendungen, die die Nutzung dieser Permission deklarieren (und dies bei der Installation bestätigt worden ist), können den Zugriff ausführen. Definiert man solche

[15] Sog. *Custom Builds*, die durch Mitglieder der Opensource Community erstellt werden, oder auch durch Provider angepasste und auf Geräten vorinstallierte Versionen von Android.

Permissions sollte man also auf der einen Seite sicherstellen, dass man tatsächlich Daten hat, die man anderen Anwendungen zur Verfügung stellen möchte (und nicht nur dem Benutzer) und auf der anderen Seite durch konsistente und logische Benennung Sorge dafür tragen, das der Zweck der *Permission* für den Benutzer erkennbar ist, wenn er bei der Installation damit konfrontiert wird.

Darüber hinaus sollte man das Hinzufügen neuer *Permissions* soweit es geht vermeiden. So ist das Definieren eigener *Permissions* nur dann wirklich notwendig, wenn Anwendungen ohne eine neuerliche Nachfrage an den Benutzer ineinander greifen sollen. Eine Alternative zur Nutzung von Permissions ist zum Beispiel der Aufruf einer Activity zusammen mit einem passenden Intent um die Gästeliste unserer Beispielanwendung zu erweitern. Der Benutzer kann dann durch die Activity gefragt werden, ob die aufgerufene Aktion (hinzufügen eines Gastes) mit den durch den *Intent* repräsentierten Parametern (den Kontaktdaten des Gastes) gestattet werden soll. Auf diese Weise bleibt das System transparent für den Benutzer (da keine permanent gültige Erlaubnis vergeben wird) und außerdem wird der Entwicklungsaufwand verringert.

2.3 Security Policy

Anwendungen enthalten oft Komponenten, auf die externe Anwendungen niemals direkten Zugriff erhalten können sollen- zum Beispiel eine Komponente, die sich um die interne Passwortverwaltung kümmert. Die Lösung, die Andorid hier bietet, ist die Möglichkeit eine Komponente als *private* zu definieren.

Die verringert die potentiellen Angriffspunkte, die eine Anwendung nach außen bietet. Auf der anderen Seite ergibt sich jedoch auch das Problem, das Komponenten bei unzureichender Spezifikation durch den Entwickler implizit als öffentlich (*public*) definiert werden. Da der Entwickler sich dessen unter Umständen nicht bewusst ist, können so Sicherheitslücken in Anwendungen entstehen.

3 System Security

3.1 Allgemein

Im folgenden Abschnitt werden die Sicherheitsmerkmale auf Systemebene, die nicht bereits im vorherigen Abschnitt besprochen wurden (wie zum Beispiel das Anlegen eines eigenen Benutzers für jede Anwendung), betrachtet.

3.2 Partitionen

Android verwaltet im laufenden Betrieb eine Reihe spezieller Partitionen, um bestimmte Daten je nach Klassifizierung voneinander zu trennen und über die auf Dateisystemebene vergebenen Zugriffsberechtigungen weiter zu schützen.

Im Einzelnen gestaltet sich dies wie folgt:

/data ist als einzige System-Partition schreibbar. Hier werden u.a. Anwendungsdaten mit den Berechtigungen des der jeweiligen Anwendung zugeordneten Users abgelegt.

/ und **/system** sind jeweils read-only . Hier befinden sich die meisten Systembibliotheken, sowie die `permissions.xml` Datei, in der mitgeführt wird, welche Berechtigungen an Anwendungen vergeben worden sind.

/sdcard ist der Mountpoint für die SD-Karte und wird mit der Option *no-exec* eingehangen. So können von hieraus keine Programme gestartet werden und kein Schadcode in das System eingeschleust werden.

Auf der SD Karte kommt aus Gründen der Kompatibilität[16] außerdem das FAT32-Dateisystem[17] zum Einsatz. Der wesentliche Nachteil ist, dass auf FAT32 keine Berechtigungen auf Dateisystemebene umgesetzt werden können. Deshalb muss man berücksichten, das Daten, die man auf der SD Karte speichert, durch jede andere Anwendung gelesen und verändert werden können.

3.3 Systemupdates

Systemupdates ermöglicht Android durch die Verwendung von Mount-Overlays. Dies ermöglicht es, die oben beschrieben Mechanismen zum Schutz der Systemdaten durch die Verwendung verschiedener Partitionen, temporär aufzuheben und Updates einzuspielen.

[16] Durch den Einsatz von FAT32 wird ermöglicht, dass die SD Karte auch in einem Kartenlesegerät unter Windows und praktisch allen anderen Betriebssystemen gelesen werden kann.

[17] Vgl. http://de.wikipedia.org/wiki/FAT32#FAT32

Update-Dateien werden zumeist direkt über die Verbindung des Gerätes zum Internet verteilt. Dazu wird ein zur Verfügung gestelltes Update automatisch auf das Gerät heruntergeladen und nach Bestätigung durch den Benutzer eingespielt. Zumeist ist ein Neustart des Gerätes notwendig. Alternativ kann ein Update auch manuell auf die SD Karte geladen und von dort aus angewendet werden.

Um zu verhindern, dass das System durch Einspielen eines veränderten Updates kompromittiert werden kann, werden Updates vor ihrer Installation auf eine gültige Signatur geprüft.[18]

[18] Auf einigen HTC Geräten, insbesondere auch auf dem G1, lässt sich diese Prüfung durch Verwendung einer sogenannten „Goldcard“ umgehen. Hierbei handelt es sich um eine mit einem bestimmten Dateisystemimage bespielten SD Karte, die die Signaturprüfung außer Kraft setzt. Hierdurch ist es möglich, modifizierte Firmware / ROM-Images auf den Geräten zu installieren. Vgl. http://revskills.de/pages/goldcard.html

4 Enterprise Security

4.1 Allgemeines

Für den Einsatz von Smartphones im professionellen Umfeld wie großen Organisationen und Unternehmen ergeben sich spezielle Anforderungen. Werden die Geräte dienstlich genutzt, finden sich sehr schnell unternehmenskritische Daten auf ihnen wieder, die vor unbefugtem Zugriff geschützt werden müssen. Dabei spielt der Spagat zwischen der Benutzbarkeit und sicherheitstechnischer Effizienz eine nicht unerhebliche Rolle.

Die folgenden Abschnitte zeigen, welche Mechanismen und Möglichkeiten für Android derzeit in Hinblick auf gängige Anforderungen in diesem Bereich vorhanden sind.

4.2 VPN

Für den Einsatz mobiler Endgeräte in Verbindung mit der Infrastruktur eines Unternehmens sind verschlüsselte Netzwerkverbindungen mit sogenannten *virtual private networks* (VPN) derzeit unverzichtbar.

Nachdem bei Erscheinen von Android im Oktober 2008 keine Unterstützung für VPN vorhanden war, erschienen im Laufe des vergangenen Jahres gepatchte und von der Community erweiterte Images in die verschiedene VPN Technologien eingebaut wurden. Diese konnten natürlich nur auf Entwicklergeräten installiert werden und waren dementsprechend nicht für den Einsatz durch Normalverbraucher oder im betrieblichen Umfeld geeignet.

Mit der im Herbst 2009 erschienenden Version 1.6 von Android findet die VPN Unterstützung ihren Weg in den offziellen Entwicklungszweig. Möglich sind dabei Verbindungen mit den folgenden Technologien[19]:

- PPTP
- L2TP
- L2TP IPSec PreShared Key (PSK)
- L2TP/IPSec certifikate based (CRT)

[19] Vgl. Android 1.6 Platform Highlights, http://developer.android.com/sdk/android-1.6-highlights.html

So können zur Authentifizierung sowohl vorab vereinbarte Schlüssel als auch Zertifikate benutzt werden und Android wird auch für den Einsatz im professionellen Umfeld interessanter. Android 1.6 wird dabei nicht nur für neue Geräte verfügbar sein, sondern voraussichtlich auch per Update für Nutzer der bisherigen Geräte, wie zum Beispiel dem T-Mobile G1, zur Verfügung stehen.

4.3 Verschlüsselung

Betrachtet man die Möglichkeit der Dateiverschlüsselung, ergeben sich für mobile Geräte ganz eigene Anforderungen. Denn im Gegensatz zu PCs oder Laptops, die beim Transport oder wenn man nicht mit ihnen arbeitet, zumeist heruntergefahren werden, sind mobile Geräte meistens im Dauerbetrieb.

Eine Lösung könnten hier Datencontainer sein, die je nach Bedarf ver- und entschlüsselt werden. Hindernisse sind hier jedoch die Zugriffszeiten und sowie begrenzten Rechenkapazitäten. Ein weiterer problematischer Punkt sind die von den meisten Verschlüsselungsalgorithmen benötigten Zufallszahlen-Generatoren, da diese auf diesen Geräten ebenfalls nur sehr eingeschränkt zur Verfügung stehen. Bekannte Ansätze wie DM-Crypt[20] und Loop-AES [21] werden durch das standardmäßige, auf die in den Geräten verwendeten Flashspeicher optimierte, YAFFS2[22] erschwert.

Außer dem genannten DM-Crypt ist zudem derzeit keine Verschlüsselungskomponente im Standard-Linuxkernel enthalten und Komponenten, die im „Userland“ betrieben würden, bringen zumeist vielseitige Abhängigkeiten mit sich. (Vgl. [6])

Es sieht also unterm Strich derzeit nicht gut aus für den Einsatz von Verschlüsselungsmechanismen unter Android. Zwar gibt es einige Ansätze, in Spezialfällen mit reinen Androidapplikationen zu arbeiten (Z. B. „Secrets for Android“ [23], das Sicherheit für textbasierte Geheimnisse bietet, jedoch nicht mit Binärdaten umgehen kann.), jedoch stoßen diese schnell an (performance-) Grenzen, wie das Beispiel des Astro Filemanager[24], der zeitweise über eine Option zur Verschlüsselung von Dateien verfügte, diese wurde jedoch wieder aus der Featureliste gestrichen.

Welche Möglichkeiten es aus dem Bereich der Forensik gibt, (als Angreifer) Zugriff auf die Daten eines Telefons zu erhalten, zeigt unter anderem das Paper *Android Forensics* unter

[20] Vgl. http://de.wikipedia.org/wiki/Dm-crypt bzw. http://www.saout.de/misc/dm-crypt/
[21] Vgl. http://sourceforge.net/projects/loop-aes/
[22] *Yet Another Flash Filesystem 2*, ein Dateisystem speziell für Flashspeicher, das deren besonderen Eigenschaften (insb. die physikalischen, wie z.B. Abnutzung der Speicherzellen) berücksichtigt.
[23] „Secrets for Android“ Homepage: http://code.google.com/p/secrets-for-android/
[24] Vgl. Arto Filemanager http://androidfeeder.com/808/astro-file-manager/

[11] und soll hier nicht weiter behandelt werden, um den Umfang dieser Arbeit nicht zu sprengen.

4.4 Smartcards

Die Verwendung von Smartcards, um den Zugriff und die Verwendung der mobilen Endgeräte zu ermöglichen, scheint nahezuliegen und würde einen zusätzlichen Vertrauensanker ins Spiel bringen. Verfolgt man diese Idee weiter, ergeben sich jedoch zwei wesentliche Probleme - zum einen fehlt nicht nur die nötige Softwareunterstützung in Android bisher völlig, sondern auch die aus der Sun Java VM bekannte PKCS11 API. Des Weiteren würde das Einbinden der nötigen Hardware native Komponenten im Betriebssystem benötigen.

Zum anderen ist in den bisherigen Geräten auch keine Hardware, sprich kein passender Kartenleser, vorhanden. Alternativen könnten hier Geräte mit USB On-the-go[25] Fähigkeiten oder die Bluetooth Schnittstelle bieten.

Eine weitere Möglichkeit könnten Produkte wie die *certgate SmartCard microSD*[26] sein, die über einen eigenen Zufallszahlengenerator, eine eigene API verfügt und über den microSD Slot des Endgeräts ohne weitere Hardware genutzt werden könnte. [6]

4.5 Zugriffsschutz

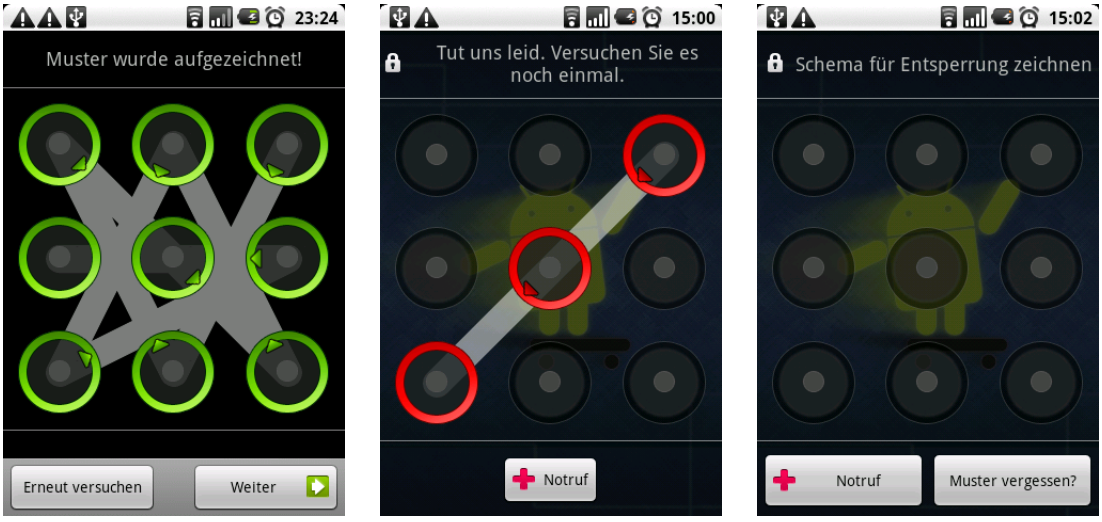
Um den Zugriff auf das Gerät, während es kurz unbeaufsichtigt ist, zum Beispiel im Büro, sicherzustellen, bietet Android die Möglichkeit, ein Entsperrungsmuster festzulegen.

Geht das Gerät in den Standbymodus bzw. wird die Tastensperre aktiv, muss der Benutzer zum Entsperren anstatt einer standardmäßigen Tastenkombination (z.B. Stern- und Menütaste bei Nokia) ein zuvor festgelegtes Muster auf ein Eingabefeld aus 3x3 Kontaktpunkten zeichnen. Das durch den Benutzer gewählte Muster muss dabei mindestens 4 Punkte miteinander verbinden und kann bei Bedarf auch alle Punkte beinhalten. Dabei können nicht nur benachbarte Punkte, sondern auch entfernte Punkte verwendet werden. Ein Beispiel zeigt Abbildung 2 (a), der Ausgangspunkt ist oben rechts, der Endpunkt links in der Mitte.

[25] Durch USB On-the-go (OTG) können entsprechend ausgerüstete Geräte miteinander kommunizieren, indem eines der beiden eine eingeschränkte Host-Funktionalität übernimmt. Das G1, das erste Gerät, das mit Android ausgeliefert wurde und bisher auch einzige, das als Entwicklergerät erhältlich ist, verfügt zum Beispiel über eine USB Schnittstelle, die sowohl im Client- als auch im Host-Modus agieren kann.

[26] Vgl. Produkthomepage unter http://www.certgate.com/index.php?id=71.

Durch diese große Flexibilität bei der Definition des Musters scheint der Schutz vor Brute-Force-Angriffen ausreichend hoch, zumal die Anzahl der Versuche, wie nachfolgend beschrieben, beschränkt ist.



(a) Muster festlegen (b) Entsperren fehlgeschlagen (c) Muster vergessen?

Abbildung 2: Screenshots „Entsperrmuster“

Ein Beispiel für einen einfachen Fehlversuch zeigt Abbildung 2 (b). Nach erfolgreicher Eingabe wird der Zugriff auf die normale Benutzeroberfläche bzw. die gerade geöffnete Anwendung freigegeben.

Nach einem Fehlversuch erhält der Benutzer zusätzlich zu dem immer verfügbaren Notruf-Schalter die Möglichkeit, mit der Schaltfläche *Muster vergessen* (2 (c)) einen Alternativweg zum Entsperren des Gerätes zu aktivieren.

In der auf dem T-Mobile G1 ausgelieferten Version von Android wird dazu auf das Googlekonto zurückgegriffen, das auf dem Gerät konfiguriert ist. Zum Entsperren muss der Benutzer anstatt des Musters das Kennwort dieses Kontos angeben.

Werden hingegen mehr als fünf Fehleingaben in Folge getätigt, werden weitere Versuche nur nach einer Verzögerung von 30 Sekunden zugelassen. Diese Verzögerung gilt für jeden weiteren Versuch in Folge, auch ein zwischenzeitliches Anwählen des Notrufmodus setzt den Timer nicht vorzeitig zurück.

So verfügt Android über einen wirksamen Schutz gegen spontane Versuche, bei kurzer Abwesenheit des Eigentümers auf die im Gerät gespeicherten Daten zuzugreifen.

Der Ansatz, graphische Kennwörter anstatt von klassischen Kennwörtern aus Ziffern und Zeichen zu verwenden, findet immer mehr Anklang. Dies zeigt auch das Projekt *Fraktale*

Kennwörter[27] oder das Projekt "Background Draw a Secret"(BDAS) [28]. Grafische Kennwörter und Muster gelten als einfacher zu merken und werden so durch den Benutzer leichter aktzeptiert, wodurch auch die Bereitschaft, komplexere Kennwörter zu wählen, steigen soll.

Somit liefert Android einen Zugriffsschutz mit, den man auf anderen Plattformen erst durch zusätzliche Software oder nur mit herkömmlichen Mechanismen (bei BlackBerry zum Beispiel die klassische PIN) erreichen kann.

[27] Durchgeführt im Sommersemester 2009 an der Hochschule der Medien Stuttgart, vgl. http://www.hdm-stuttgart.de/multimedial/medianight/archiv_ss_09/medianightprojekt20090331103805/projektview

[28] Vgl. http://www.golem.de/0807/60770.html

5 Zusammenfassung und Ausblick

Diese Ausarbeitung zeigt, dass Android über ein ausgereiftes und konsequent umgesetztes Securitymodell verfügt.

Dies scheint sich neben der reinen Betrachtung der vorgestellten Mechanismen und Funktionen auch daran zu manifestieren, dass die Befürchtungen von F-Secure[29] bisher nicht eingetreten sind. F-Secure hatte erwartet, dass Anwendungen, die nicht von einer zentralen Stelle reviewed werden und direkten Zugriff auf weite Teile der API haben, zu großen Sicherheitslücken führen.

Die Abfrage bzw. Bestätigung der Zugriffsrechte in Kombination mit der Reputation durch Userkommentare und Verbreitung von Anwendungen durch Empfehlungen und Berichterstattung scheint bis dato in Kombination einen ausreichenden Schutz vor Schadsoftware zu bieten.

Darüber hinaus sind bisher auch kaum Angriffe und Sicherheitslücken bekannt geworden, die konzeptionelle Schwächen vermuten lassen. Für Aufsehen und Unruhe sorgte zwar der „Browserbug“[30], der jedoch schnell behoben werden konnte und bisher keine weiteren Sicherheitslücken nach sich gezogen hat. Zumal hier zwar die Kontrolle über den Browser durch fremde „übernommen“ werden konnte, weitere Teile des Systems jedoch nicht betroffen waren. Dies ist ein deutlicher Unterschied zu anderen Plattformen, wie zum Beispiel dem iPhone OS. Gerade im Jahr 2009 waren dort eine ganze Reihe von Sicherheitlücken aufgedeckt wurden, die zum Teil auch für Angriffe ausgenutzt wurden.[31]

Dies gilt für den aktuellen Stand der Android-Entwicklung. Da jedoch immer mehr Hersteller mit immer unterschiedlicheren Geräten aufwarten, scheint sich eine Zersplitterung, zumindest in Feinheiten, bereits abzuzeichnen. Jeder Hersteller darf - und wird, um seine Produkte hervorzuheben, das auf seinen Geräten verwendete Android anpassen. Wichtig wird sein, dass die Kompatibilität zumindest soweit gewahrt bleibt, dass die meisten Anwendungen funktionieren.

Sicherheitstechnisch erschwert diese Variantenvielfalt die Betrachtung natürlich, nicht zuletzt, weil Erweiterungen nicht unbedingt als Quelltext veröffentlicht werden müssen.

Es ist also damit zu rechnen, dass man in Zukunft Android als gemeinsamen Nenner betrachten kann, auf den Geräten jedoch Varianten zum Einsatz kommen. Ein Vergleich zu Linux mit seiner Distributionsvielfalt liegt hier nahe, mit dem Unterschied, das die Software der Mobiltelefone an die Hardware gebunden ist.

[29] Vgl. http://searchmobilecomputing.techtarget.com/news/article/0,289142,sid40_gci1283171,00.html
[30] Vgl. http://www.heise.de/security/meldung/Bug-im-Android-Browser-195993.html
[31] Vgl. z.B. http://www.heise.de/security/meldung/iPhone-OS-3-0-schliesst-46-Sicherheitsluecken-183353.html

Das Android wird es wohl im Wesentlichen nur noch im zentralen Coderepository geben. Solange die Sicherheitsmerkmale dieser Version aber weiterhin so ausgereift bleiben, oder im Optimalfall noch weiter verbessert werden (z.B. das Handling von Permissions aus Entwicklersicht oder die Möglichkeit, einer installierten Anwendung auch nur einen Teil der geforderten Rechte zu geben), ist das schon eine sehr gute Ausgangsbasis für sichere Endgeräte.

Der Blick im Abschnitt *Enterprise Security* auf die Features, die für den praktischen Einsatz im Unternehmen interessant sein könnten, hat auch gezeigt, das Android hier auf einem guten Weg ist, die notwendigen Funktionalitäten bereitzustellen. Eine Ausbreitung auf diesen Nutzerkreis ist also ebenfalls wahrscheinlich.

Literatur

[1] Google Inc., "Android Online Documentation", Google Inc., http://code.google.com/android, zuletzt abgerufen am 10.12.2009

[2] Jesse Burns, "Developing secure mobile applications for Android", iSec Partners, 2008, https://website.isecpartners.com/files/iSEC_Securing_Android_Apps.pdf

[3] Ed Burnette, "Hello, Android - Introducing Google's Mobile Platform", Raleight, Norh Carolina 2008, ISBN 978-1-93435617-3

[4] Chris Haseman, "Android Essentials", APress 2008, ISBN 978-1-4302-1064-1

[5] Reto Meier, "Professional Android Application Development", Wiley Publishing 2009, ISBN 978-0-470-34471-2

[6] Christian Küster, "Android Security", Tarent GmbH, Vortrag auf LinuxDay 2009 und FrOsCon 2009, http://blogs.fsfe.org/christian_kuester/files/2009/07/android-security-linuxtag-tarent.pdf

[7] William Enck, Patrick McDaniel, "Understanding Android's Security Framework", Systems and Internet Infrastructure Security, Computer Science and Engineering Department at Penn State University, October 2008, http://siis.cse.psu.edu/slides/android-sec-tutorial.pdf

[8] Prof. Dr. Roland Schmitz, "Mobile Malware Evolution and the Android Security Model", Hochschule der Medien Stuttgart, Vortrag auf der droidCon 2009, Berlin

[9] Markus Schlichting, "Die Android Plattform: Aktueller Stand und Ausblick", Hochschule der Medien Stuttgart, August 2008, http://www.mynethome.de/content/Android_Plattform-Stand_und_Ausblick-082008-MSchlichting.pdf

[10] Dominik Gruntz, "Android Application Model", Fachhochschule Nordwestschweiz, Dezember 2008, http://www.fhnw.ch/technik/imvs/publikationen/de/publikationen/vortraege/080312_gruntz_android_programmiermodell.pdf

[11] Andrew Hoog, “Android Forensics”, Mobile Forensics World 2009, 29.05.2009, http://www.viaforensics.com/android

[12] Wikipedia, Symbian OS, http://de.wikipedia.org/wiki/Symbian_OS, zuletzt abgerufen am 15.11.2009

[13] Wikipedia, Android (Betriebssystem), http://de.wikipedia.org/wiki/Symbian_OS, zuletzt abgerufen am 05.01.2010