Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3446

Integration der PMP in das Android OS

Philipp Scholz

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr.-Ing.. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Christoph Stach
begonnen am:	25.Oktober 2012
beendet am:	26. April 2013
CR-Klassifikation:	K.4.1, D.4.6

Kurzfassung

Die Privacy Management Platform stellt ein neues Berechtigungskonzept für Android vor, das es ermöglicht persönliche Daten zu verschleiern oder Berechtigungen einer App zur Laufzeit zu ändern. Jedoch ist die PMP als eigenständige App nicht in der Lage Zugriffe für Apps zu steuern, die Daten nicht bei der PMP anfragen, sondern direkt beim Betriebssystem. In dieser Arbeit wird ein Ansatz vorgestellt um die PMP in das Betriebssystem zu integrieren. Dazu werden verschiedene Integrationsstrategien verglichen und die Implementierung derjenigen vorgestellt, die als Sicherste angesehen wird. Eine Integration zusammen mit der Unterbindung von direkten Zugriffen auf Telefondaten garantiert, dass Berechtigungsprüfungen ausschließlich durch die PMP erfolgen. Zusätzlich wurden zwei Apps implementiert, die die Funktionsweise der integrierten PMP und dem modifizierten Betriebssystem demonstrieren.

Abstract

The Privacy Management Platform introduces a new permission model for Android to provide enhanced security and privacy. This includes cloaking personal data and changing access permissions at run-time. However, the PMP as a standalone application is not capable of enforcing access policies for apps that do not initiate requests to the PMP but to the operating system itself. This paper presents an approach in order to integrate the PMP into the Android OS by comparing various integration strategies and describing the implementation of the one that is deemed to be the most secure. This integration along with the inhibition of direct access to the phone's data ensures that permission checks can only be performed by the PMP. Additionally, two apps have been created to demonstrate the usability of both the integrated PMP and the modified operating system.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

Der Funktionsumfang von Mobiltelefonen ist mittlerweile mehr als das, was der Name zunächst vermuten lässt. Bereits seit den späten 90er Jahren können mobile Geräte Aufgaben übernehmen, die zuvor nur von einem PC aus möglich waren. Dazu zählen Funktionen wie Kalender, Adressbuch oder Internetzugang. Daher entstand für solche mobilen Endgeräte der Begriff Smartphone. Mit der Einführung von Android im Oktober 2008 stand für Smartphones ein freies, quelloffenes Betriebssystem zur Verfügung. Zeitgleich wurde der Google Market ins Leben gerufen, um zahlreiche Zusatzanwendungen bereitzustellen. Die Erweiterung eines Smartphones um neue Funktionen ist seitdem nicht mehr nur bei Technikbegeisterten, sondern bei allen Nutzern sehr beliebt. Anfang des Jahres 2013 konnte Android bereits einen Marktanteil von 67 Prozent [zdn13] verzeichnen. Die Zahl der installierbaren Anwendungen (*Apps*) im Google Play Store (ursprünglich: Google Market) stieg zu dieser Zeit laut [app13] auf über 600.000.

Die einfache Installation von Apps und der steigende Marktanteil bieten jedoch auch eine Grundlage für schädliche Software. Aus dem *Mobile Threat Report* der Firma *F-Secure* für das 4. Quartal 2012 [FS13] geht hervor, dass sich die Zahl der schädlichen Anwendungen jedes Jahr fast verdoppelt. Die meisten schädlichen Apps sind als Hilfsprogramme oder Unterhaltungssoftware getarnt. Sie lesen persönliche Daten aus, die über die Internetverbindung des Smartphones versendet und auf Servern gesammelt werden. Auf modernen Smartphones können viele persönliche Daten gespeichert sein. Die Kontaktliste gibt Auskunft über den Freundeskreis, die Liste der Telefonate zeigt sogar die Personen zu denen eine engere Verbindung besteht. Das GPS-Modul gibt den aktuellen Aufenthaltsort an, besuchte Internetseiten offenbaren persönliche Interessen. Gespeicherte Kurznachrichten enthalten private Informationen, die nur für den Adressaten bestimmt sind.

Android besitzt ein Berechtigungssystem, um unerwünschte Zugriffe auf das Gerät zu verhindern. Jede Funktion, die persönliche Daten ausliest oder kostenpflichtige Dienste nutzen kann, ist mit einer Berechtigung versehen. Eine App, die auf solche Funktionen zugreift, muss für jede Funktion die entsprechende Berechtigung anmelden. Die angemeldeten Berechtigungen werden bei der Installation der App angezeigt. Stimmt der Benutzer diesen Berechtigungen nicht zu, wird die Installation abgebrochen. Berechtigungen werden ausschließlich während der Installation vergeben und verbleiben bis zur Deinstallation der App.

Wie in [FHE+12] untersucht wurde, bringt das Berechtigungsmanagement in Android einige Probleme mit sich. Zum einen kennen viele Nutzer den Zusammenhang zwischen den in Android formulierten Berechtigungen und den möglichen Risiken nicht. Das führt dazu, dass das Gefahrenpotenzial einer App falsch eingeschätzt wird. Zum anderen gibt es in Android

über 100 Berechtigungen. Daher sinkt bei den Nutzern die Bereitschaft, die Berechtigungen zu kontrollieren und mögliche Risiken abzuwägen. Ein weiteres Problem ist, dass Benutzer bei der Installation einer App allen Berechtigungen gleichzeitig zustimmen müssen. Gerade für Benutzer, die in der Lage sind die Risiken einzelner Berechtigungen richtig einzuschätzen, wird dadurch die Auswahl an brauchbaren Apps eingeschränkt. Zahlreiche Diskussionen, die in Internetforen wie [htt] zu lesen sind, zeigen, dass viele Nutzer mit dem aktuellen Berechtigungssystem von Android nicht zufrieden sind und Handlungsbedarf sehen.

In dieser Arbeit wird eine Möglichkeit vorgestellt, um das Berechtigungssystem in Android auf der Basis der *Privacy Management Platform*, kurz: PMP, zu verbessern. Die PMP bietet eine umfassende Kontrolle der Zugriffe auf private Daten, die durch eine intuitive Bedienoberfläche repräsentiert wird. Darüber hinaus bietet die PMP ein Konzept, um den Nutzer über die Funktionen einer App und deren Risiken genau zu informieren. [SM13] Um eine Telefonfunktion zu nutzen, registriert sich die App bei der PMP, und fordert den Zugriff auf eine bestimmte Telefonfunktion an. Ist eine entsprechende Berechtigung vorhanden, stellt die PMP der App eine Implementierung für den Zugriff zur Verfügung.

Im derzeitigen Entwicklungsstand können nur solche Zugriffe verweigert werden, die bei der PMP angefordert werden. Zugriffe, die direkt über Funktionen des Betriebssystems durchgeführt werden, unterliegen dem Berechtigungssystem von Android. Daher entstand die Idee, die PMP in das Betriebssystem zu integrieren und es so zu modifizieren, dass direkte Zugriffe auf Telefonfunktionen unterbunden werden.

Gegenstand dieser Arbeit ist Entwurf einer sinnvollen Integration der Privacy Management Platform in das Betriebssystem Android sowie deren technische Umsetzung.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Berechtigungen in Android In diesem Kapitel wird das Prinzip der Berechtigungen in Android erklärt.

Kapitel 3 – Verwandte Arbeiten befasst sich mit bereits vorhandener Software zum manuellen Setzen von Berechtigungen in Android.

Kapitel 4 – Die PMP stellt die Privacy Management Platform vor.

Kapitel 5 – Integrationsstrategien nennt und vergleicht verschiedene Möglichkeiten der Integration eines alternativen Berechtigungssystems.

Kapitel 6 – Integration der PMP beschreibt die umgesetzte Eingliederung der PMP in das Android Betriebssystem.

Kapitel 7 – Demonstrator behandelt zwei implementierte Anwendungen, die die Funktionalität der integrierten PMP zeigen.

2. Berechtigungen in Android

Das Betriebssystem Android bietet eine API, über die Zugriffe auf alle Daten eines Mobiltelefons möglich sind. Das können Sensoren sein, Netzwerkpakete oder gespeicherte Nutzerdaten wie Kontakte, Kurznachrichten, Bilder oder besuchte Internetseiten. Einige dieser Daten lassen Rückschlüsse auf die Identität des Nutzers zu. Die Weitergabe dieser Daten an Dritte wäre daher ein tiefer Eingriff in die Privatsphäre. Eine App könnte, ob absichtlich oder unabsichtlich, die Funktionalität beeinträchtigen. Sie könnte sogar ein Gerät komplett stilllegen, so dass nur noch eine Neuinstallation des Betriebssystems hilft. Darüber hinaus können über die API Anrufe gestartet oder SMS verschickt werden. Eine unkontrollierte Nutzung dieser Funktionen kann im schlimmsten Fall also auch Kosten nach sich ziehen.

Um Daten und Funktionen des Mobiltelefons gegen unerlaubte Zugriffe zu schützen setzt Android ein Berechtigungssystem ein. In diesem Kapitel wird das Konzept der Berechtigungen näher untersucht. Es wird gezeigt, wie Android sicherstellt, dass eine Anwendung nur Zugriff auf Funktionen und Daten hat für die eine entsprechende Berechtigung vorhanden ist.

2.1. Android Permissions

Einige Funktionen der Android API dürfen nicht ohne spezielle Berechtigungen ausgeführt werden. Diese werden durch Konstanten der Klasse `Manifest.permission` repräsentiert. Zum Beispiel steht der String `BATTERY_STATS` für die Berechtigung, Informationen über den Akku des Geräts auszulesen. Die Berechtigung zu prüfen, ob das Gerät mit dem Internet verbunden ist, wird durch den String `ACCESS_NETWORK_STATE` dargestellt.

Apps benutzen diese String-Konstanten, um Berechtigungen anzumelden. Die Paketdatei jeder App enthält die Konfigurationsdatei `AndroidManifest.xml`. Darin stehen unter anderem Informationen wie Name und Symbol, erforderliche Android-Version und die Angabe, welche Klasse den Startbildschirm implementiert. Außerdem müssen in dieser Datei alle Berechtigungen aufgelistet werden, die für die App nötig sind, um die gewünschten Zugriffe auf das Telefon durchführen zu dürfen. Die Angabe einer Berechtigung wird durch das XML-Element `<uses-permission>` signalisiert. Der Name der Berechtigung steht im zugehörigen Attribut `android:name`.

Die Angaben der Berechtigungen in der Datei `AndroidManifest.xml` werden unter anderem dazu verwendet dem Benutzer vor der Installation einer App mitzuteilen, welche Zugriffe auf Telefonfunktionen und Nutzerdaten diese App vornehmen darf. Verlangt eine

Listing 2.1 `/packages/apps/Camera/AndroidManifest.xml` (Auszug)

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.SET_WALLPAPER" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_SMS" />
```

App beispielsweise die Berechtigung `INTERNET`, sieht der Benutzer nur deren Beschreibung `uneingeschränkter Internetzugriff`. Die Android API besitzt viele Funktionen, die nur mit der Berechtigung `INTERNET` zugänglich sind. Welche das sind und welche diese App nutzt wird vor der Installation nicht angegeben. Der Benutzer kann die Installation an dieser Stelle abbrechen oder bestätigen, dass er mit allen angeforderten Berechtigungen einverstanden ist und die Installation fortsetzen.

Die Berechtigungen einer App mit den Erklärungen können auch nach der Installation im Menü *Einstellungen* → *Anwendungen* angezeigt werden. Hier können auch Berechtigungen für vorinstallierte Apps nachgelesen werden.

Android fasst Berechtigungen, die ähnliche Bereiche schützen, in Gruppen zusammen. Diese Gruppen werden durch die Klasse `Manifest.permission_group` repräsentiert. Sie haben eine geringe Bedeutung für Entwickler, da sie in der Datei `AndroidManifest.xml` nicht angegeben werden. Sie sorgen bei der Installation einer App für eine übersichtliche Darstellung der angeforderten Berechtigungen.

Die in Listing 2.1 angeforderten Berechtigungen werden dem Benutzer bei der Installation gruppiert angezeigt:

- **Meinen Standort:** genauer GPS-Standort
- **Ihre Nachrichten:** SMS oder MMS lesen
- **Speicher:** SD-Karten-Inhalt ändern/löschen
- **Hardware-Steuerelemente:** Audio aufnehmen, Bilder und Videos aufnehmen
- **System-Tools:** Standby-Modus deaktivieren, Hintergrund festlegen

`WAKE_LOCK` und `SET_WALLPAPER` werden in der Gruppe *System-Tools* zusammengefasst, `CAMERA` und `RECORD_AUDIO` gehören der Gruppe *Hardware-Steuerelemente* an.

2.2. Klassifizierung

In Android 2.3.7 gibt es 136 Berechtigungen. Die meisten davon sind im Betriebssystem verankert. Eine App kann auch ihre eigenen Berechtigungen deklarieren, wie etwa die mitgelieferte App *Browser*. Hier werden für das Laden und Speichern von Lesezeichen eigene Berechtigungen eingeführt und der Gruppe `PERSONAL_INFO` zugeordnet.

Android teilt Berechtigungen in vier Stufen ein, die *Protection Levels* genannt werden:

Normal Berechtigungen dieser Stufe schützen Funktionen, die für das System, andere Anwendungen oder den Benutzer als harmlos gelten. Sie werden während der Installation standardmäßig ausgeblendet und erst sichtbar, wenn der Benutzer auf das Feld *alle anzeigen* tippt. Sie werden einer App nach der Installation ohne weitere Prüfung gewährt. Das Risiko wird als gering eingestuft, da eine App mit Berechtigungen wie beispielsweise `VIBRATE`, `SET_WALLPAPER` oder `BATTERY_STATS` das Verhalten eines Geräts nur geringfügig verändern kann.

Dangerous Solche Berechtigungen schützen das System vor Zugriffen, die als riskant eingestuft werden. Dazu gehört das Auslesen von Nutzerdaten, Standort und Telefondaten, sowie der Zugriff auf Telefonhardware und Speicherkarte. Oft angeforderte Berechtigungen dieser Stufe sind `BLUETOOTH`, `READ_CONTACTS`, `ACCESS_NETWORK_STATE`, `ACCESS_FINE_LOCATION` und `CHANGE_WIFI_STATE`. Apps mit den Berechtigungen `SEND_SMS` und `CALL_PHONE` können für den Benutzer sogar Kosten entstehen lassen. Berechtigungen dieser Stufe werden bei der Installation standardmäßig angezeigt und müssen vom Benutzer bestätigt werden. Nachträglich installierte Apps können keine Berechtigungen erlangen, die höher eingestuft sind als *dangerous*.

Signature Paketdateien von Apps mit der Endung `.apk` werden durch einen `private key` signiert. Dadurch kann sichergestellt werden, dass mehrere Softwarekomponenten von demselben Hersteller oder Entwickler stammen. Ohne eine gültige Signatur kann eine App nicht installiert werden. Um Apps während der Entwicklungsphase lokal zu testen, wird vom Android SDK automatisch ein `debug key` erzeugt. Will ein Entwickler seine App veröffentlichen, muss ein `private key` erzeugt werden, der bei kommenden Updates die Herkunft des Pakets bestätigt. Verlangt eine App bei der Installation eine Berechtigung der Stufe *Signature*, wird geprüft, welche App die Berechtigung deklariert hat. Stimmen die Signaturen der beiden Apps überein, wird die Berechtigung erteilt, ohne den Benutzer zu fragen oder zu informieren. Die meisten Deklarationen von Berechtigungen stammen vom Betriebssystem selbst. Daher wird in den meisten Fällen die Signatur der zu installierenden App mit der des Geräteherstellers verglichen.

SignatureOrSystem Berechtigungen dieser Stufe werden nach demselben Kriterium erteilt wie die der Stufe *Signature*. Der Unterschied ist, dass die Berechtigung auch erteilt wird, wenn die Anfrage von einer App kommt, die Teil des Betriebssystems ist, selbst wenn die Signaturen nicht übereinstimmen.

Für benutzerdefinierte Berechtigungen wird die Stufe *SignatureOrSystem* nicht empfohlen. Sie wurde eingeführt, um den Datenaustausch zwischen Apps zu gewährleisten, die Teil des gemeinsamen Betriebssystems sind, aber von unterschiedlichen Herstellern stammen.

2.3. Zugriff auf Telefonfunktionen

In [FCH+11] wird die Kommunikation zwischen einer App und dem Betriebssystem beschrieben. Jeder App liegt eine *API Library* bei. Sie enthält eine *Public API* mit einem *RPC Stub*, der Aufrufe an das Betriebssystem weiterleitet. Die *API Library* hat dieselben Berechtigungen wie die Anwendung und läuft im selben Prozess mit dem selben Benutzernamen. Parallel dazu laufen unter dem Benutzernamen `root` Systemprozesse, in denen die Implementierung der *Public API* sitzt. Ein Teil der Funktionen in der *Public API* sind als `private` deklariert. Sie können nur über Java Reflections erreicht werden. Der Zugriff über Reflections wird aber nicht empfohlen. Funktionen, die als `private` deklariert sind, werden nur von Google-Anwendungen benutzt und können in kommenden Versionen nicht mehr verfügbar sein.

Um eine Funktion des Telefons nutzen zu können, wird die entsprechende Funktion in der beiliegenden *Public API* aufgerufen. Der Aufruf wird an den *RPC Stub* weitergeleitet, der dann die Implementierung der *Public API* im Systemprozess aufruft. Hier findet die eigentliche Prüfung der Berechtigung statt. Abbildung 2.1 zeigt die Anbindung einer App an einen Systemprozess.

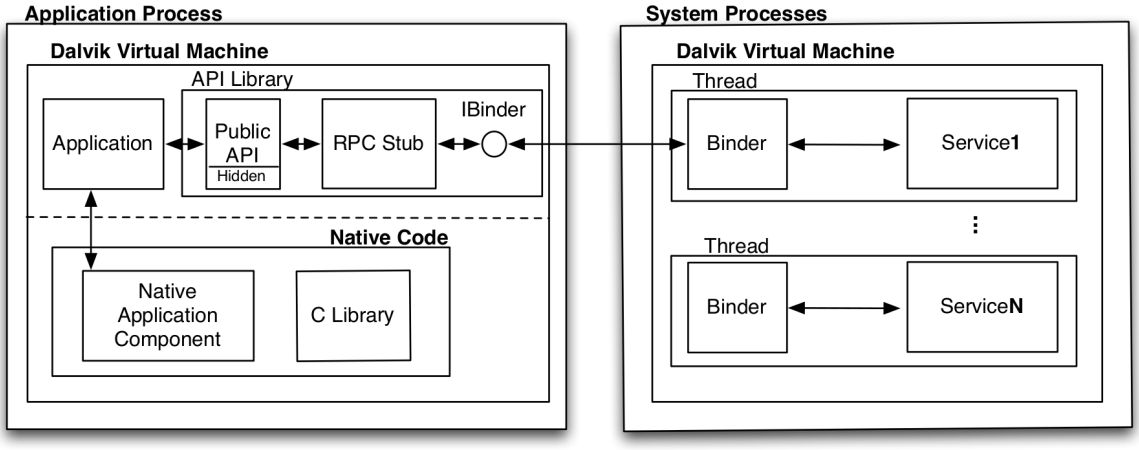

Abbildung 2.1.: Kommunikation zwischen einer App und dem Android-System

2.4. Prüfmechanismus für Apps

Bei vielen Zugriffen auf Telefonfunktionen muss geprüft werden, ob eine entsprechende Berechtigung vorliegt. In Android sind Berechtigungen für Apps an den Benutzernamen geknüpft. Jeder App wird bei der Installation ein Benutzername und eine Benutzernummer (`UID`) zugewiesen. Die Nummer richtet sich nach der Reihenfolge der Installation und reicht von `10000-99999`. Die entsprechenden Benutzernamen lauten `app_0` bis `app_9999`. Der Administrator `root` hat die `UID 0`. Es existieren weitere Benutzernamen und `UID`s für die Ausführung verschiedener Systemkomponenten. Diese sind fest im Android Quellcode verankert.

Der Prüfmechanismus wird in der API Implementierung in Gang gesetzt. Über die Schnittstelle `Context`, von der zentrale Klassen wie `Activity` oder `Service` abgeleitet sind, können Informationen über eine App abgefragt werden. Die Implementierung von `Context` ist Teil des Betriebssystems und enthält die wichtigen Funktionen zur Überprüfung von Berechtigungen.

Fragt eine App eine Ressource an, leitet der RPC Stub die Anfrage an die entsprechende Funktion in der API-Implementierung weiter. Bevor diese den Wert an die App zurückliefert, wird die `enforce()`-Funktion der Implementierung von `Context` aufgerufen. Diese testet über den `ActivityManager` und den `PackageManager`, ob die gegebene `UID` die geforderte Berechtigung hat. Ist dies nicht der Fall, wird eine `SecurityException` ausgelöst, die die App beendet. Beispielhaft für die Abfrage der IMEI-Nummer eines Geräts zeigt die Abbildung 2.2 die Kette der Aufrufe vom `TelephonyManager` bis zur Prüfung durch den `PackageManager`.

2.5. Speicherung bei der Installation

Die mitgelieferte App `PackageInstaller` stellt die Benutzeroberfläche für die Installation neuer Apps bereit. In der Klasse `PackageInstallerActivity` Wird der Dialog aufgebaut, der dem Benutzer die Berechtigungen anzeigt, die von der zu installierenden App angefordert werden. Bestätigt der Benutzer die Installation, wird der `PackageManager` aufgerufen. Die Funktion `InstallPackage` führt die komplette Installation durch. Es wird zunächst geprüft, ob die Berechtigung `INSTALL_PACKAGES` beim `PackageManager` oder `PackageInstaller` vorliegt, dann wird der Installationsmechanismus in Gang gesetzt.

Der Ablauf der Installation ist durch einen `Handler` realisiert. Ein `Handler` kann eine ihm zugehörige Liste von Aufgaben bearbeiten. Diese Liste heißt `MessageQueue`. Ihre Elemente sind entweder Objekte der Klasse `Message` oder Objekte von Klassen, die die Schnittstelle `Runnable` implementieren. Ein `Handler` und die zugehörige `MessageQueue` werden benutzt, um Aufgaben des eigenen Prozesses zu einem späteren Zeitpunkt auszuführen oder einem anderen Prozess eine Aufgabe zu übergeben.

Der `Handler` im `PackageManager` verarbeitet zuerst die Nachricht `INIT_COPY`. Hier wird geprüft, ob der Dienst `Media Container Service` verfügbar ist und gegebenenfalls neu gestartet. Die weitere Installation erfolgt durch die Verarbeitung der Nachricht `MCS_BOUND`.

Die Methode handleStartCopy startet den Kopiervorgang. Danach verarbeitet die Methode handleReturnCode das Ergebnis des Kopiervorgangs. War der Vorgang erfolgreich, wird durch die Methode ScanPackageLI Paketinformationen ausgelesen. Darin stehen die Berechtigungen des Pakets. Die Methode updateSettingsLI schreibt die gelesenen Informationen in das Objekt mSettings. Um diese Informationen dauerhaft zu speichern, werden sie durch die Methode writeLP im XML-Format auf dem Dateisystem des Geräts abgelegt. Abbildung 2.3 zeigt die Systemaufrufe von der Bestätigung des Benutzers, die Installation zu beginnen bis zur Speicherung der Berechtigungen des installierten Pakets.

2.6. Einlesen von Berechtigungen beim Systemstart

Die erste Java-Komponente die beim Bootvorgang eines Android-Geräts ausgeführt wird, ist die Methode SystemServer.run(). Darin werden alle Hintergrunddienste wie Status Bar, Bluetooth Service oder Package Manager gestartet. Der Dienst Package Manager liest beim Starten alle relevanten Informationen über installierte Pakete und deren gespeicherten Berechtigungen.

Die Methode readPermissions() durchsucht den Ordner /system/etc/permissions/. Hier befinden sich XML-Dateien. Die meisten davon haben die Namensstruktur von Java Packages, zum Beispiel android.hardware.touchscreen.multitouch.xml. Sie stehen für die Hardware-Features eines Geräts und definieren die dazugehörigen Berechtigungen. Diese Dateien werden zuerst gelesen. Anschließend wird die Datei platform.xml gelesen. Sie weist Systembenutzern wie shell oder media ihre Berechtigungen zu.

Als nächstes erzeugt der PackageManager ein Objekt der Klasse Settings. Darin werden globale Informationen über installierte Pakete abgelegt. Die Methode Settings.readLP() liest die Datei /system/etc/permissions/packages.xml. Sie enthält eine Liste aller global verfügbaren Berechtigungen sowie Informationen zu jedem mitgelieferten und nachinstallierten Paket:

- Anzahl der Signaturen
- Schlüssel der Signatur
- Installationsort- und Datum
- Benutzer-ID UID mit der das Paket ausgeführt wird
- Berechtigungen, die dem Paket gewährt wurden

Danach wird für jedes Paket über die Methode addUserIdLP() ein Eintrag im Settings-Objekt erzeugt. Dieser Eintrag wird dazu genutzt, zur Laufzeit zu prüfen, ob eine Anwendung eine bestimmte Berechtigung besitzt. Die Indizierung der Einträge erfolgt durch die Benutzernummer UID.

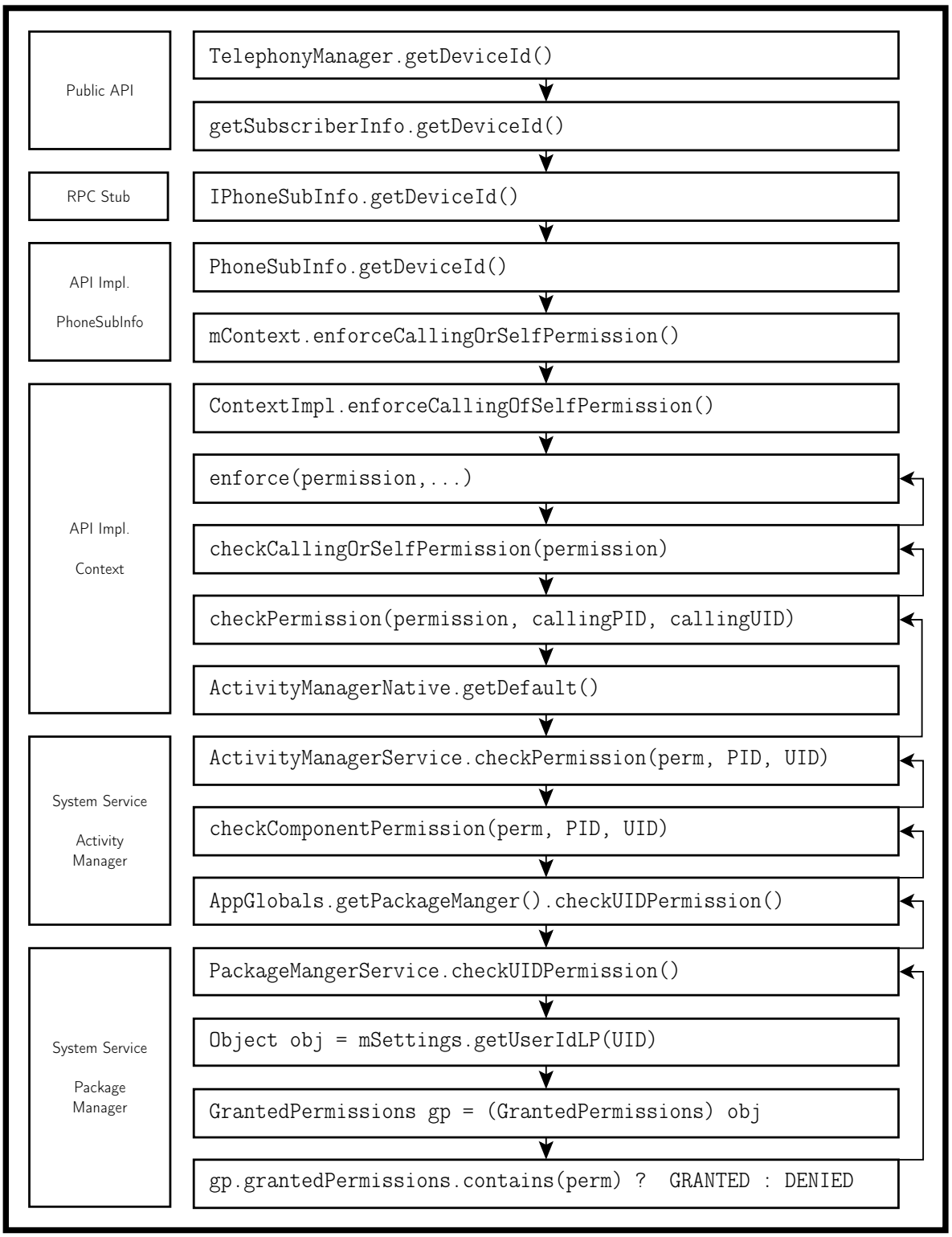

Abbildung 2.2.: Funktionsaufrufe der Berechtigungs-Prüfung. Pfeile nach oben signalisieren Rückgabewerte.

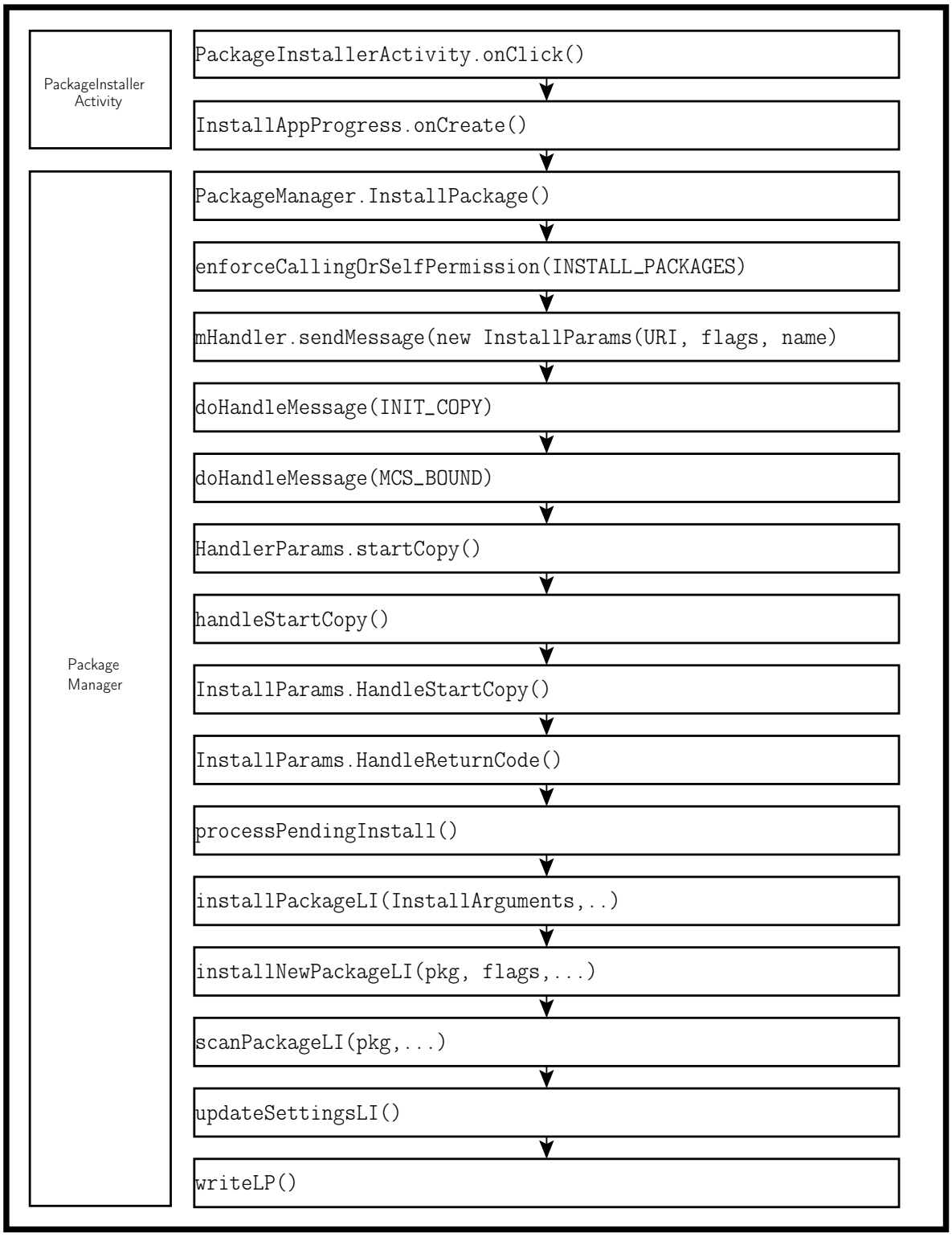

Abbildung 2.3.: Funktionsaufrufe bei der Installation einer App

3. Verwandte Arbeiten

3.1. Wissenschaftliche Ansätze

Es existieren bereits einige wissenschaftliche Arbeiten zu den Themen Sicherheit und Privatsphäre auf Android-Systemen. In diesem Abschnitt wird eine Auswahl davon untersucht.

3.1.1. Apex

In [NKZ10] wird ein Verfahren beschrieben, wie sich Berechtigungen für eine App während und nach der Installation aktivieren oder deaktivieren lassen. *Apex* steht für *Android Permission Extension* und bezeichnet ein Framework, das aus der Modellierung und Erweiterung des Konzepts der Permissions in Android entstanden ist. Damit können neue Zugangsbedingungen (*Policies*) an Android Permissions geknüpft werden. Diese werden in einer eigenen Programmiersprache formuliert.

Damit der Benutzer sich nicht mit dieser Programmiersprache auseinandersetzen muss, wurde die Installation von Apps modifiziert. Die neue Installationsroutine heißt *Poly*. Sie passt sich optisch an die mitgelieferte Installationsroutine an und übernimmt auch deren Aufgaben. Zusätzlich kann jede Android Permission, die eine App anfordert, durch Antippen deaktiviert oder mit zusätzlichen Bedingungen versehen werden. Um den Benutzer nicht zu überfordern, kann die Installation auch wie gewohnt durch einmalige Bestätigung fortgesetzt werden, ohne die Rechte der App zu verändern.

Konkrete Quelltexte liegen für diesen Ansatz nicht vor. Nach [NKZ10] wurden die Klassen `Activity Manager` und `Package Manager` sowie der Mechanismus zur Berechtigungsprüfung geändert.

3.1.2. Aurasium

Das in [XSA12] beschriebene Verfahren kommt ohne Veränderung des Betriebssystems aus. Aurasium verfeinert das bestehende Berechtigungssystem in vielen Bereichen. Für Internetverbindungen wird eine Liste mit verbotenen IP-Adressen gespeichert. Bevor eine App eine SMS sendet wird geprüft, ob es sich um eine teure SMS an eine Premium-Nummer handelt. Werden sensible Daten wie Kontakte oder den Standort angefordert, greift entweder eine vorher gespeicherte Einstellung oder der Benutzer wird gefragt.

Grundidee ist die Modifikation von Paketdateien. Aurasium benutzt das frei zugängliche Hilfsprogramm `apktool`, um aus der Datei `classes.dex` aus den Paketdateien eine Verzeichnisstruktur mit Java Klassen zu erstellen. Zu diesen Klassen werden die erweiterten Sicherheitsprüfungen hinzugefügt. Um zu überwachen, wann ein Zugriff auf Telefondaten erfolgt, wird nativer C-Code in das Paket mit eingebunden. Anschließend wird das Paket wieder eingepackt und kann installiert werden.

3.2. Experimentelle Software

Seit bekannt wurde wie sich auf Android-Geräten Administratorrechte freischalten lassen, werden stetig neue Anwendungen oder Modifikationen entwickelt, die tiefgreifende Veränderungen im System zulassen. Durch das sogenannte „rooten" eines Geräts können Anwendungen wie ein Dateimanager, eine Kommandozeile oder eine Firewall installiert werden. Fortan entstanden auch modifizierte Varianten des kompletten Betriebssystems.

Unter diesen Anwendungen und Betriebssystem-Modifikationen befinden sich auch solche, die sich im Hinblick auf die Privatsphäre des Benutzers um das Setzen und Entziehen von Berechtigungen kümmern. Eine Auswahl daraus soll im Folgenden behandelt werden.

3.2.1. CyanogenMod 7

Der CyanogenMod ist eine Modifikation des Betriebssystems Android, die 2009 erstmals zum Download angeboten wurde. Während der Entwickler Steve Kondik anfangs die meisten Codezeilen selbst schrieb, hat der CyanogenMod inzwischen eine große Entwickler- und Nutzergemeinde. Mit über 3,8 Millionen aktiven Installationen[1] ist der CyanogenMod aktuell die meistverbreitetste Android-Modifikation. Zu jeder Version von Android existiert eine Version des CyanogenMod. Die im Folgenden untersuchte Version 7.2 basiert auf Android 2.3.

Neben zahlreichen Veränderungen in Bereichen wie Benutzeroberfläche, Netzwerk und Konfigurierbarkeit bietet der CyanogenMod 7.2 auch die Möglichkeit, einzelne Berechtigungen für Anwendungen nach dem Installationsprozess anzuzeigen und zu verändern. Ruft der Benutzer Zusatzinformationen zu einer installierten App auf, wird unter den Informationen zum Speicherverbrauch die Liste der Berechtigungen angezeigt, die die App während der Installation angefordert hat. Der Benutzer kann hier auf eine Berechtigung tippen. Deren Text wird dann durchgestrichen dargestellt und die App darf keine Funktionen mehr verwenden, die diese Berechtigung erfordern.

Der CyanogenMod ist ein gut dokumentiertes, quelloffenes Projekt. Es gibt eine informative Homepage und ein gut besuchtes Forum. Außerdem sind alle Versionen des CyanogenMod auf `github.com` [2] abgelegt. So kann sich jeder Interessierte entweder online durch den

[1] stats.cyanogenmod.com
[2] https://github.com/CyanogenMod

```
public int checkPermission(String permName, String pkgName) {
    synchronized (mPackages) {
        PackageParser.Package p = mPackages.get(pkgName);
        if (p != null && p.mExtras != null) {
            PackageSetting ps = (PackageSetting)p.mExtras;

            if (ps.sharedUser != null) {
                if (ps.sharedUser.grantedPermissions.contains(permName)) {
                    return PackageManager.PERMISSION_GRANTED;
                }
            } else if (ps.grantedPermissions.contains(permName)) {
                return PackageManager.PERMISSION_GRANTED;
            }
        }
    }
    return PackageManager.PERMISSION_DENIED;
}
```

```
public int checkPermission(String permName, String pkgName) {
    synchronized (mPackages) {
        PackageParser.Package p = mPackages.get(pkgName);
        if (p != null && p.mExtras != null) {
            PackageSetting ps = (PackageSetting)p.mExtras;
            int uid = Binder.getCallingUid();
            if (ps.sharedUser != null) {
                if (ps.sharedUser.grantedPermissions.contains(permName)) {
                    return checkRevoked(permName, ps.sharedUser, uid, ps.sharedUser.userId);
                }
            } else if (ps.grantedPermissions.contains(permName)) {
                return checkRevoked(permName, ps, uid, ps.userId);
            }
        }
    }
    return PackageManager.PERMISSION_DENIED;
}
```

Abbildung 3.1.: Zentrale Änderungen im CyanogenMod7 (unten) im Vergleich zum Android-Quellcode (oben) für das Entziehen von Berechtigungen

kompletten Quellcode jeder Version navigieren oder den Quellcode herunterladen und in einer Entwicklungsumgebung betrachten.

Auch die Plattform für Code Reviews des CyanogenMod ist frei zugänglich[3]. Dort können die Änderungen, die dazu führen, dass Android-Berechtigungen editierbar sind, nachvollzogen werden.

Dazu führen die Entwickler des CyanogenMod eine Liste mit entzogenen Berechtigungen ein. Deaktiviert der Benutzer eine Berechtigung für eine App, wird in der Klasse `PackageManagerService` die Liste `revokedPermissions` um diese Berechtigung erweitert. An der Stelle im `PackageManagerService`, wo Android die Überprüfung einer Berechtigung für eine App abgeschlossen hat und die Konstante `PERMISSION_GRANTED` liefert, wird zusätzlich geprüft, ob sich die betreffende Berechtigung in der Liste der entzogenen Berechtigungen befindet (Abb. 3.1). Ist das der Fall, wird trotz erfolgreicher Prüfung durch das Android-System die Konstante `PERMISSION_DENIED` zurückgeliefert.

[3] http://review.cyanogenmod.org

Mit dem CyanogenMod steht also ein benutzerfreundliches System zur Verfügung, das es erlaubt, Berechtigungen für bereits installierte Anwendungen jederzeit einzusehen und zu verändern. Doch die Art der Implementierung bringt Nachteile mit sich. Will eine App auf Funktionen zugreifen, für die keine Berechtigung erlangt werden kann, wird in den meisten Fällen ein Fehler ausgelöst. Da der Entwickler der App davon ausgehen kann, dass die erforderlichen Rechte vorhanden sind, ist für fehlende Rechte normalerweise keine Fehlerbehandlung vorgesehen. Es ist daher sehr wahrscheinlich, dass eine App, deren Berechtigungen nachträglich eingeschränkt wurden, mit einer Fehlermeldung beendet wird. So ist zwar sichergestellt, dass eine App keinen Zugriff auf nicht erwünschte Bereiche hat, doch wird durch das Entziehen der Berechtigungen der Funktionsumfang der App beeinträchtigt.

Bisher ist dieses Feature nur in CyanogenMod 7.2 verfügbar. Versionen des CyanogenMod, die auf Android 4 basieren, bieten keine Änderungen der Berechtigungen nach der Installation. Für den CyanogenMod 9 und 10 ist eine solche Berechtigungsverwaltung derzeit auch nicht vorgesehen.

Zusammenfassung

- Vorteile:
  - Benutzerfreundliche Konfiguration aller Berechtigungen
  - Weit verbreitet, dadurch gute Unterstützung
  - Viele gute Anleitungen zur Installation verfügbar
- Nachteile:
  - Anwendungen stürzen bei verweigerten Berechtigungen oft ab.

3.2.2. PDroid

PDroid ist wie der CyanogenMod eine Modifikation des Android-Betriebssystems. Das Projekt ist quelloffen, aber wenig dokumentiert. Die Software wird nicht als komplettes Android-Image ausgeliefert, sondern als Patch. Um PDroid zu installieren, muss also ein Image einer passenden Android-Version vorhanden sein. Darauf wird der PDroid-Patch angewendet und anschließend das modifizierte Betriebssystem auf dem Handy installiert. Für Windows 7 (XP, Vista) ist das Programm *Automatic ROM Patcher* verfügbar, um die Installation zu erleichtern.

PDroid kann derzeit nicht auf Android-Versionen angewendet werden, die von Herstellern mit einem Gerät ausgeliefert werden. Es wird eine modifizierte Version von Android 2.3 „Gingerbread“ benötigt. Die Unterstützung für neuere Android-Versionen soll folgen.

Die Benutzeroberfläche von PDroid bietet die Möglichkeit, für jede App andere Einstellungen der Berechtigungen vorzunehmen. Die meisten Berechtigungen lassen sich aktivieren oder

```
    private TelephonyManager getTelephonyManager() {
        synchronized (mSync) {
            if (mTelephonyManager == null) {
                mTelephonyManager = new TelephonyManager(getOuterContext());


            }
        }
        return mTelephonyManager;
    }
```

```
    private TelephonyManager getTelephonyManager() {
        synchronized (mSync) {
            if (mTelephonyManager == null) {
                // BEGIN privacy-modified
                mTelephonyManager = new PrivacyTelephonyManager(getOuterContext());
                // END privacy-modified
            }
        }
        return mTelephonyManager;
    }
```

Abbildung 3.2.: Wichtige Änderung in PDroid: Wird ein `TelephonyManager` angefordert, liefert PDdroid (unten) die modifizierte Version `PrivacyTelephonyManager`.

deaktivieren. Einige Berechtigungen wie das Auslesen der IMEI-Nummer des Geräts können so konfiguriert werden, dass entweder ein Zufallswert oder eine vom Benutzer angegebene Zahl zurückgegeben wird.

Die technische Umsetzung erfolgt durch neue Klassen wie `PrivacyTelephonyManager`, `PrivacyAccountManager` oder `PrivacyLocationManager`. Diese sind von den Klassen `TelephonyManager`, `AccountManager` oder `LocationManager` abgeleitet und implementieren solche Funktionen neu, die für die Privatsphäre des Benutzers relevant sind. Dazu gehören beispielsweise das Auslesen der Telefonnummer, der gespeicherten Kurznachrichten oder der besuchten Internetseiten.

Zusammenfassung

- Vorteile:
  - Übersichtliche Benutzeroberfläche
  - Einige persönliche Daten konfigurierbar, z.B. IMEI-Nummer
  - Anwendungen laufen bei verweigerten Berechtigungen weiter.
- Nachteile:
  - Nicht alle sicherheitsrelevanten Daten werden geschützt.
  - Bei einem Update muss das komplette Betriebssystem neu aufgesetzt werden.

3.2.3. Privacy Protector

Es gibt auch Software für das Verwalten von Berechtigungen, die kein Android-Gerät mit Administratorrechten erfordert. Ein Beispiel dafür ist die Anwendung PrivacyProtector. Sie verwaltet den Zugriff auf den mobilen Internetzugang und die Standortbestimmung.

Beim Starten von Privacy Protector wird eine Liste aller Anwendungen angezeigt, die auf dem Gerät installiert sind. Für jede Anwendung kann der Benutzer über ein Kontrollkästchen auswählen, ob sie Zugriff auf die Internet- oder GPS-Funktionen haben soll.

Die einfache Funktionsweise von Privacy Protector ist der Grund, warum keine Administratorrechte erforderlich sind. Der Privacy Protector startet einen Hintergrunddienst, der einmal pro Sekunde überprüft, welche Anwendung momentan im Vordergrund läuft. Sobald eine neue Anwendung im Vordergrund registriert wird, werden die benutzerdefinierten Einstellungen für die neue Anwendung geladen. Wurden dieser Anwendung die Rechte für die Internet- oder GPS-Funktionen entzogen, werden WLAN sowie mobile Zugangspunkte oder der GPS-Empfänger deaktiviert.

Das offensichtliche Problem bei diesem Vorgehen ist, dass die Deaktivierung der Internet- oder GPS-Funktion sich systemweit auswirkt. Das Android-System ist so ausgelegt, dass Anwendungen nicht beendet, sondern angehalten werden, wenn sie nicht mehr sichtbar sind. Daher sind von einer Deaktivierung meistens mehrere Anwendungen betroffen. Darunter können sich auch solche befinden, die laut Privacy Protector auf die Internet- und GPS-Dienste zugreifen dürfen.

Auf die gleiche Weise wie der Privacy Protector ohne Administratorrechte die Internetverbindung eines Geräts ausschalten kann, darf eine andere Anwendung sie einschalten. Dazu wird die Berechtigung `CHANGE_WIFI_STATE` benötigt. Die Aktivierung der Standortbestimmung durch eine Anwendung ist im Android-System nicht vorgesehen und soll nur durch Benutzereingabe erfolgen. Jedoch kann durch eine Sicherheitslücke [sta] im *Power Management Widget* der GPS-Empfänger auch ohne Benutzerinteraktion eingeschaltet werden.

Bei jeder Anwendung, die beim Start überprüft, ob erforderliche drahtlose Verbindungen ausgeschaltet sind und sie gegebenenfalls aktiviert, ist der Privacy Protector machtlos.

Zusammenfassung

- Vorteile:
  - Einfache Installation über den Google Play Store
  - Keine Administratorrechte erforderlich
- Nachteile:
  - Behandelt nur die Internet- und GPS-Funktionen
  - Funktionen werden systemweit abgeschaltet.
  - Leicht zu umgehen

4. Die PMP

4.1. Motivation

Die Liste der persönlichen Daten, die ein Mobiltelefon beherbergen kann, ist lang. Umso wichtiger ist es auch, sie zu schützen. In Android sind daher alle Bereiche, die persönliche Daten enthalten können, mit Berechtigungen versehen. Eine App oder Teile des Betriebssystems selbst dürfen nur auf eine geschützte Funktion zugreifen, wenn eine entsprechende Berechtigung vorliegt. Alle Berechtigungen, die eine App anfordert müssen bei der Installation gewährt werden. Es ist weder bei der Installation noch zu einem späteren Zeitpunkt möglich, einzelne Berechtigungen zu deaktivieren. Ist der Benutzer mit den Berechtigungen einer App nicht einverstanden, gibt es also nur die Möglichkeit, die Installation abzubrechen. Manche Berechtigungen wie die für den Internetzugriff eröffnen einer App viele Möglichkeiten. Wofür eine App diese Berechtigung im Detail braucht, bleibt dem Benutzer bei der Installation verborgen.

Aus der Motivation heraus, das zu ändern ist im Rahmen eines Studienprojektes die **Privacy Management Platform** (kurz: PMP) entstanden. Der Benutzer soll nicht nur wissen, auf welche Daten eine App zugreift, sondern auch zu welchem Zweck.

4.2. Funktionen

Die PMP besitzt zwei Betriebsmodi. Der einfache Modus (*Simple Mode*) beinhaltet nur die wichtigsten Funktionen der PMP, um unerfahrene Benutzer nicht zu überfordern. Der Expertenmodus (*Expert Mode*) richtet sich an erfahrene Benutzer. Die Konfiguration der Berechtigungen für eine App ist aufwändiger, dafür kann hier der volle Funktionsumfang der PMP genutzt werden.

4.2.1. Einfacher Modus

Die Standardeinstellung ist der Einfache Modus. Er deckt die wichtigsten Funktionen der PMP ab. Wird eine PMP-kompatible App installiert, sieht der Benutzer eine Liste der angeforderten Zugriffe auf das Gerät. Zu jedem Zugriff werden von der App weitere Informationen bereitgestellt. Diese weiteren Informationen können beliebig ausführlich sein. Sie sollen dem Benutzer mitteilen, welche Daten mit dem angeforderten Zugriff vom Gerät

gelesen werden können und wie diese Daten weiterverarbeitet werden. Zusätzlich kann sich der Benutzer eine genauere Beschreibung der App anzeigen lassen.

Diese Informationen sind eine wichtige Grundlage für die nächste Funktion der PMP. Der Benutzer kann jetzt entscheiden, welche Zugriffe erlaubt sein sollen und welche nicht. Hat der Benutzer alle unerwünschten Zugriffe abgeschaltet, wird die App geöffnet und kann sofort genutzt werden. Im Gegensatz zu vielen bereits vorhandenen Programmen zur Regulierung von Berechtigungen muss weder die PMP noch das Gerät neu gestartet werden.

Ein weiterer Vorteil gegenüber bereits vorhandener Software zeigt sich beim Betrieb der App. Einige Ansätze wie der CyanogenMod 7, der im Kapitel 3.2.1 vorgestellt wurde, entziehen den Apps Berechtigungen und führen zu häufigen Abstürzen, da Android auf einen Zugriff, für den eine entsprechende Berechtigung fehlt, mit einer `SecurityException` reagiert. Erfolgt dieser Zugriff schon beim Start der App, wird sie vollständig unbrauchbar. Die PMP forciert für jeden nicht autorisierten Zugriff eine Fehlerbehandlung. Die App kann den Zugriff so nicht vornehmen, läuft aber weiter.

Die PMP erlaubt es außerdem, die Zugriffsrichtlinien für Apps auch nach der Installation einzusehen und zu verändern. Auch in diesem Fall ist kein Neustart notwendig.

4.2.2. Expertenmodus

Im Expertenmodus basiert die Steuerung von Zugriffsrechten auf *Presets*. Ein *Preset* ist eine Sammlung von Telefonfunktionen. Zu jeder Telefonfunktion im *Preset* wird gespeichert, wie diese genutzt werden darf. Die meisten Funktionen können entweder aktiviert oder deaktiviert werden. Bei einigen Funktionen kann es sinnvoll sein, die Berechtigung abzustufen. Bei der Standortbestimmung bietet die PMP die Einstellungen *Straße*, *Stadt*, *Land* oder *keine*.

Beim Einsatz von *Presets* können Berechtigungen auch an Bedingungen geknüpft werden. Derzeit werden ortsbezogene und zeitbezogene Bedingungen unterstützt. So können Telefonfunktionen nur zu bestimmten Uhrzeiten oder an bestimmten Tagen freigegeben werden. Ähnlich kann eine Telefonfunktion nur dann zugänglich gemacht werden, wenn sich der Benutzer in einem Umkreis um einen bestimmten Punkt befindet. Durch den Einsatz mehrerer *Presets* für dieselbe App können bei ortsbezogenen oder zeitbezogenen Bedingungen Konflikte entstehen. Diese werden von der PMP automatisch erkannt und durch ein Ausrufezeichen signalisiert.

Durch die zusätzlichen Funktionen im Expertenmodus verändert sich auch die Installation einer neuen App. Wie im Einfachen Modus sieht der Benutzer alle Zugriffe auf das Gerät, die die App vornehmen will. Anstatt diese Zugriffe zu aktivieren oder zu deaktivieren, muss jeder erlaubte Zugriff einer Telefonfunktion in einem *Preset* zugeordnet werden. Dazu kann ein neues *Preset* erstellt oder ein Bestehendes verwendet werden. Alle *Presets*, die während der Installation einer App verwendet werden, werden automatisch dieser App zugeordnet. Die Menge aller Telefonfunktionen dieser *Presets* legt also den erlaubten Funktionsumfang der App fest.

In einem *Preset* kann der Benutzer nicht nur festlegen, welche Telefonfunktionen erlaubt sind, sondern auch, welche Art von Daten an die App geliefert werden. Die PMP unterstützt drei Arten von Daten:

NORMAL Die Applikation erhält von der PMP unveränderte Daten, die vom Telefon ausgelesen wurden.

MOCK Die PMP liefert der App Daten, die nicht vom Telefon stammen, sondern zufällig generiert oder auf einen bestimmten Wert festgesetzt sind. Die PMP teilt der App mit, dass es sich um verfälschte Daten handelt. Die App kann also den Benutzer darüber informieren, dass die Daten gefälscht sind.

CLOAK Die Applikation erhält ebenfalls veränderte Daten mit dem Unterschied, dass die PMP der App nicht mitteilt, dass die Daten falsch sind. Die App verhält sich also genauso, wie es mit echten Daten der Fall wäre.

4.3. Aufbau

Die Hauptfunktion der PMP besteht darin Anwendungen den Zugriff auf Telefonfunktionen zu erlauben oder zu verweigern. Grundvoraussetzung dafür ist, dass sich Apps nach deren Installation bei der PMP registrieren. Telefonfunktionen sind in Android durch Berechtigungen geschützt, die zum Zeitpunkt der Installation erteilt werden und nicht entzogen werden können. Das Konzept der Berechtigungen in Android wird in Kapitel 5 näher beschrieben. Um den Zugriff auf Telefonfunktionen steuern zu können, stellt die PMP eigene Methoden für den Zugriff bereit, die von den registrierten Apps genutzt werden.

4.3.1. Ressourcen und Ressourcengruppen

Die von der PMP angebotenen Methoden für den Zugriff auf Telefonfunktionen werden in sogenannte *Ressourcen* gegliedert. Eine Ressource ist also eine Implementierung von Zugriffen auf mehrere inhaltlich und logisch zusammengehörende Telefonfunktionen. Ressourcen mit ähnlichen Funktionen gehören zu einer *Ressourcengruppe*. Die Zugehörigkeit von Funktionen zu Ressourcen oder von Ressourcen zu Ressourcengruppen muss sich nicht nach der Implementierung im Android Betriebssystem richten. Eine einzelne Methode einer Ressource kann auch Aufgaben von mehreren Android-Methoden zusammenfassen. Wichtig ist nur, dass die Funktionen, die von den installierten Apps benötigt werden, in den Ressourcengruppen der PMP enthalten sind.

4.3.2. Privacy Settings

Jede Ressourcengruppe definiert eigene Regeln für den Zugriff auf ihre Funktionen. Diese Regeln heißen `Privacy Settings`. Sie schützen die Funktionen von Ressourcengrouppen vor ungewünschten Zugriffen, sind also im Ansatz vergleichbar mit Android Permissions. Ein Unterschied zu Android Permissions ist, dass Privacy Settings verschiedenartige Werte annehmen können.

- `BooleanPrivacySetting`: „true“ oder „false“
- `IntegerPrivacySetting`: Eine Ganzzahl
- `EnumPrivacySetting`: Eine Aufzählung
- `SetPrivacySetting`: Eine Menge an beliebigen Objekten

Dadurch können Zugriffe auf Funktionen nicht nur verboten, sondern beliebig abgestuft oder näher spezifiziert werden. Für einen abgestuften Zugriff auf Kurznachrichten könnte zum Beispiel die Aufzählung {„Keine“, „Gesendet“, „Empfangen“, „Alle“} verwendet werden. Eine GPS-Ressourcengruppe könnte die Genauigkeit der Standortbestimmung durch den Typ `IntegerPrivacySetting` spezifizieren.

4.3.3. Presets und Contexts

Die Presets, die sich im Expertenmodus anlegen lassen, sind eine Sammlung von Privacy Settings zusammen mit deren Wert. Diese können aus unterschiedlichen Ressourcengruppen sein. Wenn einer App mehrere Presets zugeordnet sind, kann es vorkommen, dass deren Privacy Settings sich überschneiden. Sind die gemeinsamen Privacy Settings auf unterschiedliche Werte gesetzt, entsteht ein Konflikt. Konflikte werden von der PMP erkannt und können angezeigt werden.

Um festzulegen, ob echte oder gefälschte Daten an die App weitergegeben werden, besitzt jede Ressourcengruppe ein spezielles `EnumPrivacySetting` mit dem Namen `Mode` und den möglichen Werten `NORMAL`, `MOCK` oder `CLOAK`. Dieses Privacy Setting kann zu einem Preset hinzugefügt werden. Der hier eingestellte Wert gilt für alle Privacy Settings des Presets, die aus derselben Ressourcengruppe stammen.

Die in Abschnitt 4.2.2 erwähnten Bedingungen sind durch `Contexts` realisiert. Contexts sind mit Privacy Settings verbunden. Sie bestehen aus einer Bedingung für Zeit oder Ort und einem Wert. Wird die Bedingung wahr, überschreibt dieser Wert den Wert, auf den das Privacy Setting im Preset gesetzt wurde.

An ein Privacy Setting können beliebig viele ortsbezogene oder zeitbezogene Contexts geknüpft sein. Überschneiden sich Zeit oder Ort zweier Contexts, kann ein Konflikt entstehen.

4.3.4. Service Features

Die PMP ermöglicht dem Benutzer, Zugriffe einer App auf Telefonfunktionen zu verhindern. Dazu muss der PMP übermittelt werden, welche Zugriffe eine App vornehmen will. Zu diesem Zweck müssen für jede App `Service Features` definiert werden. Service Features sind in ihrer Funktion vergleichbar mit den Einträgen der Form <`uses-permission`> in der Datei `AndroidManifest.xml`, mit denen Apps beim Android Betriebssystem Berechtigungen anfordern (Abschnitt 2.1). Die besitzen jedoch einen frei wählbaren Namen und eine Beschreibung. So können dem Benutzer für jeden Zugriff, den eine App vornehmen will, detaillierte Informationen gegeben werden.

Um der PMP mitzuteilen welche Funktionen ein Service Feature nutzt, sind für jedes Service Feature die erforderlichen Privacy Settings und deren erforderlicher Wert notiert. Jedes Service Feature muss mindestens ein Privacy Setting definieren. Die Anzahl der Privacy Settings ist unbegrenzt. Eine App könnte ein einziges Service Feature anbieten, das alle benötigten Privacy Settings enthält. Je mehr Service Features eine App anbietet, desto feiner kann der Benutzer die Zugriffe auf die PMP Ressourcengruppen konfigurieren.

Die Architektur der PMP wird in [SM13] detailliert beschrieben.

4.4. Interaktion mit einer App

Bei ihrer Installation registrieren sich Apps bei der PMP. Dabei wählt der Benutzer aus, welche Zugriffe erlaubt sein sollen und welche nicht. Im Einfachen Modus wählt der Benutzer die erlaubten Service Features aus. Im Expertenmodus legt der Benutzer ein oder mehrere Presets an, die alle Privacy Settings beinhalten, die von den Service Features der App benötigt werden.

Für die Speicherung der Auswahl an erlaubten Zugriffen spielt der Betriebsmodus keine Rolle. Die Tabellen der Datenbank, dargestellt in Tabelle 4.1, bleiben gleich.

Tabellenname	Zweck
ResourceGroup	Attribute speichern
PrivacySetting	PrivacySetting ↔ Resourcegroup
App	Attribute speichern
ServiceFeature	Service Feature ↔ App
Preset	Attribute speichern
ServiceFeature_RequiredPrivacySettingValue	Service Feature ↔ Privacy Setting
Preset_GrantedPrivacySettingValue	Preset ↔ Privacy Setting
Preset_AssignedApp	Preset ↔ App
Context_AnnotatedPrivacySettingValue	Context ↔ Privacy Setting

Tabelle 4.1.: Tabellen der PMP-Datenbank. Ein Doppelpfeil symbolisiert eine Zuordnung

Über die Methode `getResource` kann eine App auf Funktionen einer PMP-Ressourcengruppe zugreifen. Eine Ressourcengruppe besitzt für jede Funktion drei Implementierungen. Eine für normale Werte (`NORMAL`), eine für Zufallswerte (`MOCK`) und eine für Zufallswerte, die wie echte Werte behandelt werden (`CLOAK`). Welche davon aufgerufen wird, entscheidet das Privacy Setting `Mode`. Jede Funktion einer Ressource prüft zu Beginn, ob die erforderlichen Privacy Settings für die App gelten, die eine Implementierung anfordert. Ist dies nicht der Fall, wird eine SecurityException ausgelöst und die Fehlerbehandlung in der App startet.

4.5. Benutzeroberfläche

Für die Konfiguration der PMP steht eine übersichtliche grafische Oberfläche zur Verfügung. Der Startbildschirm zeigt das Hauptmenü. Wie in Abbildung 4.3 dargestellt, führen vier große Schaltflächen zu den untergeordneten Menüs:

Apps Hier werden alle Apps aufgelistet, die bei der PMP registriert sind. Durch die Auswahl einer App kann der Benutzer deren Beschreibung lesen und deren Service Features aktivieren oder deaktivieren. Im Expertenmodus werden hier für Service Features notwendige Privacy Settings zu Presets hinzugefügt. Der Benutzer kann hier auch festlegen, welche Presets der App zugeordnet sind. Abbildung 4.1 zeigt die Details einer App.

Abbildung 4.1.: Anzeige der Details zur App „Information Read“

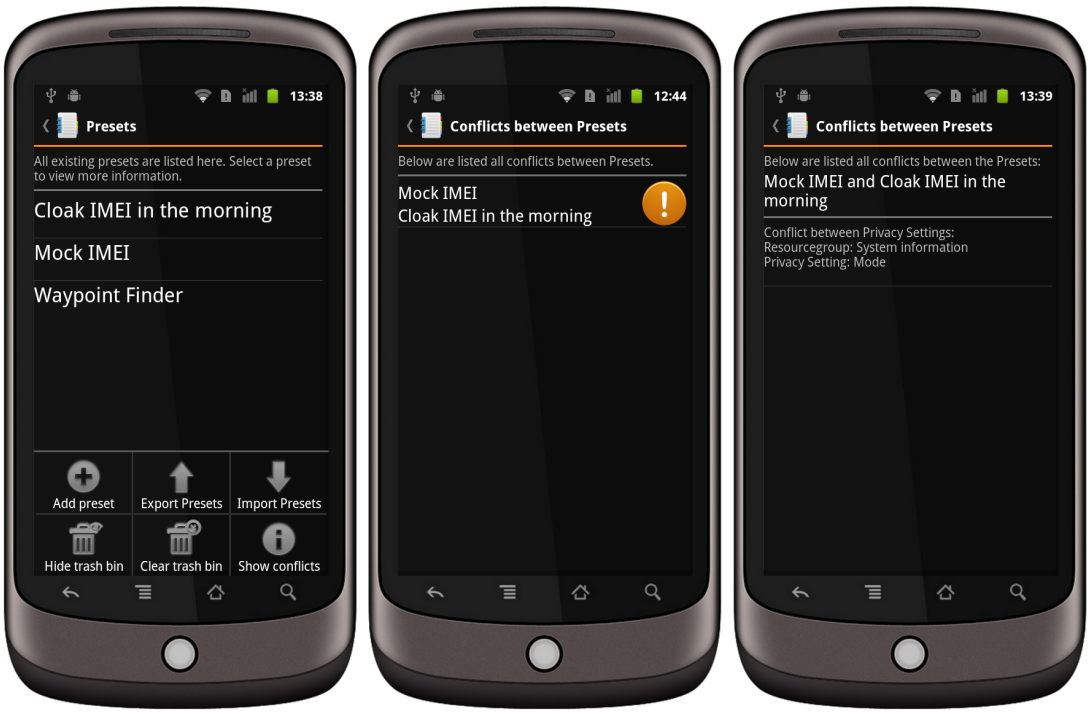

Abbildung 4.2.: Anzeige von Konflikten in Presets

Ressourcen In diesem Dialog sind alle installierten Ressourcen angezeigt. Der Benutzer kann sich zu jeder Ressource Informationen anzeigen lassen oder Ressourcen deinstallieren. Über den Reiter *verfügbar* können neue Ressourcen von einem Server heruntergeladen und installiert werden.

Presets Dieses Untermenü ist nur im Expertenmodus zugänglich. Hier sind alle existierenden Presets aufgelistet. Für ein Preset können zugehörige Privacy Settings angezeigt, hinzugefügt oder gelöscht werden. Über den Menü-Knopf des Geräts erscheinen weitere Optionen. So kann der Benutzer Presets importieren oder exportieren, Konflikte anzeigen oder ein neues Preset erstellen. Abbildung 4.2 zeigt weitere Optionen für Presets und die Erkennung von Konflikten.

Einstellungen Hier kann zwischen dem Einfachen Modus und dem Expertenmodus gewechselt werden. Die weiteren Optionen legen fest, welche Aktionen der PMP einen Log-Eintrag erzeugen.

Abbildung 4.3.: Das mittlere Bild zeigt das Hauptmenü der PMP. Durch die Auswahl der Schaltflächen *Apps*, *Ressourcen*, *Presets*, oder *Einstellungen* werden die außen angeordneten Benutzeroberflächen angezeigt.

5. Integrationsstrategien

Dieses Kapitel befasst sich mit verschiedenen Möglichkeiten für eine Integration der PMP in Android. Dazu wird zunächst die Architektur des Betriebssystems untersucht. Um eine geeignete Strategie zu ermitteln, werden anschließend verschiedene Ansätze zur Integration eines Berechtigungssystems analysiert und verglichen.

5.1. Android Architektur

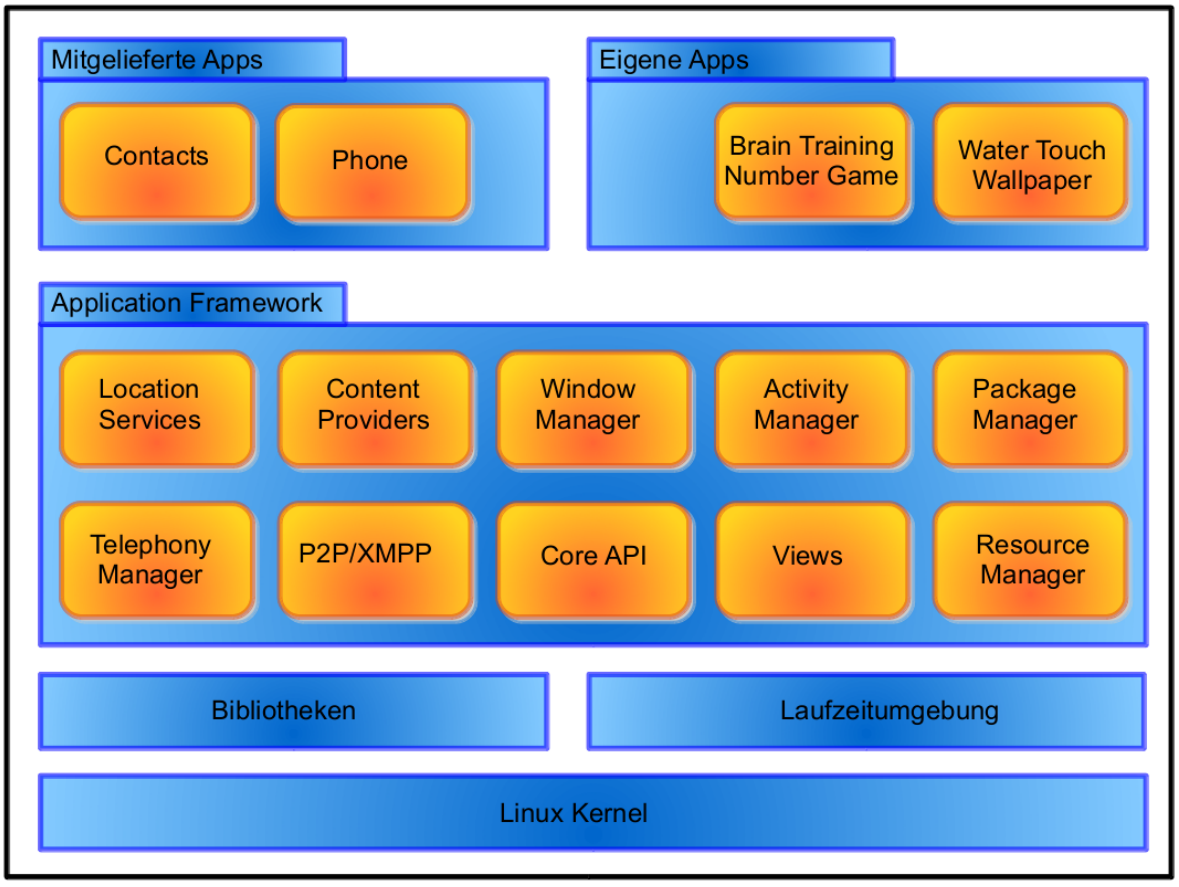

Abbildung 5.1.: Android Software Stack

Abbildung 5.1 zeigt die Architektur des Android Betriebssystems mit ihren wichtigsten Komponenten:

Linux Kernel Die Basis bildet eine abgewandelter Linux Kernel der Version 2.6. Er übernimmt grundsätzliche Aufgaben wie Prozessverwaltung oder Speicherverwaltung. Er dient mit den Gerätetreibern für Flash-Speicher, WLAN, Bildschirm oder Kamera als Schnittstelle zwischen Hardware und Software.

Bibliotheken Android liefert eine Sammlung von Bibliotheken, geschrieben in C/C++. Sie enthält unter anderem eine speziell angepassten C-Laufzeitumgebung, Unterstützung für die gängigsten Bild- und Tonformate, Unterstützung für 2D- und 3D Grafik und eine `SQLite` Datenbank.

Laufzweitumgebung Jede Android-Anwendung läuft in ihrer eigenen Instanz der *Dalvik Virtual Machine*. Sie basiert auf der quelloffenen Java-VM *Apache Harmony* und ist dahingehend optimiert, dass viele Prozesse mit geringem Speicherverbrauch parallel laufen können. Die Dalvik-VM verarbeitet kompilierten Java-Code, der mit dem Hilfsprogramm `dx` in das Format `.dex` übersetzt wird.

Application Framework Das *Application Framework* besteht aus Java-Klassen und bietet Entwicklern die Möglichkeit, Android um eigene Apps zu erweitern. Durch die `Views` können grafische Oberflächen entwickelt werden. Der `Resource Manager` erleichtert die Integration von Texten, Bildern und Layouts. Über einen `Content Provider` kann eine App Daten für andere Apps bereitstellen. Mit einer Vielzahl von System-Diensten wie `TelephonyManager` oder `LocationManager` übernimmt das Application Framework den Zugriff auf Gerätehardware. Die `Core API` enthält die Basisklassen für Android Apps wie `Activity`, `Application` oder `Context`. Ebenfalls im Framework enthalten sind die in Kapitel 2 untersuchten Mechanismen zum Laden, Speichern und Überprüfen von Berechtigungen.

Mitgelieferte Apps Mitgelieferte Apps wie *Browser*, *Kontakte* oder *Telefon* werden zusammen mit dem Android Betriebssystem gebaut und ausgeliefert. Sie liegen auf der System-Partition des Geräts und können daher vom Benutzer nicht deinstalliert werden.

Eigene Apps Unter diese Kategorie fallen alle nachträglich installierten Zusatzanwendungen. Sie liegen auf der Partition `userdata` und können auf ein externes Speichermedium verschoben oder gelöscht werden.

5.2. Integration

Die im Folgenden untersuchten Integrationsstrategien beziehen sich auf den Teil von Android, der in Java implementiert ist, also die Anwendungsschicht und das Application-Framework.

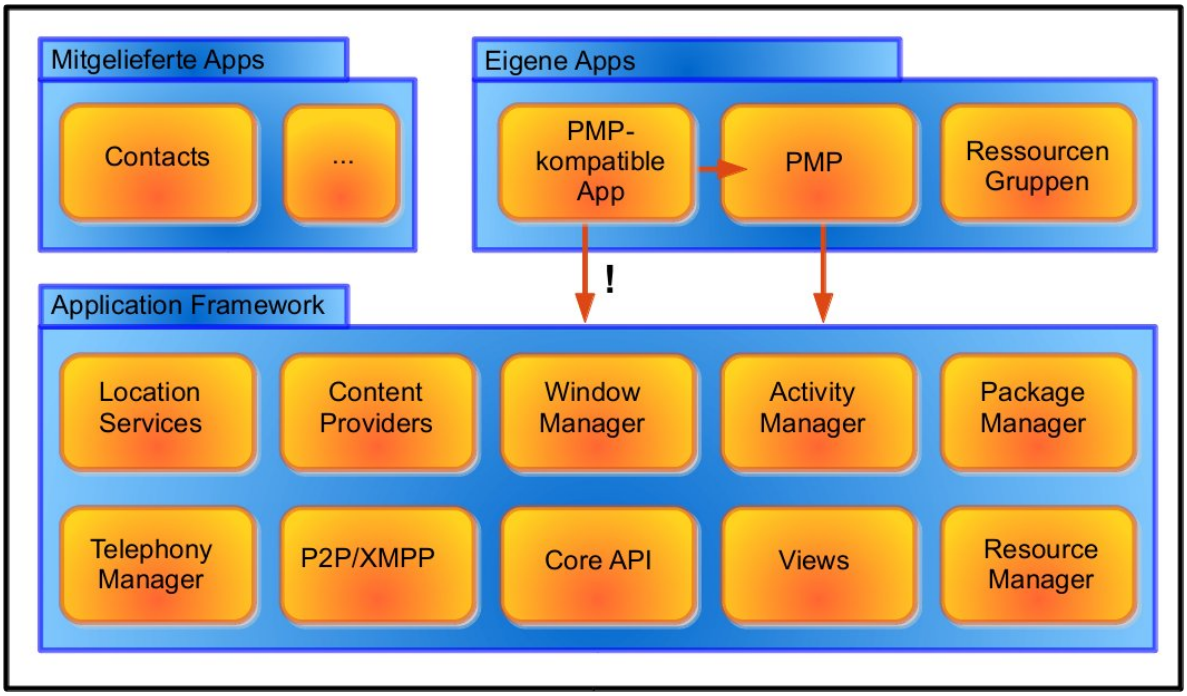

Abbildung 5.2.: PMP als eigenständige App. Der direkte Zugriff auf das Application Framework kann nicht kontrolliert werden.

5.2.1. Berechtigungssystem als App

Um herauszufinden, welche Vorteile eine Integration bringt, wird zunächst der aktuelle Stand untersucht. Die PMP wurde als installierbare App entwickelt. Die Ressourcengruppen, die ebenfalls als Apps vorliegen, implementieren die Zugriffe auf Daten und Funktionen des Geräts. Zugriffe auf das Gerät können also nur für solche Apps gesteuert werden, die sich bei der PMP registrieren und deren Ressourcengruppen benutzen. Die Ressourcengruppen führen selbst keine Zugriffe durch. Sie stellen der PMP Implementierungen für Zugriffe auf das Application Framework zur Verfügung.

Das Android Betriebssystem wird nicht verändert. Die PMP als App lässt sich also auf jedem Gerät mit Android 2.1 oder höher installieren. Das bedeutet gleichzeitig, dass das System der Permissions nicht verändert wird. Apps können weiterhin Permissions in der Datei `AndroidManifest.xml` deklarieren und auf geschützte Telefonfunktionen zugreifen. Dies ist sogar für Apps möglich, die bei der PMP registriert sind.

Bewertung

Die PMP benötigt zwar kein modifiziertes Betriebssystem, bietet aber nur ein geringes Maß an Sicherheit, da weiterhin direkte Zugriffe auf das Application Framework möglich sind.

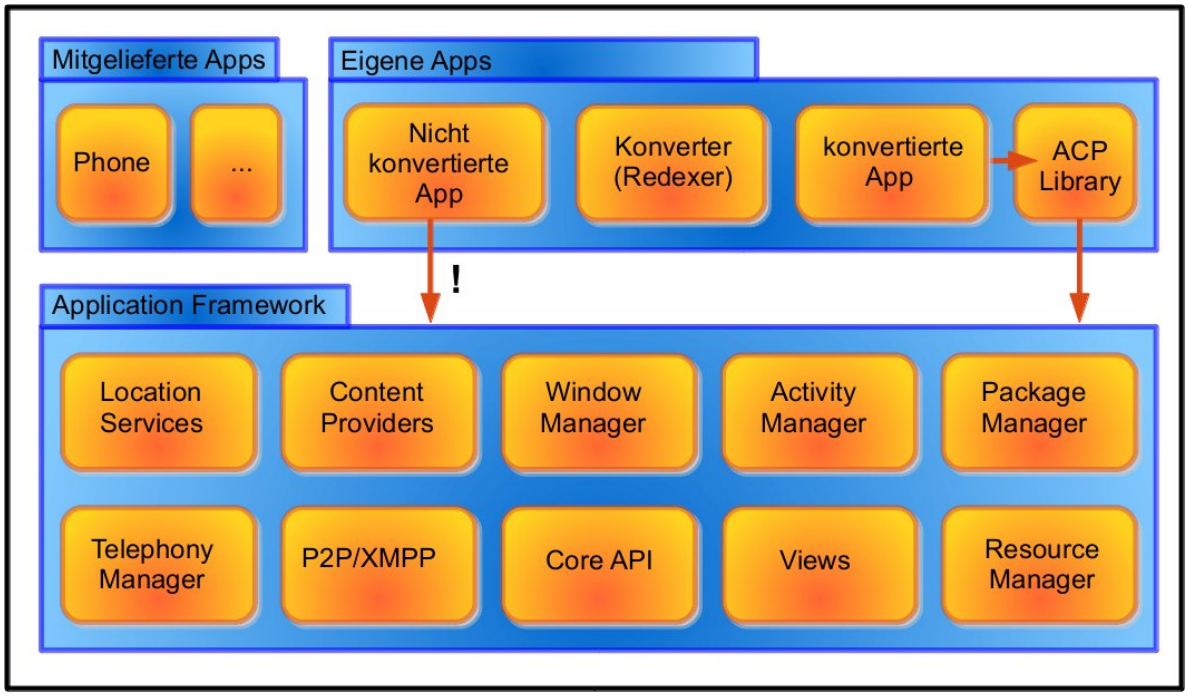

Abbildung 5.3.: Konvertierte Apps benutzen die ACP Library, um auf das Application Framework zuzugreifen. Für unkonvertierte Apps sind direkte Zugriffe möglich.

5.2.2. Konverter

In [RJV+11] wird ein Ansatz beschrieben, der ebenfalls ohne Modifikation des Betriebssystems auskommt. Das Konzept besteht darin, die ressourcenorientierten Android Permissions um die in Tabelle 5.1 aufgelisteten, anwendungsorientierten Berechtigungen zu erweitern.

Dazu wurden zwei Komponenten entwickelt:

1. Die **ACPlib** implementiert die Dienste für die Berechtigungen `InternetURL`, `ScanBarcode` und `LocationBlock`. Sie können nur von Apps angefordert werden, die eine entsprechende Berechtigung haben.
2. Der **Redexer** kann Paketdateien von Android Apps einlesen und Aufrufe zum Application Framework so modifizieren, dass stattdessen die entsprechenden Dienste in der ACPlib benutzt werden.

Wie Abbildung 5.3 zeigt, werden diese Komponenten als Apps installiert. Der Benutzer muss den Konverter (Redexer) auf jede App anwenden, die durch die neuen Berechtigungen geschützt sein soll. Das Betriebssystem wird bei diesem Ansatz nicht verändert. Apps, die nicht konvertiert wurden, benutzen weiterhin die Ressourcenorientierten Permissions von Android, können also nicht kontrolliert werden.

Konverter für die PMP

Die Anwendung dieses Konzepts auf die PMP hätte den Vorteil, dass Berechtigungen für beliebige Apps durch die PMP gesteuert werden könnten. Ein Konverter, der jede App kompatibel zur PMP macht, könnte wie folgt vorgehen:

1. Der Konverter bekommt als Eingabe eine Paketdatei.
2. Darin wird nach Aufrufen von Funktionen aus dem Application Framework gesucht, die eine Berechtigung erfordern.
3. Für jeden Aufruf hat der Konverter Zugang zu einem ausformulierten Service Feature. Darin sind auch die benötigten Privacy Settings und Ressourcengruppen enthalten.
4. Das App Information Set (`/assets/ais.xml`) wird aus den passenden Service Features zusammengestellt.
5. Fehlende Ressourcengruppen werden nachinstalliert.
6. Jeder gefundene Aufruf wird durch `getResource(...)` ersetzt. Es muss die benötigte Ressource übergeben werden, sowie eine überschriebene Version der Methode `onReceiveResource()`
7. Das Ergebnispaket wird erstellt.

Die größte Herausforderung liegt in Schritt 6. Das Anfordern einer Ressource der PMP bewirkt die Erstellung eines neuen Threads. Das Ergebnis der Funktion, die in `onReceiveResource()` aufgerufen wird, kann nicht direkt zugewiesen werden, sondern muss an einer anderen Stelle zwischengespeichert werden. Außerdem ist eine Ressource der PMP nicht sofort nach einer Anfrage verfügbar. Beim Einlesen des Zwischenspeichers muss also geprüft werden, ob die gewünschten Daten schon existieren.

Permission	Details
AdsPrivate	Werbung wird angezeigt, ohne persönliche Daten zu versenden
AdsGeo	Werbung wird angezeigt, nur Standortdaten dürfen gesendet werden
AnonUsage	Anonyme Nutzerstatistiken senden
InternetURL	Verbindung zu einer Webseite und deren Teilbereiche erlauben
LocationBlock	Ortung auf 150m genau zulassen
LocationVisible	Genaue Ortung, solange die Anwendung im Vordergrund ist
MobileBilling	Bezahlvorgänge über die Rechnung des Netzbetreibers zulassen
ScanBarcodes	Kamera darf für Strichcodes und QR-Codes verwendet werden
ToggleGPS	GPS aktivieren oder deaktivieren
SDCardOwnFiles	Zugriff auf eigene Dateien der App erlauben
SDCardSharedFiles	Zugriff auf gemeinsame Dateien wie Fotos oder Musik erlauben

Tabelle 5.1.: Anwendungsorientierte Berechtigungen und deren Bedeutung

Bewertung

Interessant an diesem Ansatz ist, dass auf einem unmodifizierten Betriebssystem alle Apps eine neue Berechtigungsprüfung nutzen können. Der Benutzer muss allerdings für jede App den Konverter ausführen und sie danach neu installieren. Apps, die nicht konvertiert wurden, haben weiterhin direkten Zugriff auf Telefonressourcen. Es kann also keine absolute Sicherheit garantiert werden. Weiterhin ist der Aufwand, einen Konverter für die PMP zu schreiben, kaum abzuschätzen, dementsprechend lässt sich auch schwer eine Aussage über die Erweiterbarkeit treffen.

5.2.3. Ersatz für das Berechtigungssystem

Damit die PMP das bestehende Berechtigungssystem ersetzen kann, muss zunächst dafür gesorgt werden, dass sie nicht deinstalliert werden kann. Dazu wird die PMP in die mitgelieferten Apps aufgenommen, wie in Abbildung 5.4 veranschaulicht wird. Der Quellcode der PMP wird an der richtigen Stelle im Android Quellcode abgelegt, so dass ein Betriebssystem mit vorinstallierter PMP erzeugt wird.

Außerdem muss sichergestellt sein, dass die PMP nicht umgangen werden kann. Direkte Zugriffe auf geschützte Funktionen im Application Framework müssen verhindert werden. Apps dürfen beim Zugriff auf geschützte Funktionen also keine Android Permissions mehr haben. Dazu gibt es verschiedene Ansätze:

Vorhandene Permissions löschen In Abschnitt 2.6 wird beschrieben, wo Permissions für Apps persistent gespeichert werden. Aus der Datei `/system/etc/permissions/packages.xml` kann ermittelt werden, welche Apps nicht zu den Mitgelieferten gehören. Für solche Apps werden Einträge der Form `<perms><item name="..."/></perms>` gelöscht. Dieser Vorgang wird jedesmal ausgeführt, sobald die Installation einer neuen App beendet wurde. Ein großer Nachteil ist, dass die Änderung erst nach einem Neustart des Geräts wirksam wird. Ein Neustart direkt nach der Installation ist vor allem bei der Entwicklung und beim Test neuer Apps hinderlich.

Permissions nicht erteilen Wie in Abschnitt 2.5 gezeigt, wird bei der Installation einer App deren Manifest-Datei ausgelesen. Die dort deklarierten Permissions und andere Paketinformationen werden in der Datei `/system/etc/permissions/packages.xml` abgelegt. Die Methode `writeLP()` kann so modifiziert werden, dass die gelesenen Permissions nicht abgespeichert werden. Da `writeLP()` auch für die Initialisierung mitgelieferter Apps benutzt wird, muss auch bei diesem Ansatz geprüft werden, ob die gelesene App zu den nachinstallierten Apps gehört. Mitgelieferte Apps, die ihre Permissions verlieren, würden bewirken, dass das Betriebssystem nahezu unbenutzbar wird. Jede App würde beim Starten eine `SecurityException` auslösen.

Vorhandene Permissions ignorieren Abschnitt 2.4 beschreibt den Prozess der Berechtigungsprüfung in Android. Um die Prüfung zu modifizieren, kann die Methode `checkPermission(...)` aus der Implementierung der Schnittstelle `Context` verändert werden. Anstatt über den `ActivityManager` herauszufinden, ob eine App über eine Permission verfügt, kann geprüft werden, ob es sich bei der App um eine nachinstallierte App handelt. Ist dies der Fall, kann ohne weitere Prüfung die Konstante `PERMISSION_DENIED` zurückgeliefert werden.

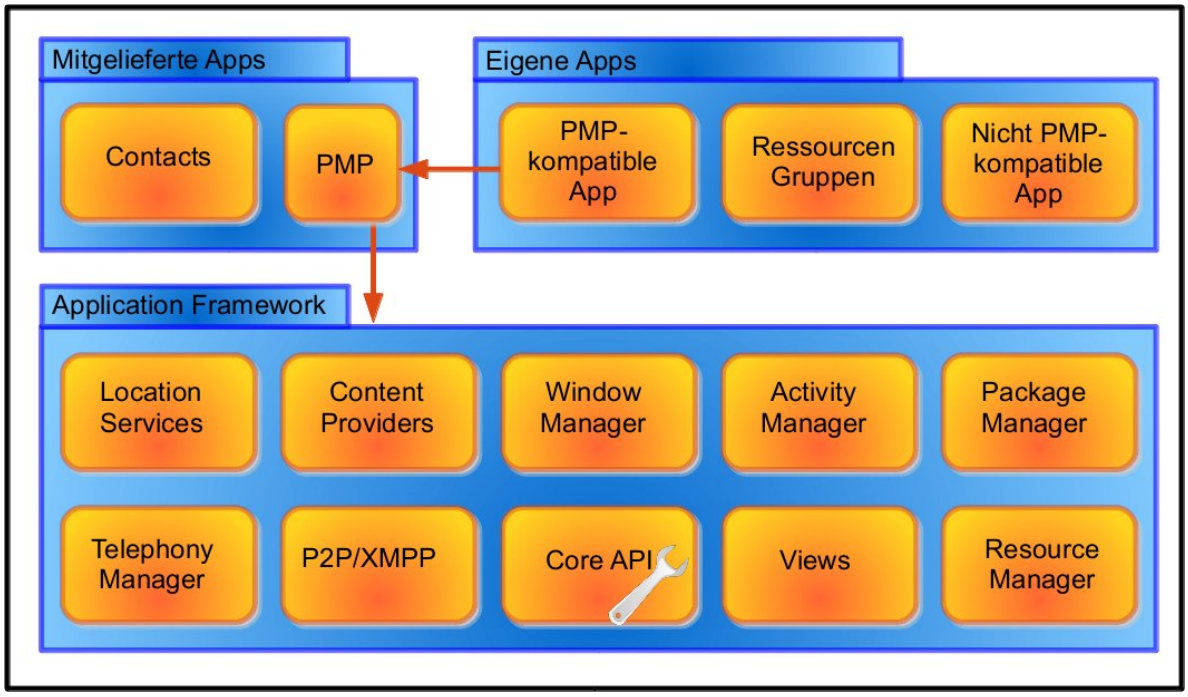

Abbildung 5.4.: Die PMP als mitgelieferte App. Der Direkte Zugriff auf das Application Framework ist durch Modifikation der API gesperrt.

Bewertung

Dieser Ansatz ist insofern einzigartig, dass er garantieren kann, dass keine App auf Telefonfunktionen zugreifen kann, solange die PMP dies nicht erlaubt.

5.2.4. Berechtigungssystem parallel zum bestehenden

Die Berechtigungen der zur PMP kompatiblen Apps sollen durch die PMP gesteuert werden, während für die Apps, die nicht zur PMP kompatibel sind, der in Android eingebaute Mechanismus greifen soll. Wie in Ansatz 5.2.3 wird auch hier die PMP in den Android Quellcode integriert und als mitgelieferte App gebaut (Abbildung 5.5). Ebenfalls müssen

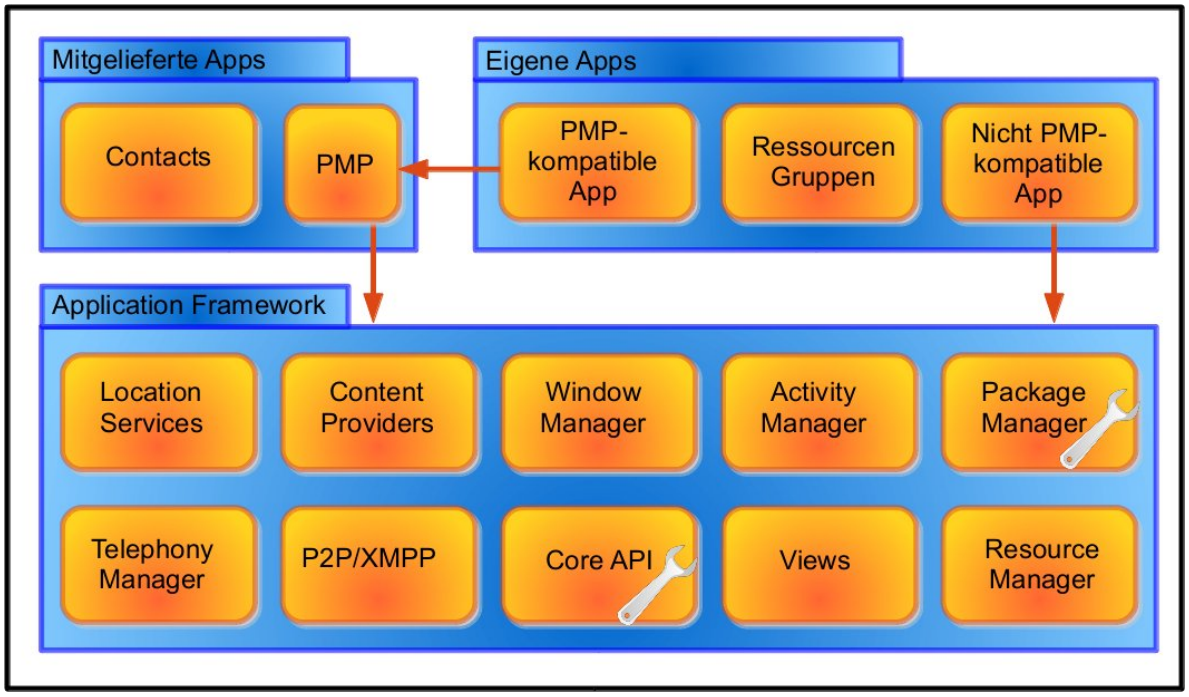

Abbildung 5.5.: Die PMP als mitgelieferte App. Der Direkte Zugriff ist für PMP-Apps gesperrt. Für andere Apps greift der Prüfmechanismus von Android.

Apps ihre Android Permissions verlieren. Der Unterschied ist, dass die Permissions nur den App entzogen werden dürfen, die zur PMP kompatibel sind. Es muss also unterschieden werden, ob eine App zur PMP kompatibel ist oder nicht.

Während der Installation einer App wird die Methode `parsePackage(..)` aufgerufen. In dieser Methode wird jeder gefundene XML-Tag der Manifest-Datei ausgewertet und anschließend ein Objekt der Klasse Package zurückgegeben, das alle relevanten Informationen enthält. Wird hier ein Aufruf der der `RegistrationActivity` der PMP gefunden, handelt es sich um eine App, die zur PMP kompatibel ist. In diesem Fall kann die Liste `requestedPermissions` des `Package`-Objekts geleert werden. Die Folge ist, dass für diese App keine Android Permissions abgespeichert werden. Dieser App wird also der direkte Zugriff auf geschützte Funktionen verweigert.

Bewertung

Diese Hybrid-Lösung würde die Kompatibilität zu Apps aus dem Google Play Store garantieren. Da diese Apps weiterhin dem Berechtigungssystem von Android unterliegen, sind die Unterschiede zur PMP als App minimal. Für Apps, die zur PMP kompatibel sind wäre der direkte Zugriff auf das Application Framework gesperrt, jedoch wäre es ohnehin widersprüchlich, eine App für die PMP zu entwickeln, die die PMP umgeht.

5.3. Zusammenfassung

Die Tabelle 5.2 fasst die Bewertungen der einzelnen Strategien für die Integration eines Berechtigungssystems in Android zusammen. Die Kriterien sind:

Kompatibel zu Standard Android Kann die Lösung angewendet werden, ohne ein modifiziertes Betriebssystem zu benutzen, also ohne ein Gerät mit Administratorrechten zu verwenden?

Kompatibel zu Standard Apps Arbeiten Standard-Apps, wie sie aus dem Google Play Store zu beziehen sind, mit dem neuen Berechtigungssystem zusammen?

Umfassende Kontrolle Werden alle installierten Apps von dem neuen Berechtigungssystem erfasst?

Sicherheit (kompatible Apps) Kann Sicherheit für Apps garantiert werden, die mit dem neuen Berechtigungssystem zusammenarbeiten?

Sicherheit (kompatible Apps) Kann Sicherheit für Apps garantiert werden, die nicht mit dem neuen Berechtigungssystem zusammenarbeiten?

Kriterium	als App	Konverter	Als Ersatz	Parallel
Kompatibel zu Standard Android	ja	ja	nein	nein
Kompatibel zu Standard Apps	nein	ja	nein	ja
Umfassende Kontrolle	nein	ja	ja	nein
Sicherheit (kompatible Apps)	ja	ja	ja	ja
Sicherheit (Standard Apps)	nein	nein	ja	nein

Tabelle 5.2.: Bewertung der Integrationsstrategien

Da nur maximale Sicherheit garantiert werden kann, wenn die PMP als Ersatz für das bestehende Berechtigungssystem dient, wird dieser Ansatz implementiert.

6. Integration der PMP

Die Dienste der PMP sollen ein vollwertiger Ersatz für Permissions in Android sein. Es muss also dafür gesorgt werden, das die PMP nicht umgangen werden kann. Zum einen darf es nicht möglich sein, die PMP zu deinstallieren, zum anderen sollen Zugriffe auf Funktionen und Daten des Telefons nur über die PMP erfolgen. Mit der PMP als nachträglich installierte App ist das nicht realisierbar. Es ist also notwendig, die PMP ins Betriebssystem zu integrieren. Das Betriebssystem selbst muss so modifiziert werden, dass der direkte Zugriff auf geschützte Android-Funktionen für nachträglich installierte Apps verweigert wird.

6.1. Integration des PMP-Kernsystems

Die PMP wird so ins Betriebssystem integriert, dass sie beim Kompilieren des Android Quellcodes mit gebaut wird. Im resultierenden Android-Image ist sie fester Bestandteil des Betriebssystems wie die mitgelieferten Apps *Browser* oder `Phone`.

6.1.1. Der Android Quellcode

Android ist ein quelloffenes Projekt. Mit dem Programm `repo`, eine Erweiterung der Versionsveraltungssoftware `git`, kann jede Android-Version heruntergeladen werden. Die PMP benötigt mindestens Android 2.1. Die jüngste und am meisten verbreitete Version von Android aus der Reihe 2.x ist die Version 2.3.7. Diese Version wird daher für die Integration der PMP verwendet. Mit den Befehlen

```
repo init -u git://android.git.kernel.org/platform/manifest.git -b android-2.3.7_r1
repo sync
```

wird der komplette Quellcode von Android 2.3.7 heruntergeladen. Nicht im Quellcode enthalten sind gerätespezifische proprietäre Treiber für Hardware, wie Grafikprozessor, Kamera oder WLAN. Mit kleinen Anpassungen lässt sich auch ohne diese Treiber ein lauffähiges Betriebssystem bauen. Dieses ist jedoch kaum brauchbar, da es einige Grafikfehler aufweist, die Kamera nicht funktioniert und die Bedienoberfläche sehr langsam reagiert.

Der Android Quellcode stellt ein Skript bereit, um die erforderlichen proprietären Treiber von einem Handy zu laden, auf dem ein Betriebssystem mit diesen Treibern läuft.

Listing 6.1 Verzeichnisstruktur des Android Quellcodes (Auszug)

```
/bionic              Quellen für die C-Laufzeitumgebung. Wie 'glibc' auf Linux-Systemen
/bootable            Dateien für den Startvorgang, z.B. das Fastboot-Protokoll
/build               Implementierung des Build-Systems, wichtige Makefiles
/cts                 Kompatibilitätstests
/dalvik              Implementierung der 'Dalvik Virtual Machine'
/development         Integration in Entwicklungsplattformen wie Eclipse
/device              Gerätespezifische Makefiles
/external            Externe quelloffene Projekte wie 'SQLite'
/frameworks/base     Das Application Framework
  /core/java         Zentrale Klassen wie 'Context', 'Application' und 'Activity'
  /services/java     Wichtige Services wie 'PackageManager' und 'SystemService'
/hardware            Hardware Abstraction Layer, quelloffene Teiber z.B. für GPS oder WLAN
/ndk                 Das 'Native Development Kit' zum einbinden von C-Quellcode in Apps
/out                 Ausgabe-Verzeichnis
  /target/product    Gerätespezifische Betriebssystem-Images
    /passion         'boot.img' und 'system.img' für das Nexus One.
/packages            Mitgelieferte Pakete
  /apps              Quellcode und Makefiles der mitgelieferten Apps.
/prebuild            Binärdateien für verschiedene Betriebssysteme
/sdk                 Das Android 'Software Development Kit'
/system              Das stark verkleinerte Linux-System. Basis für die 'Dalvik VM'.
/vendor              Gerätespezifische Konfigurationen und proprietäre Treiber
```

Um den Android Quellcode bearbeiten zu können, muss dessen Struktur näher untersucht werden. In Listing 6.1 wird der Inhalt der wichtigsten Verzeichnisse erklärt.

6.1.2. Integration der PMP-Dateien

Die PMP wurde als installierbare App entwickelt und soll so integriert werden, dass sie im erzeugten Android Image als mitgelieferte App bereitsteht. Fester Bestandteil der PMP ist die PMP API. Sie wurde als separates Projekt entwickelt, das in Eclipse als Bibliothek eingebunden werden kann. Wie jedes Android-Projekt besitzt sie eigene Ressourcen für die Texte in verschiedenen Sprachen, das Layout der Benutzeroberfläche und die darin verwendeten Grafiken. In Eclipse können Projekte auf Ressourcen eingebundener Bibliotheken zugreifen. Im Android Quellcode werden aus den Projekten `PMP` und `PMP-API` auch zwei getrennte Apps. Es ist möglich aus der PMP API eine Java-Klassenbibliothek (`pmp-api.jar`) zu bauen, die in den Klassenpfad der PMP aufgenommen wird. Diese enthält jedoch keine Ressourcen-Dateien. Auch wenn sie manuell zur `pmp-api.jar` hinzugefügt werden, sind Ressourcen der PMP API von der PMP aus unerreichbar. Die Projekte `PMP` und `PMP-API` müssen also zusammengeführt werden. Da Java im Gegensatz zu anderen Programmiersprachen wie C# keine partiellen Deklarationen von Klassen zulässt, können die zwei Verzeichnisbäume in `/src` ohne Anpassungen zusammengeführt werden. Die Ressourcen in `/res` haben teilweise identische Dateinamen. In solchen Fällen wird der Inhalt der Dateien vereinigt.

Das Projekt `PMP` enthält nun die Klassen und Ressourcen der PMP API und kann in den Android Quellcode integriert werden. Wie in Listing 6.1 gezeigt, werden Quelldateien für

mitgelieferte Apps in `/packages/apps` abgelegt. Um das Projekt in das Build-System von Android einzugliedern, muss in dem neuen Verzeichnis `/packages/apps/PMP` die Datei `Android.mk` liegen. Dort wird das Verzeichnis der lokalen Quelldateien angegeben sowie und alle Java-Klassenbibliotheken deklariert, die nötig sind, um die PMP zu bauen.

In der Datei `/build/target/product/core.mk` wird festgelegt, welche Pakete aus `/packages/apps` bei der Erzeugung eines Android Images mit einbezogen werden sollen. Zu der Variable `PRODUCT_PACKAGES` muss ein Eintrag für die PMP hinzugefügt werden.

6.1.3. Konfiguration von ProGuard

Als letzter Schritt vor dem Bauen muss *ProGuard* konfiguriert werden. ProGuard ist ein Hilfsprogramm, das Java-Quellcode schrittweise optimiert.

1. **SHRINK**: Ungenutzte Klassen, Variablen, Methoden und Attribute werden erkannt und entfernt.
2. **OPTIMIZE**: Der `Bytecode` der Methoden wird analysiert und optimiert.
3. **OBFUSCATE**: Bezeichner werden durch kurze, unlesbare Zeichenketten ersetzt.
4. **PREVERIFY**: Der Code wird auf Kompatibilität mit Java ME geprüft.

Durch die Optimierung mit ProGuard werden Android Apps kleiner, schneller und schwer zu analysieren. Beim Entwickeln von Apps mit dem Android SDK muss ProGuard explizit eingeschaltet werden. Im Build-System von Android ist ProGuard integriert und wird für jede mitgelieferte App ausgeführt, sofern das Android-Image nicht im Debug-Modus gebaut wird. Die voreingestellte Konfiguration für ProGuard liegt in `/build/core/proguard.flags`.

Durch das Entfernen von Klassen durch ProGuard kann es passieren, dass eine App nach der Optimierung nicht mehr lauffähig ist. Daher kann für jede mitgelieferte App eine eigene Konfiguration für ProGuard erstellt werden. Im Fall der PMP wurden durch das Hinzufügen der Optionen `-dontshrink` der Optimierungsschritt `1` übersprungen.

6.1.4. Erzeugen der Image-Dateien

Mit diesen Vorbereitungen kann der Quellcode kann in ein Android Image mit integrierter PMP übersetzt werden. Die Befehle

```
. build/envsetup.sh
lunch full_passion-user
make -j4
```

erzeugen die erforderlichen Image-Dateien für das *HTC Nexus One*. Über das Programm `fastboot` aus dem Android SDK lassen sich die Dateien `boot.img` und `system.img` auf die Partitionen `boot` und `system` des Geräts übertragen.

6.2. Integration der Ressourcengruppen der PMP

Bis jetzt sind Ressourcengruppen als eigenständige Apps implementiert. Sie könnten in den Android Quellcode integriert werden, doch es müssen zunächst die Folgen betrachtet werden:

Ressourcengruppen dynamisch nachinstallieren Die PMP kann benötigte Ressourcengruppen bei Bedarf von einem angegebenen Server herunterladen und nachinstallieren. Dieses Feature könnte ohne Modifikation der PMP nicht übernommen werden.

Ressourcengruppen deinstallieren Die PMP kann benötigte Ressourcengruppen deinstallieren. Um dies für integrierte Ressourcengruppen zu ermöglichen, müsste die PMP ebenfalls modifiziert werden.

Entwicklung neuer Ressourcen Neue Ressourcengruppen können leichter entwickelt werden, wenn sie als Apps behandelt werden. Eine tiefere Integration hätte zur Folge, dass entweder die PMP modifiziert werden muss oder die Ressourcengruppen mit der PMP und dem Android Quellcode zusammen gebaut werden. Im letzteren Fall müsste jeder Entwickler von Ressourcengruppen den kompletten Android Quellcode haben und bei jedem Test der Ressourcengruppe bauen.

Eine mögliche Modifikation könnte so aussehen, dass Ressourcengruppen weiterhin als Apps installiert und bei der PMP registriert werden. Danach werden sie von der PMP innerhalb des Android-Dateisystems verschoben. Dies wäre jedoch ein erheblicher Mehraufwand, der keine Vorteile mit sich bringt. Aus diesem Grund werden die Ressourcengruppen für die PMP weiterhin als eigenständige Apps behandelt.

6.3. Berechtigungsprüfung nur durch die PMP

Die Dienste der PMP sollen das bisherige Berechtigungssystem von Android ersetzen. Es darf nicht möglich sein, Telefondaten, die durch Android Permissions geschützt sind, direkt anzufordern. Dazu wird der Mechanismus zur Überprüfung der Berechtigungen von Apps verändert. In Abschnitt 2.4 wurde die Berechtigungsprüfung für Apps beschrieben. Um diesen Prozess zu verändern, soll die Methode `checkPermission(...)` so modifiziert werden, dass nachträglich installierten Apps grundsätzlich keine Berechtigungen gewährt werden. Listing 6.2 zeigt die Methode nach den vorgenommenen Änderungen.

Bei jedem Aufruf wird ermittelt, welche Pakete der übergebenen UID zugeordnet sind. In fast allen Fällen handelt es sich um ein einzelnes Paket. Sollte der seltene Fall eintreten, dass mehrere Pakete einer UID zugeordnet sind, gibt es zwei Möglichkeiten:

1. Beide Pakete sind mitgelieferte System-Apps und liegen in `/system/app`
2. Beide Pakete sind nachinstallierte Apps und liegen in `/data/app`

Listing 6.2 Modifikation in `/frameworks/base/core/java/android/app/ContextImpl.java`. Die Methode `checkPermission(...)` verweigert nachinstallierten Apps alle Berechtigungen

```
@Override
public int checkPermission(String permission, int pid, int uid) {
    if (permission == null) {
        throw new IllegalArgumentException("permission is null");
    }

    if (!Process.supportsProcesses()) {
        return PackageManager.PERMISSION_GRANTED;
    }

    // Deny all permissions from apps in /data/app
    PackageManager mPM = this.getPackageManager();
    String[] packageNames = mPM.getPackagesForUid(uid);
    try {
        String currentPackageName = packageNames[0];
        String installLocation = mPM.getApplicationInfo(currentPackageName, 0).sourceDir;
        if (installLocation.startsWith("/data/app")) {
            return PackageManager.PERMISSION_DENIED;
        }
    } catch (NameNotFoundException e) {
        // Just in case. An UID should always have an associated package.
        return PackageManager.PERMISSION_DENIED;
    } catch (NullPointerException e) {
        // While installing a package, its name doesn't yet exist
        return PackageManager.PERMISSION_GRANTED;
    }

    // Treat apps in /system/apps normally
    try {
        int checkResult = ActivityManagerNative.getDefault().checkPermission(
                permission, pid, uid);
        return checkResult;
    } catch (RemoteException e) {
        return PackageManager.PERMISSION_DENIED;
    }

}
```

Eine nachinstallierte App kann sich mit einer System-App keine UID teilen. Es spielt keine Rolle, über welches der Pakete das Installationsverzeichnis ermittelt wird. Daher wird das erste Paket ausgewählt, das von der Methode `getPackagesForUid(int uid)` geliefert wird. Ist das Paket in `/data/app` abgelegt, handelt es sich um eine nachträglich installierte App. In diesem Fall wird ohne weitere Prüfung die Konstante `PackageManager.PERMISSION_DENIED` zurückgeliefert.

Mit dieser Modifikation kann aus dem Android Quellcode erneut eine Image-Datei erzeugt werden. Ein Gerät, auf dem dieses Image installiert ist, verweigert allen nachinstallierten Apps den Zugriff auf geschützte Funktionen, es sei denn der Zugriff erfolgt über die PMP.

7. Demonstrator

Dieses Kapitel stellt zwei Apps vor, die entwickelt wurden, um die Funktionalität der integrierten PMP zu demonstrieren. Zusätzlich wurden zwei Ressourcengruppen geschrieben, so dass die PMP die von diesen Apps benötigten Funktionen bereitstellen kann.

7.1. Auslesen der IMEI

Durch die *International Mobile Equipment Identity* (IMEI) kann ein Mobiltelefon eindeutig identifiziert werden. Die 15-stellige Seriennummer kann im Fall eines Diebstahls dazu dienen, das Gerät sperren zu lassen, sofern der Netzbetreiber diesen Dienst anbietet. Die IMEI ist ein beliebtes Ziel für Spionage-Apps. Besonders seit bekannt wurde, dass der weit verbreitete Nachrichtendienst *WhatsApp* die Telefonnummer als Benutzername und die IMEI als Passwort für die Anmeldung verwendet.

7.1.1. Funktionsweise der App *Information Read*

Die als Demonstrator entwickelte App *Information Read* liest die IMEI auf zwei Arten aus:

- Durch Anfrage bei der PMP
- Über den `TelephonyManager` im Android Application Framework

Die App zeigt, dass die PMP bestimmen kann, ob der Zugriff auf die IMEI erfolgen darf oder nicht. Läuft die PMP im Expertenmodus, lässt sich konfigurieren, ob die korrekte IMEI vom Telefon gelesen wird oder ob die PMP einen definierten Wert zurückliefert. Die Implementierung `Mock` liefert hier den Wert „`111111111111111`“.

Die App soll weiterhin zeigen, dass es durch die in Kapitel 6 vorgestellte Modifikation der Berechtigungsprüfung für nachinstallierte Apps nicht möglich ist, Telefondaten zu erlangen, die durch Permissions geschützt sind. Um die Methode `TelephonyManager.getDeviceId()` aufzurufen, forciert das Betriebssystem die Permission `READ_PHONE_STATE`. Obwohl sie in der Manifest-Datei von *Information Read* angegeben wurde, schlägt der Zugriff fehl. Bei unautorisierten Zugriffen wird vom Betriebssystem eine `SecurityException` ausgelöst. Um einen Absturz der App zu vermeiden, wurde der Fehler abgefangen und im Textfeld mit der Überschrift „IMEI from Android“ ein entsprechender Hinweis angezeigt.

Listing 7.1 `assets/rgis.xml` (Auszug)

```
<?xml version="1.0" encoding="UTF-8"?>
<resourceGroupInformationSet>
   <resourceGroupInformation
       identifier="de.unistuttgart.ipvs.pmp.resourcegroups.systeminformation"
       icon="res/drawable/icon.png" className="SystemInformationResourceGroup">
      <name lang="en">System information</name>
      <name lang="de">Systeminformationen</name>
      <description lang="en">Provides access to the system information functions from
          TelephonyManager.</description>
      <description lang="de">Bietet Zugriff auf Funktionen des TelephonyManager.</description>
   </resourceGroupInformation>
   <privacySettings>
      <privacySetting identifier="getDeviceId" validValueDescription="'true', 'false'"
          requestable="true">
         <name lang="en">Access to the device id</name>
         <name lang="de">Zugriff auf die IMEI</name>
         <description lang="en">Permits access to the device id.</description>
         <description lang="de">Genehmigt den Zugriff auf die IMEI.</description>
         <changeDescription lang="en">By this Privacy Setting an App gets access to the
             device id.</changeDescription>
         <changeDescription lang="de">Durch dieses Privacy Setting wird einer App der Zugriff
             auf die IMEI erlaubt.</changeDescription>
      </privacySetting>
   </privacySettings>
   // ... more Privacy Settings here ...
</resourceGroupInformationSet>
```

7.1.2. Benötigte Ressourcengruppe

Wie in Kapitel 4.3.1 beschrieben, braucht die PMP Ressourcengruppen, deren Ressourcen die von einer App angeforderten Funktionen implementieren. Für die App *Information Read* wurde die Ressoucengruppe *System Information* erstellt. Die folgenden Schritte waren nötig:

Definition von Privacy Settings

Einer Ressourcengruppe sind Privacy Settings zugeordnet. Um sie zu definieren, wird ein `ResourceGroupInformationSet` (RGIS) angelegt, also eine Datei `rgis.xml`, die im Verzeichnis `/assets` liegen muss. Die im RGIS enthaltenen Erklärungen zur Ressourcengruppe und zu den Privacy Settings können in mehreren Sprachen abgelegt werden. Eine Angabe in Englisch ist Pflicht. Listing 7.1 zeigt das RGIS der App *Information Read* in verkürzter Form.

Erstellen der Klassen für die Ressourcengruppe und die Ressourcen

Die PMP bietet die Basisklasse `de.unistuttgart.ipvs.pmp.resource.ResourceGroup`, die abgeleitet werden kann, um eine Ressourcengruppe zu erzeugen. Zur Identifikation wird der String `PACKAGE_NAME` definiert. Um die in `rgis.xml` definierten Privacy Settings der

Listing 7.2 `IfSystemInformation.aidl`

```
package de.unistuttgart.ipvs.pmp.resourcegroups.systeminformation.interfaces;
interface IfSystemInformation {
    String getDeviceId();
    String getDeviceSoftwareVersion();
    String getNetworkCountryIso();
    String getNetworkOperator();
    String getNetworkOperatorName();
    String getSimCountryIso();
    String getSimOperator();
    String getSimOperatorName();
    String getSimSerialNumber();
    int getSimState();
    String getSubscriberId();
    String getVoiceMailAlphaTag();
    String getVoiceMailNumber();
    boolean isNetworkRoaming();
}
```

Ressourcengruppe hinzuzufügen sowie deren Typ zu bestimmen (Abschnitt 4.3.2), wird die Methode `registerPrivacySetting()` aufgerufen.

Eine Ressourcengruppe muss mindestens eine Ressource beinhalten. Dazu wird eine von `de.unistuttgart.ipvs.pmp.resource.Resource` abgeleitete Klasse erstellt. Eine Ressource der PMP muss die Methoden `getAndroidInterface()`, `getMockedAndroidInterface()` und `getCloakedAndroidInterface()` zur Verfügung stellen, die die Implementierungen für die Modi `NORMAL`, `MOCK` und `CLOAK` liefern. Die Methode `registerResource()` fügt die Ressource zur Ressourcengruppe hinzu.

Schnittstellendefinition

Die PMP und alle Apps, die ihre Funktionen verwenden, laufen in verschiedenen Prozessen. Für die Kommunikation zwischen Prozessen wird eine Schnittstellendefinition erstellt. in diesem Fall eine Datei mit dem Namen `IfSystemInformation.aidl` AIDL steht für *Android Interface Definfition Language*. Die Syntax ist mit der Syntax in Java praktisch identisch. Erst wird das Paket angegeben, in dem die Definition liegt, dann werden die Klassen für nicht-primitive Datentypen importiert. Im folgenden Block, der den Namen der Schnittstelle definiert, muss für jede Funktion, die nach außen, also für Apps sichtbar sein soll, der Name und der Rückgabetyp angegeben werden. Der Rückgabetyp kann entweder ein primitiver Datentyp sein oder eine Klasse, die die Schnittstelle `Parceable` implementiert. Im Unterschied zur Schnittstellendefinition in Java darf in einer AIDL-Datei nur eine Schnittstelle definiert werden. Listing 7.2 zeigt den Aufbau der Schnittstellendefinition für die App *Information Read*.

Implementierungen für die Modi NORMAL, MOCK und CLOAK

Aus der Datei IfSystemInformation.aidl wird vom Android SDK eine gleichnamige Datei mit der Endung .java generiert. Sie enthält die Klasse IfSystemInformation.Stub, von der jede der drei Implementierungen der Ressource abgeleitet ist.

Demnach muss jede Implementierung für alle Methoden, die in der AIDL-Datei aufgelistet sind, einen Wert des dort definierten Typs zurückliefern. Im Fall der Implementierung für den Modus NORMAL wird der TelephonyManager angefordert, der aus dem Gerät die echten Daten, zum Beispiel die IMEI ausliest. Die Methoden der Implementierungen für die Modi MOCK und CLOAK liefern konstante Werte zurück.

Jede Methode ist durch Privacy Settings geschützt. Es wird in jeder Methode der Implementierungen festgelegt, welche das sind und in welcher Stufe sie für die App, die die Ressource anfragt, vorliegen müssen. Reicht die Stufe nicht aus, wird eine SecurityException ausgelöst. Um den Quellcode übersichtlicher zu gestalten, wurde die Prüfung in der Methode PermissionValidator.validate zusammengefasst. So reduziert sich die Prüfung pro Privacy Setting auf eine Zeile.

7.1.3. Kompatibilität der App zur PMP

Um die App Information Read zur PMP kompatibel zu machen, mussten Anpassungen vorgenommen werden.

Registrierung bei der PMP

Jede App muss sich einmalig bei der PMP registrieren. Die erste Möglichkeit ist, in der Manifest-Datei der App die RegistrationActivity der PMP als Startbildschirm zu definieren. Wird die App das erste Mal gestartet, öffnet sich der Bildschirm der PMP, in dem der Benutzer die erlaubten Zugriffe auf das Gerät auswählen kann. Bei jedem weiteren Start der App wird das Event ALREADY_REGISTERED ausgelöst. Der PMPRegistrationHandler sorgt dann dafür, dass nicht die RegistrationActivity, sondern der Startbildschirm der App erscheint. Es ist auch möglich, die App zur Laufzeit über die Methode IPMP.register() zu registrieren. Dabei muss beachtet werden, dass die Registrierung in einem separaten Thread erfolgt. Solange dieser nicht abgeschlossen ist, kann keine Ressource angefordert werden.

Für die Kommunikation mit der PMP muss in der Manifest-Datei der AppService der PMP angegeben werden. Über diesen Service ist die App mit der PMP verbunden und kann Implementierungen von Ressourcen anfordern.

Listing 7.3 Das App Information Set für die App *Information Read*

```
<?xml version="1.0" encoding="UTF-8"?>
<appInformationSet>
   <appInformation>
      <name lang="en">Information Read</name>
      <name lang="de">Information Read</name>
      <description lang="en">Displays the device IMEI number</description>
      <description lang="de">Zeigt die IMEI-Nummer des Telefons an</description>
   </appInformation>
   <serviceFeatures>
      <serviceFeature identifier="imei">
         <name lang="en">Get IMEI</name>
         <name lang="de">IMEI abfragen</name>
         <description lang="en">Read the IMEI number from the device.</description>
         <description lang="de">IMEI-Nummer des Telefons auslesen.</description>
         <requiredResourceGroup
             identifier="de.unistuttgart.ipvs.pmp.resourcegroups.systeminformation"
             minRevision="1970-01-01 01:01:01:001 MEZ">
            <requiredPrivacySetting
                identifier="getDeviceId"><![CDATA[true]]></requiredPrivacySetting>
         </requiredResourceGroup>
      </serviceFeature>
   </serviceFeatures>
</appInformationSet>
```

Service Features definieren

Jede zur PMP kompatible App besitzt ein `App Information Set` (AIS). Dies muss in der Datei `/assets/ais.xml` liegen. Im AIS wird definiert, welche Service Features die App anbietet und welche Privacy Settings dafür auf welche Stufe gesetzt sein müssen. Zu jedem Service Feature werden zusätzliche Erklärungen angeboten, um dem Benutzer die Entscheidung, ob das Service Feature aktiviert werden soll zu erleichtern. Analog zum RGIS einer Ressourcengruppe (Abschnitt 7.1.2) können die Erklärungen in verschiedenen Sprachen abgelegt werden. Eine englische Version muss vorhanden sein. Das App Information Set von *Information Read* ist in Listing 7.3 zu sehen.

Zugriff auf PMP-Ressourcen

Der direkte Zugriff auf Telefonfunktionen, die durch Android Permissions geschützt sind, ist für nachträglich installierte Apps gesperrt. Zugriffe müssen daher über die zuständigen Ressourcen der PMP erfolgen. Die Methode `PMP.get(Application app)` liefert ein Objekt, das die PMP API für eine bestimmte Anwendung repräsentiert. Dessen Methode `getResource(PMPResourceIdentifier res, PMPRequestResourceHandler handler)` startet die Anfrage einer Ressource. Das Ergebnis kann durch Überlagern der Methode `onReceiveResource()` des `PMPRequestResourceHandler` abgerufen werden. Zu beachten ist, dass für die Anfrage einer Ressource ein neuer Thread erstellt wird. Es ist also nicht garantiert, dass das Ergebnis unmittelbar nach der Anfrage bereit steht.

Abbildung 7.1.: Screenshots der App *Information Read*. Die obere Reihe zeigt verschiedene Einstellungen in der PMP. Im jeweiligen Bild darunter ist die Auswirkung auf die App sichtbar.

7.2. Auslesen von Standortinformationen

Die zweite App zur Demonstration sollte sowohl mehrere Ressourcengruppen benutzen als auch eine Anwendung im Alltag haben. Die App *Waypoint Finder* benutzt den GPS-Empfänger des Geräts, um Richtung und Entfernung zu beliebigen Orten anzuzeigen. Die App besteht aus einer Richtungsanzeige, einer Entfernungsanzeige und einem Knopf über den der Benutzer seinen aktuellen Standort als Zielpunkt setzen kann. Der Waypoint Finder kann dazu verwendet werden, in fremder Umgebung zu einem Treffpunkt oder zu einem geparkten Auto zurückzufinden. Die App benötigt zwei Ressourcengruppen:

1. Die Ressourcengruppe **Location** zur Bestimmung des Standorts wurde bereits mit der vorhandenen Version der PMP mitgeliefert.
2. Um den benutzerdefinierten Zielort dauerhaft zu speichern, wurde die Ressourcengruppe **Filesystem** geschrieben. Sie enthält Funktionen zum Lesen und Schreiben von Dateien. Außerdem kann überprüft werden, ob eine gegebene Datei existiert.

Der Aufbau einer Ressourcengruppe wurde bereits in Abschnitt 7.1.2 beschrieben. Daher werden die Ressourcengruppen an dieser Stelle nicht im Detail behandelt.

7.2.1. Funktionsweise der App

Um die Zielrichtung zu bestimmen, wird zunächst der `Orientation Sensor` ausgelesen. Der `Orienation Sensor` liefert einen Vektor, der die Lage des Geräts im Raum beschreibt. Die erste Koordinate des Vektors liefert die *Peilung* $\alpha_{TELEFON}$. Die Peilung wird auch `Kurs` genannt und ist eine wichtige Größe in der Schifffahrt. Sie gibt die horizontale Abweichung der Richtung vom Betrachter zu einem gedachten Ziel gegenüber der Nordrichtung in Grad an. Bei jeder Lageänderung des Geräts wird gleichzeitig die aktuelle GPS-Position bei der Ressourcengruppe `Location` angefordert.

Für die Erdkoordinaten der aktuellen Position (ρ_1,λ_1) und der Zielposition (ρ_2,λ_2) kann laut [Ven10] mit dem Erdradius R der Abstand d berechnet werden:

$$a = \sin^2\left(\frac{\Delta\rho}{2}\right) + \cos(\rho_1) \cdot \cos(\rho_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$d = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a}) \cdot R$$

Anschließend muss noch die Peilung zum Zielpunkt bestimmt werden. Die Peilung ist entlang der Wegstrecke auf einer Kugeloberfläche nicht konstant. Die *Haversine Formula* aus [Ven10] liefert die Peilung am Startpunkt der Wegstrecke:

$$\alpha_{ZIEL} = atan2\left[(\sin\Delta\lambda \cdot \cos\rho_2), (\cos\rho_1 \cdot \sin\rho_2 - \sin\rho_1 \cdot \cos\rho_2 \cdot \cos\Delta\lambda)\right] \cdot \frac{180^\circ}{\pi}$$

Der Winkel im Uhrzeigersinn, um den ein nach oben zeigender Pfeil auf dem Display des Geräts gedreht werden muss, ist:

$$\alpha_{PFEIL} = \alpha_{ZIEL} - \alpha_{TELEFON}$$

Der sich drehende Pfeil ist als Objekt der Klasse `ImageView` implementiert, dem eine Bilddatei mit transparentem Hintergrund zugeordnet ist. Auf dieses Objekt werden Matritzen zur Translation, Skalierung und anschließend zur Rotation angewendet.

7.2.2. Screenshots

Die Abbildung 7.2 zeigt die App *Waypoint Finder* im Zusammenspiel mit der PMP. Links sind die Service Features der App zu sehen, im mittleren Bild wird die PMP so konfiguriert, dass die Ressourcengruppe `Location` gefälschte Daten liefert. Wie das rechte Bild zeigt, läuft die App, obwohl der GPS-Empfänger nicht eingeschaltet ist.

Abbildung 7.2.: Screenshots der App *Waypoint Finder* und der PMP.

8. Zusammenfassung und Ausblick

Das Betriebssystem Android hat sich seit seiner Einführung im Jahr 2008 rasend schnell verbreitet. Anfang des Jahres 2013 verfügte es bereits über einen Marktanteil von 67 Prozent und mehr als 600.000 installierbare Apps. Darunter befinden sich auch Apps, deren Ziel es ist, private Daten auszuspähen oder über den heimlichen Versand von Premium SMS Geld zu stehlen. Zugriffe auf sensible Daten sind in Android durch Berechtigungen geschützt. Bei der Installation einer App sieht der Benutzer alle erforderlichen Berechtigungen und muss allen zustimmen, um die App installieren zu können. Ein großer Nachteil des Berechtigungssystems in Android ist, dass nicht klar ersichtlich ist, wozu eine App ihre Berechtigungen braucht und welche Daten tatsächlich ausgelesen werden. Weiterhin hat die große Zahl an Berechtigungen (137 in Android 2.3.7) dazu geführt, dass Nutzer den Überblick verlieren und den angeforderten Berechtigungen bei der Installation immer weniger Beachtung schenken. Es herrscht also Bedarf zu forschen, wie das System der Berechtigungen verbessert werden kann.

Dazu wurde untersucht, wie Android sicherstellt, dass eine App nur Zugriff auf eine Telefonfunktion hat, wenn eine entsprechende Berechtigung vorliegt. Durch Analyse des Quellcodes des Betriebssystems wurde ermittelt, wie das Gerät erteilte Berechtigungen speichert, beim Start wiederherstellt und auf Anfrage überprüft. Es hat sich gezeigt, dass der Mechanismus aus einer Kette von Aufrufen besteht, die sich quer durch das Application Framework zieht. Somit wurden mögliche Ansatzpunkte für eine Modifikation der Berechtigungsprüfung lokalisiert.

Aus dem Wunsch nach mehr Kontrolle sind einige Konzepte für ein verbessertes Berechtigungssystems entstanden. Nahezu jeder Entwurf sieht vor, dass der Benutzer einzelne Berechtigungen deaktivieren kann. Dazu gibt es die Idee, Apps vor der Installation zu konvertieren, so dass sie einer neuen Berechtigungsprüfung unterliegen. Auch die Änderung des Prüfmechanismus in Android wurde mehrfach umgesetzt, so dass ein neues Betriebssystem entsteht. Es hat sich herausgestellt, dass die untersuchten Konzepte entweder nicht umfassend sind oder Nachteile wie häufige Abstürze haben.

Mit der PMP wurde ein Konzept vorgestellt, das viele Vorteile bietet. Service Features einer App bieten dem Nutzer einen detaillierten Überblick darüber, welche Daten vom Telefon angefragt werden. Zudem können sie zum Zeitpunkt der Installation, nach der Installation und sogar im laufenden Betrieb der App deaktiviert werden. Der Status von Service Features kann zusätzlich durch zeitbezogene und ortsbezogene Bedingungen automatisiert werden. Einziger Nachteil der PMP ist, dass sie als App leicht umgangen werden kann. Die PMP soll also so integriert werden, dass sie fester Bestandteil des Betriebssystems ist.

Anschließend wurde geprüft, wo ein alternatives Berechtigungssystem in Android am besten anzusiedeln ist. Dazu wurden verschiedene Strategien untersucht. Die PMP kann als App, als Alternative für das bestehende Berechtigungssystem oder als dessen Ersatz dienen. Das Ergebnis war, dass die PMP nur dann für maximale Sicherheit sorgen kann, wenn sie das vorhandene Berechtigungssystem ersetzt.

Für die Implementierung dieses Ansatzes mussten die Projekte PMP und PMP-API zusammengeführt und in den Android Quellcode eingebaut werden. Die PMP wird zusammen mit dem Betriebssystem gebaut und kann nicht mehr deinstalliert werden. Um zu verhindern, dass Apps den direkten Zugriff auf das Application Framework erlangen können, wurde zusätzlich der Mechanismus der Berechtigungsprüfung modifiziert. Damit steht ein Android Betriebsystem zur Verfügung auf dem die PMP als einzige Autorität zur Erteilung von Berechtigungen ist.

Zur Demonstration der Ergebnisse wurden zwei Apps entwickelt. Die App *Information Read* beweist, dass Zugriffe auf geschützte Funktionen wie das Auslesen der IMEI ohne die PMP nicht mehr möglich sind. Die etwas umfangreichere App *Waypoint Finder* gibt Richtung und Entfernung zu einem benutzerdefinierten Wegpunkt an. Die Zugriffsrechte für beide Apps lassen sich über die Benutzeroberfläche der PMP steuern.

Ausblick

Das Resultat dieser Arbeit ist ein Betriebssystem, auf dem nachinstallierte Apps nur über die PMP Berechtigungen für Zugriffe auf geschützte Telefonfunktionen erlangen können. Das bedeutet, dass den Apps, die nicht zur PMP kompatibel sind, grundsätzlich alle Zugriffsrechte verweigert werden. Um dieses Betriebssystem im Alltag nutzen zu können, müsste dafür gesorgt werden, dass alle Apps PMP-kompatibel sind. Ein möglicher Ansatz wäre ein Konverter, wie in Abschnitt 5.2.2 beschrieben wird. Ein Konverter könnte auch in die PMP integriert sein. Er könnte von der `PackageInstallerActivity` aus aufgerufen werden und Pakete von Apps so präparieren, dass die App nach der Installation zur PMP kompatibel ist.

Weiterhin ist die Entwicklung von PMP-kompatiblen Apps dadurch erschwert, dass jede angeforderte Ressource in einem neuen Thread geliefert wird. Werte, die über die Ressource ausgelesen werden, müssen also zwischengespeichert werden. Hilfreich wäre eine Komponente, die das Anfordern einer Ressource so kapselt, dass direkt auf Funktionen in einer Ressource zugegriffen werden kann, zum Beispiel `getResource(LOCATION_RG).getLatitude()`.

Für das Setzen von ortsbezogenen Bedingungen wird in der PMP die Google Maps Library verwendet. Da die Google Maps Library nicht quelloffen ist, kann sie nicht mit einer selbst kompilierten Version von Android ausgeliefert werden, außer dies geschieht von Google selbst. Es sollte nach einem Ersatz gesucht werden, der es ermöglicht ortsbezogene Bedingungen komfortabel zu erstellen.

A. Anhang

Während der Arbeit mit der PMP sind folgende Ungereimtheiten aufgetreten:

Preset erstellen Beim Eintragen des Namens springt der Cursor an den Anfang des Textfeldes statt wie gewohnt an das Ende des vorhandenen Textes. Die Eingabe eines Namens kann nicht beendet werden, ohne die Bildschirmtastatur manuell auszublenden, da der Knopf zur Bestätigung verdeckt wird.

Preset editieren Werden beim Editieren eines Presets nur Kleinbuchstaben durch Großbuchstaben ersetzt, erscheint die Meldung „Preset existiert schon“.

Preset editieren Beim Editieren des Privacy Settings „Mode“ tritt ein nicht reproduzierbarer Fehler auf, der die PMP beendet. Der Fehlertext lautet:

„No keyboard for id 131074. Using default keymap : /system/usr/keychars/qwerty.kcm.bin“

Ob der Fehler von der PMP oder vom kompilierten Image verursacht wird, konnte nicht festgestellt werden. Bei anderen Anwendungen trat er noch nicht auf.

Registration Activity Wird während der Registrierung einer App die Bildschirmausrichtung verändert, verschwindet der Dialog mit dem Knopf „Set initial Service Features“.

Literaturverzeichnis

[and13] android.com. *Android Security Overview*, 2013. URL http://source.android.com/tech/security/.

[app13] appbrain.com. Number of available Android applications, 2013. URL http://www.appbrain.com/stats/number-of-android-apps/. (Zitiert auf Seite 9)

[FCH+11] A. P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner. Android Permissions Demystified. In *CCS '11*. University of California, Berkeley, 2011. (Zitiert auf Seite 14)

[FHE+12] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior. In *SOUPS '12*. 2012. (Zitiert auf Seite 9)

[FS13] F-Secure. Mobile Threat Report Q4 2012. Technischer Bericht, F-Secure, 2013. (Zitiert auf Seite 9)

[htt] http://androidforums.com. How do you edit an app's permissions? URL http://androidforums.com/android-applications/185217-how-do-you-edit-apps-permissions.html. (Zitiert auf Seite 10)

[NKZ10] M. Nauman, S. Khan, X. Zhang. Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In *ASIACCS '10*. 2010. (Zitiert auf Seite 19)

[RJV+11] N. Reddy, J. Jeon, J. A. Vaughan, T. Millstein, J. S. Foster. Application-centric security policies on unmodified Android. Technischer Bericht, University of California, Los Angeles, 2011. (Zitiert auf Seite 36)

[SM13] C. Stach, B. Mitschang. Privacy Management for Mobile Platforms - A Review of Concepts and Approaches. In *Proceedings of the 14th International Conference on Mobile Data Management*, S. 1–9. IEEE Computer Society Conference Publishing Services, 2013. URL http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2013-11&engl=. (Zitiert auf den Seiten 10 und 29)

[sta] stackoverflow.com. Enable GPS programatically like Tasker. URL http://stackoverflow.com/questions/4721449/enable-gps-programatically-like-tasker. (Zitiert auf Seite 24)

[Ven10] C. Veness. Calculate distance and bearing between two Latitude/Longitude points using Haversine formula, 2010. URL http://www.movable-type.co.uk/scripts/latlong.html. (Zitiert auf Seite 55)

[XSA12] R. Xu, H. Saïdi, R. Anderson. Aurasium: Practical Policy Enforcement for Android Applications. In *USENIX Security '12*. 2012. (Zitiert auf Seite 19)

[zdn13] zdnet.de. Android erreicht in Deutschland einen Marktanteil von 67 Prozent, 2013. URL http://www.zdnet.de/88140679/android-erreicht-in-deutschland-marktanteil-von-67-prozent/. (Zitiert auf Seite 9)

Alle URLs wurden zuletzt am 25. 04. 2013 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Philipp Scholz)