Matthias Horbach (Hrsg.): INFORMATIK 2013,
2072 Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2013

Wie funktioniert Datenschutz auf Mobilplattformen?

Christoph Stach
IPVS Universität Stuttgart, Universitätsstraße 38, D-70569 Stuttgart
Christoph.Stach@ipvs.uni-stuttgart.de

Abstract: Die wachsende Verbreitung von mobilen Geräten, bei denen einerseits sehr viele Kontextdaten und andererseits zahlreich persönliche Informationen anfallen, macht diese zu einem hervorragenden Ziel für Angriffe auf die Privatsphäre. Verschärft wird diese Situation dadurch, dass jede Mobilplattform eine eigene Strategie zum Schutz dieser Daten verfolgt, allerdings ohne dabei den Nutzer in geeignetem Maße mit einzubeziehen. Aus diesem Grund gibt es eine Vielzahl an Erweiterungen für diese Berechtigungssysteme. Für den Nutzer bleibt dabei weiterhin die Frage, für welche Plattform und für welches Berechtigungssysteme er sich entscheiden soll. In diesem Papier beschäftigen wir uns daher mit den Eigenschaften aktueller Mobilplattformen und einiger aktueller Berechtigungssysteme. Wir stellen mit der *Privacy Management Platform (PMP)* unser eigenes Berechtigungssystem vor, das sich stark an den Bedürfnissen der Nutzer orientiert. Neben dem Modell für die Berechtigungsrichtlinien hat allerdings auch die Art, wie diese Modell in die Mobilplattform eingebunden wird, entscheidenden Einfluss auf die Eigenschaften des Systems. Durch die Gegenüberstellung dieser Eigenschaften, wird dem Nutzer die Wahl einer Plattform und eines Berechtigungssystems, das seinen Bedürfnissen genügt, erleichtert.

1 Einleitung

Mobiltelefone sind seit Jahren ein fester Bestandteil unseres Alltags. Das Leistungsspektrums dieser Geräte steht heutzutage Desktop-PCs in nichts nach – und so werden sie auch genutzt: Man verwaltet damit Termine und Kontakte, lässt sich navigieren und kann mittels NFC sogar an der Kasse bezahlen. Diese Vielseitigkeit wird durch ein großes Angebot an Drittsoftware weiter begünstigt. Bei der Verwendung jener Anwendungen, den sog. *Apps*, gehen viele Nutzer oft unbedacht vor, wodurch es häufig zu einer unerwünschten Verwendung von privaten Daten (z.B. Fotos, Kontaktdaten oder Mitteilungen) kommt [Ba12a]. Ein zuverlässiges Berechtigungssystem, das den Zugriff auf private Daten regelt, kann daher ein entscheidendes Kaufargument für oder gegen eine Mobilplattform darstellen. Jede Plattform setzt dabei allerdings auf eine andere Strategie zum Schutz privater Daten (im Folgenden von uns als *Datenschutz* bezeichnet). Aus Nutzersicht ist jedoch keiner dieser Ansätze befriedigend, wie zahlreiche Diskussionen (z.B. `http://goo.gl/keCdf`) zeigen.

Aber gerade für Apps, die mit hochgradig vertraulichen Daten arbeiten, wird ein verlässliches Berechtigungssystem akut benötigt. Ein Beispiel für solche Apps sind u.a. *mobile Health-Games*, Spiele mit therapeutischem Ansatz, wie *Candy Castle (CC)* [SS12]. Mit CC wird für Kinder die Blutzuckermessung spielerisch in deren Alltag eingebaut und den Ärzten werden alle behandlungsrelevanten Daten übersichtlich präsentiert. CC hat dabei nicht nur Zugriff auf die Blutzuckerwerte eines Nutzers, sondern auch auf dessen Positionsdaten und weitere private Informationen, wie den Kontaktdaten des behandelnden Arztes. Dazu benötigt die App u.a. die Android-Berechtigungen

ACCESS_FINE_LOCATION, READ_CONTACTS oder CALL_PRIVILEGED. Wegen der Häufung dieser Berechtigungen, besitzt CC ein hohes Gefahrenpotential, weshalb Nutzer vor einer Verwendung zurückschrecken könnten. Obgleich einige Berechtigungen nur für nebensächliche Features von CC benötigt werden, auf die viele Nutzer verzichten würden, müssen dennoch alle gewährt werden. Trotz der vielen Android-Berechtigungen gibt es keine Berechtigung, die den Umgang mit medizinischen Daten beschränkt.

In diesem Papier beschäftigen wir uns daher damit, wie aus Nutzersicht der Datenschutz auf mobilen Geräten besser geregelt werden kann. Hierzu analysieren wir zunächst in Abschnitt 2 die Berechtigungsmodelle der unterschiedlichen Mobilplattformen. Anschließend werden in Abschnitt 3 Erweiterungen des Android-Berechtigungssystems miteinander verglichen und in Abschnitt 4 wird unser Ansatz für ein erweiterbares, kontextsensitives, absturzsichers Berechtigungssystem präsentiert, das Veränderungen der Berechtigungsrichtlinien zur Laufzeit zulässt und private Daten auf Wunsch randomisiert. Da die Implementierung eines solchen Systems großen Einfluss auf die Nutzbarkeit hat, werden die möglichen Implementierungsalternativen in Abschnitt 5 vorgestellt. Abschließend fasst Abschnitt 6 die wichtigsten Aspekte dieser Arbeit zusammen.

2 Aktuelle Mobilplattformen

Um eine Klassifizierung von Mobilplattformen hinsichtlich ihres Umgangs mit potentieller Malware zu ermöglichen, führen Barrera et al. [BO11] drei Modelle ein: das *Walled Garden-Modell*, das *Guardian-Modell* und das *End-user Control-Modell*.

Walled Garden-Modell. Im *Walled Garden-Modell* schirmt ein Plattformanbieter das System vollständig ab. Er allein entscheidet, welche Apps für die Plattform verfügbar sind. Installierte Apps können jederzeit vom Anbieter ferngelöscht werden und neue Apps können aufgespielt werden, ohne dass der Nutzer eingreifen kann. Der Nutzer muss darauf vertrauen, dass der Anbieter dabei in seinem Interesse handelt, da alle privaten Daten diesem vollständig ausgeliefert sind. Ist der Anbieter allerdings zuverlässig und geht er bei der Prüfung der Apps gewissenhaft vor, so ist das Walled Garden-Modell selbst für unerfahrene Nutzer sehr sicher, da diesen jedwede Entscheidung abgenommen wird.

Guardian-Modell. Das sogenannte *Guardian-Modell* ist weniger präskriptiv. Ein Plattformentwickler überträgt nahezu sämtliche fundamentale Berechtigungsentscheidungen an eine Kontrollinstanz, den *Guardian*. Dieser kann ein Partner des Systemanbieters (wodurch es dem Walled Garden-Modell nahe kommt), der Mobilfunkanbieter oder ein vertrauenswürdiger Dritter (z.B. ein Administrator) sein. Häufig wird die Entscheidungsgewalt auch unter diesen drei Parteien aufgeteilt. Der Nutzer selbst wird im Guardian-Modell kaum bei sicherheitsrelevanten Fragen in den Entscheidungsprozess miteinbezogen. Er wird als unkalkulierbares Sicherheitsrisiko angesehen, da ihm oft die nötige Kompetenz sowie die gebotene Sorgfalt fehlt, um wirksame Datenschutzeinstellungen vorzunehmen.

End-user Control-Modell. Den beiden obigen Modellen, die den Nutzer mehr oder weniger entmündigen, steht das *End-user Control-Modell* gegenüber. Hier erhält der Nutzer (nahezu) die

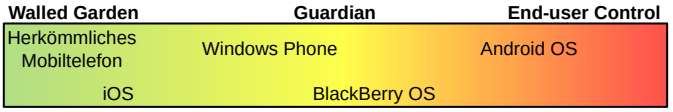


Abb. 1: Klassifikation der heutigen Mobilplattformen (vgl. [BO11])

alleinige Entscheidungsgewalt über das System. Zusammen mit diesen Freiheiten werden ihm gleichzeitig auch alle Pflichten bezogen auf den Datenschutz übertragen. Damit er dieser Verantwortung gerecht werden kann, muss sichergestellt werden, dass er mit ausreichend Informationen über die Apps, die auf seinem System laufen, und deren Datennutzung versorgt wird. Nur so kann er die für ihn bestmöglichen Entscheidungen treffen. Auch muss das System über ein eingängiges Konfigurationssystem verfügen, das es dem Nutzer ermöglicht die getroffenen Entscheidungen entsprechend in die Tat umzusetzen.

App-Entwickler wollen allerdings nicht, dass Elemente einer App deaktiviert werden können (z.B. Werbeeinblendungen, die eine App finanzieren [Le12]). Daher fallen Informationen über das Verhalten von Apps dürftig aus und die Möglichkeit zur Rechtevergabe ist überaus limitiert. Der Nutzer wird vor eine Alles-oder-Nichts-Entscheidung gestellt – entweder er nutzt eine App mit allen Berechtigungen oder die App wird nicht installiert.

Betrachtet man die Einordnung aktueller Mobilplattformen (Abbildung 1), so stellt man fest, dass Android dem End-user Control-Modell am nächsten kommt. Daher ist es auch nicht verwunderlich, dass sich viele Datenschutzstudien mit Android beschäftigen, da es hier dem Nutzer obliegt, für die Sicherheit relevanter Daten zu sorgen.

Shabtai et al. [Sh10] stellen bei ihrer Studie fest, dass das in Android implementierte Berechtigungssystem unzureichend ist: Apps können sogar installiert werden, ohne dass der Nutzer ihre Berechtigungen akzeptiert. Aber auch ohne diese offensichtliche Sicherheitslücke, sind viele Nutzer überfordert die Berechtigungen und deren Bedeutung zu verstehen und setzen sich daher erst gar nicht damit auseinander [Fe12]. Dies ist allerdings dringend erforderlich, da viele Apps überprivilegiert sind [Fe11] und sogar heimlich Berechtigungen untereinander austauschen [CHY12]. Lt. Enck et al. [En10] sind dies keine Einzelfälle, sondern 20 von 30 zufällig gewählten Apps aus dem Google Play Store zeigen ein auffälliges Verhalten im Umgang mit sensiblen Daten. Im Folgenden werden daher unterschiedliche Ansätze zur Verbesserung des Berechtigungssystems von Android vorgestellt.

3 Erweiterungen des Android-Berechtigungssystems

Viele wissenschaftliche Arbeiten befassen sich mit erweiterten Berechtigungssystemen für Android. Nauman et al. [NKZ10] untersuchten, von welchen Berechtigungen das größte Bedrohungspotential für den Nutzer ausgeht. Zur Minimierung dieser Bedrohungen führen sie ein neues Modell für Berechtigungsrichtlinien ein, bei dem Berechtigungen mit Bedingungen verknüpft werden können. Eine solche Bedingung könnte sein, dass eine App pro Tag maximal fünf SMS verschicken darf. Um diese Regeln durchsetzen zu können, wird mit *Apex* ein neues Berechtigungssystem für Android präsentiert.

Neben der Einführung der restriktionsbasierten Regeln ermöglicht es Apex, dass einer App auch nur eine Teilmenge der angeforderten Berechtigungen genehmigt oder dieser ein Berechtigung zur Laufzeit hinzufügt respektive entzogen werden kann. Allerdings wird die App mit einer unverständlichen Fehlermeldung beendet, wenn eine Funktion aufgerufen wird, für die benötigte Berechtigungen fehlen.

AppFence [Ho11] befasst sich mit dem Problem des unkontrolliertem Austauschs von geschützten Daten. Hierzu kann der Nutzer die Übermittlung dieser Daten komplett sperren und nicht vertrauenswürdige Apps vorsorglich mit verfälschten Daten versorgen. Allerdings können Apps AppFence unterwandern und untereinander die (vermeintlich) geschützten Daten austauschen. Außerdem gibt AppFence keine Informationen darüber, wozu eine App die Daten benötigt, weshalb der Nutzer keine opportune Konfiguration vornehmen kann.

Aurasium [XSA12] und *Dr. Android & Mr. Hide* [Je12] verwenden einen App-Konverter (siehe Abschnitt 5.5), um datenschutzkritische Aufrufe einer App aufzuspüren und in Anfragen an eine mitgelieferte Bibliothek umzuwandeln. Für die Verwendung dieser Bibliothek kann der Nutzer feingranulare, anwendungsbezogene Berechtigungen definieren. Allerdings wird er von Dr. Android & Mr. Hide nicht darüber informiert, welche Auswirkungen diese Datenschutzeinstellungen auf das Verhalten der App haben. Bei Aurasium werden Apps unvermittelt beendet, wenn die zugestandenen Berechtigungen nicht ausreichend sind.

Da der Zugriff auf sensible Daten noch keinen Schaden verursachen kann, sondern erst deren Verarbeitung, führen Beresford et al. [Be11] mit *MockDroid* ein Berechtigungssystem ein, bei dem der Nutzer festlegt, ob eine App korrekte oder verfälschte Daten erhalten soll. Dies ist allerdings nur für einen Teil der Android-Berechtigungen möglich und manche Apps verlieren dadurch vollständig ihren Sinn (z.B. wenn eine Navigationssoftware nur mit randomisierten Positionsdaten versorgt wird).

Banuri et al. [Ba12b] adressieren mit *SEAF* das Problem, dass es eine Rolle spielen kann, in welcher Reihenfolge eine App Zugriff auf bestimmte Systemfunktionen erlangt, wodurch scheinbar harmlose Berechtigungen den Datenschutz gefährden können. Daher erweitern sie das Modell für Berechtigungsrichtlinien, so dass für jede App spezifiziert werden kann, in welcher Abfolge diese die angeforderten Berechtigungen erhält. SEAF untersucht zur Laufzeit, ob eine App eine potentiell gefährliche Sequenz von Berechtigungsanforderungen aufweist und informiert den Nutzer anschließend darüber. Dieser kann daraufhin den Berechtigungsrichtlinien eine neue Regel hinzufügen. Wird eine App allerdings als unkritisch eingestuft, so bietet SEAF dem Nutzer kein anderes Leistungsspektrum, als das originäre Berechtigungssystem von Android.

Fragkaki et al. [Fr12] führen ein formales Modell zur Beschreibung von generischen Berechtigungen ein. Im Gegensatz zu dem Berechtigungsmodell von Android kann man mit diesem Modell die Lebensdauer von Berechtigungen modellieren, sowie Berechtigungseskalationen und unkontrollierte Informationsflüsse unterbinden. Mit *Sorbet* wird ein Berechtigungssystem vorgestellt, in dem dieses Berechtigungsmodell implementiert ist. Allerdings ist es mit Sorbet nicht möglich, kontextsensitive Regeln zu formulieren.

Mit *YAASE* nehmen sich Russello et al. [Ru11] der unkontrollierten Berechtigungsweitergabe zwischen mehreren Apps an. Hierfür führen sie eine Notation für feingranulare Berechtigungsregeln ein, ohne

Tab. 1: Vergleich der Berechtigungssysteme

	Kontext-sensitiv	Laufzeit-änderungen	Absturz-sicher	Dummy Daten	Feed-back
Apex	√	√	×	×	×
AppFence	×	√	√	√	×
Aurasium	×	√	√	×	√
Dr. Android & Mr. Hide	√	×	√	×	×
MockDroid	×	√	√	√	√
SEAF	√	√	×	×	√
Sorbet	×	√	×	×	×
YAASE	×	√	×	×	×
CyanogenMod	×	√	×	×	×
PDroid	×	√	√	√	×
Privacy Protector	×	√	×	×	×
PMP	√	√	√	√	√

dass diese zu komplex wird. Somit gewinnen die Regeln an Mächtigkeit und bleiben für den Nutzer dennoch beherrschbar. YAASE kann allerdings nicht verhindern, dass Apps anstelle der Berechtigungen direkt die geschützten Daten untereinander austauschen.

Neben diesen Ansätzen, die aus dem wissenschaftlichen Kontext stammen, beschäftigt sich auch die Android-Community mit möglichen Verbesserungen für das Berechtigungssystem von Android. *CyanogenMod* ist eine der bekanntesten Erweiterungen des originären Android-Systems. Neben einigen Features, die außerhalb des Fokusses dieser Arbeit liegen, ermöglicht CyanogenMod es dem Nutzer, einer App beliebige Berechtigungen – auch zur Laufzeit – entziehen zu können. Dies führt allerdings unweigerlich zu einen Absturz der betroffenen App, wenn sie versucht, auf Daten zuzugreifen, ohne die dafür benötigten Berechtigung zu haben, wodurch die Stabilität des Systems sehr leidet. Daher wurde dieses Feature mit der Version 9.0 ersatzlos eingestellt. *PDroid Privacy Protection* versucht das Problem der Systemabstürze bei fehlenden Berechtigungen dadurch zu umgehen, dass in diesen Fällen wahlweise Zufallswerte oder frei gewählte Konstanten an die App weiterleitet werden. Völlig konträr zu CyanogenMod und PDroid geht *Privacy Protector (No root)* vor. Bei diesem Ansatz werden ausschließlich Standortdaten und Netzwerkzugriffe als datenschutzkritische Faktoren angesehen. Der Nutzer kann daher für eine App nur diese beiden Funktionen deaktivieren. Stellt Privacy Protector fest, dass diese App gestartet wird, werden die Ortungsfunktionen bzw. der Netzwerkzugang systemweit deaktiviert, wodurch auch alle anderen Apps in Mitleidenschaft gezogen werden.

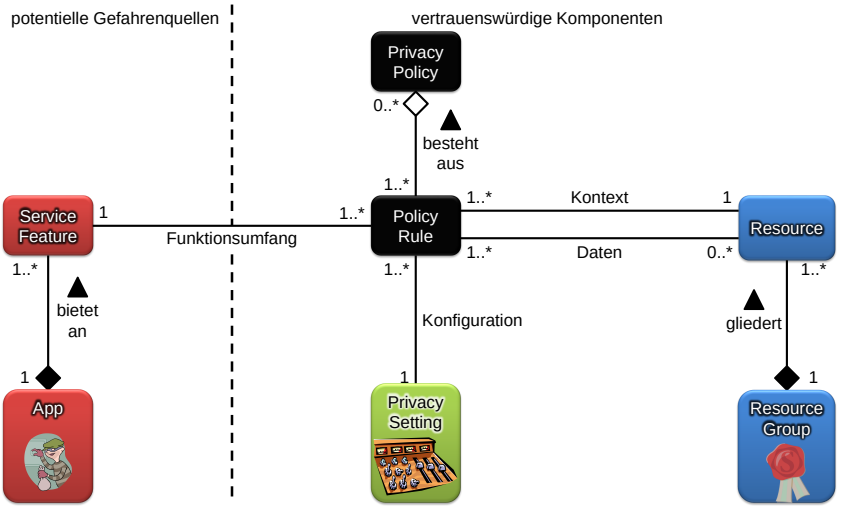


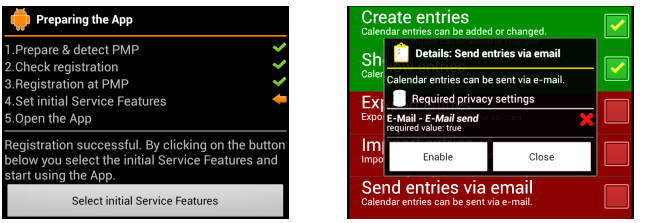
Abb. 2: Das Berechtigungsrichtlinienmodell der PMP

4 Die Privacy Management Platform

Tabelle 1 zeigt, dass keines der vorgestellten Berechtigungssysteme alle Nutzerwünsche erfüllt. Aus diesem Grund führen wir mit der **P**rivacy **M**anagement **P**latform (*PMP*) [SM13] ein kontextsensitives und absturzsicheres Berechtigungssystem ein, das Berechtigungsanpassungen zur Laufzeit zulässt und private Daten auf Wunsch beliebig randomisiert. Dabei wird der Nutzer stets über die Konsequenzen, die seine Einstellungen auf den Leistungsumfang einer App haben, informiert. Abbildung 2 zeigt in einer an UML angelehnten Notation, dass bei einer PMP-Datenschutzregel vier Komponenten eine zentrale Rolle spielen:

Service Feature. Eine App kann die von ihr angebotenen Funktionen in sogenannten Service Features kapseln. Berechtigungen werden nicht einer App, sondern deren Service Features zugeordnet. Der Nutzer kann diese Features beliebig aktivieren bzw. deaktiviern und somit auch die Berechtigungen, die eine App benötigt, ausweiten oder einschränken.

Resource. Ressourcen stellen aus App-Sicht Schnittstellen zu geschützten Systemdaten und -funktionen dar. Berechtigungsabhängig geben sie einer App die angeforderten Daten korrekt, in aggregierter Form oder völlig randomisiert weiter. Die Ressourcen selbst sind dabei allerdings kein fester Bestandteil der PMP oder der Android-Plattform, sondern eigenständige Services. Somit ist es möglich auch nachträglich weitere Ressourcen zu installieren oder bestehende Ressourcen zu aktualisieren. Darum muss sich der Nutzer allerdings nicht kümmern, sondern die PMP installiert eine Ressource bei Bedarf automatisch aus einem geschützten Datenarchiv – auf diese Weise kann gleichzeitig sichergestellt werden, dass sämtliche Ressourcen frei von Malware sind. Inhaltlich zusammengehörige Ressourcen können in Gruppen, den sogenannten *Resource Groups*, zusammengefasst werden.



(a) Installationsdialog (b) Datenschutzeinstellungen

Abb. 3: PMP Installationsprozess

Privacy Setting. Die Datenschutzeinstellungen werden von den Ressourcen definiert und durch den Nutzer konfiguriert. Eine Datenschutzeinstellung gibt dabei an, wie feingranular Berechtigungen für eine Ressource vergeben werden können. Dadurch ist es dem Nutzer beispielsweise möglich, die Verwendung von Positionsdaten zu verbieten, auf eine Genauigkeit von x Meter einzuschränken oder zu randomisieren.

Privacy Rule. Eine Datenschutzregel kombiniert diese Komponenten: Sie besteht aus einem Akteur (Service Feature), der Zugriff auf Daten (Ressource) erlangen will, und der Berechtigung, die der Nutzer (Datenschutzeinstellung) erteilt hat. Mittels kontextuellen Beschränkungen kann eine Regel noch weiter detailliert werden – z.B. dass eine Regel nur an bestimmten Orten oder nur zu festen Zeiten gelten soll. Existiert für einen Zugriff keine Regel, so wird, unter Annahme der Weltabgeschlossenheit, dieser Zugriff verweigert.

Aus Nutzersicht werden durch dieses erweiterte Berechtigungsrichtlinienmodell geringfügige Änderungen am App-Installationsprozess nötig. Abbildung 3a zeigt den neuen Verlauf. Zunächst wird überprüft, ob die PMP installiert ist und läuft. Im Erfolgsfall meldet sich die App anschließend bei der PMP an und der Nutzer kann einen initialen Regelsatz festlegen. Hierzu werden alle Service Features aufgelistet und der Nutzer kann sie einzeln aktivieren oder deaktivieren. Bei jeder Aktion wird er darüber informiert, welche Berechtigungen die App dafür erhält bzw. ihr entzogen werden, wie es in Abbildung 3b zu sehen ist. Über die GUI der PMP kann diese Konfiguration auch nachträglich jederzeit geändert werden.

Ein Berechtigungssystem ist nur sinnvoll, wenn es verständlich und ergonomisch ist [Fe12]. Daher bietet die PMP ein intuitives Nutzerinterface und eine Importfunktion für vorkonfigurierte Regelwerke. Darüber hinaus besitzt die PMP für Laien und Experten zwei unterschiedliche Nutzermodi, damit die einen nicht überfordert werden und die anderen dennoch das volle Leistungsspektrum ausschöpfen können. Weitere Details zur PMP finden sich in [SM13] und [St13].

5 Implementierungsstrategien

Betrachtet man, wie die in Abschnitt 3 vorgestellten Berechtigungssystemerweiterungen in die Android-Plattform eingebunden werden, so stellt man fest, dass dabei gänzlich heterogene Strategien verfolgt

wurden. Soll ein neues Berechtigungssystem – beispielsweise die PMP – eingeführt werden, stellt sich daher immer die Frage, welche Implementierungsstrategie zu präferieren ist. Diese Entscheidung hat allerdings gravierenden Einfluss auf die Nutzbarkeit des Systems. Daher werden in diesem Abschnitt unterschiedliche Implementierungsstrategien vorgestellt und miteinander verglichen.

5.1 Erweiterung des Berechtigungssystems von Android

Mit neuen Android-Versionen führt Google sporadisch neue Berechtigungen ein, erweitert oder verfeinert bestehende und entfernt als unnötig erachtete wieder. So ist beispielsweise die `READ_EXTERNAL_STORAGE`-Berechtigung zwar mit Android 4.1 vorgestellt, jedoch noch nicht aktiviert worden – d.h., Apps können diese Berechtigung bereits anfordern, aber der lesende Zugriffe auf den Externspeicher ist vorerst auch ohne diese möglich. Dadurch wird eine Übergangsphase geschaffen, die es Entwicklern erlaubt, für ihre Apps Updates zu veröffentlichen, in denen diese Berechtigung verwendet wird. Für den Nutzer ist hierbei der Vorteil, dass er sich an kein neues Berechtigungssystem gewöhnen muss und dennoch einen verbesserten Schutz erhält. Allerdings können auf diesem Weg keine kontextsensitiven Regeln eingeführt oder Datenschutzeinstellungen zur Laufzeit geändert werden. Der Nutzer erhält nach wie vor nur generische Informationen, wozu eine angeforderte Berechtigung benötigt wird. Daher ist der Einfluss, den eine solche Erweiterung auf den Datenschutz hat, nur sehr gering. Außerdem ist es möglich, dass nach einem automatischen Systemupdate manche Apps nicht mehr funktionieren, wenn mit dem Update neue Berechtigungen aktiv wurden. Der Nutzer muss demnach immer dafür Sorge tragen, dass er auch alle Apps auf den neusten Stand bringt – vorausgesetzt, seine Apps werden kontinuierlich an neue Android-Versionen angepasst. Ebenso problematisch ist es, wenn eine App nach einem Update Berechtigungen einer neuen Android-Version verwendet, für den Nutzer allerdings kein Systemupdate zur Verfügung steht – folglich kann er die App nicht mehr ausführen.

Aufgrund der schwerwiegenden Nachteilen und des marginalen Erweiterungspotentials, ist diese Implementierungsstrategie für die Einführung eines neuen Berechtigungssystems ungeeignet und stellt lediglich für Google eine Möglichkeit der sukzessiven Anpassung an Systemänderungen (z.B. durch die Ablösung veralteter APIs) oder zur Reaktion auf veränderte Gegebenheiten (z.B. durch die Einführung neuer Technologien, wie NFC) dar.

5.2 Anwendungsbasierte Implementierung

Die anwendungsbasierte Implementierung eines Berechtigungssystems kann man mit der Installation einer Sicherheitssoftware (z.B. eine Firewall) auf einem PC vergleichen. Das Betriebssystem verbleibt dabei unverändert und die Sicherheitssoftware unterliegt den gleichen Restriktionen und Bestimmungen, wie alle anderen Apps. Abhängig von diesen Rechten, kann sie andere Programme überwachen und, bei Bedarf, deren Interaktionen mit dem System einschränken oder vollständig unterbinden.

Für ein anwendungsbasiertes Berechtigungssystem bedeutet das, dass es Berechtigungen für den Zugriff auf alle zu schützenden Daten benötigt. Im Falle von Android müssen daher im Manifest

`uses-permission`-Tags für alle definierten Berechtigungen gesetzt sein. Mit diesen Rechten ausgestattet, kann es auf alle Daten zugreifen und diese bei Bedarf – wenn es der Nutzer erlaubt – an andere Apps – eventuell gekürzt oder randomisiert – weiterleiten. Außerdem muss sichergestellt werden, dass das Berechtigungssystem direkt bei Systemstart geladen wird und danach nicht mehr beendet werden kann.

Für Apps, die mit einem solchen System zusammenarbeiten, besteht die Notwendigkeit, dass diese sich bei jedem Start der App bei dem Berechtigungssystem anmelden, z.B. mittels einer Registrierungsaktivität, die der eigentlichen App vorgeschaltet wird. Auf diese Weise kann zeitgleiche sichergestellt werden, dass das Berechtigungssystem läuft bzw., sollte dies nicht der Fall sein, automatisch mit der App gestartet wird. Benötigt eine App Zugriff auf geschützte Daten, so erfolgt dies nicht über das Android-Application-Framework, sondern direkt über das Berechtigungssystem. D.h., die App fordert bei dem Berechtigungssystem beispielsweise die aktuelle Positionsdaten an, das Berechtigungssystem prüft, ob diese App aktuell die dafür benötigten Rechte besitzt und leitet, wenn dem so ist, anschließend die Daten – eventuell aggregiert oder randomisiert – weiter. Somit benötigen die Apps selbst keine `uses-permission`-Tags im Manifest mehr.

Ein großer Vorteil aus Nutzersicht ist hierbei, dass neben allen Apps, die mit dem Betriebssystem mitgeliefert werden (*System-Apps*), auch alte, unangepasste Drittsoftware (*Legacy-Apps*) weiterhin funktioniert. Legacy-Apps rufen das Berechtigungssystem nicht auf und erhalten benötigte Daten weiterhin direkt vom Android-System. Ein weiterer, sicherlich nicht zu unterschätzender, Vorteil dieser Methode ist, dass keinerlei Manipulationen an der Mobilplattform vorgenommen werden, da dies erhebliche Probleme, bis hin zu Hardware-Schäden nach sich ziehen kann und daher nur für erfahrene Nutzer zu empfehlen ist. Da das Berechtigungssystem jederzeit installiert bzw. entfernt werden kann, ohne dass dadurch sämtliche Systemeinstellungen erneut vorgenommen werden müssen, ist die Einstiegshürde für den Nutzer auch erheblich niedriger.

Allerdings ergeben sich bei dieser Strategie auch sicherheitsrelevante Nachteile. Da die Rechte des Berechtigungssystems nicht über die einer normalen App hinausgehen, kann es bei einer solchen Implementierung keinen universellen Schutz geben. Es kann nicht verhindert werden, dass Apps ihre Daten direkt vom Android-System beziehen. Dies kann der Nutzer allerdings bemerken, da in diesem Fall die App bei der Installation, wie bislang auch, Berechtigungen anfordern würde. Jedoch ist es fraglich, ob der Durchschnittsnutzer sich dessen bewusst ist. Hinzu kommt, dass der Nutzer zunächst das Berechtigungssystem von Hand installieren muss, da es nicht von Haus aus Teil der Plattform ist.

5.3 Ersetzung des Berechtigungssystems von Android

Soll das Android-Berechtigungssystem vollständig ersetzt werden, so muss bei der Installation jeder App sichergestellt werden, dass die App keine Android-Berechtigungen mehr verwendet. Hierfür gibt es grundsätzlich zwei unterschiedliche Herangehensweisen:

Manipulation der Berechtigungsinformationen. Jede apk-Datei muss vor der Installation entpackt, ihr Manifest um jedwede `uses-permission`-Tags erleichtert und anschließend neu verpackt und

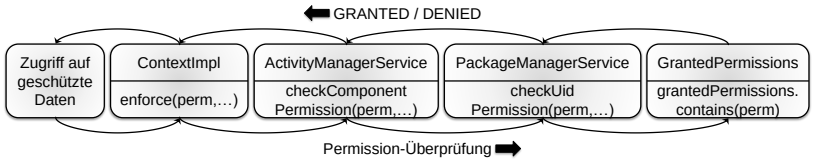


Abb. 4: Die Berechtigungsüberprüfung von Android

signiert werden. Dies kann z.B. ein App-Konverter (Abschnitt 5.5) bewerkstelligen, wodurch diese Strategie aus Nutzersicht die Vor- und Nachteile der Implementierung mit App-Konverter aufweist. Allerdings ergibt sich ein weiterer Nachteilen dadurch, dass bei dieser Strategie jeder App systematisch alle Berechtigungen entzogen werden. Daher muss bei Legacy-Apps mit häufigen Abstürzen gerechnet werden.

Manipulation der Berechtigungsprüfung. Die Android-Methode `enforce` überprüft, ob einer bestimmten App eine angegebene Berechtigung gewährt wurde. Abhängig davon werden ihr Datenzugriffe bewilligt oder verweigert. Der hierbei entstehende Informationsfluss ist in Abbildung 4 skizziert. Ein sehr einfacher, aber dennoch performanter, Lösungsweg besteht darin, dass die Methoden `enforce` derart manipuliert wird, dass sie konstant den Wert `PERMISSION_DENIED` zurückliefert. Dies hätte allerdings zur Folge, dass alle Apps das erweiterte Berechtigungskonzept unterstützen müssen bzw. keinerlei Berechtigungen zum Betrieb benötigen dürfen – alle System-Apps eingeschlossen!

Letzteres kann durch die Differenzierung zwischen System-Apps und Drittsoftware verhindert werden. Um diese Unterscheidung zu realisieren, genügt es den Speicherort einer App zu betrachten: System-Apps befinden sich in dem Verzeichnis `/system/app`, während Drittsoftware in `/data/app` installiert wird. Die Methode `enforce` muss daher lediglich die Paketinformationen einer App abfragen, zu denen u.a. auch der Dateipfad der apk-Datei gehört. Abhängig von dem Speicherort werden Berechtigungsanfragen von `enforce` nur dann an den *ActivityManagerService* weitergegeben, wenn diese von System-Apps kommen. Anfragen von Drittsoftware werden per se abgelehnt.

Bei dieser Variante ist für den Nutzer der Vorteil, dass alle System-Apps weiterhin funktionieren und Drittsoftware ausschließlich über das neue Berechtigungssystem Zugriff auf geschützte Daten erlangen kann.

Dieser Ansatz birgt allerdings auch einige Nachteile. Der gravierendste Nachteil für den Nutzer ist, dass auf diese Weise keine Legacy-Apps mehr lauffähig sind, da diesen alle Berechtigungen entzogen werden. Ein weiterer, zunächst nicht so offensichtlicher Nachteil betrifft die System-Performance. Die zusätzliche Überprüfung des Speicherorts erzeugt einen Performance-Overhead, der, für sich betrachtet, unbedeutend scheint. Allerdings wird die Methode `enforce` vom Android-System im Hintergrund mit sehr hoher Frequenz aufgerufen. Daher wirkt sich bereits dieser minimale Overhead hochgradig negativ auf das Laufverhalten und die Stabilität des kompletten Systems aus!

5.4 Implementierung als Alternative zu dem Berechtigungssystem von Android

Weniger rigoros, als die vollständige Ersetzung des Berechtigungssystems von Android (siehe Abschnitt 5.3), ist der parallele Betrieb des bestehenden Systems mit einem neu eingeführten, alternativen Berechtigungssystems. Hierfür muss es eindeutig sein, ob für eine App das alte oder das neue Berechtigungssystem zum Einsatz kommen soll. Eine Möglichkeit zur Identifikation, um welchen der beiden Typen es sich handelt, stellt die Analyse des Manifests dar, wenn das neue System keine `uses-permission`-Tags verwendet. Die Android-Methode `parsePackage` wertet zur Laufzeit einer App jeden Eintrag des Manifests aus und fasst die relevanten Informationen in einem `Package` zusammen. Diese Unterscheidung kann allerdings sehr aufwendig werden – man bedenke, dass eine Absenz von `uses-permission`-Tags nicht damit gleichzusetzen ist, dass eine App das neue Berechtigungssystem verwendet.

Für die PMP wäre diese Identifizierung sehr einfach, da sich eine App beim Start bei der PMP registriert. Für die Registrierungsaktivität ist ein Eintrag im Manifest nötig. Findet die Methode `parsePackage` daher diesen Eintrag, können alle `uses-permission`-Tags, die sich fälschlicherweise ebenfalls in dem Manifest befinden könnten, nicht an das System weitergegeben. Ohne diese Registrierungsaktivität lässt sich die PMP nicht nutzen und es kann folglich davon ausgegangen werden, dass das ursprüngliche Berechtigungssystems von der App genutzt wird. Da ein weiterer Bestandteil der von `parsePackage` ermittelten Informationen eine Liste (`requestedPermissions`) ist, in der alle Berechtigungen aus dem Manifest aufgeführt sind, reicht es aus, diese Methode so zu manipulieren, dass sie für Legacy-Apps die korrekte (und vollständige) Liste und für alle anderen Apps eine leere Liste übergibt.

Ein wesentlicher Vorteile bei dieser Implementierungsstrategie ist aus Nutzersicht, dass Legacy-Apps weiterhin uneingeschränkt lauffähig sind. Auch wenn die Identifikation des verwendeten Berechtigungssystems u.U. aufwendig ist, so muss diese Überprüfung nur einmalig beim Start einer App stattfinden. Daher fällt dieser Overhead weniger ins Gewicht.

Allerdings birgt diese Methode für den Nutzer auch Nachteile. Da die Identifikation des verwendeten Berechtigungssystems nicht zwangsläufig eindeutig ist, können *false positives* (die App verwendet das originär Berechtigungssystem, aber sie wird fälschlicherweise dem neuen System zugeordnet) oder *false negatives* (die App verwendet das neu eingeführte System, dies wird allerdings nicht erkannt) auftreten.

Bei der PMP können false negatives komplett ausgeschlossen werden – existiert die Registrierungsaktivität nicht, so kann die App nicht mit der PMP kommunizieren. False positives können hingegen auftreten, wenn eine Legacy-App eine Aktivität mit einer Signatur besitzt, die identisch mit der der PMP-Registrierungsaktivität ist. Da dies jedoch sehr unwahrscheinlich ist, kann die Zahl der false positives daher als vernachlässigbar klein angenommen werden. Neben diesem technischen Problem, existiert zusätzlich ein Akzeptanzproblem seitens der App-Entwickler. Da auch das alte Berechtigungssystem verwendet werden kann, das wesentlich weniger restriktiv bei der Vergabe von Zugriffsrechten ist, besteht aus Entwicklersicht keinerlei Veranlassung dazu, das neue System zu verwenden. Auch wenn die Konfigurationsmöglichkeiten hinsichtlich der zu schützenden Daten den

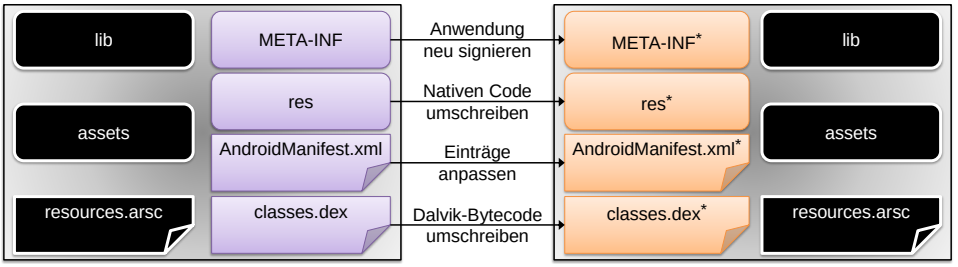


Abb. 5: Konzept für einen Android App-Konverter

Kundenwünschen entsprechen, würden nur wenige Apps an das neue System angepasst bzw. für dieses entwickelt werden.

5.5 Implementierung mit App-Konverter

Wie man in den obigen Abschnitten 5.1 bis 5.4 erkennen konnte, stellt der Umgang mit Legacy-Apps stets ein Problem dar, wenn ein bestehendes Berechtigungssystem erweitert bzw. ersetzt werden soll. Je nach Implementierungsstrategie bedeutet dies, dass jene Apps entweder weiterhin das alte (unzureichende) Berechtigungssystem verwenden oder nicht mehr lauffähig sind. Aus Nutzersicht sind beide Alternativen inakzeptabel.

Ein anderer Ansatz im Umgang mit Legacy-Apps stellt die Einführung eines neuen Berechtigungssystems in Kombination mit einem App-Konverter dar. Dieser Konverter durchsucht eine App nach datenschutzkritischen Systemaufrufen, die spezielle Berechtigungen benötigen (z.B. eine Positionsanfrage), und ersetzt diese im Code durch äquivalente Anfragen an das neue Berechtigungssystem. Zusätzlich entfernt der App-Konverter nicht länger benötigten Code aus der App (z.B. `uses-permission`-Tags aus dem Manifest) und fügt zusätzlichen Code hinzu, sofern dies für das neue Berechtigungssystem von Nöten ist.

Abbildung 5 skizziert, welche Anpassungen durchgeführt werden können. Sowohl der Dalvik-Bytecode (*classes.dex*), als auch die nativen Codefragmente (*res*) einer App können adaptiert werden, um die Datenzugriffe neu zu regeln, während in der *AndroidManifest.xml* Berechtigungen oder Aktivitäten eingetragen oder entfernt werden können. Anschließend muss die App erneut verpackt und signiert werden (*META-INF*). Bei diesem Prozess ergeben sich zwei Probleme:

Der Android-Verifizierer. Bei der Übersetzung von Java Quellcode in Dalvik-Bytecode nimmt der Compiler Codeoptimierungen und Variablenindexierungen vor. Der Android-Verifizierer prüft zur Laufzeit u.a., ob bei diesen Indizes Duplikate auftreten. Die Integration von vorkompiliertem Code in eine bestehende App kann allerdings zu Anomalien bei diesen Indexstrukturen führen. Ist dies der Fall, kann die App nicht mehr ausgeführt werden.

Die Einbindung von Bibliotheken. Aus App-Sicht sind Anfragen an das neue Berechtigungssystem Funktionsaufrufe einer importierten Bibliothek. Dies ist in Android allerdings erst möglich, wenn eine App vollständig initialisiert wurde, d.h. nach Abarbeitung der `onCreate`-Methode. Funktionsaufrufe, für die Berechtigungen erforderlich sind, werden daher aus dieser Methode ausgelagert. Unmittelbar nach der Initialisierung muss der ausgelagerte Code ausgeführt werden.

Reddy et al. [Re11] zeigen mit dem sogenannten *Redexer* allerdings, dass ein solches Werkzeug, das selbst die Manipulation von Dalvik-Bytecode, von dem kein Quellcode vorliegt, ermöglicht, dennoch implementiert werden kann.

Dieser Ansatz bietet dem Nutzer viele Vorteile. Zum einen bietet er die gleich Sicherheit, wie einer der in das Betriebssystem eingebetteten Ansätze (siehe Abschnitt 5.1, 5.3 und 5.4), da sichergestellt ist, dass Zugriffe auf geschützte Daten nur über das neue Berechtigungssystem erfolgen können. Hierzu sind bei diesem Ansatz allerdings keinerlei Manipulationen am Betriebssystem selbst erforderlich. Zum anderen ist ein nach diesem Ansatz entwickeltes Berechtigungssystem kompatibel zu Legacy-Apps – besser gesagt, es ist nicht nur sichergestellt, dass alle Legacy-Apps weiterhin funktionsfähig sind, sondern sie unterstützen, nachdem sie konvertiert wurden, das neue Berechtigungssystem.

Für den Nutzer besteht der größte Nachteil dieser Implementierungsstrategie darin, dass er bei jeder Neuinstallation und bei jeder Aktualisierung einer App diese erneut konvertieren muss – vergisst er dies, wird das alte Berechtigungssystem verwendet! Nicht zu unterschätzen ist die Tatsache, dass die Manipulation des Dalvik-Bytecodes zum Teil auf Heuristiken beruht. Dabei kann es in seltenen Fällen allerdings zu Fehlannahmen kommen, die gravierende Folgen für die Systemsicherheit haben können. Der Entwickler eines Berechtigungssystems muss dafür Sorge tragen, dass er das System und den App-Konverter an jedes Update der Mobilplattform anpasst. Der Nutzer muss darauf achten, dass er die für seine Plattform passende Version des Berechtigungssystems installiert.

Ein weniger offensichtlicher Nachteil ist, dass sich, abhängig von der Implementierung des App-Konverters, die Signatur der App ändert. Dies hat direkten Einfluss auf die Zugriffsrechte einer App, da Apps mit der selben Signatur Daten und sogar Codefragmente untereinander austauschen können. Eine Veränderung der Signatur kann daher zu Sicherheitsrisiken führen, die für den Nutzer praktisch nicht überschaubar sind.

5.6 Vergleich der Implementierungsstrategien aus Nutzersicht

In Tabelle 2 werden die wichtigsten Eigenschaften der vorgestellten Implementierungsstrategien gegenübergestellt. Es wird verglichen, ob Berechtigungssysteme, die nach der jeweiligen Strategie implementiert werden, nachträglich erweitert werden können (*Erweiterbarkeit*), ob Legacy-Apps damit lauffähig sind (*Kompatibilität*), ob die Mobilplattform manipuliert werden muss (*Systemupdate*), ob sich eine App nur über das Berechtigungssystem Zugriff auf geschützte Daten verschaffen kann (*Sicherheit*) und ob das Berechtigungssystem direkt einsatzfähig ist, ohne dass der Nutzer die Apps zunächst darauf vorbereiten muss (*Automatische Anwendung*). Offensichtlich gibt es keine optimale Implementierungsstrategie, sondern nur auf bestimmte Einsatzzwecke bezogene Optima.

Tab. 2: Vergleich der Implementierungsstrategien

Strategie	Erweiterbarkeit	Kompatibilität	Systemupdate	Sicherheit	Automatische Anwendung
5.1	×	×	√	√	√
5.2	√	√	×	×	√
5.3	×	×	√	√	√
5.4	×	√	√	√	√
5.5	√	√	×	√	×

6 Zusammenfassung und Ausblick

In diesem Papier stellen wir die wesentlichen Datenschutzmerkmale aktueller Mobilplattformen vor. Da für alle Plattformen, bei denen der Nutzer Kontrolle über – und damit auch Verantwortung für – den Datenschutz hat, ein gutes Berechtigungssystem immens wichtig ist, betrachten wir einige Berechtigungssysteme für Android. Wir analysieren, welche Einflüsse das Modell der Berechtigungsrichtlinien und die Implementierungsart auf die Charakteristika des Systems haben. So kann der Nutzer selbst entscheiden, welches System für seine Bedürfnisse am passendsten ist. Da keines der existierenden Systeme kontextsensitiv und absturzsicher ist, Veränderungen der Berechtigungsrichtlinien zur Laufzeit zulässt, Daten randomisieren kann und den Nutzer stets über die Auswirkungen der Berechtigungseinstellungen informiert, führen wir mit der PMP ein solches System ein.

Aktuelle Analysen zeigen, dass Nutzer mobiler Geräte entweder gar keine Verantwortung für den Schutz privater Daten (iOS) oder volle Kontrolle über diese Daten (Android) haben wollen [Vi13]. Da Android derzeit kein zufriedenstellendes Berechtigungssystem hat, gewinnen alternative Berechtigungssysteme, wie die PMP, immer mehr an Bedeutung. Weiterführende Evaluationen und Nutzerstudien sollen zeigen, inwiefern man die PMP besser auf die Bedürfnisse der Nutzer ausrichten kann und welche Implementierungsstrategie im Fall der PMP den größten Zuspruch erhält.

Danksagung. Die PMP wurde in einer engen Zusammenarbeit mit Google München entwickelt und fertiggestellt. Dabei erhielten wir in der Planungsphase nützliche Ratschläge und Denkanstöße und Unterstützung bei der Implementierung. Daher möchten wir uns bei Google für diese Hilfe und Unterstützung bei unserer Arbeit an der PMP sowie deren hilfreiche Ratschläge und Verbesserungsvorschläge bedanken. Außerdem danken wir unseren Kollegen Nazario Cipriani, Christoph Gröger und Carlos Lübbe für ihre Unterstützung sowie unseren Studenten Thorsten Berberich, Jakob Jarosch, Tobias Kuhn, Philipp Scholz und Marcus Vetter für deren Hilfe bei der Implementierung der PMP.

Literatur

[Ba12a] Backes, M.: New Approach Uncovers Data Abuse on Mobile End Devices, Techn. Ber., University Saarland, 2012.

[Ba12b] Banuri, H. et al.: An Android runtime security policy enforcement framework. Personal and Ubiquitous Computing 16/, S. 631–641, 2012.

[Be11] Beresford, A. R. et al.: MockDroid: trading privacy for application functionality on smartphones. In: HotMobile '11. 2011.

[BO11] Barrera, D.; van Oorschot, P.: Secure Software Installation on Smartphones. IEEE Security & Privacy 9/, S. 42–48, 2011.

[CHY12] Chan, P. P.; Hui, L. C.; Yiu, S. M.: DroidChecker: Analyzing Android Applications for Capability Leak. In: WISEC '12. 2012.

[En10] Enck, W. et al.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: OSDI '10. 2010.

[Fe11] Felt, A. P. et al.: Android Permissions Demystified. In: CCS '11. 2011.

[Fe12] Felt, A. P. et al.: Android Permissions: User Attention, Comprehension, and Behavior. In: SOUPS '12. 2012.

[Fr12] Fragkaki, E. et al.: Modeling and Enhancing Android's Permission System, Techn. Ber., Carnegie Mellon University, 2012.

[Ho11] Hornyack, P. et al.: These Aren't the Droids You're Looking For: Retrofitting Android to Protect Data from Imperious Applications. In: CCS '11. 2011.

[Je12] Jeon, J. et al.: Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In: SPSM '12. 2012.

[Le12] Leontiadis, I. et al.: Don't kill my ads! Balancing Privacy in an Ad-Supported Mobile Application Market. In: HotMobile '12. 2012.

[NKZ10] Nauman, M.; Khan, S.; Zhang, X.: Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In: ASIACCS '10. 2010.

[Re11] Reddy, N. et al.: Application-centric security policies on unmodified Android, Techn. Ber., University of California, Los Angeles, 2011.

[Ru11] Russello, G. et al.: YAASE: Yet Another Android Security Extension. In: PASSAT '11. 2011.

[Sh10] Shabtai, A. et al.: Google Android: A Comprehensive Security Assessment. IEEE Security & Privacy 8/, S. 35–44, 2010.

[SM13] Stach, C.; Mitschang, B.: Privacy Management for Mobile Platforms - A Review of Concepts and Approaches. In: MDM '13. 2013.

[SS12] Stach, C.; Schlindwein, L. F.: Candy Castle - A Prototype for Pervasive Health Games. In: PERCOM '12. 2012.

[St13] Stach, C.: How to Assure Privacy on Android Phones and Devices? In: MDM '13. 2013.

[Vi13] VisionMobile: Developer Economics 2013, Techn. Ber., Developer Economics, 2013.

[XSA12] Xu, R.; Saïdi, H.; Anderson, R.: Aurasium: Practical Policy Enforcement for Android Applications. In: USENIX Security '12. 2012.