Dynamische Analyse von Apple iOS Apps – Überwachung datenschutzrelevanter Vorgänge in Smartphone-Anwendungen

Masterarbeit im Fach Informatik

vorgelegt von

Christoph Settgast

geb. *13.11.1986* in *Kassel*

angefertigt am

Institut für Informatik
Lehrstuhl für Informatik 1 (IT-Sicherheitsinfrastrukturen)
Prof. Dr.-Ing. Felix Freiling
Friedrich–Alexander–Universität Erlangen–Nürnberg

Prüfer: *Prof. Dr.-Ing. Felix Freiling*
Betreuer: *Andreas Kurtz*

Abgabe der Arbeit: *21.05.2013*

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den *21.05.2013*

Christoph Settgast

Inhaltsverzeichnis

Abkürzungsverzeichnis iv

Abbildungsverzeichnis v

Tabellenverzeichnis vi

Listings viii

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Aufgabenstellung . . . 2

2 Hintergrund 5
- 2.1 Apple App Store . . . 5
- 2.2 Hardware . . . 8
- 2.3 Datenschutz . . . 10
  - 2.3.1 Personenbezogene Daten . . . 11
  - 2.3.2 Identifizierende Daten . . . 12
  - 2.3.3 Zugriffsautorisierung durch den Benutzer . . . 14
- 2.4 Code von Drittanbietern in iOS-Anwendungen . . . 15
  - 2.4.1 Personalisierte Werbung . . . 15
  - 2.4.2 Nutzungsstatistiken . . . 15
- 2.5 Technischer Hintergrund . . . 16
  - 2.5.1 Objective-C und C . . . 17
  - 2.5.2 Objective-C Runtime . . . 18
  - 2.5.3 Frameworks . . . 18
  - 2.5.4 Sandbox . . . 20
  - 2.5.5 Sicherheitsmaßnahmen in Anwendungen . . . 20
  - 2.5.6 API-Hooking . . . 22
  - 2.5.7 Jailbreaks . . . 23
  - 2.5.8 Entwicklungsumgebung Theos . . . 24
  - 2.5.9 Unit-Tests mit dem iOS-Simulator . . . 24
  - 2.5.10 Taint Tracking . . . 25

3 Aktueller Stand der Anwendungsanalyse 28
3.1 PiOS: Detecting Privacy Leaks in iOS Applications . . . 28
3.2 Challenges for Dynamic Analysis . . . 29
3.3 PSiOS: Bring Your Own Privacy & Security to iOS Devices . . . 30
3.4 Vision: Automated Security Validation of Mobile Apps at App Markets . . 30
3.5 AppsPlayground: Automatic Security Analysis . . . 31
3.6 App Genome Report . . . 32
3.7 Clueful . . . 32
3.8 Appthority . . . 33
3.9 SiRA: Automation in iOS Application Assessments . . . 34

4 Entwurf und Implementierung des Analyseverfahrens 36
4.1 Vergleich von statischer und dynamischer Analyse . . . 36
4.2 Konzept . . . 38
4.3 Auswahl der Anwendungen . . . 40
4.4 Vorgehensweise je Anwendung . . . 40
4.5 Implementierung des Analysetweaks . . . 41
4.5.1 Beobachter-Entwurfsmuster . . . 42
4.5.2 Testgetriebene Entwicklung . . . 42
4.6 Vorbereitung des Analysegeräts . . . 44
4.7 Realisierung der Ergebnis-Datenbank . . . 45

5 Implementierung der Analysekriterien 47
5.1 Personenbezogene Daten . . . 47
5.1.1 Adressbuch . . . 48
5.1.2 Kalender und Erinnerungen . . . 48
5.1.3 Fotos und Videos . . . 49
5.1.4 Aufenthaltsort, Geofencing und Location Monitoring . . . 50
5.1.5 Soziale Netzwerke . . . 50
5.1.6 Anruf-Historie . . . 51
5.2 Identifizierende Daten . . . 52
5.2.1 Unique Device Identifier . . . 52
5.2.2 Identifier for Vendor und Advertising Identifier . . . 53
5.2.3 MAC-Adressen . . . 53
5.2.4 Telefonnummer und ICCID . . . 54
5.3 Nutzungsstatistiken und Werbenetzwerke . . . 55
5.4 Sensoren . . . 57
5.4.1 Mikrofon . . . 57
5.4.2 Kamera . . . 58
5.4.3 Beschleunigungssensor, Rotationssensor, Kompass . . . 58
5.5 Sicherheitsmaßnahmen in Anwendungen . . . 60
5.5.1 Dateiverschlüsselung . . . 60

5.5.2 Transportverschlüsselung . . . 61
5.5.3 Passwortverwaltung . . . 61
5.6 Netzwerkverkehr . . . 62
5.6.1 HTTP-Requests . . . 62
5.6.2 Vollständiger Netzwerkverkehr . . . 62
5.7 Aktoren . . . 64
5.7.1 Lautsprecher . . . 65
5.7.2 LED-Kamera-Blitz . . . 65
5.7.3 Vibrationsmotor . . . 65

6 Test des Analyseverfahrens und Analyseergebnisse 67
6.1 Test des Analyseverfahrens . . . 67
6.1.1 Performance-Benchmark . . . 67
6.1.2 Beispielanalyse der Referenzanwendung SpyPhone . . . 68
6.1.3 Test durch Anwendung PrivateData . . . 70
6.2 Analyseergebnisse . . . 70
6.2.1 Top 300 der kostenlosen Anwendungen . . . 71
6.2.2 Vergleich zwischen Kategorien . . . 77

7 Zusammenfassung und Ausblick 81
7.1 Zusammenfassung . . . 81
7.2 Ausblick . . . 83

A Anhang 84
A.1 SpyPhone: Geöffnete Dateien . . . 84
A.2 Sandbox-Profil aus iOS-Version 4.0 . . . 86
A.3 Liste der analysierten Top 300 Anwendungen . . . 89
A.4 Listen der Top 50 Anwendungen aus ausgewählten Kategorien . . . 98

Literaturverzeichnis 108

Abkürzungsverzeichnis

ABI	Application Binary Interface
API	Application Programming Interface
ARC	Automatic Reference Counting
ASID	Advertising Identifier
CUFUA	CompleteUntilFirstUserAuthentication
CUO	CompleteUnlessOpen
ECID	Exclusive Chip ID
ER-Diagramm	Entity-Relationship-Diagramm
GDB	GNU Debugger
HTTP	Hypertext Transfer Protocol
ICCID	Integrated Circuit Card ID
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
IPC	Inter-Process Communication
MAC-Adresse	Media-Access-Control-Adresse
PIE	Position-Independent Executable
REST	Representational State Transfer
SSL	Secure Sockets Layer
UDID	Unique Device Identifier
VID	Identifier for Vendor
XMPP	Extensible Messaging and Presence Protocol

Abbildungsverzeichnis

1.1 Gesamtsystem DiOS 4

2.1 Anzahl der im Apple App Store verfügbaren Anwendungen 6
2.2 Funktionsweise des API-Hooking 23

4.1 Konzept des Analyseverfahrens 39
4.2 Beobachter-Entwurfsmuster 42
4.3 ER-Diagramm der Ergebnisdatenbank 46

5.1 Inhalt der Datei com.apple.commcenter.plist 55
5.2 Aufzeichnung des Netzwerkverkehrs 64

6.1 Testanwendung SpyPhone 69

Tabellenverzeichnis

2.1 Anzahl der Anwendungen pro Kategorie im Apple App Store 7
2.2 Preisverteilung im Apple App Store 8
2.3 Technische Daten des iPhone 4 9
2.4 Voraussetzungen für den Zugriff auf Sensoren 9
2.5 Voraussetzungen für den Zugriff auf Aktoren 10
2.6 Voraussetzungen für den Zugriff auf personenbezogene Daten 12
2.7 Voraussetzungen für den Zugriff auf identifizierende Daten 14
2.8 Dienstleister für Nutzungsstatistiken und Werbenetzwerke 16
2.9 Verfügbare Objective-C Klassen und Methoden 19
2.10 Unterschiede zwischen den Dateiverschlüsselungsmodi 21

3.1 Zusammenfassung des Forschungsstands 35

5.1 Beispielhafte call_history.db 52
5.2 Einträge der flags in call_history.db 52

6.1 Benchmark des Analyseverfahrens 68
6.2 Ergebnisse der Analysekriterien für die Anwendung SpyPhone 69
6.3 Zugriff auf identifizierende Daten für die Top 300 72
6.4 Zugriff auf personenbezogene Daten für die Top 300 72
6.5 Verwendete Sensoren der Top 300 73
6.6 Verwendete Aktoren der Top 300 73
6.7 Verwendete Sicherheitsmaßnahmen der Top 300 73
6.8 Verwendete Werbenetzwerke und Nutzungsstatistiken der Top 300 74
6.9 Netzwerkverkehr der Top 300 75
6.10 Meistgenutzte Domains im HTTP/S-Netzwerkverkehr der Top 300 76
6.11 Zugriff auf personenbezogene Daten im Kategorienvergleich 78
6.12 Zugriff auf identifizierende Daten im Kategorienvergleich 78
6.13 Verwendete Sensoren im Kategorienvergleich 79
6.14 Verwendete Werbenetzwerken und Nutzungsstatistiken im Kategorienvergleich 79
6.15 Verwendete Sicherheitsmaßnahmen im Kategorienvergleich 80

A.1 Geöffnete Dateien der Anwendung SpyPhone 84
A.2 Liste der Top 300 Anwendungen 89

A.3 Education Top 50 98
A.4 Entertainment Top 50 100
A.5 Lifestyle Top 50 102
A.6 Books Top 50 104
A.7 Business Top 50 106

Listings

2.1 Bildung des UDID 13
2.2 Beispiel für die Übersetzung von Objective-C Aufrufen in C-Aufrufe . . . 17

4.1 Testfall eines Kriteriums für die Erfassung einer Klassenmethode 43

5.1 Minimale API-Aufrufe zur Verwendung des Mikrofons 57

1 Einleitung

1.1 Motivation

Für Smartphones gibt es eine Vielzahl von populären Anwendungen[1], die teilweise kostenlos angeboten werden. Die kostenlosen Anwendungen finanzieren sich meist über Werbung. Diese wird möglichst passend auf den Nutzer zugeschnitten, um die Wahrscheinlichkeit zu steigern, dass der Nutzer dem Werbelink folgt. Daher werden vom Werbeanbieter möglichst viele Informationen über den Nutzer gesammelt.

Auf Smartphones gibt es in der Regel viele personenbezogenen Daten wie Telefonnummern, Adressen, Kalendereinträge, Anrufe/SMS, Fotos und den aktuellen Aufenthaltsort. Außerdem sind in den Geräten viele eindeutig identifizierende Merkmale wie die Telefonnummer, die Seriennummer, der „Unique Device Identifier“ (UDID), die „International Mobile Equipment Identity““ (IMEI), die „International Mobile Subscriber Identity“ (IMSI), die „Integrated Circuit Card ID“ (ICCID) sowie die MAC-Adresse der WLAN- und Bluetooth-Adapter vorhanden. Diese Merkmale identifizieren das Smartphone eindeutig. Da ein Smartphone meist nur von einer Person genutzt wird, wird die Annahme getroffen, dass vom Gerät auf dessen Nutzer geschlossen werden kann.

Android, Googles Betriebssystem für mobile Endgeräte, regelt den Zugriff auf diese Informationen über ein feingranulares Berechtigungskonzept. Dabei präsentiert das Betriebssystem bei der Installation einer Anwendung eine Liste der Berechtigungen, welche die Anwendung zur Ausführung benötigt. Der Nutzer kann dann entscheiden, ob er der Anwendung alle Berechtigungen erteilt und die Anwendung installiert. Andernfalls wird die Installation abgebrochen.

Bei iOS, Apples Betriebssystem für Smartphones und Tablets, wird dem Nutzer bei der Installation von Anwendungen keine Information zur Verfügung gestellt, auf welche Daten eine Anwendung zugreifen wird. Erst vor dem ersten Zugriff auf personenbezogene Daten wird pro Anwendung einmalig eine Berechtigung vom Nutzer eingeholt. Ob eine Anwendung mehrfach auf personenbezogene Daten zugreift, ist für den Nutzer nicht nachvollziehbar. Alle weiteren Daten, Sensoren und Aktoren werden nur in der Eingangskontrolle des App Stores überprüft. Der Nutzer ist daher vorwiegend von der Qualität dieser Eingangskontrolle abhängig.

[1] englisch „applications“, daher umgangssprachlich „Apps“ genannt

Da dieses Prüfverfahren nicht öffentlich dokumentiert ist, sind weder die eingesetzten Methoden noch die Prüfkriterien bekannt. So soll sichergestellt werden, dass die Anwendungen nicht gezielt durch entsprechende Programmierung die Kriterien umgehen. Der Nachteil hiervon ist, dass sich die Qualität des Analyseverfahrens nur schwer beurteilen lässt.

Außerdem besteht die Gefahr, dass unberechtigte Zugriffe auf personenbezogene Daten im Rahmen dieses Apple-Prüfverfahrens nicht erkannt werden. In der Vergangenheit kam es bereits vor, dass Datenschutzprobleme erst nach Veröffentlichung einer Anwendung festgestellt wurden. Die entsprechenden Anwendungen wurden dann nachträglich aus dem App Store entfernt. Ein Beispiel dafür ist die Anwendung des sozialen Netzwerks „Path“, welches das vollständige Adressbuch der Nutzer ohne Nachfrage an die Server von Path übertrug. Nachdem dies von Thampi festgestellt wurde [52], musste das soziale Netzwerk nachbessern. An diesem Beispiel wird deutlich, dass es automatisierter, öffentlich dokumentierter Analysemethoden für Smartphone-Anwendungen bedarf.

1.2 Aufgabenstellung

Die Arbeit ist Teil des Projekts „Dynamic Analysis of iOS Applications“ (DiOS). In diesem Projekt wird eine Infrastruktur zur automatisierten, dynamischen Analyse von iOS-Anwendungen konzipiert. Die Komponenten des Gesamtsystems sind in Abbildung 1.1 dargestellt.

Im Rahmen dieser Arbeit wird die dynamische Analyse von beliebigen Anwendungen konzipiert und realisiert. Die Automatisierung der Anwendungsausführung wird in der Arbeit von Weinlein [54] konzipiert und in dieser Arbeit zur Installation, Start, Beenden und Deinstallation von Anwendungen genutzt.

In der dynamischen Analyse werden folgende Kriterien berücksichtigt:

- Das Auslesen personenbezogener Daten wie der Inhalt des Adressbuchs und Kalenders, Fotos, Videos und der aktuelle Aufenthaltsort
- Das Auslesen eindeutig identifizierender Merkmale wie UDID und MAC-Adresse
- Das unverschlüsselte Speichern von Dateien
- Verwendung der im Gerät gespeicherten Account-Daten für Soziale Netze wie Facebook oder Twitter
- Die Übertragung von personenbezogenen Daten über das Netzwerk

Zuerst wird festgelegt, welche API-Funktionsaufrufe diese Analysekriterien erfassen sollen und dann werden diese API-Funktionen per API-Hooking abgefangen. Für die Methodik des API-Hooking wird beim Start einer Anwendung zusätzlich eine in dieser Ar-

beit entworfene dynamische Bibliothek geladen. In dieser Bibliothek werden bestimmte Funktionen der System-API überschrieben. Aufrufe der Anwendung werden so abgefangen und in die Bibliothek umgeleitet. Dort werden die API-Aufrufe protokolliert.

Bereits das Auslesen von Informationen wird ermittelt und festgehalten. Darüber hinaus ist beispielsweise eine Übertragung personenbezogener Daten zum Hersteller der Anwendung von Interesse. Hier werden im Rahmen der Arbeit Möglichkeiten aufgezeigt, Taint Tracking für iOS zu implementieren. Mittels Taint Tracking werden Daten an Quellen markiert, sodass die Markierung an anderer Stelle erkannt werden kann. Damit ist eine Erkennung der Übertragung von personenbezogenen Daten möglich.

Abschließend wird in einer Datenbank festgehalten, welche Analysekriterien für die jeweilige Anwendung zutreffen. Die Datenbank soll zur besseren Skalierung und Datensicherheit unabhängig vom Gerät bereitstehen. Die Analysebibliothek und die Datenbank kommunizieren per Web-Schnittstelle (REST).

In der Datenbank wird jeder Untersuchungslauf mit seinen Ergebnissen festgehalten und der jeweiligen Anwendung in der untersuchten Version zugeordnet. Es ist dann möglich, die Ergebnisse nach einzelnen Kriterien zu filtern, um so eine übersichtliche Auswertung in zusammengefasster Form zu gewinnen.

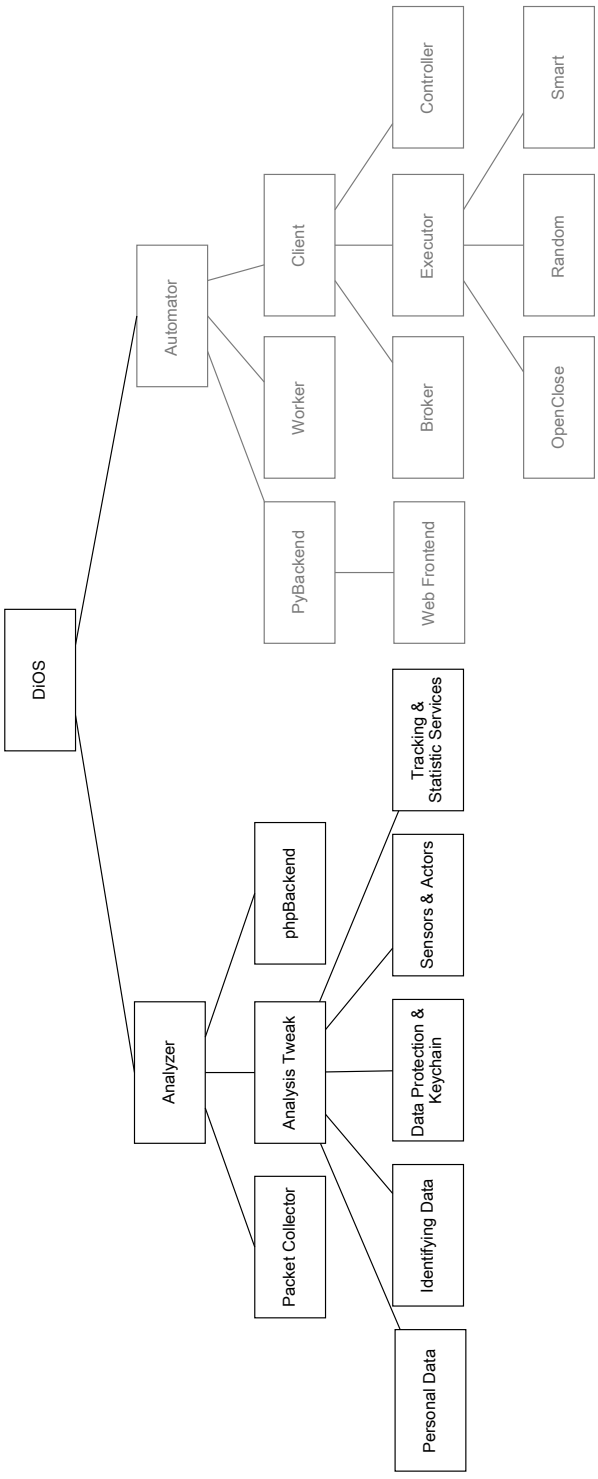

Abbildung 1.1: Gesamtsystem DiOS. In dieser Arbeit werden die Komponenten von „Analyzer“ zur dynamischen Analyse konzipiert und realisiert. Die Komponenten von „Automator“ werden in der Arbeit von Weinlein [54] konzipiert und in dieser Arbeit genutzt.

2 Hintergrund

Die Zahl der Smartphone-Nutzer in Deutschland stieg im Zeitraum von Januar 2009 bis Dezember 2012 von 6,3 auf 31 Millionen [21]. Geräte mit Apple iOS besitzen daran im Januar 2013 21,3 % Marktanteil [22]. Aus dieser großen Verbreitung von iOS-Smartphones resultiert auch ein entsprechend großes Angebot an Anwendungen, die von Drittanbietern programmiert wurden.

In diesem Kapitel werden die Grundlagen der Apple Smartphones mit ihrem zugehörigen App Store vorgestellt. Darauf folgt eine Beschreibung von Datenschutzaspekten bei Smartphones. Des Weiteren werden die technischen Rahmenbedingungen angesprochen, die bei Smartphones zu beachten sind. Außerdem werden die grundlegenden Techniken erläutert, die eine Analyse von Smartphone-Anwendungen ermöglichen. Abschließend wird der Stand der Technik zur Analyse von Smartphone-Anwendungen vorgestellt.

2.1 Apple App Store

Anwendungen von Drittanbietern für das iPhone werden ausschließlich im Apple App Store angeboten. Dort wurden März 2013 bereits 800.000 Anwendungen angeboten [1]. Im Zeitraum von März 2012 bis März 2013 kamen dabei über 215.000 neue Anwendungen hinzu [5]. Die rasant ansteigende Anzahl der verfügbaren Anwendungen wird in Abbildung 2.1 deutlich.

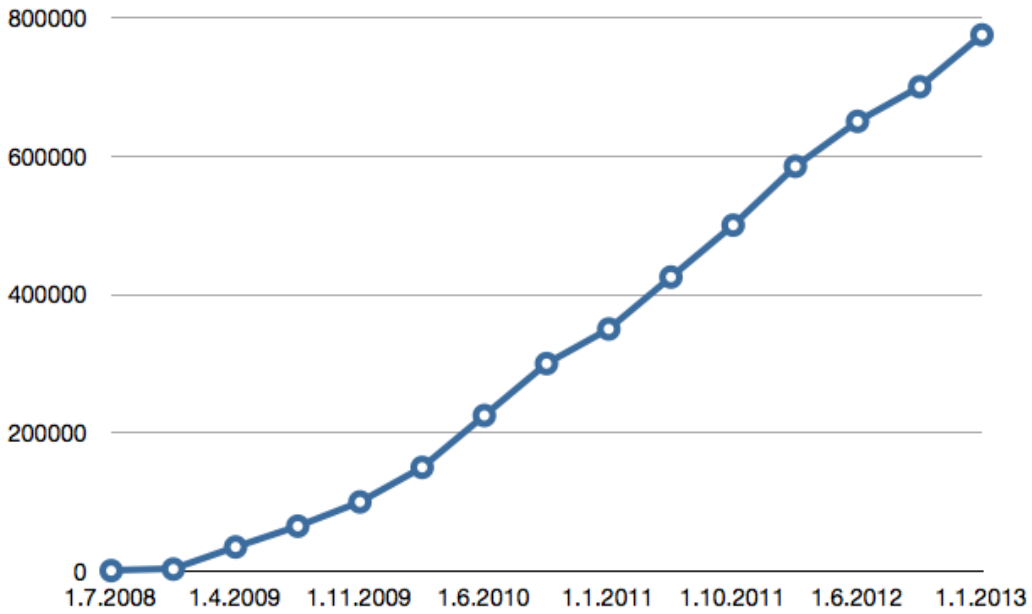

Abbildung 2.1: Anzahl der im Apple App Store weltweit verfügbaren Anwendungen von Juni 2008 bis Januar 2013 [5].

Die Anwendungen werden im Apple App Store in 22 unterschiedlichen Kategorien angeboten. Eine Anwendung ist dabei immer genau einer Kategorie zugeordnet. Die zahlenmäßig größten Kategorien sind Spiele, Bildungs- und Unterhaltungs-Anwendungen (siehe Tabelle 2.1).

Kategorie	Anzahl
Games	142.136 (16,80%)
Education	90.861 (10,74%)
Entertainment	75.655 (8,94%)
Lifestyle	68.963 (8,15%)
Books	55.823 (6,60%)
Business	55.702 (6,58%)
Utilities	50.207 (5,94%)
Travel	43.218 (5,11%)
Music	31.161 (3,68%)
Reference	26.174 (3,09%)
Sports	24.758 (2,93%)
Productivity	24.039 (2,84%)
News	23.033 (2,72%)
Healthcare & Fitness	22.659 (2,68%)
Photography	21.548 (2,55%)
Finance	19.859 (2,35%)
Medical	18.370 (2,17%)
Social Networking	16.788 (1,98%)
Food & Drink	14.019 (1,66%)
Navigation	12.486 (1,48%)
Catalogs	3.958 (0,47%)
Weather	3.588 (0,42%)

Tabelle 2.1: Anzahl der Anwendungen pro Kategorie im US-amerikanischen Apple App Store, Stand 15.04.2013 [1]

Alle so angebotenen Anwendungen werden vor ihrer Veröffentlichung von Apple in einem nicht-öffentlichen Verfahren bezüglich Datenschutz und Datensicherheit begutachtet. Dabei wird nach Beschreibung von Apple die Funktionsbeschreibung der Anwendungen mit den tatsächlichen Funktionen abgeglichen. Dies soll die Nutzer vor Schadsoftware, wie Trojanern, bewahren [26]. Da über das Prüfungsverfahren keine Details publiziert werden, kann dessen Wirkung nur schwierig eingeschätzt werden.

Preis P in US-Dollar	Anzahl	Anteil in %
0	482.839	57,08 %
$0 < P \leq 1$	179.786	21,05 %
$1 < P \leq 2$	74.072	8,76 %
$2 < P \leq 3$	36.232	4,28 %
$3 < P \leq 5$	36.679	4,34 %
$5 < P \leq 10$	22.789	2,69 %
$10 < P \leq 25$	8.664	1,02 %
$P > 25$	4.849	0,57 %
Gesamtzahl	845.910	100,00 %

Tabelle 2.2: Preis P der Anwendung und Anzahl sowie Anteil in % im US-amerikanischen Apple App Store, Stand 15.04.2013 [1]

57% der Anwendungen werden kostenlos angeboten (siehe Tabelle 2.2). Ein Teil der Anwendungen wird von Programmierern in ihrer Freizeit erstellt und kostenlos zur Verfügung gestellt.

Ein weiterer Teil der kostenlosen Anwendungen generieren auf andere Art Einnahmen, da sie Bestandteil einer Dienstleistung sind, die nicht über den App Store bezahlt wird. Als Beispiel sei eine Anwendung eines öffentlichen Nahverkehrsbetriebs genannt, der Fahrplaninformationen bereitstellt. In solchen Fällen muss die Anwendung selbst keine Einnahmen generieren.

Wenn die Anwendung nicht Teil einer Dienstleistung ist und kostenlos angeboten wird und die Entwickler Geld verdienen oder wenigstens ihre Entwicklungskosten kompensieren möchten, muss die Anwendung über ihre Nutzer Einnahmen erzielen. Hierzu wird Werbung eingesetzt, die teils auf den Nutzer zugeschnitten wird. Mit dieser Personalisierung wird versucht, die Wahrscheinlichkeit zu erhöhen, mit welcher der Nutzer dem Werbelink folgt. Diese Werbung wird von Werbenetzwerken angeboten.

In dieser Arbeit werden ausschließlich kostenlose Anwendungen analysiert, da sie die Mehrzahl der Anwendungen im App Store sind, da die Hersteller von kostenlosen Anwendungen über Werbung oder Profilbildung Geld einnehmen müssen und da die Analyse von kostenlosen Anwendungen mit geringem finanziellen Aufwand möglich ist.

2.2 Hardware

Bei Smartphones existieren hinsichtlich der Ausstattung sehr unterschiedliche Modelle. Für Analysen mit Geräten der Apple iOS-Plattform sind zur Zeit das iPhone 4, das

iPhone 4s und das iPhone 5 erhältlich. Als Hardwareplattform steht für die folgenden Analysen das Apple iPhone 4 zur Verfügung.

Die technischen Daten des Apple iPhone 4 sind in Tabelle 2.3 aufgeführt.

Prozessor	A4 CPU, 1,0 GHz, armv7-Instruktionssatz
Arbeitsspeicher	512 MB
Persistenter Speicher	8 GB Flash
Netzwerkadapter	WLAN, GPRS/UMTS, Bluetooth
Sensoren	GPS, Kompass, Mikrofon, Kamera, Beschleunigungssensor, Rotationssensor, Helligkeitssensor, Abstandssensor
Aktoren	Lautsprecher, Vibrationsmotor, LED-Kamera-Blitz

Tabelle 2.3: Technische Daten des iPhone 4

In Tabelle 2.4 sind die Voraussetzungen angegeben, unter denen eine Anwendung auf die Sensoren zugreifen kann. Auf die Aktoren kann eine Anwendung gemäß den Voraussetzungen in Tabelle 2.5 zugreifen.

Sensor	Version	Zugriff
Abstandssensor	ab 3.0	I
Beschleunigungssensor	ab 2.0	I
Helligkeitssensor	–	X
Mikrofon	ab 2.0	I
Kamera	ab 4.0	I
Kompass	ab 3.0	A
	ab 4.0	I
Rotationssensor	ab 4.0	I

Tabelle 2.4: Voraussetzungen für den Zugriff auf die Sensoren: ohne Autorisierung (I), nach einmaliger Autorisierung (A) oder kein Zugriff (X). Mit angegeben sind die Versionen der iOS-Software, bei denen die Zugriffsregelung verändert wurde [6].

Aktor	Version	Zugriff
Lautsprecher	ab 2.0	I
LED-Kamera-Licht	ab 4.0	I
Vibrationsmotor	ab 2.0	I

Tabelle 2.5: Voraussetzungen für den Zugriff auf die Aktoren: ohne Autorisierung (I). Mit angegeben sind die Versionen der iOS-Software, bei denen die Zugriffsregelung verändert wurde [6].

Diese Sensoren lassen sich aufgrund ihrer Präzision zweckentfremden. Marquardt et al. zeigen dies anhand des Beschleunigungssensors [41]. Sie verwenden ein Smartphone mit Hilfe des Beschleunigungssensors als Passwort-Logger. Dazu messen sie mit dem Sensor die Erschütterungen, welche durch Tastaturanschläge einer daneben liegenden Tastatur verursacht werden. Aus der Messung rekonstruieren sie die eingegebenen Zeichen und speichern diese.

Mit der zur Verfügung stehenden CPU-Leistung und der meist ständig verfügbaren Netzwerkverbindung lassen sich die Sensoren auf vielfältige Weise miteinander verknüpfen. Wie am Beispiel des Beschleunigungssensors gezeigt, sind dabei Szenarien denkbar, die vom Nutzer weder bemerkt noch nachvollzogen werden können. Deshalb ist eine Untersuchung jeder einzelnen Anwendung auf ihre Nutzung von Sensoren, Aktoren und Daten notwendig.

2.3 Datenschutz

Im deutschen Bundesdatenschutzgesetz (BDSG) sind die personenbezogenen Daten das Schutzobjekt. Diese sind folgendermaßen definiert: „Personenbezogene Daten sind Einzelangaben über persönliche oder sachliche Verhältnisse einer bestimmten oder bestimmbaren natürlichen Person (Betroffener).“ (§3 (1) BDSG[1]).

Moderne Smartphones speichern aufgrund ihrer Konzeption als Kombination diverser Einzelgeräte im Verlauf ihrer Benutzung eine Vielzahl solcher personenbezogener Daten. In einem Smartphone sind beispielsweise Adressbuch, Kalender, Foto- und Videokamera, Browser, Email-Programm und weitere Anwendungen standardmäßig integriert. Jede dieser Komponenten enthält personenbezogene Daten. Daneben stellen Smartphones aufgrund ihrer computerähnlichen Konzeption diverse identifizierende Merkmale ihren Anwendungen zur Verfügung.

[1] Bundesdatenschutzgesetz in der Fassung der Bekanntmachung vom 14. Januar 2003 (BGBl. I S. 66), zuletzt geändert durch Artikel 1 des Gesetzes vom 14. August 2009 (BGBl. I S. 2814)

2.3.1 Personenbezogene Daten

Ein Smartphone ist mit den vorinstallierten Anwendungen schon darauf vorbereitet, eine Vielzahl an personenbezogenen Daten zu speichern. Darüber hinaus können durch den Benutzer installierte Anwendungen weitere personenbezogene Daten vorhalten. Um die Analyse einzugrenzen, wird nur auf die Datenkategorien eingegangen, die schon vom Hersteller vorgesehen sind.

Das Adressbuch enthält nach der Definition aus §3 (1) BDSG ausschließlich personenbezogene Daten und der Inhalt fällt daher unter den Datenschutz. Der Kalender mit Einträgen zu Terminen enthält ebenso personenbezogene Daten. Der Aufenthaltsort des Smartphonebesitzers und die Historie der Aufenthaltsorte stellen personenbezogene Daten dar, welche somit unter den Datenschutz fallen. Da der aktuelle Ort zur Aufnahmezeit in den aufgenommenen Fotos und Videos hinterlegt wird, sind diese für den Benutzer ebenfalls schützenswert.

Apple iOS bietet eine „Location Monitoring“ genannte Funktion an. Mit dieser wird eine Anwendung benachrichtigt, sobald sich der Aufenthaltsort signifikant verändert hat. Das Einbuchen in eine neue Mobilfunk-Basisstation wird dabei als signifikante Änderung des Ortes vom Betriebssystem festgelegt. „Geofencing“ ist eine weitere Funktion, mit der Anwendungen benachrichtigt werden können, sobald das Gerät einen vorher festgelegten Bereich betritt oder verlässt. Aus den Ortsinformationen des „Location Monitoring“ und „Geofencing“ lässt sich ein Bewegungsprofil erstellen. Deswegen sind diese Informationen personenbezogene Daten, die unter den Datenschutz fallen.

In Tabelle 2.6 sind die Voraussetzungen angegeben, unter denen eine Anwendung auf personenbezogene Daten zugreifen kann.

Daten	Version	Zugriff
Adressbuch	bis 5.1.1	I
	ab 6.0	A
Anruf-Historie	bis 4.3.5	I
	ab 5.0	X
Aufenthaltsort	ab 2.0	A
Emails	–	X
Erinnerungen	bis 5.1.1	I
	ab 6.0	A
Fotos, Videos	bis 5.1.1	I
	ab 6.0	A
„Geofencing“	ab 4.0	A
Kalender	bis 5.1.1	I
	ab 6.0	A
Kurznachrichten	–	X
Lesezeichen	–	X
„Location Monitoring“	ab 4.0	A
Musik-Metadaten	ab 3.0	I
Notizen	–	X
Web-Historie	–	X
Zugang zu sozialen Netzwerken	ab 5.0	A

Tabelle 2.6: Voraussetzungen für den Zugriff auf personenbezogene Daten: ohne Autorisierung (I), nach einmaliger Autorisierung (A) oder kein Zugriff (X). Mit angegeben sind die Versionen der iOS-Software, bei denen die Zugriffsregelung verändert wurde [6].

Personenbezogene Daten werden von der Analyse ausgenommen, wenn sie Anwendungen von Drittanbietern nicht zugänglich sind. Hierzu gehören die Historie der besuchten Seiten im Webbrowser, Lesezeichen, Notizen, gesendete und empfangene Emails und Kurznachrichten.

2.3.2 Identifizierende Daten

„Identifizierende Daten“ werden in dieser Arbeit als solche Daten definiert, mit denen es Anwendungen möglich ist, ein Gerät über die Zeit hinweg und über Anwendungsgrenzen hinweg wiederzuerkennen. Da ein Smartphone in der Praxis nur von einer Person verwendet wird, kann mit der Zuordnung zu einem Gerät auch die Zuordnung zu einer Person als gegeben angesehen werden. Identifizierende Daten werden für die Auswahl von Werbung für den jeweiligen Nutzer verwendet.

Für Anwendungen von Drittanbietern eignen sich zur Identifizierung des Geräts prinzipiell die Seriennummer, die Telefonnummer, der „Unique Device Identifier“ (UDID), MAC-Adressen von WLAN- und Bluetoothschnittstelle, die „International Mobile Station Equipment Identity“ (IMEI), die „International Mobile Subscriber Identity“ (IMSI) und die „Integrated Circuit Card ID“ (ICCID). Diese bleiben für die Lebenszeit des Geräts konstant, im Gegensatz zu den identifizierenden Daten, welche im Browser verwendet werden, beispielsweise Cookies. Daher ist es von besonderem Interesse, wie viele und welche Anwendungen diese unveränderlichen identifizierenden Daten nutzen.

Der UDID wird dabei als Hash-Wert aus unveränderlichen Daten generiert [53]. Listing 2.1 zeigt das Vorgehen schematisch. Hintereinander gefügt werden die Seriennummer, dann bei Geräten vor dem Verizon iPhone 4 die IMEI, bei späteren Geräten stattdessen die „Exclusive Chip ID“ (ECID), danach die WLAN-MAC-Adresse und die Bluetooth-MAC-Adresse. Über diese wird ein SHA1-Hash gebildet. Die hexadezimale Darstellung dieses Hash-Werts ergibt den UDID.

```
1 // Smartphones vor dem Verizon iPhone 4
2 // (inklusive des in der Analyse verwendeten iPhone~4)
3
4 UDID = SHA1(Seriennummer + ECID + WLAN-MAC-Adresse + Bluetooth-MAC-Adresse)
5
6 // Smartphones nach dem Verizon iPhone 4
7 UDID = SHA1(Seriennummer + IMEI + WLAN-MAC-Adresse + Bluetooth-MAC-Adresse)
```

Listing 2.1: Bildung des UDID

In iOS existieren seit Version 6.0 zwei weitere Kennungen, welche aber während der Lebensdauer des Geräts verändert werden können. Die erste Kennung ist der „Identifier for Vendor“ (VID), welcher das Gerät jeweils für einen Hersteller von Anwendungen eindeutig identifiziert. Dieser ist eindeutig, so lange mindestens eine Anwendung des Herstellers auf dem Gerät installiert ist. Die zweite Kennung ist der sogenannte „advertisingIdentifier“ (ASID), der über alle Anwendungshersteller hinweg eindeutig ist. Diese kann in iOS Version 6.0 durch das vollständige Zurücksetzen des Geräts verändert werden. Ab Version 6.1 kann sie gezielt in den Systemeinstellungen zurückgesetzt werden. Durch die Einführung dieser speziellen Kennungen soll der Handel mit an Kennungen gebundenen Nutzungsprofilen eingeschränkt werden.

In Tabelle 2.7 sind die Voraussetzungen angegeben, unter denen eine Anwendung auf identifizierende Daten zugreifen kann. Dabei ist ersichtlich, dass der Zugriff auf identifizierende Daten entweder immer ohne Nachfrage oder gar nicht möglich ist. Die WLAN-MAC-Adresse, der UDID und dessen designierte Nachfolger ASID und VID sind dabei grundsätzlich verfügbar, während die Seriennummer des Geräts oder die im Mobilfunk verwendeten Kennungen wie IMEI und IMSI für Anwendungen nicht zur Verfügung stehen.

Darüber hinaus waren in iOS Version 3 und 4 die Telefonnummer in internationaler Schreibweise und die ICCID für alle Anwendungen immer verfügbar.

Daten	Version	Zugriff
ASID	ab 6.0	I
Bluetooth-MAC-Adresse	–	X
ICCID	bis 4.3.5	I
	ab 5.0	X
IMEI	–	X
IMSI	–	X
Seriennummer	–	X
Telefonnummer	bis 4.3.5	I
	ab 5.0	X
UDID	ab 2.0	I
VID	ab 6.0	I
WLAN-MAC-Adresse	ab 2.0	I

Tabelle 2.7: Voraussetzungen für den Zugriff auf identifizierende Daten: ohne Autorisierung (I) oder kein Zugriff (X). Mit angegeben sind die Versionen der iOS-Software, bei denen die Zugriffsregelung verändert wurde [6].

2.3.3 Zugriffsautorisierung durch den Benutzer

Bei Apple iOS bis einschließlich Version 5.1 musste der Nutzer nur den ersten Zugriff auf den aktuellen Ort durch eine Anwendung autorisieren. Bei Folgezugriffen wurde eine implizite Zustimmung des Nutzers angenommen. Die Zustimmung gilt bis zu ihrem expliziten Widerruf über die Systemeinstellungen.

Ab Version 6.0 wird die Zustimmung des Nutzers auch für jeden ersten Zugriff auf das Adressbuch, den Kalender, die Erinnerungen, Facebook- und Twitter-Accounts und Fotos und Videos eingeholt. Die Autorisierung jedes weiteren Zugriffs einer Anwendung wird implizit angenommen. Dabei ist anzumerken, dass Anwendungen teils alle Autorisierungen beim ersten Start einholen. Zu diesem Zeitpunkt kann der Benutzer möglicherweise nicht feststellen, für welche Aktion der Anwendung die Autorisierung eingeholt wird.

An einem Beispiel wird dies deutlich: Eine fiktive Anwendung zur Pizza-Bestellung fragt direkt beim ersten Start, ob auf den aktuellen Ort zugegriffen werden darf. Der Benutzer gibt seine Zustimmung in der Annahme, dass die Information nur dazu verwendet wird, ihm nahegelegene Pizzalieferdienste vorzuschlagen. Die Anwendung kann die einmal eingeholte Autorisierung im weiteren Verlauf der Benutzung jedoch noch einmal verwenden, um ortsbezogene Werbung anzuzeigen.

Im Gegensatz dazu ist der Zugriff auf alle identifizierenden Daten, Sensoren und Aktoren für jede Anwendung ohne Autorisierung durch den Benutzer möglich.

2.4 Code von Drittanbietern in iOS-Anwendungen

Sowohl für das Anzeigen von Werbung als auch zur Erfassung von Nutzungsstatistiken stehen Herstellern von iOS-Anwendungen verschiedene Drittanbieter-Frameworks zur Verfügung. Diese Frameworks werden statisch in den Code der Anwendung integriert und bei ihrer Initialisierung aufgerufen. So ist zur Ausführungszeit nicht mehr festzustellen, ob ein API-Aufruf vom Hersteller selbst programmiert wurde oder ob ein Drittanbieter-Framework diesen ausführt.

Im folgenden Abschnitt werden die Details und Zusammenhänge von personalisierter Werbung und Nutzungsstatistiken erläutert.

2.4.1 Personalisierte Werbung

Neben „iAd“, der von Apple mit iOS Version 4.0 eingeführten Möglichkeit, Werbung in Anwendungen zu zeigen und so Einnahmen auch mit kostenlosen Anwendungen zu generieren, gibt es weitere Anbieter von Werbebannern in Anwendungen. Die Vergütung für die Werbung erfolgt in der Regel pro Klick. Um die Wahrscheinlichkeit zu erhöhen, dass ein Nutzer auf ein Werbebanner klickt, wird die angezeigte Werbung nach Möglichkeit auf die Interessen und den aktuellen Ort des Nutzers abgestimmt. Um die Interessen der Nutzer zu ermitteln, wird Profilbildung eingesetzt. Diese Profilbildung kann sowohl durch Werbeanbieter als auch durch Anbieter von Nutzungsstatistiken erfolgen. Manche Anbieter wie Flurry bieten beide Dienstleistungen aus einer Hand an. Um diese Profile dann über Anwendungs- und Anbietergrenzen hinweg zuordnen zu können, werden die identifizierenden Daten aus Kapitel 2.3.2 verwendet.

In bisherigen Analysen von Egele et. al [26] ist der weit verbreitetste Anbieter für solche Werbeformen „AdMob“, der in 38 % der 1407 untersuchten Anwendungen vorgefunden wurde. Weitere Anbieter wie „Mobclix“ und „AdWhirl“ wurden nur in 3 % beziehungsweise 1 % der Anwendungen vorgefunden.

2.4.2 Nutzungsstatistiken

In der Webentwicklung sind mit Google Analytics und vergleichbaren Anbietern sehr detaillierte Nutzungsstatistiken etabliert. Auch bei Anwendungen für Smartphones gibt es eine Vielzahl entsprechender Dienstleister. Diese bieten den Anwendungsherstellern Informationen über Häufigkeit und Art der Nutzung einer Anwendung. Gleichzeitig ermöglichen diese Statistiken die Erstellung von Nutzerprofilen. Dabei ist es wichtig, Nutzer eindeutig zu identifizieren, beispielsweise anhand der in Kapitel 2.3.2 genannten Daten wie MAC-Adresse und UDID.

In der Analyse von Egele et al. [26] sind diese Anbieter nur sehr selten vertreten. Pinchmedia ist in 6 %, Flurry in 4 % der untersuchten Anwendungen vorhanden.

In dieser Arbeit werden die in Tabelle 2.8 genannten Anbieter von Werbung und Nutzungsstatistiken erfasst. Dies sind sowohl die noch aktiven Anbieter aus [26] als auch weitere Anbieter, die sich seit den bisherigen Analysen am Markt etablierten.

Dienstleister	Art
Flurry, Version 3 & 4	Werbung & Nutzungsstatistik
Google Analytics, Version 1 & 2	Nutzungsstatistik
Admob	Werbung
TestFlight	Nutzungsstatistik
MobAge	Werbung
MobileAppTracker	Nutzungsstatistik
Mixpanel	Nutzungsstatistik
KISSMetrics	Nutzungsstatistik
Apsalar	Werbung & Nutzungsstatistik
Tapjoy	Werbung
Medialets	Werbung
Localytics	Nutzungsstatistik
Bango	Nutzungsstatistik
Distimo	Nutzungsstatistik
InMobi	Werbung
Smaato	Werbung
AdColony	Werbung
Appsfire	Werbung
Chartboost	Werbung
JumpTap	Werbung
PinchMedia	Nutzungsstatistik

Tabelle 2.8: In der Analyse erfasste Dienstleister für Nutzungsstatistiken und Werbenetzwerke

2.5 Technischer Hintergrund

Die Softwareentwicklung für Smartphones bietet Anwendungsherstellern eine Vielzahl von Möglichkeiten und ist nicht mit Anwendungsentwicklung für klassische Mobiltelefone vergleichbar. Sowohl die Hardware als auch das Betriebssystem und bereitgestellte Bibliotheken leiten sich bei Smartphones von der PC-Architektur ab. Googles Android

basiert beispielsweise auf dem Linux-Kernel und Apples iOS verwendet wie Mac OS X den Mach Microkernel und ein BSD-basiertes Subsystem.

Alle Analysen von Smartphone-Anwendungen beschränken sich in dieser Arbeit auf das Apple iPhone mit iOS, da die Unterschiede in der Softwarearchitektur zwischen iOS und Android eine übergreifende Analyse unverhältnismäßig aufwändig machen würde.

Daher wird in diesem Kapitel zuerst die Software-Architektur für iOS-Anwendungen dargestellt. Danach folgen die technischen Hintergründe, die eine Analyse von Anwendungen ermöglichen.

2.5.1 Objective-C und C

Alle Anwendungen für Apple iOS werden in C und Objective-C geschrieben. Objective-C ist dabei eine Obermenge von C, in welche die Konzepte der Objektorientierung aus Smalltalk integriert sind [9].

Die Sprache ist dabei aufgrund dynamischer Objekttypisierung und dynamischem Binden von Methoden zur Laufzeit dynamisch konzipiert. Bei primitiven Datentypen ist Objective-C wie C statisch typisiert. Bei Klassenobjekten hingegen wird eine dynamische Typisierung zur Laufzeit verwendet.

Auch die Zuordnung von Methoden zu ihren Implementierungen wird erst zur Laufzeit dynamisch aufgelöst. Dazu verwendet Objective-C ein Nachrichtenkonzept. Bei diesem wird zum Kompilierungszeitpunkt ein Methodenaufruf in eine Nachricht mit Namen, Sender und Empfänger übersetzt. Diese Nachricht wird als Aufruf der C-Funktion `objc_msgSend()` übersetzt und dann erst zur Laufzeit an das Empfängerobjekt zugestellt. Das Empfängerobjekt entscheidet zur Laufzeit, ob und wie es auf diese Nachricht reagiert. Die Auflösung zur Laufzeit wird von der Objective-C-Runtime vorgenommen. Die Namen der Nachrichten sind dabei als C-Strings in den Nachrichten hinterlegt.

```
1 // Objective"~C Nachricht
2 [someObject aMethodWithInt:13];
3
4 // Uebersetzung in objc_msgSend()-Aufruf
5 objc_msgSend(someObject, "aMethodWithInt", 13);
```

Listing 2.2: Beispiel für die Übersetzung von Objective-C Aufrufen in C-Aufrufe

In Listing 2.2 ist die Übersetzung von Objective-C-Nachrichten in `objc_msgSend()`-Aufrufe exemplarisch für eine Nachricht mit einem Integer-Parameter gezeigt.

Es existieren zwei Versionen des Objective-C-Application-Binary-Interface (ABI) und der Objective-C-Runtime. Bei Mac OS X wird für 32-Bit-Programme die ABI in Version 1.0 verwendet, für 64-Bit-Programme die Version 2.0. In iOS wird ausschließlich die ABI in Version 2.0 verwendet.

2.5.2 Objective-C Runtime

Die Objective-C Runtime ist zentraler Bestandteil jedes Objective-C Programms [10]. Die Runtime stellt die zur Laufzeit dynamischen Funktionen der Sprache zur Verfügung, wie beispielsweise das dynamische Binden von Methoden. Im Gegensatz zu C-basierten Programmen ist daher ein mit den Objektkonzepten von Objective-C geschriebenes Programm ohne die Runtime nicht lauffähig.

Die Runtime selbst ist in der Sprache C programmiert und stellt folgende Funktionen zur Verfügung: Alle Nachrichten, die für den Methodenaufruf versendet werden, werden über die C-Funktion `objc_msgSend()` durch die Runtime verarbeitet. Der Dispatcher der Runtime sucht dabei nach einer passenden Implementierung für die jeweilige Nachricht [10].

Außerdem werden jeder Klasse ihre Methoden erst zur Laufzeit hinzugefügt. Hierfür besitzt die Runtime die entsprechenden C-Funktionen `class_addMethod`, `class_addIvar`, `class_addProtocol`. Dies ermöglicht eine Erweiterung von existierenden Klassen zur Laufzeit, ohne dass die bereits bestehende Klasse neu übersetzt werden muss. So können in Objective-C-Anwendungen auch Klassen der bestehenden Frameworks für eigene Zwecke erweitert werden. Diese Erweiterungen werden in Objective-C „Kategorien“ genannt.

Zuletzt lassen sich mit der Objective-C Runtime auch die Implementierungen zu bestehenden Klassen zur Laufzeit austauschen. Hierzu existieren folgende C-Funktionen:

- `method_getImplementation`
- `method_getNumberOfArguments`
- `method_getArgumentType`
- `method_getReturnType`
- `method_setImplementation`
- `method_exchangeImplementations`

2.5.3 Frameworks

Um die Vielfalt der zur Verfügung stehenden API-Funktionen deutlich zu machen, wird im Folgenden ein Überblick über die Frameworks gegeben. Diese fassen mehrere API-Funktionen zu einer logischen Einheit zusammen. Insgesamt existieren in iOS Version 6.1 51 Frameworks.

Das Betriebssystem und die bei iOS vorhandenen Frameworks lassen sich in vier Schichten einteilen: Core OS, Core Services, Media und Cocoa Touch. Alle vier Schichten

können aus den Anwendungen verwendet werden. Von Apple wird lediglich empfohlen, für jeden Zweck eine möglichst hohe Softwareschicht zu verwenden [3].

In der Core OS-Schicht befinden sich die aus anderen C-basierten Betriebssystemen bekannten POSIX- und BSD-APIs für Multithreading, Netzwerk, Standard-IO, Speicherverwaltung und mathematischen Funktionen. Daneben finden sich in der Core OS-Schicht Frameworks, die vor allem zur Kommunikation mit Geräten und zur Anwendungssicherheit dienen. Diese sind: Accelerate, CoreBluetooth, ExternalAccessory, GSS und Security.

Die Core Services-Schicht bietet die grundlegenden Funktionen für die Anwendungsprogrammierung wie abstrakte Datentypen, Zeichenkettenfunktionen, Datums- und Uhrzeitfunktionen und Multithreading. Sie sind im Foundation-Framework (Objective-C basiert) und CoreFoundation-Framework (C-basiert) gebündelt. Außerdem sind folgende Frameworks Teil der Core Services-Schicht: Accounts, AddressBook, AdSupport, CFNetwork, CoreData, CoreLocation, CoreMedia, CoreMotion, CoreTelefony, EventKit, MobileCoreServices, NewsstandKit, PassKit, QuickLook, Social, StoreKit und SystemConfiguration.

Die Media-Schicht bietet Audio-, Video- und Bildbearbeitungsfunktionen. Diese werden gebündelt in folgenden Frameworks angeboten: AssetsLibrary, AVFoundation, CoreAudio, CoreGraphics, CoreImage, CoreMIDI, CoreText, CoreVideo, ImageIO, GLKit, MediaPlayer, OpenAL, OpenGLES und QuartzCore.

In der Cocoa Touch-Schicht werden abschließend alle Funktionen für die grafische Oberfläche von Anwendungen gebündelt. Im Wesentlichen stellt das UIKit-Framework diese Funktionen zur Verfügung. Es besitzt dabei aus historischen Gründen neben den grafischen Elementen und der Interaktion mit dem Benutzer auch Funktionen, die Zugriff auf den Beschleunigungssensor oder den Batteriezustand ermöglichen.

Anhand der großen Zahl von Frameworks ist erkennbar, welche Bandbreite von API-Funktionen für Anwendungsentwickler verfügbar ist. Die exakte Zahl der zur Verfügung stehenden API-Funktionen ist von Apple nicht dokumentiert. Daher wird in Tabelle 2.9 eine Näherung für die Zahl von API-Funktionen gegeben, welche auf der Zählung aller in Objective-C verfügbaren Funktionen der iOS-Versionen 5 und 6 basiert.

iOS-Version	Zahl der Klassen	Zahl der Methoden
5.0	2233	42376
5.1	2248	42634
6.0	3398	62035
6.1	3421	62414

Tabelle 2.9: Verfügbare Objective-C Klassen und Methoden, eigene Erhebung

Eine Anwendung kann mit Hilfe dieser großen Zahl an Funktionen, die über viele Frameworks verteilt sind, ihre eigene Funktionalität realisieren. Da die dynamische Analyse pro Funktion vorgenommen wird, muss eine Auswahl von diesen rund 62.000 verfügbaren Funktionen getroffen werden. Zusätzlich sind aus historischen Gründen vorhandene Doppelstrukturen zu berücksichtigen.

2.5.4 Sandbox

Auf dem Gerät wird jede Anwendung in einer sogenannten Sandbox ausgeführt. Diese Sandbox dient dazu, gegenseitige Zugriffe zwischen Anwendungen zu verhindern. Die Sandbox ist dabei so ausgelegt, dass standardmäßig jeder Zugriff verweigert wird, wenn er nicht explizit auf einer Positivliste vermerkt ist. Eine Anwendung hat nur innerhalb ihres eigenen Ordners Lese- und Schreibrechte. Des Weiteren wird eine Anwendung so daran gehindert, Interrupts an andere Prozesse zu senden. Außerdem ist es für Anwendungen nicht möglich, weitere Prozesse zu starten [7].

Nur in zweiter Linie dient die Sandbox dazu, Anwendungen den Zugriff auf personenbezogene Daten zu verweigern. Anwendungen können nicht auf diese zugreifen, da die Datenbanken für Emails, Kurznachrichten oder Notizen nicht auf der Positivliste der Sandbox stehen. Schnittstellen, die vom Betriebssystem oder von Frameworks ohne Autorisierung des Benutzers angeboten werden, können jedoch von jeder beliebigen Anwendung genutzt werden.

Ab Version 3 von iOS existiert die konkrete Positivliste der Sandbox nicht mehr in Textform. Sie ist nur in vorkompilierter Form im Sandbox-Dienst hinterlegt. Für Version 4.0 wurde sie von Zovi [24] aufwändig aus dem Dienst extrahiert und dokumentiert. Sie findet sich im Anhang A.2. Darin ist ersichtlich, dass in Version 4 sowohl die Datenbankdatei des Adressbuchs als auch der Speicherort für Fotos in der Positivliste verzeichnet waren. In Version 6 wurden diese Einträge aus der Positivliste entfernt und ein Zugriff auf die Dateien wird daher von der Sandbox unterbunden.

2.5.5 Sicherheitsmaßnahmen in Anwendungen

Apple stellt seit iOS Version 4.0 eine API zur Dateiverschlüsselung für Anwendungen zur Verfügung.

Die Dateiverschlüsselung ist vom Betriebssystem für Anwendungen transparent gestaltet. Anwendungen können so die normalen Funktionen zum Lesen und Schreiben von Dateien nutzen, wenn diese gerade entschlüsselt sind. Ob eine Datei verschlüsselt werden soll, wird über Dateiattribute durch die Anwendung festgelegt. Die Verschlüsselung und Entschlüsselung wird dann je nach Modus vom Betriebssystem bei bestimmten Aktionen durchgeführt. Diese Aktionen sind das Aktivieren und Deaktivieren des Sperrbildschirms

und der Neustart des Geräts. Wenn kein Code für den Sperrbildschirm vom Benutzer gesetzt wird, ist die Verschlüsselung vollständig deaktiviert.

Wenn eine Anwendung versucht, während des aktivierten Sperrbildschirms auf eine verschlüsselte Datei zuzugreifen, wird vom Betriebssystem ein allgemeiner Eingabe-/Ausgabefehler zurückgeliefert. Im entsperrten Zustand kann eine Anwendung aufgrund der transparenten Realisierung der Verschlüsselung nur anhand des Verschlüsselungsmodus überprüfen, ob eine Datei bei aktiviertem Sperrbildschirm verschlüsselt wird.

Die Dateien können mit unterschiedlichen Verschlüsselungsmodi gespeichert werden. In Tabelle 2.10 sind die Unterschiede zwischen den Modi zusammengefasst.

Modus	Beschreibung
None	Keine Verschlüsselung (Standardeinstellung)
CUFUA	Datei ist während des Bootvorgangs des Geräts verschlüsselt. Sobald der Nutzer einmal den Code am Sperrbildschirm eingibt, wird die Datei entschlüsselt.
CUO	Datei ist verschlüsselt, solange der Sperrbildschirm aktiv ist und die Datei nicht geöffnet war, als der Sperrbildschirm aktiviert wurde
Complete	Datei ist verschlüsselt, solange der Sperrbildschirm aktiv ist. Sobald der Code eingegeben wird, wird die Datei entschlüsselt

Tabelle 2.10: Unterschiede zwischen den Dateiverschlüsselungsmodi

Der Modus „Complete“ ist der Modus, in dem die Dateien zeitlich gesehen die längste Zeit verschlüsselt sind, da sie immer verschlüsselt werden, sobald der Sperrbildschirm am Gerät aktiviert wird. Der Modus „CompleteUnlessOpen“ (CUO) unterscheidet sich nur dadurch, dass Dateien, wenn sie beim Aktivieren des Sperrbildschirms gerade geöffnet waren, nicht geschlossen und verschlüsselt werden. Alle nicht geöffneten Dateien werden verschlüsselt, sobald der Sperrbildschirm aktiviert wird. Im Modus „CompleteUntilFirstUserAuthentication“ (CUFUA) werden Daten während des Bootvorgangs bis zum ersten Entsperren verschlüsselt. Im Modus „None“ werden Dateien nie verschlüsselt.

Da die Verschlüsselung maximal bei aktiviertem Sperrbildschirm aktiv ist, ist während des aktiven Gebrauchs des Geräts keine Dateiverschlüsselung vorgesehen. Somit liegen auch alle Daten nicht gestarteter Anwendungen unverschlüsselt vor und können über das Betriebssystem ausgelesen werden. Wegen der Sandbox können Anwendungen von Drittanbietern jedoch nicht darauf zugreifen. Ein Angreifer mit physikalischem Zugriff auf ein nicht gesperrtes Gerät kann jedoch auf alle Dateien zugreifen.

Wenn Anwendungsentwickler die Daten ihrer Anwendung schützen wollen, während andere Anwendungen ausgeführt werden, können sie daher nicht auf die von iOS bereitgestellte Dateiverschlüsselung zurückgreifen. Sie müssen selbst die notwendigen Verschlüsselungsmechanismen realisieren. Da in der automatischen Analyse keine Berücksichtigung von Einzelfällen möglich ist, können diese Mechanismen nicht berücksichtigt werden.

2.5.6 API-Hooking

Mit Hooking kann der Aufruf von beliebigen Funktionen einer Anwendung abgefangen und verändert werden. API-Hooking wendet diesen Ansatz auf API-Funktionen an. Eine Anwendung kann dabei nicht feststellen, ob der Aufruf verändert oder ausgetauscht wurde. Daher ist Hooking für die Anwendung transparent.

API-Hooking für Objective-C wird von Petrichs „Captain Hook“ [43] über die Funktionen der „Objective-C Runtime“ realisiert [10]. Die Objective-C Runtime erlaubt es, Methoden zur Laufzeit zu Klassen hinzuzufügen und zu entfernen und darüber hinaus Implementierungen von Methoden zur Laufzeit auszutauschen.

Das API-Hooking für C-Funktionen wird von MobileSubstrate über Assembler analog realisiert [35].

Für Objective-C wird dazu die eigentliche Funktionsimplementierung in einer Variable abgespeichert, sodass sie unter einem weiteren Namen erreichbar ist. In der neuen Implementierung ist dann die Ausführung eigenen Codes inklusive des Aufrufs der Original-Implementierung möglich. Dann wird mit Hilfe der Objective-C-Runtime die bisherige Implementierung durch die neue Implementierung ersetzt. API-Hooking muss für jede zu ersetzende Funktion einzeln durchgeführt werden. Das Vorgehen ist in Abbildung 2.2 dargestellt.

Zur Realisierung von API-Hooking wird eine Bibliothek dynamisch zur Anwendung dazu geladen. Dies ist über die Umgebungsvariable `DYLD_INSERT_LIBRARIES` möglich. Durch MobileSubstrate [30] ist es möglich, dynamische Bibliotheken in den Ordner `/Library/MobileSubstrate/DynamicLibraries/` zu platzieren und über einen Filter festzulegen, welche Bibliothek zu welcher Anwendung geladen wird. Diese weiteren Bibliotheken werden Tweaks genannt.

In der Arbeit wird API-Hooking verwendet, um Funktionsaufrufe von beliebigen Anwendungen zu protokollieren. Der Tweak ist nicht an eine spezielle Anwendung angepasst, sondern kann zu jeder beliebigen Anwendung hinzugeladen werden und diese dynamisch analysieren.

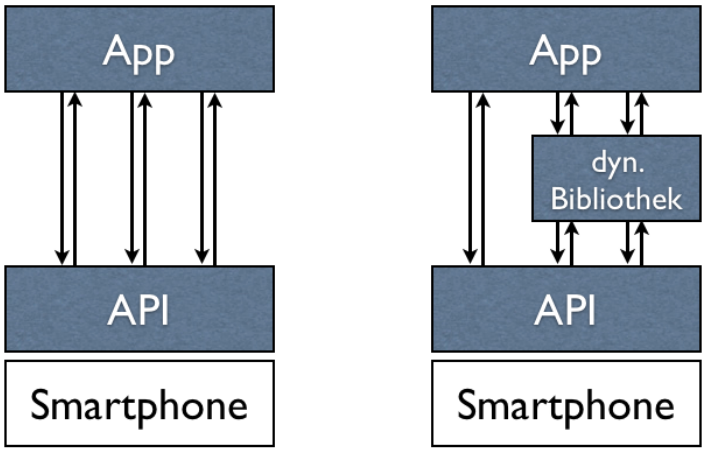

Abbildung 2.2: API-Aufrufe ohne API-Hooking (links) und mit API-Hooking (rechts).

In der weiteren Beschreibung wird in Ermangelung eines deutschen Fachbegriffs das englische Verb „hook“ nach deutscher Beugung für die Anwendung des API-Hooking-Prinzips verwendet.

2.5.7 Jailbreaks

Im Auslieferungszustand sind bei Apple iOS-Geräten diverse Restriktionen in Kraft: Nur von Apple signierter Code kann ausgeführt werden, der Zugang als Benutzer root ist nicht möglich und es können keine Umgebungsvariablen gesetzt werden. Die Prüfung der Signaturen wird vor dem Start von Anwendungen im Kernel sichergestellt.

API-Hooking ist aufgrund dieser Restriktionen im Auslieferungszustand nicht möglich.

Jailbreaks heben diese Restriktionen auf und ermöglichen dadurch die Ausführung von nicht von Apple signiertem Code. Dazu wird die Überprüfung der Signaturen im Kernel außer Kraft gesetzt. Außerdem wird ein Zugang mit root-Rechten eingerichtet und Anwendungen können auch ohne Sandbox gestartet werden. Da auch das Setzen von Umgebungsvariablen möglich wird, können dynamische Bibliotheken zu Anwendungen geladen werden.

Für einen Jailbreak werden Sicherheitslücken im Betriebssystem ausgenutzt, um root-Rechte zu erlangen und dann den Kernel zu verändern. Da die Lücken nach Veröffentlichung des Jailbreaks nach Möglichkeit von Apple geschlossen werden, stehen die Jailbreaks für aktuelle iOS-Versionen häufig nicht lange zur Verfügung. Eine Ausnahme bilden Jailbreaks, die Sicherheitslücken in nicht-veränderlichen Teilen des Betriebssys-

tems wie dem Bootrom ausnutzen. Dies ermöglicht die Analyse auch bei neu erschienen iOS-Versionen.

Es gibt dabei zwei Arten von Jailbreaks: Einerseits kabel-ungebundene Jailbreaks, die auch nach einem Neustart des Geräts weiterhin aktiv sind („untethered“), und andererseits kabelgebundene Jailbreaks, die nur über die USB-Verbindung zu einem Computer während des Bootvorgangs Sicherheitslücken ausnutzen können und dann aktiv sind („tethered“).

Ein Beispiel für einen kabel-ungebundener Jailbreak ist „evasi0n“ für iOS-Version 6.0 bis 6.1.2 [14]. Mit Erscheinen von iOS Version 6.1.3 wurden die ausgenutzten Sicherheitslücken von Apple geschlossen und der Jailbreak steht daher für diese Version nicht mehr zur Verfügung.

„Limera1n“ ist ein Beispiel für ein kabelgebundenen Jailbreak. Er nutzt eine Sicherheitslücke im Bootrom des A4-Prozessors aus. Da der Bootrom des A4-Prozessors nicht verändert werden kann, steht dieser Jailbreak für alle Geräte mit diesem Prozessor zur Verfügung.

In der vorliegenden Analyse wird ein iPhone 4 und der kabel-ungebundene Jailbreak „evasi0n“ verwendet.

2.5.8 Entwicklungsumgebung Theos

Da die von Apple bereitgestellte Entwicklungsumgebung Xcode die Entwicklung von Tweaks nicht unterstützt, wird das von Howett [34] veröffentlichte Werkzeug „Theos“ verwendet. Dieses stellt Makefile-Vorlagen für Tweaks zur Verfügung. Damit kann ein Tweak kompiliert und installiert werden. Die Bearbeitung und Verwaltung des Quelltexts kann dann mit Xcode oder einem beliebigen Texteditor geschehen.

2.5.9 Unit-Tests mit dem iOS-Simulator

Der iOS-Simulator wird von Apple für die Entwicklung und Test von iOS-Anwendungen bereitgestellt. Er verfügt größtenteils über die gleichen Frameworks wie die Smartphones, ist aber auf der x86-Architektur lauffähig. Dabei wird eine zu testende Anwendung in den x86-Instruktionssatz übersetzt und dann ausgeführt. Daher können im Simulator nur Anwendungen ausgeführt werden, für die dem Entwickler der Quelltext vorliegt.

Im Unterschied dazu bietet Android für die Anwendungsentwicklung einen Emulator. Bei diesem wird eine CPU mit ARM-Instruktionssatz mit Hilfe von QEMU [16] emuliert. Innerhalb dieses Emulators können daher auch Anwendungen aus dem Google Market ausgeführt werden, die nur in kompilierter Form vorliegen.

Zusätzlich ist beim iOS-Simulator zu beachten, dass manche Komponenten nicht vorhanden sind oder sich anders als auf dem tatsächlichen Gerät verhalten können [8]. Nicht vorhanden sind folgende Daten und API-Funktionen:

- Beschleunigungssensor
- Rotationssensor
- Kamera
- Mikrofon
- Abstandssensor
- Push-Benachrichtigungen
- Autorisierungen für den Zugriff auf Adressbuch, Kalender, Fotos, Erinnerungen
- Dateiverschlüsselung (DataProtection-API)
- iCloud Synchronisation
- `AVCaptureDevice` des AudioFoundation-Frameworks
- MessageUI-Framework
- Datenbankdateien für Anruf-Historie, Adressbuch, Kalender und Fotos unter den genannten Pfaden

Der iOS-Simulator lässt sich auch für den Test des Tweaks benutzen, der in dieser Arbeit entwickelt wurde und API-Hooking verwendet. Da CaptainHook ausschließlich Funktionen der Objective-C-Runtime zur Realisierung des API-Hookings für Objective-C-Funktionen nutzt, funktionieren die API-Hookings auch im Simulator. Für C-Funktionen wird das MobileSubstrate-Framework zum kompilierten Testfall geladen. Damit lässt sich auch im Simulator API-Hooking für C realisieren, da die Funktion `MSHookFunktion()` dann zur Verfügung steht.

Mit Hilfe des iOS-Simulators lässt sich so die Methodik der testgetriebenen Entwicklung, die in Kapitel 4.5.2 genauer vorgestellt wird, auch für Tweaks benutzen, die API-Hooking verwenden.

2.5.10 Taint Tracking

Über API-Hooking ist es möglich, die Quellen von sensitiven Daten zu überwachen. Damit ist jedoch keine Aussage möglich, wie die Daten im weiteren Verlauf verwendet werden.

Um dies beantworten zu können, ist sogenanntes „Taint Tracking“ notwendig. Dieses Vorgehen basiert auf der Propagierung von Markierungen für Speicherbereiche. Quellen von sensitiven Informationen versehen ihre Speicherbereiche mit einer Markierung, bevor

sie diese an die Anwendungen übergeben. Während des Ablaufs der Anwendung wird diese Markierung weiterpropagiert, wenn der Wert kopiert wird. Bei der Verarbeitung von Werten in mathematischen Funktionen werden die Markierungen mit OR verknüpft, sodass eine einmal bestehende Markierung auch dort weiterpropagiert wird. An Senken von Informationen, also beispielsweise der Netzwerkschnittstelle, kann anhand der Markierungen festgestellt werden, aus welcher Quelle diese Informationen stammen.

Existierende Taint Tracking-Systeme [19, 27, 18, 28] werden durch eine volle Emulation des zu untersuchenden Systems mit Werkzeugen wie Bochs [38] oder QEMU [16] realisiert, wodurch sie aufgrund der damit verbundenen Leistungseinbußen für die Anwendung auf heutigen Smartphones ungeeignet sind[18].

Android (Java)

Android-Anwendungen werden in der Programmiersprache Java geschrieben und in der Dalvik Virtual Machine ausgeführt.

Für Android existiert mit „TaintDroid“[28] ein Taint-Tracking-System, das Informationsflüsse innerhalb des Geräts und Informationsabflüsse über Senken wie der Netzwerkschnittstelle erkennen kann. Laut den Autoren ist dies ohne große Leistungseinbußen realisierbar, sodass TaintDroid zur Laufzeit während der normalen Benutzung des Geräts verwendet werden kann.

TaintDroid arbeitet mit vier Genauigkeitsstufen: Innerhalb der Dalvik Virtual Machine werden einzelne Variablen markiert. Zwischen Anwendungen werden ganze IPC-Messages markiert, wenn diese Message mindestens eine markierte Variable enthält. Für Systembibliotheken außerhalb der Dalvik Virtual Machine werden Methoden markiert und für persistente Speicherung im Dateisystem werden Dateien markiert.

Das Tainting auf Variablen-Ebene innerhalb der Dalvik Virtual Machine ist möglich, da jede Android-Anwendung innerhalb dieser virtuellen Maschine ausgeführt wird. Da die virtuelle Maschine Open Source ist, kann sie so verändert werden, dass sie das Tracking auf Variablen-Ebene beherrscht.

Daher lässt sich TaintDroid ohne die volle Emulation realisieren, welche bei anderen Taint-Tracking-Systemen notwendig ist.

C

Für C-basierte Programme gibt es zwei Ansätze von Taint Tracking. Der erste verwendet eine volle Emulation des zu trackenden Systems. Die zweite Lösung verändert die Programme vor ihrer Ausführung, um so auf x86-Intruktionsebene Daten zu markieren und Markierungen zu propagieren [19].

Da die vollständige Emulation des Systems mit Bochs oder QEMU aufwändig ist, zeigen spätere Arbeiten [18, 20], dass sich für das Taint Tracking auch Emulatoren konzipieren lassen, die auf Prozessebene anstatt auf Systemebene emulieren. Sie können die mit der Emulation einhergehenden Leistungseinbußen auf einen Faktor 1,5 bis 3 beschränken und sind laut den Autoren daher für den Praxiseinsatz geeignet [18].

„Binary instrumentation“ [20, 46, 48] ist eine weiterer Möglichkeit, bei generischen x86-Programmen Taint Tracking anzuwenden. Hierbei wird der Binärcode vor der Ausführung so modifiziert, dass der zum Taint Tracking notwendige Code in den existierenden Binärcode eingefügt wird. Dann wird dieser modifizierte Binärcode ausgeführt.

Dieser Ansatz vermeidet zwar die vollständige Emulation, bringt aber aus anderen Gründen Leistungseinbußen mit sich. Auch Registerinhalte müssen markiert werden können. Diese Markierung erfolgt aus Performance-Gründen in anderen Registern, beispielsweise SSE-Registern [18] oder 64-Bit-Registern [46]. Daher stehen diese Register der eigentlichen Anwendung nicht mehr zur Verfügung.

Auch für den ARM-Instruktionssatz existiert „binary instrumentation“ [33] als Erweiterung des von Intel entwickelten Frameworks „PIN“ [40]. Anders als in der Arbeit von Hazelwood et al. angegeben, ist die ARM-Variante von PIN nicht öffentlich zugänglich.

Taint Tracking ist aufgrund seiner Realisierung an die technische Infrastruktur wie die virtuelle Maschine bei Android und den verwendeten Instruktionssatz gebunden. Da iOS-Anwendungen nicht innerhalb einer virtuellen Maschine ausgeführt werden, ist der „TaintDroid“-Ansatz nicht übertragbar. Daher soll in dieser Arbeit auf Taint Tracking verzichtet werden.

3 Aktueller Stand der Anwendungsanalyse

3.1 PiOS: Detecting Privacy Leaks in iOS Applications

Egele et. al [26] führen eine statische Analyse des Binärcodes von iOS-Anwendungen durch. Bei einer statischen Analyse wird der vorliegende Binärcode in einen Kontrollflussgraphen überführt und dann werden innerhalb dieses Graphen Datenflussanalysen durchgeführt. Bei iOS-Anwendungen ist der Binärcode grundsätzlich verschlüsselt und wird erst zur Ausführung im Smartphone entschlüsselt. Daher muss der Binärcode für eine statische Analyse eigens entschlüsselt werden. In der Arbeit werden die Anwendungen dazu mit einem Debugger geladen. Bevor die Anwendung startet, hält der Debugger die Ausführung an und der entschlüsselte Binärcode kann aus dem Hauptspeicher gelesen werden.

In der Analyse wird folgendes Kriterium als Datenschutzproblem definiert: Jeder durch den Benutzer nicht autorisierte Zugriff auf personenbezogene Daten gilt als Datenschutzproblem. Es wird angenommen, dass eine Autorisierung besteht, sobald ein beliebiges Dialogfeld zwischen Zugriff und weiterer Verwendung der personenbezogenen Daten dem Benutzer angezeigt wird.

Von den 1407 untersuchten Anwendungen wurden 825 aus dem Apple App Store und 582 aus dem Cydia Store entnommen. Der Cydia Store ist auf den Smartphones nach einem Jailbreak verfügbar. Anders als beim Apple App Store werden die Anwendungen nicht durch den Betreiber kontrolliert, bevor sie verfügbar gemacht werden.

Die statische Analyse förderte einige Datenschutzprobleme zu Tage. 44,9 % der 1407 untersuchten Anwendungen verwenden Werbenetzwerke, die ihrerseits zur Anzeige möglichst zielgerichteter Werbung die Geräte-ID an das Werbenetzwerk übertragen. Darüber hinaus greifen noch einmal 14 % Anwendungen ohne ein Werbenetzwerk auf die Geräte-ID zu. Lediglich 0,4 % der Anwendungen greifen auf das Adressbuch zu.

Ein Nachteil der Arbeit liegt in der Auswahl der Anwendungen. Es wird nicht beschrieben, wie die 1407 Anwendungen ausgewählt wurden. Außerdem ist die Liste der ana-

lysierten Anwendungen und ihre verwendete Version nicht dokumentiert. Daher ist ein direkter Vergleich der Ergebnisse nicht möglich.

Als weiteren Nachteil führen die Autoren an, dass die statische Analyse aufgrund der dynamischen Aufrufmöglichkeiten der Objective-C Runtime nicht immer das genaue Ziel eines Methodenaufrufs rekonstruieren kann. Der Grund hierfür ist einerseits, dass die Objective-C Runtime zur Laufzeit den Austausch von Methodenimplementierungen erlaubt. Außerdem kann ein zur Laufzeit bestimmter Methodenname oder Parameter in der statischen Analyse nicht erfasst werden. Dies erschwert beispielsweise die Erfassung, welche Dateien eine Anwendung genau öffnet.

3.2 Challenges for Dynamic Analysis

Aufbauend auf PiOS analysieren Szydlowski et. al [51] iOS-Anwendungen dynamisch mit Hilfe des GNU Debuggers (GDB). Dieses Verfahren führt alle Anwendungen aus und analysiert sie zur Laufzeit.

Die große Herausforderung ist dabei, alle Methoden mindestens einmal auszuführen. Im Software-Engineering ist dieses Kriterium als vollständige Methodenüberdeckung bekannt. Um das Kriterium zu erfüllen, müssen alle Funktionen einer Anwendung tatsächlich aufgerufen werden. Daher ist es notwendig, die Benutzerinteraktion mit der jeweiligen Anwendung zu automatisieren. Szydlowski et. al verwendeten hierfür eine auf dem Remote-Desktop-Protokoll „VNC“ aufbauende Lösung. Der vollständige Bildschirminhalt wird auf den Analyserechner übertragen und dort analysiert, um über das Remote-Desktop-Protokoll Interaktionen auszulösen. Die Interaktionen setzen sich dabei aus zufälligen Positionen auf dem Bildschirm und einer Mustererkennung des Bildschirminhalts zusammen. Mit Hilfe letztgenannter wird versucht, grafische Elemente der Benutzeroberfläche zu erkennen, um gezielt durch eine Anwendung navigieren zu können.

Die dynamische Analyse wird anhand einer prototypischen Anwendung getestet. In dieser Anwendung sind mehrere Elemente einer typischen Benutzeroberfläche wie Schaltflächen und Eingabefelder vorhanden. Der Prototyp ist dabei so gestaltet, dass ein Eingabefeld erst nach der Aktivierung einer Schaltfläche erscheint. Dieses Eingabefeld ist in einer statischen Analyse schwer zu erkennen und zeigt die Stärken der dynamischen Analyse. Bei der dynamischen Analyse des Prototyps wird eine Methodenabdeckung von 69 % erreicht.

Laut den Autoren ist die Vorgehensweise jedoch für eine großangelegte Analyse nicht praktikabel. Die Ausführung mit dem Debugger ist nicht performant genug, um eine große Zahl von Anwendungen zu analysieren. Außerdem können Anwendungen das Vorhandensein eines Debuggers feststellen und dann Code nicht ausführen, der auf personenbezogene Daten zugreift. Des Weiteren werden Systemaufrufe mit dem GDB nicht

überwacht. In ihrem Fazit schlagen die Autoren eine Analyse auf Ebene der Objective-C-Runtime vor.

3.3 PSiOS: Bring Your Own Privacy & Security to iOS Devices

Werthmann et al. [55] zeigen mit „PSiOS“ ein Werkzeug zur Verhinderung von Datenschutzproblemen für iOS-Anwendungen. Dazu verwenden sie das Sandbox-Konzept von iOS und erweitern es um passgenaue Sandbox-Profile für einzelne Anwendungen. Der Nutzer kann damit für jede Anwendung Zugriffe auf einzelne API-Funktionen autorisieren und bei Bedarf widerrufen. Von der Sandbox werden sowohl Objective-C- als auch C-Funktionen unterstützt.

Die Arbeit baut auf der Control-Flow-Integritätsprüfung MoCFI [25] auf. Mit dieser wird sichergestellt, dass API-Aufrufe nur so ausgeführt werden, wie der Anwendungshersteller dies vorgesehen hat. Damit werden die Angriffs-Methoden des „Return-Oriented Programming“ verhindert. Die Idee dieser Angriffe auf Software ist es, den Kontrollfluss eines Programm so zu verändern, dass Code aus dem Programm und den geladenen Bibliotheken so kombiniert wird, dass ein Angreifer damit beliebige Funktionen ausführen kann. Mittels MoCFI lässt sich dieser veränderte Kontrollfluss erkennen und verhindern.

PSiOS ist kein automatisiertes Analysewerkzeug, sondern zur Einschränkung einzelner Anwendungen konzipiert. Daher werden nur Ergebnisse für 15 Anwendungen präsentiert, aber keinerlei statistische Auswertung einer großen Zahl von Anwendungen.

Anhand der Anwendung SpyPhone [49] wird exemplarisch die Wirksamkeit von PSiOS gezeigt. Des Weiteren werden 14 populäre Anwendungen mit PSiOS ausgeführt, um zu demonstrieren, dass PSiOS den Zugriff auf personenbezogene oder identifizierende Daten unterbinden kann. Wie mit Benchmarks gezeigt wird, geschieht dies mit Leistungseinbußen im Bereich zwischen 0.18% und 8%.

3.4 Vision: Automated Security Validation of Mobile Apps at App Markets

Gilbert et al. [32] präsentieren mit „AppInspector“ eine Vorgehensweise zur dynamischen Analyse von Android-Anwendungen. Die als Vision skizzierte Vorgehensweise baut auf den Ansätzen aus „TaintDroid“ [28] und „PiOS“ [26] auf.

Die Autoren schlagen für eine bessere Skalierbarkeit der Analysen vor, die Anwendungen in virtuellen Maschinen anstatt auf Smartphones auszuführen. Dabei sollen mit Hilfe von

Taint Tracking sowohl Sicherheits- als auch Datenschutzprobleme aufgezeigt werden. Ein Sicherheitsproblem ist dabei der unberechtigte Zugriff auf Daten durch eine Anwendung. Ein Datenschutzproblem wird so definiert, dass eine Anwendung auf persönliche Daten ohne Zustimmung des Nutzers zugreift. Der Arbeit zufolge kann nicht generisch bestimmt werden, ob ein Zugriff auf persönliche Daten ein Datenschutzproblem darstellt. Dies ist eine Einzelfallentscheidung pro Anwendung und Nutzungsszenario.

Für neun Anwendungen werden Ergebnisse hinsichtlich der Anweisungsabdeckung gezeigt. Diese neun Anwendungen sind die am häufigsten heruntergeladenen Anwendungen aus den neun Kategorien des Android Markets. Jede Anwendung wird 30 Minuten ausgeführt und mit zufälligen Eingaben gesteuert. Es zeigen sich große Unterschiede zwischen den analysierten Anwendungen: Die Werte für die Anweisungsabdeckung liegen im Bereich von 1 % bis 40 %.

Da der rein zufällige Ansatz zur Anwendungsausführung laut den Autoren nicht zielführend ist, wird Symbolic Execution als Methode vorgeschlagen. Die Idee dieser Vorgehensweise ist, Verzweigungen in Anwendungen direkt anzuspringen und auszuführen, anstatt über unterschiedliche Benutzerinteraktionen zu versuchen, alle Zweige auszuführen. Für 1100 Anwendungen wird die Zahl der Verzweigungen ermittelt. Sie liegt für 90 % der Anwendungen jeweils unter der Schwelle von 4187 Zweigen.

Die Autoren führen die Symbolic Execution nicht durch und präsentieren auch keine Ergebnisse für die anfangs definierten Datenschutz- und Sicherheitsprobleme.

3.5 AppsPlayground: Automatic Security Analysis of Smartphone Applications

Rastogi et al. [47] zeigen mit „AppsPlayground“ eine Vorgehensweise zur dynamischen Analyse von Android-Anwendungen, welche auf „TaintDroid“ [28] aufbaut. Der Fokus der Arbeit liegt dabei auf der Automatisierung der Anwendungsausführung und die Autoren zeigen, wie in möglichst kurzer Zeit möglichst alle Anweisungen einer Anwendung tatsächlich ausgeführt werden können. Dazu werden auch Ereignisse wie das Eintreffen neuer Nachrichten oder Anrufe simuliert.

Die Analyse wird dabei in virtuellen Maschinen anstatt auf Smartphones durchgeführt. Um eine Erkennung der Analyseumgebung durch die untersuchte Anwendung zu erschweren, werden möglichst realistische Daten für die Telefonnummer, IMEI und IMSI simuliert und virtualisierungsspezifische Dateien verborgen.

Den Autoren gelingt es, die Ergebnisse von TaintDroid automatisiert mittels AppsPlayground nachzustellen. Dabei können neun der 30 Anwendungen, die bei TaintDroid untersucht wurden, nicht erneut untersucht werden. Diese Anwendungen sind zum aktualisierten Android-Emulator inkompatibel.

Darüber hinaus werden mit AppsPlayground 3968 Anwendungen hinsichtlich der Übertragung von identifizierenden Daten über das Netzwerk untersucht. 14,6 % der Anwendungen übermitteln die Android ID, 8,2 % die IMEI, 1,6 % die Telefonnummer und 0,1 % die WLAN-MAC-Adresse. Insgesamt übermitteln 23,8 % Anwendungen mindestens eine der genannten identifizierenden Informationen über das Internet. Außerdem übertragen 5,3 % Anwendungen den aktuellen Aufenthaltsort über das Netzwerk.

Zur Auswahl der untersuchten Anwendungen wird keine Aussage gemacht, daher ist ein direkter Vergleich der Ergebnisse nicht möglich.

3.6 App Genome Report

Das auf IT-Sicherheit spezialisierte Unternehmen Lookout [39] führt statische Analysen des Binärcodes für kostenlose Anwendungen sowohl auf der iOS- als auch der Android-Plattform durch. Hierbei werden ausschließlich aggregierte Analysen veröffentlicht und keine Aussagen über einzelne Anwendungen.

Ein Ergebnis ihrer statischen Analysen ist, dass der Anteil der Anwendungen rückläufig ist, die Code für Zugriffe auf den aktuellen Ort oder das Adressbuch enthalten. Im August 2010 enthielten 36,5 % der Anwendungen Code für Zugriff auf den aktuellen Ort und 14,8 % der Anwendungen Code für den Zugriff auf das Adressbuch. Stand Februar 2011 reduzierte sich der Anteil für den aktuellen Ort auf 33,9 % und für das Adressbuch auf 11,2 % der Anwendungen.

Die Analysen von Lookout enthalten keine Aussagen über weitere personenbezogene Daten wie Kalender oder Fotos. Identifizierende Daten wie UDID und MAC-Adressen werden in den Analysen ebenfalls nicht behandelt.

3.7 Clueful

Das Virenschutz- und Internet-Sicherheit-Unternehmen Bitdefender [17] veröffentlicht mit „Clueful“ statische Analysen von kostenlosen iOS-Anwendungen. Hierbei werden folgende Kriterien erfasst:

- Adressbuch auslesen/verändern
- Kann den Kalender verändern
- Kann den aktuellen Ort verwenden
- Ortung kann die Akkulaufzeit beeinflussen
- Verbindet sich mit Twitter oder Facebook

- Verwendet den UDID
- Lädt den UDID hoch
- Verwendet den ASID
- Kann Werbung anzeigen
- Nutzungsstatistiken (Flurry, Mobclix, Medialets und Google Analytics)
- Verschlüsselt gespeicherte Daten

Die Ergebnisse werden von Bitdefender nur für einzelne Anwendungen zugänglich gemacht. Es werden keine aggregierten Werte veröffentlicht. Des Weiteren ist nicht ersichtlich, wie viele Anwendungen von Bitdefender insgesamt analysiert werden.

3.8 Appthority

Das auf IT-Sicherheit spezialisierte Unternehmen Appthority bietet Sicherheitsanalysen von iOS- und Android-Anwendungen für Mobile-Device-Management-Software an. Dabei werden die Anwendungen nach Unternehmensangaben hinsichtlich bekannter Schadsoftware sowie Sicherheits- und Datenschutzaspekten untersucht. Die Ergebnisse der Analysen werden von Appthority für ausgewählte Anwendungen in unregelmäßigen Abständen veröffentlicht.

Im Juli 2012 wurden die Ergebnisse für die 50 am häufigsten heruntergeladenen kostenlosen Anwendungen veröffentlicht [13]. 88 % der Anwendungen verwendeten Werbenetzwerke oder Statistik-Bibliotheken, 70 % der Anwendungen griffen auf den aktuellen Ort, 52 % auf das Adressbuch und 26 % auf den Kalender zu.

Im Februar 2013 wurden aus fünf beliebten Kategorien die je zehn am häufigsten heruntergeladenen kostenlosen Anwendungen analysiert [12]. Die Kategorien sind „Business“, „Education“, „Entertainment“, „Finance“, „Games“. Dabei verwendeten 60 % der Anwendungen Werbenetzwerke oder Statistik-Bibliotheken, 60 % griffen auf den aktuellen Ort, 54 % auf das Adressbuch und 14 % auf den Kalender zu. Aufgrund des veränderten Auswahlkriteriums der Anwendungen ist ein zeitlicher Vergleich nicht möglich.

Das genaue Analyseverfahren wird von Appthority nicht dokumentiert. Es wird lediglich angegeben, dass sowohl statische als auch dynamische Analysen durchgeführt werden. Darüber hinaus wird die genaue Liste der analysierten Anwendungen und ihrer Versionen nicht veröffentlicht.

3.9 SiRA: Automation in iOS Application Assessments

Engler et al. [29] präsentieren mit „SiRA“ eine statische und dynamische Anwendungsanalyse von iOS-Anwendungen hinsichtlich Sicherheits- und Datenschutzaspekten. Sie zeigen dazu einerseits einen Überblick über die zur Analyse notwendigen Schritte. Außerdem erläutern sie, welche Teile dieser Analyse teilweise oder vollständig automatisiert werden können.

Nach einem Abbild des Dateisystems wird die Anwendung installiert und entschlüsselt. Anschließend wird eine statische Analyse hinsichtlich IT-Sicherheitsaspekten durchgeführt. Dazu wird ermittelt, ob die Anwendung als „Position-Independent Executable“ (PIE) übersetzt ist. Außerdem wird überprüft, ob bei der Übersetzung Schutzmechanismen gegen die Ausnutzung von Pufferüberläufen aktiviert wurden. Des Weiteren wird erfasst, ob das „Automatic Reference Counting“ (ARC) in der Anwendung verwendet wird.

Anschließend wird eine teilautomatisierte dynamische Analyse der Anwendung durchgeführt. Dabei werden bezüglich Datenschutzaspekten Zugriffe auf das Adressbuch, den Kalender und den aktuellen Aufenthaltsort erfasst. Außerdem wird die Übertragung des UDID protokolliert. Abschließend wird wieder ein Abbild des Dateisystems erstellt und mit dem ersten Abbild verglichen.

Bei der Analyse werden weitere identifizierende Daten wie die MAC-Adresse nicht beachtet. Auch personenbezogene Daten wie die Fotos und Videos bleiben bei der Analyse durch „SiRA“ außen vor. In der Arbeit wird darüber hinaus nur das Verfahren präsentiert, es werden keine Ergebnisse für konkrete Anwendungen gezeigt. Der Quelltext des Analyseverfahrens wurde entgegen der Ankündigung in der Veröffentlichung bisher nicht publiziert.

Zusammenfassung

Aufgrund der großen und stark ansteigenden Zahl von Anwendungen im Apple App Store wird eine vollständig automatische Analyse der Anwendungen angestrebt.

Ein Vergleich der bisherigen Arbeiten im Bereich der Anwendungsanalyse für Smartphones ist in Tabelle 3.1 zu finden.

Die statische Analyse, wie sie in „PiOS“ [26] oder der „App Genome Report“ [39] durchführt, hat aufgrund der dynamischer Objekttypisierung und dynamischem Binden von Methoden in Objective-C Grenzen. Daher wird in späteren Arbeiten wie der von Szydlowski [51] oder „PSiOS“ [55] eine dynamische Analyse angestrebt. Die Herausforde-

Analyse	Platform	Anzahl	Verfahren
TaintDroid (2010)	Android	30	dynamisch
App Genome Report (2010)	iOS & Android	unbekannt	statisch
PiOS (2011)	iOS	1407	statisch
AppInspector (2011)	Android	0	dynamisch
Szydlowski (2011)	iOS	1	dynamisch
Appthority (2012)	iOS & Android	50	statisch & dynamisch
Bitdefender (2012)	iOS	unbekannt	unbekannt
SiRA (2012)	iOS	0	statisch & dynamisch
AppsPlayground (2013)	Android	3968	dynamisch
PSiOS (2013)	iOS	15	statisch & dynamisch

Tabelle 3.1: Vergleich der aktuellen Forschung bezüglich analysierter Platform, Zahl der Anwendungen und verwendetem Analyseverfahren.

rung dabei ist die Ausführung aller Anweisungen einer Anwendung. Dies ist auf der Android-Platform bereits mit den Arbeiten „TaintDroid“ [28] und „AppsPlayground“ [47] erforscht, für iOS-Anwendungen jedoch noch nicht verfügbar.

Das von Apple praktizierte Prüfverfahren und die Prüfverfahren von kommerziellen Anbietern wie Bitdefender [17] und Appthority [12, 13] sind nicht öffentlich dokumentiert. Sie untersuchen zwar teils eine große Zahl an Anwendungen, veröffentlichen aber nicht alle Details ihrer Analyseverfahren. Daher ist es wichtig, ein öffentlich vollständig dokumentiertes, für eine große Zahl von Anwendungen skalierbares Prüfverfahren zu realisieren.

Die bisherigen Analysen decken sowohl hinsichtlich Datenschutz- als auch Sicherheitsaspekten manche Daten wie die WLAN-MAC-Adresse nicht ab. Darüber hinaus wird die Verwendung von Sensoren und Aktoren in den Analysen nicht berücksichtigt. Die dynamische Analyse wurde bisher nur nicht-öffentlich durchgeführt.

Aus diesen Gründen wird in der vorliegenden Arbeit ein performantes, dynamisches Analyseverfahren gezeigt, das auch für eine große Zahl von Anwendungen Analysekriterien erfassen kann.

4 Entwurf und Implementierung des Analyseverfahrens

Ziel des Analyseverfahrens ist es festzustellen, auf welche personenbezogenen und identifizierenden Daten eine Anwendung zugreift sowie welche Sensoren und Aktoren diese verwendet.

In diesem Kapitel wird ein Überblick über den Ansatz der Analyse gegeben, mit dem Anwendungen aus dem Apple App Store analysiert werden. Dazu werden die Vorteile des dynamischen Analyseverfahrens gegenüber dem statischen Verfahren erläutert. Des Weiteren wird das generelle Konzept für die dynamische Analyse und die konkrete Vorgehensweise je Anwendung erläutert. Schließlich werden Implementierungsdetails für die Komponenten des Analyseverfahrens beschrieben.

4.1 Vergleich von statischer und dynamischer Analyse

Anwendungen realisieren den Zugriff auf Daten, Sensoren und Aktoren über die Verwendung von API-Funktionen. Um herauszufinden, auf welche Daten eine Anwendung zugreift, wird daher eine Liste der verwendeten API-Funktionen erstellt. Im Folgenden werden statische und dynamische Analyse zur Ermittlung der verwendeten API-Funktionen miteinander verglichen.

In der statischen Analyse wird der Binärcode von Anwendungen auf die Verwendung von API-Funktionen durchsucht. In der dynamischen Analyse hingegen wird zur Laufzeit festgestellt, welche API-Funktionen von einer Anwendung aufgerufen werden.

Die Anwendungen im Apple App Store werden verschlüsselt angeboten und auch auf dem Smartphone verschlüsselt abgelegt. Sie werden erst direkt vor der Ausführung entschlüsselt. Eine dynamische Analyse kann auf diese Entschlüsselung durch das Betriebssystem zurückgreifen. Für eine statische Analyse dagegen müssen die Anwendungen zunächst entschlüsselt und abgespeichert werden [26].

In der dynamischen Analyse ist es das Ziel, die tatsächlich ausgeführten API-Funktionen festzustellen. In der statischen Analyse wird lediglich das Vorhandensein von Anweisungen ermittelt. Um darüber hinaus festzustellen, ob und wann eine API-Funktion tatsäch-

lich ausgeführt wird, muss in der statischen Analyse der vollständige Kontrollflussgraph aufwändig rekonstruiert werden.

In der dynamischen Analyse kann der Netzwerkverkehr einer Anwendung aufgezeichnet werden. In einer statischen Analyse muss er sehr aufwändig aus dem Datenfluss rekonstruiert werden und selbst dann gibt es Situationen, in denen dies nicht möglich ist. Wenn eine Anwendung zuerst Daten lädt und dann basierend auf diesen Daten weiteren Netzwerkverkehr generiert, ist dies in der statischen Analyse nicht nachvollziehbar. In der dynamischen Analyse steht der gesamte Netzwerkverkehr zur Laufzeit zur Verfügung und kann aufgezeichnet und analysiert werden.

Während der dynamischen Analyse muss dafür gesorgt werden, dass möglichst alle Anweisungen der Anwendung mindestens einmal ausgeführt werden, um die Anweisungen erfassen zu können. In der statischen Analyse hingegen liegen bis auf absichtlich verschleierte Anweisungen alle Instruktionen zur Analyse offen.

Die Herausforderung, dass möglichst alle Programmteile mindestens einmal ausgeführt werden, ist mit der Testabdeckung beim Testen von Software vergleichbar. Dabei gibt es verschiedene Maße für die Testabdeckung: Die Methodenabdeckung ist als Quotient der ausgeführten Methoden zur Gesamtzahl der Methoden definiert. Die Zweigabdeckung ist als Quotient der ausgeführten Programmzweige zur Gesamtzahl der Programmzweige definiert. Die Anweisungsüberdeckung misst den Quotient der ausgeführten Anweisungen zur Gesamtzahl der Anweisungen [57].

Da eine vollständige Anweisungsüberdeckung schwer zu erreichen ist [32], bedeutet das Nicht-Zutreffen eines Analysekriteriums lediglich, dass die entsprechende Anweisung nicht ausgeführt wurde. Es ist aber möglich, dass die Anweisung im nicht ausgeführten Abschnitt der Anwendung dennoch vorhanden ist. Daher ist ein Eintrag im Protokoll nur ein hinreichendes Kriterium dafür, dass die Anweisung in der Anwendung vorhanden ist. Nur bei vollständiger Anweisungsüberdeckung wäre es ein notwendiges Kriterium. Ein Beispiel verdeutlicht die Situation: Wenn bei einer Anwendung eine API-Funktion für den Adressbuchzugriff erfasst wird, dann verwendet die Anwendung nachweislich das Adressbuch. Wenn keine API-Funktion für den Adressbuchzugriff erfasst wird, heißt das nicht, dass die Anwendung nicht doch das Adressbuch verwenden könnte. Es bedeutet nur, dass die Anwendung mit den ausgeführten Anweisungen nicht auf das Adressbuch zugegriffen hat. Der Zugriff kann noch durch andere Anweisungen erfolgen, die lediglich während der Analyse nicht ausgeführt wurden. Daher kann nur das Vorkommen eines API-Zugriffs gesichert festgestellt werden, aber nicht dessen Gegenteil.

Da nur das Ereignis eines API-Zugriffs festgestellt werden kann, nicht aber dessen Abwesenheit, zeigt die Parallelen zwischen Softwaretest und dynamischer Anwendungsanalyse. Auch im Softwaretest postuliert Dijkstra, dass beim Test von Software nur die „Existenz eines Fehlers, aber nicht dessen Abwesenheit“ [57] gezeigt werden kann.

Da die Vorteile für die dynamische Analyse überwiegen und die statische Analyse für iOS-Anwendungen bereits durchgeführt wurde [26], wird in dieser Arbeit ein dynamisches Analyseverfahren angewendet.

4.2 Konzept

Zur Realisierung der dynamischen Analyse müssen die Anwendungen einzeln ausgeführt werden. Die Anwendungen von Drittanbietern liegen nur in kompilierter Form im ARM-Instruktionssatz vor. Da keine virtuelle Maschine für iOS-Anwendungen existiert, kann die Ausführung nicht auf x86-basierten PCs durchgeführt werden. Somit wird die Analyse auf dem Smartphone durchgeführt.

Da über API-Funktionen auf personenbezogene und identifizierende Daten sowie Sensoren und Aktoren zugegriffen wird, ist das Ziel der Analyse die Protokollierung aller aufgerufenen API-Funktionen.

Für die Protokollierung wird API-Hooking verwendet. Mit API-Hooking wird das Ziel eines API-Aufrufs auf eine eigene Funktion umgeleitet. In dieser Funktion wird der Aufruf der entsprechenden API-Funktion protokolliert, anschließend die ursprüngliche API-Funktion ausgeführt und deren Ergebnis an die Anwendung zurückgeliefert. Dieses Konzept wird in Abbildung 2.2 gezeigt. Der Ansatz ist dabei für die Anwendung transparent und kann ohne Modifikation der Anwendungen für beliebige Anwendungen genutzt werden.

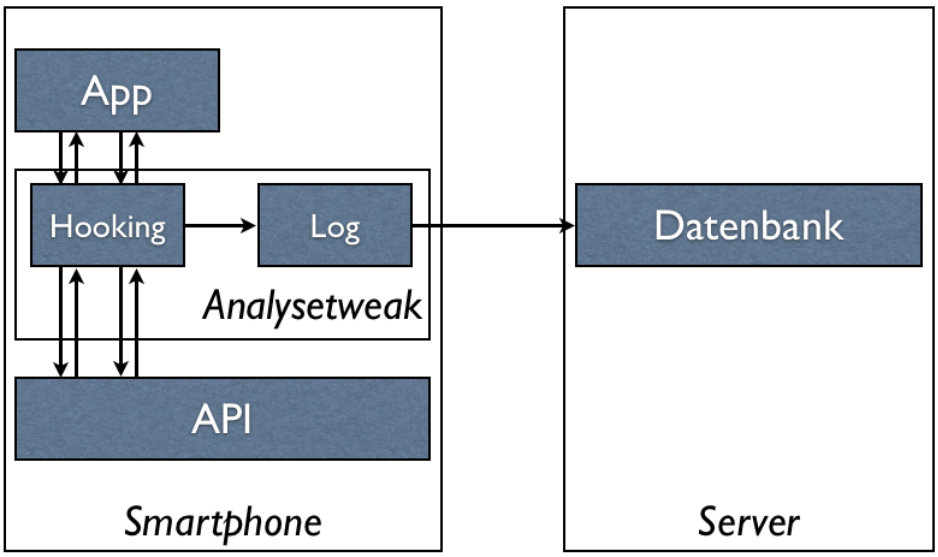

Abbildung 4.1: Konzept des Analyseverfahrens. API-Aufrufe aus der Anwendung werden über API-Hooking im Tweak abgefangen, in ein Log gespeichert und am Ende der Ausführung über eine REST-Schnittstelle an eine Datenbank übermittelt.

Zur Realisierung des API-Hooking für iOS wird für Objective-C-API-Funktionen „CaptainHook“ [43] verwendet. Dieses greift intern auf Funktionen der Objective-C-Runtime [10] zurück, um das Ziel von API-Aufrufen zu verändern. Für C-API-Funktionen wird „MobileSubstrate“ [30] zur Realisierung von API-Hooking genutzt.

API-Hooking wäre auch mit Hilfe eines Debuggers realisierbar. Szydlowski et al. [56] verwenden den Debugger GDB zur dynamischen Analyse nach diesem Ansatz. Nach Angaben der Autoren stellte sich dieser als nicht praktikabel heraus, da die Performance-Einbußen zu groß waren. Daher wird in dieser Arbeit CaptainHook und MobileSubstrate verwendet.

Der Code für das API-Hooking wird in einer dynamischen Bibliothek gespeichert. Diese muss beim Start zu den zu analysierenden Anwendungen dazugeladen werden. Die dynamische Bibliothek wird daher als sogenannter Tweak für MobileSubstrate realisiert. Mit Hilfe eines Filters lässt sich bei MobileSubstrate festlegen, welche Tweaks beim Start einer Anwendung geladen werden. Um den Tweak zu kompilieren, der in dieser Arbeit zur dynamische Analyse programmiert wurde, wird Theos [34] verwendet. Dies ist eine Sammlung von Makefiles zur Übersetzung von iOS-Anwendungen und Tweaks.

Da iOS im Auslieferungszustand das Laden von eigenen dynamischen Bibliotheken nicht unterstützt, wird ein Jailbreak angewendet. Dieser entfernt die von Apple eingestellten Restriktionen und erlaubt es deswegen, MobileSubstrate zu verwenden.

Mit dem beschriebenen Konzept wird eine dynamische Analyse umgesetzt, die den Zugriff auf personenbezogene und identifizierende Daten sowie die Verwendung von Sensoren und Aktoren über die Protokollierung der aufgerufenen API-Funktionen erfasst.

4.3 Auswahl der Anwendungen

Für die Analyse muss zunächst eine systematische Auswahl von Anwendungen getroffen werden. Diese Auswahl ist notwendig, da aus Zeitgründen nicht alle über 800.000 verfügbaren Anwendungen im Apple App Store analysiert werden können. Wenn eine Auswahl getroffen wird, ist es wichtig, die genaue Auswahl zu dokumentieren, um in weiteren Arbeiten Vergleiche zu ermöglichen.

Der Apple App Store gliedert sich in 22 Kategorien, die jeweils eine sehr unterschiedliche Zahl von Anwendungen enthalten. Spiele, Lern- und Unterhaltungs-Anwendungen sind die zahlenmäßig größten Kategorien, wie in Kapitel 2.1 beschrieben ist.

Apple bietet Ranglisten mit 300 Einträgen für die am häufigsten heruntergeladenen kostenlosen Anwendungen an. Dabei wird sowohl eine Gesamtliste über alle Kategorien als auch eine Rangliste pro Kategorie bereitgestellt [4]. Da die am häufigsten heruntergeladenen Anwendungen vermutlich von besonders vielen Menschen verwendet werden, werden diese Anwendungen in dieser Arbeit analysiert.

Bisherige Analysen von Egele et al. [26] oder Rastogi et al. [47] analysieren zwar mehrere tausend Anwendungen, geben dabei aber keinerlei Auskunft über die Auswahl der analysierten Anwendungen. Daher lassen sich deren Ergebnisse nicht generalisieren und auch nicht mit den hier gewonnenen Ergebnissen vergleichen.

4.4 Vorgehensweise je Anwendung

Zur dynamischen Analyse einer einzelnen Anwendung wird folgendes Vorgehen durchgeführt:

1. Installation der Anwendung aus dem App Store
2. Start der Anwendung
3. Beim Start: Analysetweak wird über MobileSubstrate geladen
4. Beim Laden: Konstruktor des Analysetweaks wird aufgerufen

5. Im Konstruktor: API-Hooking mit CaptainHook und MobileSubstrate
6. Ausführung von Aktionen in der laufenden Anwendung: Dabei aufgerufene API-Funktionen werden in Datenstruktur im Tweak protokolliert
7. Betätigung des Home-Buttons, wodurch das Signal „UIApplicationWillResignActiveNotification“ ausgelöst wird
8. Bei UIApplicationWillResignActiveNotification: Übertragung des Protokolls an die Datenbank
9. Beenden der Anwendung
10. Deinstallation der Anwendung

Mit diesem Vorgehen wird sichergestellt, dass immer nur eine Anwendung auf dem Gerät installiert und ausgeführt wird. Damit ist die gegenseitige Beeinflussung von Anwendungen ausgeschlossen.

Das API-Hooking wird beim Start der Anwendung durchgeführt. Der Konstruktor wird dazu mit der Compiler-Direktive `__attribute__((constructor))` markiert. Dies führt dazu, dass der Konstruktor direkt nach Laden des Tweaks initialisiert wird.

Das Protokoll der ausgeführten API-Funktionen wird innerhalb des Tweaks abgelegt. So ist es während der Analyse aus gehookten Funktionen mit geringem Aufwand erreichbar. Vor Beendigung der Anwendung wird es an die Datenbank auf dem Server übertragen. Falls jedoch eine analysierte Anwendung abstürzt, sind die Ergebnisse für diese Anwendung verloren. Dies wird in Kauf genommen, da die nochmalige Analyse einer einzelnen Anwendung mit geringem Aufwand möglich ist.

Die Schritte „Installation“, „Start der Anwendung“, „Betätigung des Home-Buttons“, „Beenden der Anwendung“ und „Deinstallation der Anwendung“ werden über die Automatisierungslösung von Weinlein [54] realisiert.

4.5 Implementierung des Analysetweaks

Der Analysetweak ist in Objective-C programmiert. Zur Umsetzung der Analysekriterien werden 46 Instanzmethoden, 25 Klassenmethoden und 26 C-Funktionen gehookt.

Bei der Implementierung des Tweaks wird das Beobachter-Entwurfsmuster für Funktionen verwendet, die von mehreren Kriterien erfasst werden müssen. Der Tweak wird mit der Methodik der „Testgetriebenen Entwicklung“ implementiert und getestet.

4.5.1 Beobachter-Entwurfsmuster

Das Beobachter-Entwurfsmuster (auch Observer-Pattern) ist ein Verhaltensmuster zur Software-Entwicklung. Es ermöglicht eine 1-zu-n Abhängigkeit zwischen Objekten, sodass eine Änderung an einem Objekt zur Benachrichtigung aller abhängigen n Objekte führt [31]. Die zu beobachtende Klasse wird als Subjekt, die abhängigen Klassen als Beobachter bezeichnet.

Die Struktur dieses Musters wird verwendet, um mehrere Kriterien über den Aufruf einer API-Funktion zu benachrichtigen. So wird allen die Möglichkeit gegeben, auf diesen Aufruf zu reagieren. Mit Hilfe dieser Vorgehensweise können mehrere Kriterien beispielsweise `open()` und ähnliche APIs hooken und nacheinander die übergebenen Parameter überprüfen.

Beispielhaft sei der `AAOpenTweak` beschrieben, der `open()`, `fopen()`, `stat()` und `lstat()` hookt. An diesem Subjekt registrieren sich die Beobachter-Klassen, um dann bei einem Aufruf der beobachteten Schnittstelle benachrichtigt zu werden.

Abbildung 4.2 zeigt den Zusammenhang zwischen Subjekt und Beobachtern am Beispiel des `AAOpenTweak`.

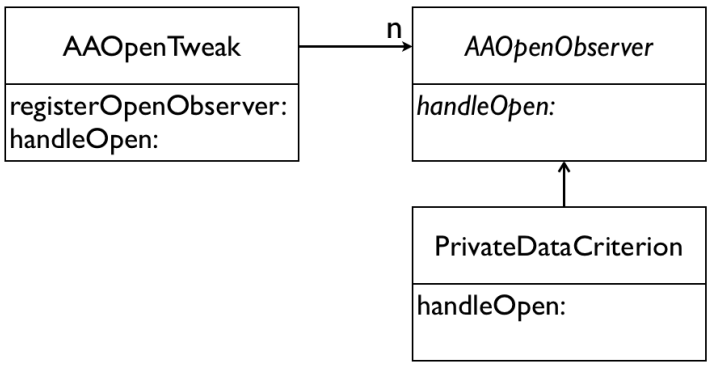

Abbildung 4.2: Beobachter-Entwurfsmuster für `open()`-API am Beispiel des Subjekts `AAOpenTweak` und des Beobachters `PrivateDataCriterion`

4.5.2 Testgetriebene Entwicklung

Zur Entwicklung des API-Hooking-Codes für die Analysekriterien wird der Ansatz der „Testgetriebenen Entwicklung“ [15] verwendet.

Bei der Umsetzung der einzelnen Kriterien ist es wichtig, dass der Programmcode alle zu erfassenden API-Aufrufe registriert. Um dies zu testen kann der Code auf dem Gerät zu einer Anwendung geladen werden, welche die entsprechenden API-Aufrufe ausführt.

Danach muss die Auswertung für diese Anwendung in der Datenbank mit den bekannten ausgeführten API-Aufrufen verglichen werden. Wenn dies übereinstimmt, ist die Funktionalität des Programmcodes für die Kriterien gegeben. Dieser Ansatz ist zeitaufwändig und aufgrund des manuellen Abgleichs fehleranfällig.

Einen weniger aufwändigen und weniger fehleranfälligen Ansatz bietet die Methode der „Testgetriebenen Entwicklung“. Hierbei werden Unit-Tests in die Entwicklung integriert und über diese die Funktionalität geprüft. Es wird folgender Entwicklungsansatz verwendet: Pro neuer Funktionalität, die implementiert werden soll, wird zuerst ein Testfall programmiert. Dieser schlägt zunächst ohne die funktionierende Implementierung bei Ausführung fehl („red“). Dann wird die eigentliche Funktion entwickelt und der Testfall schlägt bei korrekter Implementierung nicht mehr fehl („green“). Abschließend wird die Funktionalität noch soweit verbessert, dass sie den nicht-funktionalen Anforderungen genügt („refactor“) [15].

Für die Entwicklung des Tweaks bedeutet dies, dass zuerst ein Testfall für ein zu programmierendes Kriterium erstellt wird. Dann wird das API-Hooking für das Kriterium entwickelt. Abschließend wird der Programmcode beispielsweise hinsichtlich Doppelstrukturen verbessert.

Die testgetriebene Entwicklung ist für diesen Programmcode besonders gut geeignet, da alle Testfälle dem selben Schema folgen und mit wenigen Programmzeilen zu entwerfen sind. In jedem Testfall muss geprüft werden, ob eine aufgerufene API-Funktion im Protokoll erfasst wurde. Der typische Testfall für ein Kriterium, das den Aufruf `SampleClass: - (void) doIt` erfassen soll, wird in Listing 4.1 gezeigt. Dabei ist ersichtlich, dass für den Test von Instanzmethoden zwei Anweisungen, für den Test von Klassenmethoden nur eine Anweisung notwendig ist.

```
1 STAssertFalse([[AABackend sharedInstance] resultForCriterion:@"Sample");
2
3 SampleClass* sample = [[SampleClass alloc] init];
4 [sample doIt];
5
6 STAssertTrue([[AABackend sharedInstance] resultForCriterion:@"Sample"]);
```

Listing 4.1: Testfall eines Kriteriums für die Erfassung einer Klassenmethode

Diese Art der Entwicklung ist weniger fehleranfällig, da nach jeder Veränderung des Quelltexts alle bisherigen Testfälle ausgeführt werden können. So kann überprüft werden, ob die bisher funktionierenden Kriterien auch nach größeren Veränderungen weiterhin alle API-Aufrufe erfassen.

4.6 Vorbereitung des Analysegeräts

Das in den Analysen verwendete Gerät ist ein Apple iPhone 4 mit 8 GB persistentem Speicher (siehe Kapitel 2.2). Alle Analysen werden mit der iOS-Softwareversion 6.1.0 durchgeführt, die am 28. Januar 2013 veröffentlicht wurde.

Vor der Verwendung zur Analyse sind folgende Schritte zur Vorbereitung notwendig: Der nicht kabelgebundene Jailbreak „evasi0n“ [14] wird durchgeführt. Mit diesem werden die im Auslieferungszustand vorhandenen Restriktionen bezüglich nicht-signiertem Code aufgehoben. Außerdem ist dann ein uneingeschränkter Benutzer (root-Zugang) möglich. Erst nach dem Jailbreak kann MobileSubstrate installiert werden, sodass der Analysetweak geladen werden kann.

Zur weiteren Vorbereitung wird der in der Arbeit entwickelte Analysetweak und Packet Collector sowie die von Weinlein [54] entwickelte Automatisierung auf dem Gerät installiert.

Im Auslieferungszustand sind bei iOS Version 6 Datenbanken wie beispielsweise Adressbuch und Kalender ohne Inhalt. Um ein realistisches Szenario zu analysieren, werden folgende Beispieldaten im Gerät hinterlegt.

Adressbuch: Michael Mustermann

- Name: Michael Mustermann
- Telefon (Mobil): +4917691827364
- Telefon (Privat): 091156473829
- Email-Adresse: michael@mustermann.de
- Geburtstag: 01.06.1980

Kalender:

- Titel: DiesIstEinLangerTitel
- Ort: DiesIstEinLangerOrt
- Datum: 01.01.2000
- Anfang: 0:00 Uhr
- Ende: 3:00 Uhr
- Zeitzone: Europe/Berlin

Identifizierende Daten:

- UDID: c07e30a9fae7dead679ae604353df4c40da03e87
- WLAN-MAC-Adresse: AC:3C:0B:CD:15:7F

- ASID: 90C2D27C-C333-4737-9798-DF84CC4E0E12
- Telefonnummer: +4917698530662

iCloud-Account:

- Account: appleid_ger1@weinlein.info
- Aktivierte Dienste: „Dokumente und Daten“, „iPhone suchen“, „iCloud-Backup“

Facebook-Account:

- Email-Adresse: appanalysis1@meinefau.de
- Name: Max Meier

Twitter-Account:

- Benutzername: @AppAnalysisMax
- Email-Adresse: appanalysis1@meinefau.de

Der Netzwerkverkehr von Anwendungen kann nach diesen Beispieldaten durchsucht werden, um die Übermittelung von Daten über das Netzwerk festzustellen.

4.7 Realisierung der Ergebnis-Datenbank

Die Datenbank zur Speicherung der Analyseergebnisse besitzt folgende Schnittstelle:

- `POST /charts`: Rangliste aus dem App Store im JSON-Format [4] als Liste zur Analysedurchführung hinterlegen
- `POST /results`: Übermittlung eines Analyseergebnisses im JSON-Format
- `GET /results/<id>`: Resultate einer Analysedurchführung anzeigen
- `GET /missing/<id>`: Anwendungen einer Analysedurchführung auflisten, für die noch kein Analyseergebnis in der Datenbank vorliegt

Die Schnittstelle ist mit PHP programmiert und die Datenbank in einem MySQL-Datenbankserver abgelegt.

Das ER-Diagramm der Datenbank ist in Abbildung 4.3 abgebildet. Darin ist ersichtlich, dass die Liste der Anwendungen einer Analysedurchführung in der Tabelle „charts“ gespeichert wird. Die beschreibenden Attribute wie Start und Endzeitpunkt der Analysedurchführung sind in der Tabelle „schedules“ hinterlegt. Die Ergebnisse der Analyse werden in den Tabellen „results“, „trackingLibs“, „httpRequests“, „openPaths“ und „traces“ gesichert.

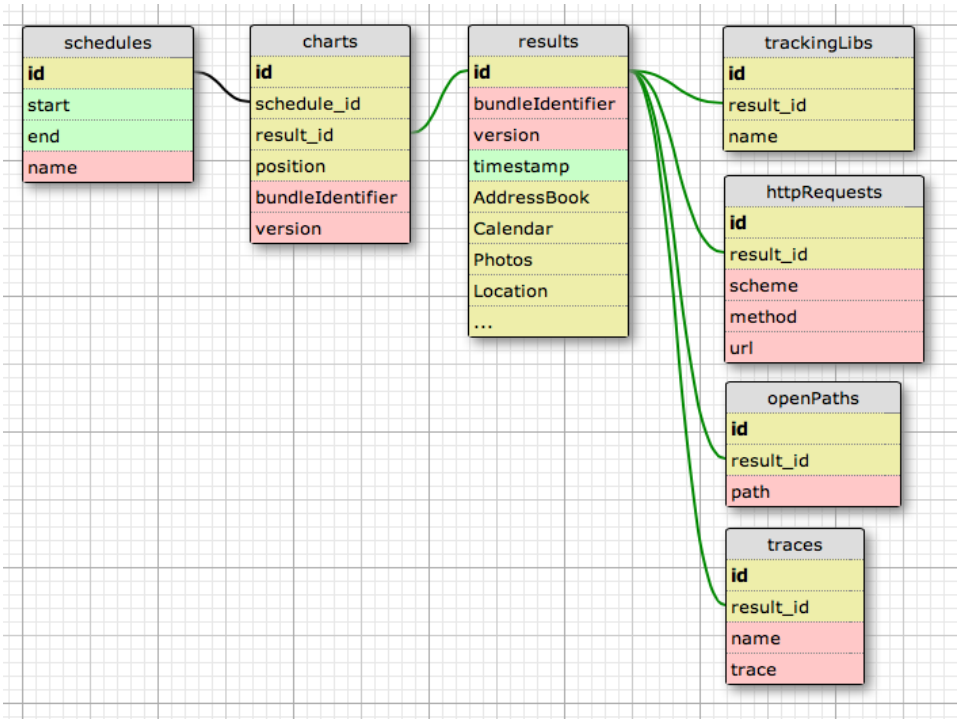

Abbildung 4.3: ER-Diagramm der Ergebnisdatenbank

5 Implementierung der Analysekriterien

In diesem Kapitel werden die Kriterien beschrieben, anhand derer die ausgewählten Anwendungen analysiert werden. Zusätzlich wird die technische Realisierung der einzelnen Kriterien dargestellt.

Die Kriterien stellen nur eine Erfassung der Aktivitäten der Anwendung dar. Es ist keine Bewertung der Anwendung möglich. Dies ist nur für den Einzelfall pro Anwendung möglich. Da die Analysen in dieser Arbeit aufgrund der großen Zahl an Anwendungen im Apple App Store automatisiert durchgeführt werden sollen, wird keine Bewertung sondern nur eine Erfassung der Aktivitäten der Anwendungen angestrebt. Ein Beispiel verdeutlicht die Einzelfallentscheidung: Wenn eine Anwendung zur Adressbuchsynchronisation auf das Adressbuch zugreift, ist dies für Nutzer legitim. Wenn aber eine Anwendung mit Taschenlampenfunktion auf das Adressbuch zugreift, ist dies anders zu bewerten. Da die Information, um welche Anwendung es sich handelt, in der automatischen Analyse nicht ermittelt werden kann, wird keine Bewertung vorgenommen.

Für die Notation werden die in den Kriterien verwendeten API-Funktionen für die C-Funktionen ohne Parameter angegeben. Für Objective-C-Funktionen werden sie mit `Klasse: Funktionstyp (Rückgabewert) Selektor` notiert. Auch in dieser Selektornotation sind die Parametertypen und -benennungen nicht aufgeführt.

5.1 Personenbezogene Daten

Die erste Gruppe von Kriterien erfasst den Zugriff auf personenbezogene Daten. Zu diesen gehören das Adressbuch, der Kalender, die Anruf-Historie, die Fotos und Videos, der Aufenthaltsort, inklusive „Geofencing“ und „Location Monitoring“, sowie die Zugangsdaten zu sozialen Netzwerken wie Facebook und Twitter.

5.1.1 Adressbuch

iOS erlaubt ab Version 2.0 über das AddressBook-Framework den Zugriff auf das Adressbuch des Nutzers. Ab Version 6.0 muss der Nutzer den ersten Zugriff auf das Adressbuch autorisieren.

Wenn diese Autorisierung besteht, wird über die API sowohl lesender als auch schreibender Zugriff ermöglicht. Es kann daher das vollständige Adressbuch ausgelesen oder verändert werden. Das Framework bietet eine C-API für das Adressbuch an, eine Objective-C-API existiert nicht.

Um den Zugriff von Anwendungen auf das Adressbuch festzustellen, werden folgende API-Funktionen gehookt:

- `open()`
- `fopen()`
- `ABAddressBookCreate()`
- `ABAddressBookCreateWithOptions()`
- `ABAddressBookRequestAccessWithCompletion()`
- `ABAddressBookGetPersonWithRecordID()`
- `ABAddressBookCopyArrayOfAllPeople()`
- `ABAddressBookCopyArrayOfAllPeopleInSource()`
- `ABAddressBookCopyArrayOfAllPeopleInSourceWithSortOrdering()`
- `ABAddressBookCopyPeopleWithName()`

Die Adressbuch-Datenbank ist auf dem Gerät als SQLite-Datenbank unter dem Pfad `/var/mobile/Library/AddressBook/AddressBook.sqlitedb` gespeichert. Mit iOS Version 6.0 ist aufgrund der Sandbox kein direkter Zugriff möglich. In früheren Versionen wäre für eine Anwendung auch der Zugriff mit der SQLite-API direkt auf diese Datenbank-Datei denkbar. Daher wird der Vollständigkeit halber auch `open()` und `fopen()` mit dem Pfad der SQLite-Datenbank als Zugriff auf das Adressbuch registriert.

5.1.2 Kalender und Erinnerungen

Über das EventKit-Framework wird unter iOS ab Version 4.0 der Zugriff auf den Kalender des Nutzers ermöglicht. Zusätzlich sind die Erinnerungen, die ein Nutzer in der Anwendung „Erinnerungen“ speichert, über das EventKit-Framework erreichbar. Ab Version 6.0 wird für die beiden Datentypen Kalendereinträge und Erinnerungen jeweils eine Autorisierung eingeholt.

Über die API ist sowohl lesender als auch schreibender Zugriff auf den Kalender des Nutzers möglich. Es existiert ausschließlich eine Objective-C-API.

Um den Zugriff auf den Kalender und die Erinnerungen festzustellen, werden folgende API-Funktionen gehookt:

- `open()`
- `fopen()`
- `EKEventStore: - (id) init`

Analog zum Adressbuch ist auch die Kalender-Datenbank auf dem Gerät als SQLite-Datei unter dem Pfad `/var/mobile/Library/Calendar/Calendar.sqlitedb` gespeichert. Mit iOS Version 6.0 ist aufgrund der Sandbox kein direkter Zugriff auf die Datei möglich. Da in früheren Versionen auf die Datenbank direkt zugegriffen werden konnte, wird der Vollständigkeit halber auch der Zugriff über `open()` oder `fopen()` auf diese Datei als versuchter Zugriff auf den Kalender registriert.

5.1.3 Fotos und Videos

Fotos und Videos werden von iOS im Datentyp „Asset“ zusammengefasst. Daher soll auch der Zugriff auf diese in einem Kriterium behandelt werden. Der Zugriff auf die Fotos und Videos des Nutzers wird unter iOS ab Version 4.0 über das AssetsLibrary-Framework ermöglicht. Seit Version 6.0 wird eine einmalige Autorisierung für diesen Zugriff eingeholt.

Bei iOS Version 4 und 5 erfolgte der Zugriff direkt auf die Foto- und Videodatenbank, welche als SQLite-Datei unter dem Pfad `/var/mobile/Media/PhotoData/Photos.sqlite` gespeichert ist. Daher wird ein API-Aufruf von `open()` oder `fopen()` auf diese Datei als versuchter Zugriff auf Fotos und Videos registriert.

Ab iOS Version 6 wird der Zugriff über den Dienst `assetsd` per Interprozesskommunikation (IPC) vollzogen, sodass auch die Objective-C-API der AssetsLibrary gehookt wird.

Für die Registrierung der Zugriffe auf Fotos und Videos werden damit folgende API-Zugriffe gehookt:

- `open()`
- `fopen()`
- `ALAssetsLibrary: - (id) init`

5.1.4 Aufenthaltsort, Geofencing und Location Monitoring

Der aktuelle Aufenthaltsort des Nutzers ist seit iOS Version 2.0 über das CoreLocation-Framework mit einmaliger Autorisierung für Anwendungen zugänglich. Der aktuelle Ort wird dabei über unterschiedliche Methoden ermittelt. Einerseits kann die ungefähre Position mit Triangulation anhand der Mobilfunkstationen ermittelt werden, welche sich in Reichweite befinden. Außerdem kann über in Reichweite befindliche WLAN-Basisstationen die Position ermittelt werden. Daneben besitzt das iPhone 4 einen GPS-Empfänger, der außerhalb von Gebäuden die beste Genauigkeit zur Positionsbestimmung liefert. Durch die Kombination der drei Techniken kann das Smartphone in fast allen Empfangsszenarien die Position bestimmen.

Auf der Positionsbestimmung bauen Techniken wie das „Geofencing“ und das „Location Monitoring“ auf. Die Autorisierung für den aktuellen Ort gilt auch für die Funktionen „Geofencing“ und „Location Monitoring“. Mittels Geofencing kann eine Anwendung benachrichtigt werden, wenn der Smartphone-Besitzer ein bestimmtes Gebiet betritt oder verlässt. Die Funktionalität wird in iOS ebenfalls vom CoreLocation-Framework angeboten.

Mittels „Location Monitoring“ ist es für eine Anwendung möglich, kontinuierlich über Standortänderungen informiert zu werden. Die Anwendung wird bei einer Standortänderung automatisch im Hintergrund gestartet und erhält zehn Sekunden Zeit, um auf die Standortänderung zu reagieren. Mit dieser Funktion kann die Anwendung ein Bewegungsprofil des Nutzers erstellen. Während dieser Zeit kann die Anwendung jedoch auch beliebige andere Programmaktionen durchführen. Die Funktion des CoreLocation-Frameworks kann daher missbraucht werden, um Anwendungen zu programmieren, die vom Nutzer nicht dauerhaft beendet werden können.

Für den Nutzer ist bei der Autorisierung nicht abzusehen, um welchen der genannten Zugriffe es sich handelt. Daher wird bei der Analyse zwischen aktuellem Ort, „Geofencing“ und „Location Monitoring“ unterschieden.

Dazu werden folgende API-Funktionen gehookt:

- `CLLocationManager: - (void) setDelegate:`
- `CLLocationManager: - (void) startUpdatingLocation`
- `CLLocationManager: - (void) startMonitoringForRegion`
- `CLLocationManager: - (void) startMonitoringSignificantLocationChanges`

5.1.5 Soziale Netzwerke

Im Betriebssystem iOS lassen sich seit Version 5.0 Zugangsdaten für Twitter und seit Version 6.0 Zugangsdaten für Facebook und SinaWeibo hinterlegen. Über diese Daten

kann Anwendungen der Zugriff auf den jeweiligen Account bei dem entsprechenden sozialen Netzwerk ermöglicht werden. Jede Anwendung muss dafür einmalig autorisiert werden. Danach ist sowohl lesender als auch schreibender Zugriff auf die Daten des Accounts möglich.

Bei diesem Verfahren werden jedoch nicht die Zugangsdaten selbst an die Anwendung herausgegeben. Stattdessen wird ein Token verwendet, mit dem der Zugriff ermöglicht wird.

Um den Zugriff von Anwendungen auf die Accounts des Nutzers in sozialen Netzwerken zu registrieren, werden folgende API-Zugriffe gehookt:

- `ACAccountStore: - (void) requestAccessToAccountsWithType: withCompletionHandler:`
- `ACAccountStore: - (void) requestAccessToAccountsWithType:options: completion:`

5.1.6 Anruf-Historie

Die Anruf-Historie ist bei iOS als SQLite-Datenbank in der Datei `/private/var/wireless/Library/CallHistory/call_history.db` gespeichert [50]. Auf diese kann in iOS Version 3 und Version 4 mit den SQLite-API-Funktionen lesend zugegriffen werden. Ab Version 5 wird ein Zugriff auf die Datei durch die Sandbox unterbunden.

In der SQLite-Datenbank sind Zeitpunkt, Dauer, Gesprächspartner, Art des Gesprächs (Mobilfunk, FaceTime) und Gesprächsinitiator für jedes Gespräch hinterlegt. Da diese Informationen für 100 Gespräche gespeichert werden, kann die Gesprächshistorie sehr genau analysiert werden [2].

In Tabelle 5.1 werden Beispieleinträge für die Datenbank gezeigt. Dabei sind Gesprächsinitiator und Art des Gesprächs in der Spalte `flags` kodiert. Die Bedeutung der bekannten Einträge wird in Tabelle 5.2 aufgelistet. Um mehrere Einträge zu kodieren, werden sie bitweise mit „oder“ verknüpft.

Um Zugriffe auf die Anruf-Historie zu erfassen, werden `sqlite3_open()`-Aufrufe auf den Pfad der Datenbank gehookt.

id	address	date	duration	flags	country_code	network_code
1	a@sample.s	1362154457	0	21	000	00
2	+49123456789	1362154695	8	4	262	07
3	09111234567	1365403137	0	5	262	07
4	09111234567	1365409697	10	5	262	07
5	b@sample.s	1366201384	0	21	262	07
6	c@sample.s	1366201397	0	21	262	07

Tabelle 5.1: Beispielhafter Inhalt der Datei `call_history.db`, gekürzt um die leeren Spalten `name`, `read`, `assisted`, `face_time_data`, `originalAddress`

flag (Dezimal)	Bedeutung
0	eingehender Anruf
1	ausgehender Anruf
4	Unbekannt
8	Unbekannt
16	FaceTime-Anruf (VoIP)
65536	Keine Netzabdeckung
131072	Unbekannt
262144	Unbekannt
524288	Unbekannt
1048576	Anruf wurde aufgrund geringer Netzabdeckung abgebrochen

Tabelle 5.2: Einträge der `flags`-Spalte der `call_history.db`. Um mehrere Einträge zu kodieren, werden sie bitweise mit „oder“ verknüpft [2].

5.2 Identifizierende Daten

Die zweite Gruppe der Kriterien erfasst den Zugriff auf identifizierende Daten. Dazu gehören der UDID, der VID und der ASID, die MAC-Adresse der WLAN-Schnittstelle und die Telefonnummer und die ICCID.

5.2.1 Unique Device Identifier

Der „Unique Device Identifier“ (UDID) steht Anwendungen seit iOS Version 2.0 über das UIKit-Framework zur Verfügung. Mit Einführung von iOS Version 5.0 ist er als veraltet markiert, steht aber weiterhin zur Verfügung.

Der UDID wird vom Betriebssystem je nach Gerät unterschiedlich gebildet (siehe Kapitel 2.3.2) und ist als Kombination von unveränderlichen Daten auch in der Gesamtheit unveränderlich.

Um den Zugriff von Anwendungen auf den UDID zu erfassen, wird folgende API gehookt:

- `UIDevice: - (NSString *) uniqueIdentifier`

5.2.2 Identifier for Vendor und Advertising Identifier

Mit iOS Version 6.0 wurden im UIKit-Framework und im AdSupport-Framework zwei Alternativen zum UDID eingeführt.

Einerseits wurde ein „Identifier for Vendor“ (VID) eingeführt, der für alle Anwendungen eines Herstellers auf einem Gerät eindeutig ist. Er dient dazu, dass Anwendungshersteller das Gerät wieder erkennen können. Im Unterschied zum UDID wird die Eindeutigkeit beim VID nur für jeweils einen Hersteller garantiert. Zwei Anwendungen von unterschiedlichen Herstellern erhalten daher auf dem selben Gerät unterschiedliche Werte für diesen Identifier. So ist die Profilbildung wie mit dem UDID über Anwendungsgrenzen nicht mehr möglich.

Außerdem wurde mit iOS Version 6.0 und dem neuen AdSupport-Framework ein „Advertising Identifier“ (ASID) eingeführt. Dieser ist über Anwendungsgrenzen hinweg konstant, er lässt sich aber vom Nutzer verändern. In iOS Version 6.0 konnte der ASID nur durch das vollständige Zurücksetzen des Geräts geändert werden, mit Version 6.1 kann der ASID gezielt über das Einstellungsmenü zurückgesetzt werden. Da die ASID über Anwendungsgrenzen hinweg konstant ist, lässt sich ein Gerät und damit ein Nutzer immer noch nachverfolgen.

Entsprechend werden für Zugriffe auf den VID und den ASID folgende APIs gehookt:

- `UIDevice: - (NSUUID*) identifierForVendor`
- `ASIdentifierManager: - (NSUUID*) advertisingIdentifier`

5.2.3 MAC-Adressen

iOS ermöglicht seit Version 2.0 den Zugriff auf die WLAN-MAC-Adresse. Der Zugriff kann über die BSD-APIs `ioctl()`, `sysctl()` oder `getifaddrs()` erfolgen. Dabei kann nur die MAC-Adresse der WLAN-Schnittstelle abgefragt werden, da die API-Funktionen zum BSD-Netzwerkstack zugeordnet sind. Die MAC-Adresse der Bluetooth-Schnittstelle kann so nicht ausgelesen werden. Eine Objective-C-API für die MAC-Adresse existiert dagegen nicht.

Um den Zugriff von Anwendungen auf die MAC-Adresse zu registrieren, werden folgende API-Funktionen gehookt [37]:

- `ioctl()`
- `sysctl()`
- `getifaddrs()`

Die Parameter dieser Funktionen werden überprüft und nur wenn tatsächlich auf die MAC-Adresse zugegriffen wird, wird ein Protokoll-Eintrag erstellt.

Auch Framework-Threads und Framework-Funktionen zur Netzwerkkommunikation greifen über die gehookten API-Funktionen auf die MAC-Adresse zu. Da diese Framework-Funktionen die MAC-Adresse nur intern verwenden und den Anwendungen nicht zur Verfügung stellen, müssen diese Zugriffe herausgefiltert werden. Dies wird über einen Abgleich mit der Aufrufhierarchie gelöst. Nur wenn aus dem Code der aktuellen Anwendung `ioctl()`, `sysctl()` oder `getifaddrs()` aufgerufen wird, wird der Aufruf auch protokolliert.

Beim Zugriff auf die MAC-Adresse wird deutlich, dass iOS im Kern nah mit seinem Desktop-Pendant OS X verwandt ist und dass die zur Verfügung stehenden BSD-APIs auch Zugriff auf identifizierende Daten ermöglichen.

5.2.4 Telefonnummer und ICCID

Die „Integrated Circuit Card ID“ (ICCID) ist eine Kennung, die die SIM-Karte des Nutzers eindeutig identifiziert. Sie ist für den Nutzer nur durch den Austausch der SIM-Karte veränderbar. Die Telefonnummer und die ICCID standen seit iOS Version 3.0 bis Version 6.0 für alle Anwendungen zur Verfügung. Sie sind in der Datei `/private/var/wireless/Library/Preferences/com.apple.commcenter.plist` gespeichert, die mit Hilfe des Foundation-Frameworks ausgelesen werden konnte. In iOS Version 6 wird dies von der Sandbox unterbunden.

In Abbildung 5.1 wird der Inhalt der Einstellungsdatei `com.apple.commcenter.plist` gezeigt beispielhaft gezeigt.

Um Zugriffe auf die Telefonnummer und die ICCID zu erfassen, werden die Aufrufe von `open()` und `fopen()` gehookt und auf den Pfad der Datei `com.apple.commcenter.plist` überprüft.

```
{
    CarrierBundleName = 12345;
    ICCID = 1234567890123456789;
    InternationalRoamingEDGE = 0;
    LASDNextUpdate = 2013-05-09 18:29:30 +0000;
    NextUpdate = 2013-05-01 09:22:20 +0000;
    PhoneNumber = "+49123456789";
    PhoneNumberChangeReport = 0;
}
```

Abbildung 5.1: Beispielhafter Inhalt der Datei `com.apple.commcenter.plist`.

5.3 Nutzungsstatistiken und Werbenetzwerke

Für iOS-Anwendungen stehen eine Vielzahl von Nutzungsstatistiken und Werbenetzwerken von Drittanbietern zur Verfügung. Da die gleichen Frameworks häufig bei vielen Anwendungen verwendet werden [26], ist es für Anbieter der Werbenetzwerke und Nutzungsstatistiken möglich, umfassende Statistiken und Profile über den Nutzer zu erstellen.

Zur Erfassung der in Kapitel 2.4 genannten Anbieter für Werbung und Nutzungsstatistiken werden folgende Funktionen gehookt:

- Flurry:
  - `Flurry: + (void) startSession`
  - `FlurryAnalytics: + (void) startSession`
- Google Analytics:
  - `GAITracker: - (id) trackerWithTrackingId:`
  - `GANTracker: + (id) sharedTracker`
- Admob: `GADRequest: + (id) request`
- TestFlight: `TestFlight: + (void)takeOff:`
- MobAge: `MBMobAge: initializeMobageWithServerEnvironment: appId:appVersion:consumerKey:consumerSecret:`
- MobileAppTracker: `MobileAppTracker: - (void) startTrackerWithAdvertiserId: advertiserKey:withError:`
- Mixpanel: `Mixpanel: + (id) sharedInstanceWithToken:`

- KISSMetrics: `KISSMetricsAPI: + (KISSMetricsAPI *) sharedAPIWithKey:`
- Apsalar:
  - `Apsalar: + (void) startSession:withKey:`
  - `Apsalar: + (void) startSession:withKey:andLaunchOptions:`
  - `Apsalar: + (void) startSession:withKey:andURL:`
- Tapjoy: `TapjoyConnect: + (void) requestTapjoyConnect:secretKey:`
- Medialets:
  - `MedialetsAnalyticsManager: - (void) initialize:`
  - `MedialetsAnalyticsManager: - (void) initializeWithAppID: appVersion:locationManager:`
- Localytics: `LocalyticsSession: - (void) startSession:`
- Bango: `BGOAnalyticsManager: + (id) sharedManager`
- Distimo: `DistimoSDK: - (void) handleLaunchWithOptions:sdkKey:`
- InMobi:
  - `IMAdRequest: + (id) request`
  - `IMAdView: - (id) initWithFrame:imAppId:imAdSize: rootViewController:`
  - `IMAdView: - (id) initWithFrame:imAppId:imSlotId:imAdSize: rootViewController:`
- Smaato:
  - `SOMABannerView: - (void) initWithDimension:`
  - `SOMAAdDownloader: - (void) addAdListener:`
  - `SOMAAdDownloader: - (void) setLocationUpdateEnabled:`
- AdColony: `AdColonyPublic: + (void) initAdColonyWithDelegate:`
- Appsfire: `AFAppBoosterSDK: + (void) connectWithAPIKey:`
- Chartboost: `Chartboost: - (void) startSession`
- JumpTap: `JTAdWidget: + (void) initializeAdService:`
- PinchMedia: `Beacon: + (void) initAndStartBeaconWithApplicationCode: useCoreLocation:useOnlyWiFi`

5.4 Sensoren

Das in der Untersuchung verwendete iPhone 4 besitzt eine Vielzahl an Sensoren, mit der es Informationen über die Umgebung erfassen kann. Marquardt et al. [41] zeigen, dass über die Sensoren auch sehr sensitive Daten generiert werden können. Sie verwenden ein iPhone mit Hilfe dessen Beschleunigungssensors als Passwort-Logger. Dazu messen sie mit dem Sensor die Erschütterungen, welche durch Tastaturanschläge einer daneben liegenden Tastatur verursacht werden. Aus der Messung rekonstruieren sie die eingegebenen Zeichen und speichern diese.

Eine noch offensichtlichere Beobachtung des Nutzers ist mit dem Mikrofon des iPhones möglich. Eine Anwendung kann bei iOS während ihrer Verwendung über das Mikrofon Gespräche aufzeichnen und an den Anwendungshersteller übertragen. Der Zugriff auf das Mikrofon erfolgt dabei ohne Autorisierung durch den Nutzer und kann damit unbemerkt bleiben.

Im Folgenden werden die Analysekriterien vorgestellt, die einen Zugriff auf das Mikrofon, die Kamera, den Beschleunigungssensor, den Rotationssensor und den Kompass erfassen.

5.4.1 Mikrofon

Auf das Mikrofon können Anwendungen können seit iOS Version 2.0 mit dem AudioUnit-Teil des CoreAudio-Frameworks zugreifen. Mit Version 3.0 wurde eine weitere Zugriffsmöglichkeit über das AVFoundation-Framework eingeführt. Dabei dürfen alle Anwendungen ohne vorherige Autorisierung auf das Mikrofon zugreifen.

Um zu zeigen, wie einfach ein Zugriff auf das Mikrofon mit Hilfe des AVFoundation-Frameworks möglich ist, wird in Listing 5.1 ein konkretes Beispiel gegeben. Dort sind die zwei notwendigen API-Aufrufe aufgeführt.

```
1  // Einstellungen
2  NSDictionary* settings = [NSDictionary dictionaryWithObjectsAndKeys:
3                            [NSNumber numberWithInt:kAudioFormatMPEG4AAC_LD],
                                 AVFormatIDKey,
4                            @44100, AVSampleRateKey,
5                            @1, AVNumberOfChannelsKey,
6                            nil];
7
8  // Eigentliche Aufnahme
9  AVAudioRecorder* recorder = [[AVAudioRecorder alloc] initWithURL:@"recording.m4a"
       settings:settings error:nil];
10 [recorder record];
```

Listing 5.1: Minimale API-Aufrufe zur Verwendung des Mikrofons

Innerhalb der AudioUnit-APIs wird nicht zwischen Mikrofon und Lautsprecher unterschieden, sondern ein virtuelles Gerät (`kAudioUnitSubType_RemoteIO`) repräsentiert beide. Der Eingabekanal ist dabei das Mikrofon, der Ausgabekanal der Lautsprecher.

Da nach der Initialisierung der AudioUnit basierend auf diesem Gerät Zugriffe auf die einzelnen Kanäle nicht mehr unterschieden werden kann, wird eine Initialisierung von `AudioComponentInstanceNew()` mit Parameter `kAudioUnitSubType_RemoteIO` als Verwendung von Mikrofon und Lautsprecher protokolliert.

Um Zugriffe auf das Mikrofon zu registrieren, werden folgende API-Funktionen gehookt:

- `AudioComponentInstanceNew()`
- `AudioUnitInitialize()`
- `AVAudioRecorder: - (id) initWithURL:settings:error:`
- `AVCaptureDevice: + (NSArray *) devicesWithMediaType:`
- `AVCaptureDevice: + (AVCaptureDevice *) defaultDeviceWithMediaType:`
- `AVCaptureDeviceInput: + (id) deviceInputWithDevice:error:`
- `AVCaptureDeviceInput: - (id) initWithDevice:error:`

5.4.2 Kamera

Das in der Untersuchung verwendete iPhone 4 besitzt zwei Kameras, eine auf der Gerätevorderseite, eine auf der Geräterückseite. Beide Kameras stehen Anwendungen zur Verfügung.

Anwendungen können seit iOS Version 4.0 mit Hilfe des AVFoundation-Framework mit einer Objective-C-API auf die Kamera zugreifen. Der Zugriff wird für alle Anwendungen ohne Autorisierung gewährt.

Um Zugriffe auf die Kamera zu registrieren, werden folgende API-Funktionen gehookt:

- `AVCaptureDevice: + (NSArray *) devicesWithMediaType:`
- `AVCaptureDevice: + (AVCaptureDevice *) defaultDeviceWithMediaType:`
- `AVCaptureDeviceInput: + (id) deviceInputWithDevice:error:`
- `AVCaptureDeviceInput: - (id) initWithDevice:error:`

5.4.3 Beschleunigungssensor, Rotationssensor, Kompass

Das iPhone besitzt seit der ersten Version einen dreiachsigen Beschleunigungssensor. Seit dem iPhone 3GS ist zusätzlich ein magnetischer Kompass verbaut. Beginnend mit dem iPhone 4 ist auch ein dreiachsiger Rotationssensor enthalten.

Anwendungen können seit iOS Version 2.0 über das UIKit-Framework auf den Beschleunigungssensor zugreifen. Mit iOS Version 3.0 kann auf den Kompass über das

CoreLocation-Framework zugegriffen werden. Seit iOS Version 4.0 sind der Rotationssensor, der Beschleunigungssensor und der Kompass für Anwendungen über das CoreMotion-Framework verfügbar. Der Zugriff auf Beschleunigungs- und Rotationssensor erfolgt dabei für alle Anwendungen ohne Autorisierung.

Beim Kompass stellt die Zugriffsautorisierung eine besondere Situation dar. Der Zugriff auf den Kompass über das CoreMotion-Framework erfolgt ohne Autorisierung. Über das CoreLocation-Framework muss der Nutzer den Zugriff jedoch einmal autorisieren. An dieser Stelle wird klar, dass die Autorisierung durch den Nutzer nicht einheitlich durchgesetzt wird, sondern historisch gewachsen ist.

Um Zugriffe auf den Beschleunigungssensor zu erfassen, werden folgende API-Funktionen gehookt:

- `UIAccelerometer: - (void) setDelegate:`
- `CMMotionManager: - (void) startAccelerometerUpdatesToQueue: withHandler:`
- `CMMotionManager: - (void) startAccelerometerUpdates`

Für den Rotationssensor werden folgende API-Funktionen gehookt:

- `CMMotionManager: - (void) startGyroUpdatesToQueue:withHandler:`
- `CMMotionManager: - (void) startGyroUpdates`

Für den Kompass werden folgende API-Funktionen gehookt:

- `CLLocationManager: - (void) startUpdatingHeading`
- `CMMotionManager: - (void) startMagnetometerUpdatesToQueue: withHandler:`
- `CMMotionManager: - (void) startMagnetometerUpdates`

Über die DeviceMotion-APIs sind kalibrierte Daten von Beschleunigungssensor, Rotationssensor und Kompass verfügbar, daher werden auch diese Funktionen gehookt:

- `CMMotionManager: - (void)startDeviceMotionUpdatesUsingReferenceFrame: toQueue:withHandler`
- `CMMotionManager: - (void)startDeviceMotionUpdatesToQueue:withHandler:`
- `CMMotionManager: - (void)startDeviceMotionUpdatesUsingReferenceFrame:`
- `CMMotionManager: - (void)startDeviceMotionUpdates`

5.5 Sicherheitsmaßnahmen in Anwendungen

Neben dem Zugriff auf Daten, Sensoren und Aktoren durch Anwendungen soll auch der Umgang der Anwendungen mit Dateien analysiert werden. Dazu ist es wichtig herauszufinden, ob Anwendungen Dateien auf dem Gerät verschlüsselt ablegen. Des Weiteren wird erfasst, ob Anwendungen die Passwortverwaltung benutzen und ob und wie Daten über das Netzwerk übertragen werden.

5.5.1 Dateiverschlüsselung

iOS bietet seit Version 4.0 Unterstützung für das verschlüsselte Speichern von Dateien auf dem Gerät. Die Details dazu sind in Kapitel 2.5.5 erläutert.

Um zu erfassen, ob Anwendungen die Dateiverschlüsselung verwenden, müssen `open()`- und `fopen()`-Aufrufe gehookt werden. Bei jedem Aufruf wird dann die aktuell gesetzte Dateiverschlüsselung der gerade zu öffnenden Datei abgefragt und gesichert.

In iOS ist es über das Foundation.framwork außerdem möglich, bei einer Datei erst nach dem Öffnen Verschlüsselungsattribute zu setzen. Die Datei wird dann nach dem Schließen vom Betriebssystem entsprechend der gesetzten Attribute abgespeichert. Daher müssen auch Funktionen von NSFileManager und NSData gehookt werden.

Insgesamt sind daher folgende API-Funktionen zu beobachten:

- `open()`
- `fopen()`
- `NSFileManager: - (BOOL) setAttributes:ofItemAtPath:error:`
- `NSData: - (BOOL) writeToURL:options:error:`
- `NSData: - (BOOL) writeToFile:options:error:`

Wenn eine Anwendung mindestens eine Datei mit den Verschlüsselungsmodi `Complete` oder `CompleteUnlessOpen` öffnet oder in eine solche schreibt, wird in der aggregierten Analyse aufgeführt, dass diese Anwendung die Dateiverschlüsselung nutzt.

Die Abfrage des gesetzten Verschlüsselungsattributs nach einem `open()`-Aufruf mittels `NSFileManager: - (NSDictionary *)attributesOfItemAtPath:error:` löst ihrerseits intern einen `open()`-Aufruf aus. Wenn dieser wie jeder andere `open()`-Aufruf behandelt werden würde, würde sich eine Rekursion ohne Abbruchbedingung ergeben. Daher muss während der Verarbeitung des ersten `open()`-Aufrufs das Hooking temporär unterbunden werden. Dies wird im `AAOpenTweak` mittels einer Sperre gelöst, die für jeden Thread gesetzt werden kann. Dieses Verfahren ist in „snoop-it“ [37] auf gleiche Art und Weise realisiert.

Diese Sperre wird in Objective-C als Kategorie für die Klasse `NSThread` realisiert. Damit erweitert die Sperre die API von `NSThread` um folgende Methoden:

- `NSThread: - (BOOL)isLocked;`
- `NSThread: - (void)setLocked:withValue:`

Die Realisierung mittels einer Kategorie bietet den Vorteil, dass kein weiteres Objekt für die Sperre benötigt wird, sondern die Sperre mit `[NSThread currentThread]` verwendet werden kann. Innerhalb der Kategorie können mehrere Sperren gesetzt werden. Die Sperren selbst werden dann mittels „Associated Objects“ [11, 37] realisiert.

5.5.2 Transportverschlüsselung

Neben dem Abspeichern von Dateien ist auch die Transportverschlüsselung von Netzwerkverkehr zu erfassen. Apple iOS bietet dazu seit Version 2.0 die C-Socket-APIs und in Objective-C die HTTP-APIs im Foundation-Framework an. Da bei der Socketprogrammierung die Verschlüsselung von der Anwendung selbst realisiert wird, kann sie nicht automatisch erfasst werden. Die NSURLConnection-APIs zur HTTP-Kommunikation hingegen bieten Unterstützung für SSL, die als Kriterium erfasst werden.

Dazu werden folgende API-Funktionen gehookt [37]:

- `NSURLConnection: + (NSURLConnection *) connectionWithRequest: delegate:`
- `NSURlConnection: + (NSData *) sendSynchronousRequest: returningResponse:error:`
- `NSURLConnection: - (id) initWithRequest:delegate: startImmediately:`
- `NSURLConnection: - (id) initWithRequest:delegate:`

Bei der Verwendung von SSL mit HTTPS werden von NSURLConnection nur Verbindungen zu Hosts aufgebaut, bei denen das Server-Zertifikat auch überprüft werden kann. Dazu muss es von einer Zertifizierungsinstanz ausgestellt worden sein, die iOS bekannt ist. Selbstsignierte Zertifikate werden nicht akzeptiert und die Verbindung wird nicht aufgebaut. Eine Anwendung kann jedoch die Prüfung von Server-Zertifikaten durch das Überladen von Methoden des NSURLConnectionDelegate außer Kraft setzen.

5.5.3 Passwortverwaltung

Anwendungen können zum Abspeichern von Passwörtern und anderen sensiblen Daten den sogenannten Schlüsselbund verwenden, der seit Version 2.0 zur Verfügung steht.

Dabei werden die Daten nicht im Ordner der Anwendung, sondern in einer zentralen Datenbank gesichert. Dort hat jeder bei Apple registrierte Hersteller einen eigenen Bereich, auf den nur seine Anwendungen zugreifen können.

Seit iOS Version 4.0 kann für jedes gespeicherte Datum im Schlüsselbund das zugehörige Zugriffsattribut gesetzt werden. Mit diesen kann festgelegt werden, ob ein Datum im Schlüsselbund für die Anwendung immer, nur nach dem ersten Eingeben des Entsperrcodes oder nur bei entsperrtem Gerät zugänglich ist. Diese Zugriffsstufen entsprechen den Dateiverschlüsselungsstufen, die im Kapitel 2.5.5 erläutert sind.

Zur Erfassung der Passwortverwaltung werden folgende API-Funktionen gehookt [37]:

- `SecItemAdd()`
- `SecItemCopyMatching()`
- `SecItemUpdate()`
- `SecItemDelete()`

5.6 Netzwerkverkehr

Die dynamische Analyse bietet die Chance, Netzwerkverkehr von Anwendungen leicht zu erfassen. Da die Smartphone-Anwendungen vielfach Inhalte nachladen oder auf andere Art Netzwerkverkehr erzeugen, ist es lohnenswert, diesen genauer zu untersuchen.

5.6.1 HTTP-Requests

Für einen Überblick über den erzeugten Netzwerkverkehr wird die Objective-C HTTP-API gehookt und die Requests mit verwendeter HTTP Methode (GET, POST, PUT, HEAD) und URL registriert. Anhand dieser Informationen lässt sich analysieren, mit welchen URLs sich eine Anwendung verbindet. Darüber lassen sich aggregierte Analysen erstellen, welche Domains über alle Anwendungen hinweg am häufigsten verwendet werden.

Die Inhalte der HTTP-Requests werden bei dieser allgemeinen Analyse nicht erfasst. Für diese wird auf die vollständige Sammlung des Netzwerkverkehrs im nächsten Abschnitt verwiesen.

5.6.2 Vollständiger Netzwerkverkehr

Unter iOS steht seit Version 2.0 die BSD-Socket-API zur Verfügung, welche es Anwendungen ermöglicht, eigene Anwendungsprotokolle unabhängig von HTTP zu realisieren.

So implementieren beispielsweise Instant-Messaging-Anwendungen das für ihren Bereich übliche XMPP.

Es existieren mehrere Ansätze, den Netzwerkverkehr einer Anwendung zu sichern. Einerseits kann über Hooking von Netzwerkfunktionen der Netzwerkverkehr aufgezeichnet werden. Da es viele Netzwerkfunktionen gibt, ist hier besondere Vorsicht geboten, um nicht Daten gar nicht oder mehrfach auf unterschiedlichen Schichten im System aufzuzeichnen.

Daher wird in der Analyse für diese Arbeit der Netzwerkverkehr an der Netzwerkschnittstelle im Betriebssystem aufgezeichnet. So wird der vollständige, tatsächliche Netzwerkverkehr erfasst, der während der Laufzeit einer Anwendung entsteht. Bei dieser Erfassungsmethode ist es daher wichtig, dass immer nur eine Anwendung gleichzeitig ausgeführt wird, da sonst eine Vermischung von Netzwerkverkehr mehrerer Anwendungen entsteht. Dies ist insofern problematisch, da sich eine solche Vermischung bei einer späteren Analyse nicht mehr erkennen ließe.

Eine weitere Schwierigkeit entsteht, wenn Netzwerkdaten an der Betriebssystemschnittstelle bereits mit SSL verschlüsselt sind. Da dies eine Analyse erschwert beziehungsweise unmöglich macht, wird ein Ansatz gewählt, der einem „Man-in-the-Middle“-Angriff gleicht. Der Netzwerkverkehr wird über einen Proxy gesendet. Dieser Proxy bedient die HTTPS-Verbindungen und stellt eine weitere Verbindung zum Zielserver selbst her und leitet die Daten weiter. Eine im Proxy hinterlegte Zertifizierungsinstanz erstellt für jede SSL-Verbindung automatisch ein zur URL passendes Zertifikat. Diese Zertifizierungsinstanz ist auch auf dem Smartphone als vertrauenswürdig eingestuft, sodass das Gerät auch die SSL-Verbindungen vom Proxy als vertrauenswürdig ansieht. Mit Hilfe dieses Ansatzes ist es möglich, den aufgezeichneten Netzwerkverkehr später mit Hilfe des privaten Schlüssels der Zertifizierungsinstanz zu entschlüsseln. So kann auch SSL-verschlüsselter HTTP-Netzwerkverkehr analysiert werden. Das Vorgehen ist in Abbildung 5.2 zusammenfassend dargestellt.

Daten, die mit eigenen Verschlüsselungsverfahren verschlüsselt werden, welches nicht auf HTTP basiert, werden nicht über den Proxy geleitet. Sie können somit zwar an der Netzwerkschnittstelle erfasst, aber nicht entschlüsselt werden.

Aufgrund der Verwendung des HTTP-Proxys sind für HTTP-Netzwerkverkehr die beteiligten IP-Adressen und Ports nicht feststellbar, da der HTTP-Verkehr auf IP-Ebene ausschließlich an den Proxy geleitet wird und von dort erst zum eigentlich Ziel. Für HTTP-Verbindungen lassen sich aus dem Netzwerkverkehr trotzdem die beteiligten Hostnamen aus den HTTP-Headern herauslesen.

Aufgrund von Sicherheitsmaßnahmen seitens Apple ist eine Verbindung zum App Store über den Proxy nicht möglich. Außerdem werden Verbindungen ebenfalls vom Proxy ausgenommen, die für die Automatisierung notwendig sind. Daher werden folgende URLs und Netze nicht über den Proxy geleitet:

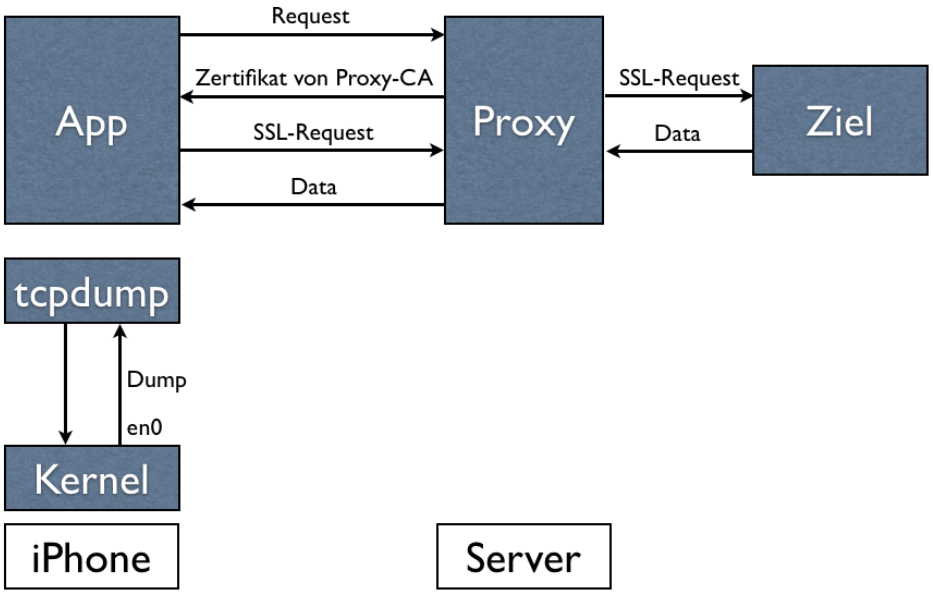

Abbildung 5.2: Aufzeichnung des Netzwerkverkehrs mit Proxy zur automatischen SSL-Zertifikatserstellung

- *.itunes.apple.com
- *.mzstatic.com
- securemetrics.apple.com
- faui1-160.informatik.uni-erlangen.de
- appanalysis.myfau.de
- 127.0.0.0/8

Zur Realisierung dieses Verfahrens wird auf dem Proxy die Software „mitmproxy“ [23] verwendet. Auf dem Gerät läuft ein eigener Dienst mit root-Rechten, der mit Hilfe der Software „tcpdump“ [36] den Netzwerkverkehr an der WLAN-Netzwerkschnittstelle des Betriebssystems aufzeichnet.

5.7 Aktoren

Der Vollständigkeit halber werden auch die Aktoren in die Analyse mit aufgenommen. Das iPhone 4 besitzt einen Lautsprecher, einem LED-Kamera-Blitz und einen Vibrationsmotor.

5.7.1 Lautsprecher

Der Lautsprecher kann seit iOS Version 2.0 über die CoreAudio-Frameworks und seit iOS Version 2.2 über das AVfoundation-Framework verwendet werden. Das iPhone 4 besitzt die Möglichkeit, dass der Nutzer die Stummschaltung aktiviert. Über das AVfoundation-Framework lässt sich der Lautsprecher jedoch auch dann nutzen, wenn der Nutzer die Stummschaltung aktiviert.

Für den Lautsprecher werden folgende APIs gehookt:

- `AudioComponentInstanceNew()`
- `AVAudioPlayer: - (id) initWithContentsOfURL:error:`
- `AVAudioPlayer: - (id) initWithData:error:`

5.7.2 LED-Kamera-Blitz

Seit dem iPhone 4 ist das Smartphone mit einer LED ausgestattet, die neben der Kamera angebracht ist. Diese kann sowohl als Blitz bei Fotoaufnahmen als auch als dauerhafte Beleuchtung bei Videoaufnahmen verwendet werden. Anwendungen mit Taschenlampenfunktion verwenden ebenfalls diese LED.

Für Anwendungen ist die LED als Blitz seit iOS Version 4.0 und als dauerhafte Beleuchtung seit iOS Version 5.0 über das AVfoundation-Framework verfügbar. Dabei sind die API-Funktionen der Klasse „AVCaptureDevice“ zugeordnet, die auch die Foto-, Video- und Tonaufnahmen erlaubt. Anwendungen können die LED wie Mikrofon und Kamera ohne Autorisierung durch den Nutzer verwenden.

Um die Verwendung der LED zu registrieren werden folgende API-Funktionen gehookt:

- `AVCaptureDevice: - (void) setTorchMode:`
- `AVCaptureDevice: - (BOOL) setTorchModeOnWithLevel:error:`
- `AVCaptureDevice: - (void) setFlashMode:`

5.7.3 Vibrationsmotor

Das iPhone besitzt einen Vibrationsmotor für die Realisierung eines Vibrationsalarms.

Der Vibrationsmotor ist für Anwendungen seit iOS Version 2.0 über eine C-API im AudioToolbox-Framework innerhalb der CoreAudio-Frameworks verfügbar. Er kann von beliebigen Anwendungen für zwei Sekunden eingeschaltet werden, ohne dass der Nutzer dies zuvor autorisieren muss.

Um die Verwendung des Vibrationsmotors zu erfassen, werden folgende API-Funktionen gehookt und auf den Parameterwert `kSystemSoundID_Vibrate` geprüft.

- `AudioServicesPlayAlertSound()`
- `AudioServicesPlaySystemSound()`

Zusammenfassung

Anhand der in diesem Kapitel vorgestellten Kriterien lassen sich Anwendungen bezüglich der verwendeten personenbezogenen und identifizierenden Daten analysieren. Des Weiteren werden Werbenetzwerke und Nutzungsstatistiken in Anwendungen protokolliert. Außerdem wird die Verwendung von Sensoren und Aktoren aufgezeichnet. Darüber hinaus werden IT-Sicherheitsaspekte wie die verwendete Datei- und Transportverschlüsselung sowie die Verwendung der Passwortverwaltung erfasst. Abschließend wird der vollständige Netzwerkverkehr aufgezeichnet, sodass er mit weiteren Werkzeugen analysiert werden kann.

Alle vorgestellten Kriterien erfassen lediglich automatisiert die Aktivitäten von Anwendungen, erlauben darüber hinaus aber keine automatische Bewertung der Anwendung.

6 Test des Analyseverfahrens und Analyseergebnisse

Mit einem Benchmark wird das in dieser Arbeit entwickelte Analyseverfahren auf generelle Geschwindigkeitseinbußen für Anwendungen getestet. Die in Kapitel 5 vorgestellten Analysekriterien werden dann jeweils auf zwei Refererenz-Anwendungen angewendet. Es folgt eine Analyse der 300 am meisten heruntergeladenen kostenlosen Anwendungen im Apple App Store. Schließlich werden die Analyseergebnisse für je 50 Anwendungen aus fünf Kategorien miteinander verglichen.

6.1 Test des Analyseverfahrens

6.1.1 Performance-Benchmark

Beim Einsatz bisheriger dynamischer Analyseverfahren sind generelle Performance-Engpässe für alle Anwendungen aufgetreten. Das Verfahren von Szydlowski [51], welches einen Debugger zur Untersuchung der ausgeführten Methoden verwendet, erwies sich für die Analyse einer großen Zahl von Anwendungen laut den Autoren als nicht ausreichend performant genug.

Das in dieser Arbeit vorgestellte Verfahren wird anhand des Benchmarks „Geekbench“ in Version 2.4.3 [45] auf allgemeine Performance-Auswirkungen getestet. Dies ist wichtig, da mit der Analyse eine große Zahl beliebiger Anwendungen untersucht werden soll. Im Gegensatz zu Analysen mit virtuellen Maschinen soll die dynamische Analyse auch für CPU- oder GPU-intensive Anwendungen wie Spiele tauglich sein.

Der Benchmark „Geekbench“ setzt sich aus vier Abschnitten zusammen [44]. Jeder Abschnitt wird einzeln getestet und am Ende errechnet Geekbench eine gewichtete Gesamtpunktzahl. Die Gesamtpunktzahl 1000 ist dabei auf die Performance eines Power Mac G5 aus dem Jahr 2003 normiert.

Der erste Abschnitt testet die Performance für Ganzzahlberechnungen und der zweite Abschnitt die für Gleitkommaberechnungen. Der dritte Abschnitt testet die Performance des Arbeitsspeichers über sequenzielles Lesen und Schreiben sowie die Allokation von

Speicherbereichen. Der vierte Abschnitt testet mit Hilfe des STREAM-Benchmarks [42] die Speicherbandbreite des Arbeitsspeichers.

Für den Benchmark werden drei Konfigurationen miteinander verglichen: das Gerät im Auslieferungszustand, das Gerät mit Jailbreak und das Gerät mit aktivem Analysetweak. Für alle Konfigurationen wird folgendes Vorgehen gewählt, um die Vergleichbarkeit der Werte sicherzustellen: Das Gerät wird gestartet, anschließend 5 Minuten abgewartet, um sicherzustellen, dass der Bootvorgang vollständig abgeschlossen ist. Erst dann wird der Benchmark ausgeführt.

Abschnitte	Auslieferungszustand	Mit Jailbreak	Analysetweak
Ganzzahl	297	296	296
Gleitkomma	372	365	371
Speicher	632	632	633
Datenstrom	306	305	306
gew. Gesamtpunktzahl	391	388	390

Tabelle 6.1: Benchmark des Analyseverfahrens

Anhand der gewichteten Gesamtpunktzahl des Geekbench-Benchmarks (siehe Tabelle 6.1) ist ersichtlich, dass das Analyseverfahren die allgemeine Rechenleistung nicht beeinträchtigt, welche für Anwendungen zur Verfügung steht. Die Schwankungen zwischen den drei untersuchten Konfigurationen liegen im Rahmen der Messgenauigkeit des Benchmarks. Auch beim Vergleich der Werte für die einzelnen Abschnitte des Benchmarks sind nur Schwankungen innerhalb der Messgenauigkeit auszumachen.

6.1.2 Beispielanalyse der Referenzanwendung SpyPhone

Die Anwendung „SpyPhone“ wurde von Seriot [50] veröffentlicht, um zu demonstrieren, auf welche Daten eine Anwendung unter iOS Version 3.1.2 ohne Autorisierung zugreifen konnte.

Die Daten sind dabei nach ihrer Art gruppiert: Email-Konten, WLAN, Telefon, Ort, Fotos, Adressbuch und Tastatur-Speicher. Abbildung 6.1 zeigt den Überblick über die Anwendung „SpyPhone“.

Die Email-Konten werden mit Kontotyp, Server und Nutzername angezeigt. Das Email-Passwort ist nicht enthalten, da es nur für die Email-Anwendung über die Passwortverwaltung zugänglich ist. Im Bereich WLAN werden die WLAN-Accesspoints aufgelistet, mit denen das Gerät bereits verbunden war. Dabei enthalten ist der letzte Zeitpunkt der Verbindung. In der Kategorie Telefon wird die ICCID, der UDID und die Anruf-Historie angezeigt. Darüber hinaus werden der Name des Mobilfunkanbieters, der „Mobile

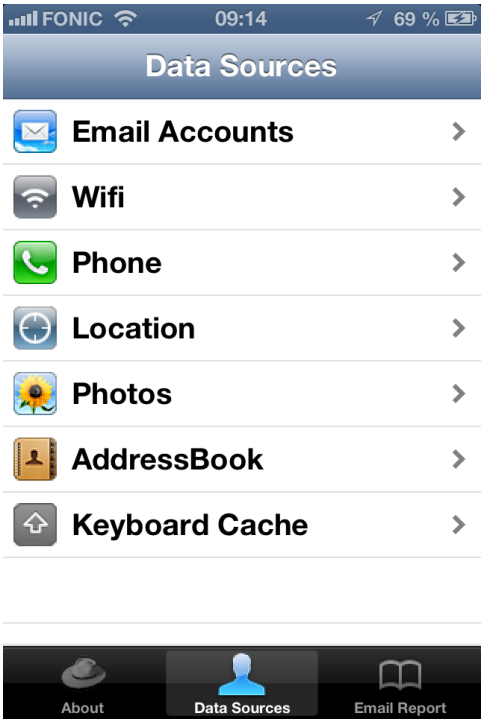

Abbildung 6.1: Datenquellen der Testanwendung „SpyPhone“, eigene Darstellung

Country Code“ und der „Mobile Network Code“ gezeigt, welche über das CoreTelefony-Framework verfügbar sind.

Der Quelltext von „SpyPhone“ ist vollständig verfügbar [49].

Analysekriterium	Ergebnis
Adressbuch	true
UDID	true
CommCenter	true
Anruf-Historie	true

Tabelle 6.2: Ergebnisse der Analysekriterien für die Anwendung SpyPhone

Bei Auswertung der Analysekriterien (siehe Tabelle 6.2) zeigt sich, dass die erwarteten Kriterien zutreffen. SpyPhone versucht, die Fotos direkt aus dem Ordner `/var/mobile/Media/DCIM/` zu lesen. Dieser Zugriff ist bei iOS Version 6 nicht möglich. Da SpyPhone somit nicht auf die Fotos zugreifen konnte, wird in den Analysekriterien kein Zugriff auf die Fotos erfasst.

In der im Anhang A.1 aufgeführten Liste der geöffneten Dateien von „SpyPhone“ ist ersichtlich, dass die Anwendung zu Beginn Grafiken lädt, die zur Anzeige notwendig sind. Beginnend mit `id 23766` werden die Einstellungsdateien geöffnet, aus denen „SpyPhone“ die Email-Konten, WLAN-Informationen und weitere Daten ausliest. Auffallend ist, dass die SQLite-Datenbankdatei nicht enthalten ist. Diese wird innerhalb von `sqlite3_open()` nicht über einen `open()` oder `fopen()`-Aufruf geöffnet.

SpyPhone versucht nicht das Mikrofon oder andere Sensoren zu verwenden, auch wenn der Name dies vermuten lassen könnte. Daher ist keines der Sensor-Kriterien positiv.

6.1.3 Test durch Anwendung PrivateData

Manche Analysekriterien lassen sich nicht mit Hilfe von „SpyPhone“ oder dem iOS-Simulator testen, wie in Kapitel 2.5.9 beschrieben. Daher wurde zu Testzwecken eine Anwendung programmiert, die auf dem Smartphone ausgeführt wird.

Diese Anwendung konzentriert sich auf folgende Daten und Funktionen:

- Dateiverschlüsselung
- `AVCaptureDevice` des AudioFoundation-Frameworks
- Datenbankdateien für Anruf-Historie, Adressbuch, Kalender und Fotos
- Sensoren wie Mikrofon, Beschleunigungssensor, Rotationssensor und Kompass
- Aktoren wie der Vibrationsmotor und die LED

Die Anwendung greift auf die Informationen zu, die von den Analysekriterien abgedeckt werden. Dabei werden auch Daten abgefragt, für die der Nutzer seine einmalige Autorisierung erteilen muss. Nach dem Zugriff werden die Daten lediglich auf der Standardausgabe ausgegeben. Es findet keine Aufbereitung der Daten wie bei „SpyPhone“ statt.

6.2 Analyseergebnisse

Im folgenden Kapitel werden die 300 am häufigsten heruntergeladenen kostenlosen Anwendungen nach den Kriterien aus Kapitel 5 untersucht. Des Weiteren werden die fünf zahlenmäßig stärksten Kategorien „Education“, „Entertainment“, „Lifestyle“, „Books“, „Business“ untereinander hinsichtlich der Kriterien verglichen. Dabei werden die je 50 am häufigsten heruntergeladenen Anwendungen berücksichtigt.

6.2.1 Top 300 der kostenlosen Anwendungen

Zur Analyse der insgesamt 300 am häufigsten heruntergeladenen Anwendungen wird folgendes Szenario zu Grunde gelegt: Ein Anwender installiert eine neue Anwendung auf seinem Smartphone und probiert diese aus. Bei iOS wird, anders als bei Android, keine Autorisierung für Daten, Sensoren und Aktoren für die Installation vorausgesetzt. Da diese erst bei der Benutzung eingeholt wird, kann ein Anwender vor der Installation nicht feststellen, welche Daten, Sensoren und Aktoren von einer Anwendung verwendet werden.

Aus diesem Szenario resultiert folgende Fragestellung: Welche Daten, Sensoren und Aktoren verwendet eine Anwendung beim ersten Start innerhalb der ersten 30 Sekunden?

Zur Untersuchung dieses Szenarios werden die 300 am häufigsten heruntergeladenen Anwendungen aus dem deutschen App Store vom Stichtag 6.5.2013 verwendet. Von diesen Anwendungen können 297 analysiert werden. Drei Anwendungen stürzen reproduzierbar direkt beim Start mit folgender Fehlermeldung ab: „Terminating app due to uncaught exception NSInternalInconsistencyException, reason: No UIWindow Found, an application should have atleast one Window object“

Die vollständige Liste der 297 analysierten Anwendungen findet sich im Anhang A.3.

Zur Analyse werden folgende Aktionen nacheinander ausgeführt:

1. Anwendung installieren
2. Anwendung starten
3. 15 Sekunden warten
4. Home-Button betätigen
5. 5 Sekunden warten
6. Anwendung wieder öffnen
7. 10 Sekunden warten
8. Analyseergebnisse vom Tweak an die Datenbank übermitteln
9. Anwendung beenden
10. Anwendung deinstallieren

Falls die Anwendung die Autorisierung für den Zugriff auf Adressbuch, Kalender oder andere zustimmungspflichtige Daten stellt, so wird die Autorisierung erteilt.

Die Ergebnisse bezüglich der identifizierenden Daten finden sich in Tabelle 6.3. Dabei werden die identifizierenden Daten wie MAC-Adresse und ASID bei mehr als der Hälfte der Anwendungen nach dem Start ausgelesen. Der ASID wird bei 55,9 % der Anwendungen, die MAC-Adresse bei 56,2 % der Anwendungen verwendet. Der UDID wird immer

noch in 44,8 % der Anwendungen ausgelesen, obwohl dieser mit Erscheinen von iOS Version 5 als veraltet markiert wurde.

identifizierende Daten	Ergebnis
UDID	44,8 %
ASID	55,9 %
VID	29,3 %
MAC-Adresse	56,2 %
ICCID & Telefonnummer	0 %

Tabelle 6.3: Zugriff auf identifizierende Daten für die 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Auf personenbezogene Daten wird von einer geringeren Zahl von Anwendungen zugegriffen (siehe Tabelle 6.4). 19,5 % der Anwendungen greifen bereits beim ersten Start auf den aktuellen Ort zu. Auf Adressbuch, Kalender und Fotos werden nur von 0,3 % bis 2,0 % der Anwendungen zugegriffen.

personenbezogene Daten	Ergebnis
Adressbuch	2,0 %
Kalender	0,7 %
Fotos	0,3 %
Anruf-Historie	0,0 %
Aktueller Ort	19,5 %
LocationMonitoring	0,0 %
Geofencing	0,0 %
Twitter	0,7 %
Facebook	0,3 %

Tabelle 6.4: Zugriff auf personenbezogene Daten für die 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Bezüglich der Sensoren (Tabelle 6.5) haben 23,9 % der Anwendungen Zugriff auf das Mikrofon. Dabei ist anzumerken, dass alle Zugriffe mit CoreAudio-APIs realisiert sind. Die CoreAudio-API ist eine API, bei der ein Duplex-Gerät sowohl Mikrofon als auch Lautsprecher darstellt. Das Mikrofon wird in diesem Gerät als Eingangskanal und der Lautsprecher als Ausgangskanal definiert. Da ein späteres Auslesen des Kanals mit Hilfe des API-Hookings nicht mehr feststellbar ist, wird die Initialisierung dieses Duplex-Geräts bereits als Zugriff auf Mikrofon und Lautsprecher definiert. Aus diesem Grund ist der Anteil der Anwendungen, die auf das Mikrofon zugreifen, vergleichsweise hoch.

Bezüglich der Lage- und Rotationssensoren ergibt sich folgendes Bild: 19,2 % der Anwendungen verwenden den Beschleunigungssensor, nur 4,7 % den Kompass und 2,7 % den Rotationssensor.

Sensoren	Ergebnis
Mikrofon	23,9 %
Kamera	3,4 %
Beschleunigungssensor	19,2 %
Rotationssensor	2,7 %
Kompass	4,7 %

Tabelle 6.5: Verwendung von Sensoren durch die 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Wie in Tabelle 6.6 ersichtlich ist, werden die Aktoren LED und Vibrationsmotor nicht innerhalb der ersten 30 Sekunden nach dem Start verwendet.

Aktoren	Ergebnis
LED	0 %
Vibrationsmotor	0 %

Tabelle 6.6: Verwendung von Aktoren durch die 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

27,9 % der Anwendungen verwenden die Passwortverwaltung und 2,0 % der Anwendungen benutzen die Dateiverschlüsselung (siehe Tabelle 6.7). Die Diskrepanz zwischen Dateiverschlüsselung und Passwortverwaltung ist bemerkenswert, da beides Funktionen zur Datenhaltung in Anwendungen sind, die üblicherweise beim Start initialisiert wird.

Sicherheitsmaßnahmen	Ergebnis
Dateiverschlüsselung	2,0 %
Passwortverwaltung	27,9 %

Tabelle 6.7: Verwendete Sicherheitsmaßnahmen der 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Bei den Werbenetzwerken und Statistikbibliotheken (Tabelle 6.8) ist innerhalb der analysierten Anwendungen Flurry mit 36,0 % am weitesten verbreitet. Flurry ist gleichzeitig Anbieter von Nutzungsstatistiken und Werbung. Die Werbenetzwerke Chartboost, Tapjoy und Google Analytics werden mit jeweils rund 10 % seltener verwendet. Alle übrigen Anbieter sind beim Start jeweils in unter 3 % der Anwendungen vertreten.

Werbenetzwerk/Nutzungsstatistik	Ergebnis
Flurry	36,0 %
Chartboost	10,1 %
Tapjoy	10,1 %
Google Analytics	9,4 %
Google AdMob	7,1 %
InMobi	2,7 %
MobileAppTracker	2,7 %
Testflight	1,7 %
AdColony	1,3 %
Mixpanel	1,0 %
Apsalar	0,7 %
Jumptap	0,7 %
Localytics	0,7 %
MobAge	0 %
KISSMetrics	0 %
Medialets	0 %
AppsFire	0 %
Distimo	0 %
Smaato	0 %
PinchMedia	0 %

Tabelle 6.8: Verwendete Werbenetzwerke und Nutzungsstatistiken der 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Im nächsten Abschnitt wird der Netzwerkverkehr der untersuchten Anwendungen näher betrachtet. Innerhalb der 300 am häufigsten heruntergeladenen Anwendungen verwenden 82,5 % innerhalb der ersten 30 Sekunden nach dem Start die NSURLConnection-API (siehe Tabelle 6.9). Nur 17,5 % erzeugen keinen HTTP/S-Netzwerkverkehr.

Netzwerkverkehr	Ergebnis
HTTP/S über NSURLConnection	82,5 %
keine Verwendung von NSURLConnection	17,5 %
Gesamtzahl der Verbindungen nach Art	**Ergebnis**
HTTP GET	2644
HTTP HEAD	4
HTTP POST	584
HTTP PUT	1
HTTPS GET	800
HTTPS POST	595
HTTPS PUT	2
HTTPS DELETE	2

Tabelle 6.9: Netzwerkverkehr der 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Bei Betrachtung der Zahl der getätigten Verbindungen ist auffällig, dass beim Start nicht nur mittels „GET“ Ressourcen nachgeladen werden, sondern ein Großteil der auftretenden Verbindungen mittels „POST“ Daten an Server von Werbenetzwerken, Nutzungsstatistiken und Anwendungsherstellern senden. Von den insgesamt 1179 POST-Verbindungen, welche die 247 Anwendungen erzeugen, sind 50,5 % per SSL verschlüsselt.

Domain	HTTP/S	Methode	Anzahl
http://www.photoappicons.info	HTTP	GET	240
http://data.flurry.com	HTTP	POST	225
http://p4.focus.de	HTTP	GET	173
https://graph.facebook.com	HTTPS	GET	169
http://kh.google.com	HTTP	GET	159
https://www.swoodoo.com	HTTPS	POST	139
http://tvs3.cellular.de.s3.amazonaws.com:80	HTTP	GET	132
https://ws.tapjoyads.com	HTTPS	GET	116
http://ads.mopub.com	HTTP	GET	82
https://www.chartboost.com	HTTPS	POST	81
http://www.googleadservices.com	HTTP	GET	70
http://iconmakercont.herewetest.com	HTTP	GET	59
http://static.watchever.de	HTTP	GET	59
http://immde-mobile.aeriagames.com	HTTP	POST	55
http://stproject.herewetest.com	HTTP	GET	51
http://wwatchmobtrkde.112.2o7.net	HTTP	GET	50
https://gdata.youtube.com	HTTPS	GET	48
http://i.ytimg.com	HTTP	GET	48
https://de.ioam.de	HTTPS	POST	42
http://appmessages3.herewetest.com	HTTP	POST	36
http://d3-wallpaper-wallhd10000.ticktockapps.com	HTTP	GET	36
https://api.crittercism.com	HTTPS	POST	35
http://mw1.google.com	HTTP	GET	32
http://d3v1lb83psg9di.cloudfront.net	HTTP	GET	31
http://media.shpock.com	HTTP	GET	30

Tabelle 6.10: Anzahl der Verbindungen pro Domain im HTTP/S-Netzwerkverkehr der 300 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 6.5.2013

Die an den Verbindungen beteiligten Domains können einen Hinweis darauf geben, zu welchem Zweck die Anwendungen auf das Netzwerk zugreifen. In Tabelle 6.10 sind die 25 häufigsten Domains mit der verwendeten HTTP-Methode aufgelistet. Darin ist erkennbar, dass die Übermittlung der Nutzungsstatistiken an http://data.flurry.com unverschlüsselt abläuft. Die Übermittlung an https://www.chartboost.com wird mit SSL-Verschlüsselung durchgeführt. Des Weiteren ist erkennbar, dass eine geringe Zahl von Domains an einem Großteil der Verbindungen beteiligt ist.

6.2.2 Vergleich zwischen Kategorien

Der Vergleich zwischen unterschiedlichen Kategorien im App Store soll folgende Fragen beantworten:

1. Verhalten sich Anwendungen aus verschiedenen Kategorien beim Start unterschiedlich hinsichtlich des Zugriffs auf personenbezogene Daten?
2. Ist es bei einer Kategorie wahrscheinlicher als bei einer anderen, dass Anwendungen auf identifizierende Daten zugreifen?
3. Verwenden Anwendungen aus einer Kategorie mehr Sensoren als Anwendungen aus einer anderen Kategorie?
4. Welche Unterschiede gibt es bei der Verwendung von Schutzmechanismen zwischen den Kategorien?

Für diese Fragen werden die 50 am häufigsten heruntergeladenen kostenlosen Anwendungen aus den Kategorien Education, Entertainment, Lifestyle, Books und Business aus dem deutschen App Store analysiert. Die Kategorien werden im Folgenden ausschließlich mit ihren englischen Namen benannt. Die vollständigen Listen der analysierten Anwendungen finden sich in Anhang A.4.

Zur Analyse werden analog zum Vorgehen aus Kapitel 6.2.1 folgende Aktionen ausgeführt:

1. Anwendung installieren
2. Anwendung starten
3. 15 Sekunden warten
4. Home-Button betätigen
5. 5 Sekunden warten
6. Anwendung wieder öffnen
7. 10 Sekunden warten
8. Analyseergebnisse vom Tweak an die Datenbank übermitteln
9. Anwendung beenden
10. Anwendung deinstallieren

Beim Zugriff auf personenbezogene Daten (siehe Tabelle 6.11) zeigt sich folgendes Bild: 22 % der Anwendungen aus der Kategorie Lifestyle greifen beim Start auf den aktuellen Ort zu, während keine der 50 Anwendungen aus der Kategorie Books, 4 % aus der Kategorie Education und 2 % aus der Kategorie Entertainment dies tun. Innerhalb der

verglichenen Kategorien greifen nur Anwendungen aus der Kategorie Business auf das Adressbuch zu.

Kategorie	Adressbuch	Kalender	Fotos	Aktueller Ort
Education	0 %	0 %	0 %	4 %
Entertainment	0 %	0 %	0 %	2 %
Lifestyle	0 %	2 %	2 %	22 %
Books	0 %	0 %	0 %	0 %
Business	6 %	2 %	0 %	12 %

Tabelle 6.11: Zugriff auf personenbezogene Daten für die Kategorien Education, Entertainment, Lifestyle, Books, Business für die 50 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 3.5.2013

Der Zugriff auf identifizierende Daten weist große Unterschiede zwischen den untersuchten Kategorien auf (siehe Tabelle 6.12). So greifen 50 % der Anwendungen aus den Kategorien Entertainment und Business auf die MAC-Adresse zu, während es in den übrigen Kategorien nur 22 % beziehungsweise 26 % sind. Das gleiche Bild zeigt sich beim ASID, auf den 58 % der Anwendungen aus der Kategorie Entertainment zugreifen, während in der Kategorie Business lediglich 26 % dieses identifizierende Datum auslesen.

Kategorie	UDID	ASID	VID	MAC-Adresse
Education	42 %	44 %	28 %	22 %
Entertainment	34 %	58 %	24 %	50 %
Lifestyle	40 %	44 %	24 %	50 %
Books	34 %	32 %	10 %	26 %
Business	22 %	26 %	12 %	26 %

Tabelle 6.12: Zugriff auf identifizierende Daten für die Kategorien Education, Entertainment, Lifestyle, Books, Business für die 50 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 3.5.2013

Der am häufigsten verwendete Sensor über alle Kategorien ist der Beschleunigungssensor (siehe Tabelle 6.13). Am häufigsten wird er in der Kategorie Books in 26 % der Anwendungen beim Start verwendet, am seltensten in der Kategorie Business in noch 8 %. Das Mikrofon wird im einstelligen Prozentbereich über alle Kategorien hinweg verwendet. Der Rotationssensor und der Kompass werden lediglich in der Kategorie Entertainment bei 4 % der Anwendungen innerhalb der ersten 30 Sekunden ausgelesen.

Kategorie	Mikrofon	Beschleunigung	Rotation	Kompass
Education	2 %	20 %	0 %	0 %
Entertainment	6 %	22 %	4 %	4 %
Lifestyle	2 %	12 %	0 %	0 %
Books	6 %	26 %	0 %	0 %
Business	4 %	8 %	0 %	0 %

Tabelle 6.13: Zugriff auf Sensoren für die Kategorien Education, Entertainment, Lifestyle, Books, Business für die 50 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 3.5.2013

Innerhalb der untersuchten kostenlosen Anwendungen wird der Nutzungsstatistik- und Werbeanbieter Flurry am häufigsten verwendet. Er wird bei 30 % der Anwendungen aus der Kategorie Lifestyle verwendet. In der Kategorie Books wird Flurry nur in 12 % der Anwendungen genutzt.

Innerhalb der ersten 30 Sekunden zeigen bereits 18 % der Anwendungen aus der Kategorie Education Werbung mit Hilfe des Werbenetzwerks AdMob. Innerhalb der Kategorie Business sind es lediglich 6 % der Anwendungen.

Kategorie	Flurry	Google Analytics	AdMob	Tapjoy
Education	22 %	12 %	18 %	0 %
Entertainment	28 %	12 %	12 %	4 %
Lifestyle	30 %	16 %	6 %	12 %
Books	12 %	4 %	8 %	0 %
Business	16 %	4 %	6 %	10 %

Tabelle 6.14: Verwendung von Werbenetzwerken und Nutzungsstatistiken für die Kategorien Education, Entertainment, Lifestyle, Books, Business für die 50 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 3.5.2013

Die von iOS bereits seit Version 4 (Erscheinungsjahr 2010) angebotene Dateiverschlüsselung wird lediglich innerhalb der Kategorie Business von 6 % der untersuchten Anwendungen verwendet. Innerhalb der Kategorien Lifestyle und Books verwenden nur 2 % der Anwendungen die Dateiverschlüsselung. Die Anwendungen der Kategorien Education und Entertainment benutzen die Dateiverschlüsselung beim Start gar nicht. Da die Datenhaltung einer Anwendung beim Start initialisiert wird, erscheint es wahrscheinlich, dass der Anteil der Anwendungen bei Analyse längerer Laufzeiten nicht signifikant ansteigt.

Die Passwortverwaltung wird innerhalb der Kategorien unterschiedlich häufig verwendet. Nur 8 % der Anwendungen aus der Kategorie Education nutzen sie, während 30 % der Anwendungen aus der Kategorie Lifestyle und 28 % der Anwendungen aus der Kategorie Business die Passwortverwaltung verwenden.

Kategorie	Dateiverschlüsselung	Passwortverwaltung
Education	0 %	8 %
Entertainment	0 %	22 %
Lifestyle	2 %	30 %
Books	2 %	18 %
Business	6 %	28 %

Tabelle 6.15: Verwendung von Sicherheitsmaßnahmen für die Kategorien Education, Entertainment, Lifestyle, Books, Business für die 50 am häufigsten heruntergeladen kostenlosen Anwendungen zum Stichtag 3.5.2013

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Ziel der Arbeit ist es, ein dynamisches und performantes Analyseverfahren für Smartphone-Anwendungen zu entwickeln und zu dokumentieren. Innerhalb dieses Analyseverfahrens werden Zugriffe auf personenbezogene Daten wie das Adressbuch, identifizierende Daten wie die MAC-Adresse, die Verwendung von Sensoren und Aktoren und der Netzwerkverkehr jeder analysierten Anwendung erfasst. Des Weiteren wird die Verwendung von Werbenetzwerken und Nutzungsstatistik-Dienstleistern durch Anwendungen erfasst. Schließlich wird auch die Verwendung von Sicherheitsmaßnahmen wie Dateiverschlüsselung und Passwortverwaltung innerhalb von Anwendungen gegen Angriffe von außen erfasst. Die Analysekriterien beschränken sich dabei auf Datentypen, die bereits bei Auslieferungszustand im Smartphone vorgesehen sind.

In der Arbeit wird analysiert und dokumentiert, auf welche Daten und Sensoren eine Anwendung unter welchen Voraussetzungen zugreifen kann. Ein Zugriff auf personenbezogene Daten ist nach einmaliger Autorisierung durch den Benutzer möglich. Bei identifizierenden Daten, Sensoren und Aktoren und Netzwerkzugriff ist eine Verwendung durch die Anwendung immer möglich.

Die vorliegende Arbeit zeigt ein auf API-Hooking basierendes Analyseverfahren. Dieses ist als MobileSubstrate-Tweak für iPhones mit Jailbreak programmiert. Die Analyseergebnisse für jede Anwendung werden über eine Web-Schnittstelle in einer Datenbank außerhalb des Geräts gespeichert.

Innerhalb dieses Analyse-Tweaks werden Zugriffe auf das Adressbuch, den Kalender, Fotos, Videos und den aktuellen Ort erfasst. Bei letzterem werden auch Techniken wie das sogenannten „Geofencing“ und „Location Monitoring“ berücksichtigt. Daneben wird das Auslesen von UDID, WLAN-MAC-Adresse, ASID und VID protokolliert. Ein Zugriff auf das Mikrofon wie auch auf Beschleunigungssensor, Rotationssensor und Kompass werden ebenso erfasst. Die Verwendung von Aktoren wie Lautsprecher, LED und Vibrationsmotor wird protokolliert. Bezüglich Profilbildung wird die Verwendung von 21 Anbietern von Werbung und Nutzungsstatistiken erfasst. Der Netzwerkverkehr der Anwendungen wird sowohl auf Anwendungsebene als auch auf Betriebssystemebene aufgezeichnet. Um zu untersuchen, ob Anwendungen Daten verschlüsselt speichern, wird die Verwendung der Dateiverschlüsselung und der Passwortverwaltung erfasst.

Das Analyseverfahren wird anhand von SpyPhone und einer eigens entwickelten Anwendung getestet. Dabei werden erwartungsgemäß die Zugriffe auf die jeweils verwendeten Daten und Sensoren erfasst. Anhand von Benchmarks wird gezeigt, dass das dynamische Analyseverfahren die Rechenleistung nicht beeinträchtigt, welche den Anwendungen zur Verfügung steht.

Bei der Analyse der 300 am häufigsten heruntergeladenen Anwendungen zeigen sich folgende Ergebnisse: 19,5 % der Anwendungen nutzen bereits beim Start den aktuellen Ort. Zugriffe auf weitere personenbezogene Daten wie Adressbuch und Kalender sind im einstelligen Prozentbereich. Die identifizierenden Daten werden hingegen von einer großen Zahl an Anwendungen bereits beim Start ausgelesen: 56,2 % der Anwendungen lesen die MAC-Adresse aus, 55,0 % den ASID, 44,8 % den UDID und 29,3 % den VID. Von den Sensoren wird vor allem der Beschleunigungssensor schon beim Start der Anwendungen verwendet: 19,2 % der Anwendungen nutzen diesen in den ersten 30 Sekunden. Der Kompass wird lediglich von 4,7 % der Anwendungen, der Rotationssensor nur von 2,7 % verwendet. Die Aktoren LED und Vibrationsmotor werden bei keiner der analysierten Anwendung innerhalb der ersten 30 Sekunden genutzt.

Bei den Anbietern für Werbung und Nutzungsstatistiken zeigt sich innerhalb der 300 Anwendungen eine Konzentration auf wenige Anbieter: Flurry wird von 36 %, Chartboost, Tapjoy und Google Analytics von je rund 10 % und Google AdMob von 7,1 % der Anwendungen genutzt. Insgesamt zeigt dies auch, dass vor allem Nutzungsstatistiken direkt beim Start der Anwendung bereits verwendet werden. Werbung wird innerhalb dieser Zeit nur von einem geringeren Anteil der Anwendungen genutzt. 82,5 % der Anwendungen erzeugen bereits in den ersten 30 Sekunden HTTP/S-Netzwerkverkehr. Bei diesem Netzwerkverkehr handelt es sich um zwei Aktionen, wie die Analyse der beteiligten Domains ergibt. Einerseits laden Anwendungen Ressourcen nach, die nicht in der Anwendung selbst vorgehalten werden. Andererseits werden die Nutzungsstatistiken übertragen. data.flurry.com ist der zweithäufigste Domainname innerhalb dieser Netzwerkverbindungen.

Bei den Sicherheitsmaßnahmen wird die Passwortverwaltung von 27,9 % der Anwendungen genutzt. Lediglich 2,0 % der untersuchten Anwendungen nutzen die seit Juni 2010 zur Verfügung stehende Dateiverschlüsselung.

Die zweite Analyse betrachtet je die 50 am häufigsten heruntergeladenen Anwendungen aus fünf Kategorien. Dabei werden die Kategorien mit den meisten Anwendungen, ausgenommen der Spiele, miteinander verglichen. Der Vergleich zeigt bei den personenbezogenen Daten große Unterschiede zwischen den Kategorien. Aus der Kategorie Lifestyle greifen 22 % der Anwendungen auf den aktuellen Ort zu, während in der Kategorie Books keine Anwendung darauf zugreift. Auf die identifizierenden Daten wird ebenfalls mit unterschiedlicher Häufigkeit zugegriffen: Innerhalb der Kategorien Entertainment und Lifestyle lesen 50 % die MAC-Adresse aus, während in der Kategorie Business lediglich 26 % der Anwendungen darauf zugreifen. Der Beschleunigungssensor wird ebenfalls

in den Kategorien Books, Education und Entertainment in rund 20 % der Anwendungen verwendet, in der Kategorie Business nur von 8 % der Anwendungen. Für die Werbenetzwerke soll exemplarisch Google AdMob stehen, dass in der Kategorie Education in 18 % der Anwendungen verwendet wird. Damit wird es in dieser Kategorie dreimal so häufig verwendet wie in Anwendungen der Kategorie Business, in der es nur von 6 % der Anwendungen genutzt wird.

Zusammengefasst wird in der Arbeit ein dynamisches Analyseverfahren entwickelt und eingesetzt, dass automatisch die Verwendung von personenbezogenen und identifizierenden Daten sowie von Sensoren und Aktoren erfasst. Aufgrund der Analyse während der Laufzeit der Anwendungen wird das Netzwerkverhalten der Anwendungen ebenfalls erfasst und analysiert.

7.2 Ausblick

In zukünftigen Arbeiten können die in dieser Arbeit entwickelten Analysekriterien für neue Versionen von Apple iOS erweitert werden. Mit neuen iOS-Versionen werden von Apple teilweise neue APIs für bereits bestehende personenbezogene und identifizierende Daten hinzugefügt. Des Weiteren können neue Kriterien für neu hinzugekommene Datentypen, Sensoren oder Aktoren entwickelt werden.

Außerdem ist eine Analyse von komplexeren Nutzungsszenarien für Anwendungen in Verbindung mit dem hier entwickelten Analyseverfahren möglich. Dazu ist es notwendig, auch diese Szenarien zu automatisieren, um weiterhin eine repräsentative Auswahl von Anwendungen untersuchen zu können.

Des Weiteren ist eine Erweiterung der Analyse auf mehr Anwendungen möglich. So wäre beispielsweise ein Vergleich aller Kategorien aus dem Apple App Store denkbar. Auch Anwendungen aus Cydia, dem App Store für Geräte mit Jailbreak, können mit dem vorgestellten Analyseverfahren untersucht werden.

Der während der Analyse aufgezeichnete Netzwerkverkehr kann in zukünftigen Arbeiten genauer untersucht werden, um beispielsweise die Übertragung von personenbezogenen Daten an Anwendungshersteller oder Werbenetzwerke feststellen zu können. Ebenso ist Konzeption und Implementierung von Taint Tracking zu Nachverfolgung von Daten innerhalb von Anwendungen denkbar.

A Anhang

A.1 SpyPhone: Geöffnete Dateien

Tabelle A.1: Geöffnete Dateien der Anwendung SpyPhone

id	path
23660	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/SpyPhone
23661	/Library/MobileSubstrate/DynamicLibraries/libstatusbar.plist
23662	/System/Library/Frameworks/UIKit.framework/Shared@2x ipad.artwork
23663	/System/Library/Frameworks/UIKit.framework/Shared@2x iphone.artwork
23664	/System/Library/Frameworks/UIKit.framework/Shared@2x.artwork
23665	/System/Library/Frameworks/UIKit.framework/Shared ipad.artwork
23666	/System/Library/Frameworks/UIKit.framework/Shared iphone.artwork
23667	/System/Library/Frameworks/UIKit.framework/Shared.artwork
23668	/var/mobile/Library/Preferences/com.apple.Accessibility.plist
23669	/System/Library/AccessibilityBundles/AccessibilitySettingsLoader.bundle/Info.plist
23670	/System/Library/AccessibilityBundles/AccessibilitySettingsLoader.bundle/AccessibilitySettingsLoader
23671	/System/Library/AccessibilityBundles/UIKit.axbundle/Info.plist
23672	/System/Library/AccessibilityBundles/UIKit.axbundle/UIKit
23673	/System/Library/PrivateFrameworks/UIAccessibility.framework/Info.plist
23674	/System/Library/PrivateFrameworks/UIAccessibility.framework/BlacklistedBundles.plist
23675	/System/Library/AccessibilityBundles/AddressBookUIFramework.axbundle/Info.plist
23676	/System/Library/AccessibilityBundles/AddressBookUIFramework.axbundle/AddressBookUIFramework
23677	/System/Library/AccessibilityBundles/MapKitFramework.axbundle/Info.plist
23678	/System/Library/AccessibilityBundles/MapKitFramework.axbundle/MapKitFramework
23679	/System/Library/AccessibilityBundles/QuickLook.axbundle/Info.plist
23680	/System/Library/AccessibilityBundles/QuickLook.axbundle/QuickLook
23681	/System/Library/AccessibilityBundles/iTunesStoreFramework.axbundle/Info.plist
23682	/System/Library/AccessibilityBundles/iTunesStoreFramework.axbundle/iTunesStoreFramework
23683	/System/Library/AccessibilityBundles/MediaPlayerFramework.axbundle/Info.plist
23684	/System/Library/AccessibilityBundles/MediaPlayerFramework.axbundle/MediaPlayerFramework
23685	/System/Library/AccessibilityBundles/VectorKit.axbundle/Info.plist
23686	/System/Library/AccessibilityBundles/VectorKit.axbundle/VectorKit
23687	/System/Library/AccessibilityBundles/GeoServices.axbundle/Info.plist
23688	/System/Library/AccessibilityBundles/GeoServices.axbundle/GeoServices
23689	/System/Library/AccessibilityBundles/MessageUIFramework.axbundle/Info.plist
23690	/System/Library/AccessibilityBundles/MessageUIFramework.axbundle/MessageUIFramework
23691	/System/Library/Fonts/CGFontCache@2x.plist
23692	/System/Library/Fonts/Cache/__H__HelveticaNeue.ttc
23693	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/MainWindow.nib
23694	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/report@2x iphone.png
23695	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/report_2only_@2x iphone.png
23696	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/report@2x.png
23697	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/report_2only_@2x.png
23698	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/report iphone.png
23699	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/report.png
23700	/System/Library/Fonts/Cache/__H__Helvetica.ttc
23701	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat@2x iphone.png
23702	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_2only_@2x iphone.png
23703	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat@2x.png
23704	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_2only_@2x.png
23705	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat iphone.png
23706	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat.png
23707	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_mask@2x iphone.png
23708	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_mask_2only_@2x iphone.png
23709	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_mask@2x.png
23710	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_mask_2only_@2x.png
23711	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_mask iphone.png
23712	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/white_hat_mask.png
23713	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/data@2x iphone.png
23714	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/data_2only_@2x iphone.png

Tabelle A.1: Geöffnete Dateien der Anwendung SpyPhone (Fortsetzung)

id	path
23715	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/data@2x.png
23716	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/data_2only_@2x.png
23717	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/data iphone.png
23718	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/data.png
23719	/System/Library/PrivateFrameworks/WebCore.framework/English.lproj/Localizable.strings
23720	/System/Library/PrivateFrameworks/WebCore.framework/English.lproj/Localizable.stringsdict
23721	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Sources.nib
23722	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/SPCell.nib
23723	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Email@2x iphone.png
23724	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Email_2only_@2x iphone.png
23725	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Email@2x.png
23726	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Email_2only_@2x.png
23727	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Email iphone.png
23728	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Email.png
23729	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Wifi@2x iphone.png
23730	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Wifi_2only_@2x iphone.png
23731	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Wifi@2x.png
23732	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Wifi_2only_@2x.png
23733	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Wifi iphone.png
23734	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Wifi.png
23735	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Phone@2x iphone.png
23736	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Phone_2only_@2x iphone.png
23737	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Phone@2x.png
23738	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Phone_2only_@2x.png
23739	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Phone iphone.png
23740	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Phone.png
23741	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Location@2x iphone.png
23742	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Location_2only_@2x iphone.png
23743	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Location@2x.png
23744	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Location_2only_@2x.png
23745	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Location iphone.png
23746	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Location.png
23747	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Photos@2x iphone.png
23748	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Photos_2only_@2x iphone.png
23749	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Photos@2x.png
23750	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Photos_2only_@2x.png
23751	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Photos iphone.png
23752	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Photos.png
23753	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/AddressBook@2x iphone.png
23754	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/AddressBook_2only_@2x iphone.png
23755	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/AddressBook@2x.png
23756	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/AddressBook_2only_@2x.png
23757	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/AddressBook iphone.png
23758	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/AddressBook.png
23759	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Keyboard@2x iphone.png
23760	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Keyboard_2only_@2x iphone.png
23761	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Keyboard@2x.png
23762	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Keyboard_2only_@2x.png
23763	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Keyboard iphone.png
23764	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/Keyboard.png
23765	/var/mobile/Applications/3BAADB39-0D1A-42BA-93D8-5C49318B6988/SpyPhone.app/SPSourceTVC.nib
23766	/var/mobile/Library/Preferences/com.apple.accountsettings.plist
23767	/Library/Preferences/SystemConfiguration/com.apple.wifi.plist
23768	/private/var/wireless/Library/Preferences/com.apple.commcenter.plist
23769	/var/mobile/Library/Preferences/com.apple.mobilephone.settings.plist
23770	/var/mobile/Library/Preferences/com.apple.mobilephone.plist
23771	/var/mobile/Library/Preferences/com.apple.PeoplePicker.plist
23772	/System/Library/Frameworks/AddressBook.framework/German.lproj/ABLocalizableDefaults.plist
23773	/var/mobile/Library/Preferences/com.apple.Maps.plist
23774	/var/mobile/Library/Preferences/com.apple.preferences.datetime.plist
23775	/var/mobile/Library/Preferences/com.apple.weather.plist
23776	/var/mobile/Library/Keyboard/de_DE-dynamic-text.dat

A.2 Sandbox-Profil aus iOS-Version 4.0

Action	Resource	Result
default	*	Deny
ipc-posix-sem	*	Allow
ipc-posix-shm	*	
file-ioctl	*	
mach-bootstrap	*	
mach-lookup	*	
network*	*	
network-inbound	*	
network-bind	*	
priv*	*	
priv-adjtime	*	
priv-netinet*	*	
priv-netinet-reservedport	*	
sysctl-read	*	
file-read*, file-write*	^/private/var/mobile/Library/AddressBook(/|$)	Allow
	^/private/var/ea/ea([.0-9])+(out|in)$	
	^/dev/null$	
	^/dev/dtracehelper$	
	^/dev/(ttys[0-9][0-9][0-9]|ptmx)$	
	^/dev/(ptyp[0-9a-f]|ttyp[0-9a-f])$	
	^/dev/(sha1_0|aes_0)$	
	^/dev/(urandom|random)$	
	^/dev/zero$	
	^/dev/(.)*$	Deny
file-read*	^/System/Library/Carrier Bundles/(.)*.png$	Allow
	^/private/var/mobile/Library/Carrier Bundles/(.)*/carrier.plist$	
	^/System/Library/Carrier Bundles/(.)*/carrier.plist$	
	^/private/var/mobile/Media/iTunes_Control/Artwork($|/)	
	^/private/var/mobile/Media/iTunes_Control/iTunes($|/)	
	^/private/var/mobile/Library/ConfigurationProfiles/PublicInfo($|/)	
	^/private/var/mobile/Library/Preferences/com.apple.carrier.plist	
	^/private/var/mobile/Library/Carrier Bundles/(.)*.png$	
	^/private/var/logs(/|$)	Deny
	^/private/var/mobile/Library/Carrier Bundles($|/)	
	^/System/Library/Carrier Bundles($|/)	
	^/private/var/mobile/Media/Photos/Thumbs$	Allow
	^/private/var/mobile/Library/Caches/MapTiles(/|$)	
	^/private/var/mobile/Library/Caches/com.apple.IconsCache(/|$)	
	^/private/var/mobile/Media/Photos/Thumbs/([^/])+.ithmb$)	

Action	Resource	Result
	^/private/var/mobile/Media/Photos/Videos($|/)	Deny
	^/private/var/mobile/Media/Photos/Thumbs($|/)	
	^/private/var/mobile/Media/Photos/com.apple.iPhoto.plist$	
	^/private/var/mobile/Media/com.apple.itdbprep.postprocess.lock$	Allow
	^/private/var/mobile/Media/(PhotoData|Photos|PhotoStreamsData)(/|$)	
	^/private/var/mobile/Media/com.apple.itunes.lock_sync$	
	^/private/var/mobile/Library/Preferences/com.apple.(books|commcenter|itunesstored|springboard|youtube|AppStore|MobileStore).plist	Deny
	^/private/var/mobile/Library/FairPlay(/|$)	
	^/usr/sbin/fairplayd$	
	^/private/var/mobile/Media/	
	^/private/var/mobile/Library/Caches/com.apple.keyboards(/|$)	Allow
	^/private/var/mobile/Library/Preferences(/|$)	
	^/private/var/mobile/Library/Keyboard(/|$)	
	^/private/var/mobile/Library/	Deny
	^/private/var/mobile/Applications/(([-0-9A-Z])*)($|/)	Allow
	^/private/var/mnt/	Deny
	^/private/var/tmp(/|$)	
	^/private/var/mobile/Applications/(.)*$	
file-write*	^/private/var/mobile/Media(/|$)	Deny
	/private/var/mobile/Library/Preferences/com.apple.Preferences.plist.[0-9A-Za-z][0-9A-Za-z][0-9A-Za-z][0-9A-Za-z][0-9A-Za-z][0-9A-Za-z][0-9A-Za-z]$	Allow
	^/private/var/mobile/Library/Preferences/com.apple.Preferences.plist$	
	^/private/var/mobile/Library/Keyboard/	
	/private/var/mobile/Applications/(([-0-9A-Z])*)/Library/Preferences/.GlobalPreferences.plist$	Deny
	^/private/var/mobile/Applications/(([-0-9A-Z])*)/Library/Preferences/com.apple.PeoplePicker.plist$	
	^/private/var/mobile/Applications/(([-0-9A-Z])*)/Documents/Inbox/	
	^/private/var/mobile/Applications/(([-0-9A-Z])*)/(tmp|Library|Documents)(/|$)	Allow
	^/private/var/mobile/Applications/(.)*$	Deny
file-write-unlink	^/private/var/mobile/Applications/(([-0-9A-Z])*)/Documents/Inbox/	Allow
iokit-open	AppleKeyStoreUserClient	Allow
	AppleMBXShared	
	IMGSGXShared_A0	
	IMGSGXGLContext	
	IMGSGXDevice	
	IOMobileFramebufferUserClient	
	IOSurfaceRootUserClient	

Action	Resource	Result
	IOSurfaceSendRight	
	IMGSGXShared	
	IMGSGXGLContext_A0	
	AppleJPEGDriverUserClient	
	AppleM2ScalerCSCDriverUserClient	
	AppleMBXDevice	
	AppleMBXUserClient	
network-outbound	^/private/tmp/launchd-([0-9])+.([^/])+/sock$	Deny
	^/private/var/tmp/launchd/sock$	
process-exec	^/private/var/mobile/Applications/(([-0-9A-Z])*)/([^/])*.app($|/)	Allow
signal	self	
system-socket	PF_SYSTEM, SYSPROTO_CONTROL	
	PF_SYSTEM	
	PF_ROUTE	
	*	Deny

Quelle: [24]

A.3 Liste der analysierten Top 300 Anwendungen

Tabelle A.2: Liste der 297 analysierten Anwendungen mit Versionsnummern, Stichtag 6.5.2013

position	bundleIdentifier	version
1	com.spinvector.fromcheese	1.3.0
2	com.orangenose.hard2.sd	4.0
3	com.lego.starwars.theyodachronicles	1
4	com.rovio.angrybirdsfriends	1.0.1
5	com.rsz.StickmanBaseJumper	1.5
6	com.edreams.flights	1.7.1
7	com.solarmoon.BatteryBoost	2.1
8	de.lotum.4pics1word	2.3
9	com.domingoflamingo.facefusionfull	26
10	com.superxstudios.GHP2	15
11	com.plussports.fitness	53
12	com.juicy-fruit.sweettalk	20
13	com.gameloft.IronMan3	1.0.0
14	com.socialquantum2.cityint	1.4.1262
15	com.ringtones4ios.ringtones	1.3
16	com.google.Maps	1.0.0.3977
17	com.appzcreative.PDFCreator	1.1
18	at.runtastic.gpssportapp	2.10.4
19	com.midasplayer.apps.candycrushsaga	1.10
20	com.xyrality.lordsofblood.itunes	1.0
21	com.google.ios.youtube	1.3.0.5707
22	com.ebaykleinanzeigen.ebc	1378
23	com.zynga.warofthefallen	1.0.15
24	com.outfit7.gingersbirthday	1.0
25	com.cremagames.InstantButtons	1.3.8
26	com.teamlava.fruit	1.0.1
27	com.fingersoft.benjibananas	1.8
28	com.brainfevermedia.AlienSky	3.1
29	com.burbn.instagram	190832
30	com.ihandysoft.barcode.qr.free	1.1.1
31	com.ebay.iphone	2.8.0
32	com.macphun.101PhotoFilters	133
33	de.payback.client.iphone	4.3.67
34	com.teamlava.restaurantstory	1.6.8

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
35	com.shazam.Shazam	581
36	com.skype.skype	4.6.0.306
37	com.leftover.CoinDozer	11.3
38	com.runtastic.iphone.situp.lite	1.3
39	com.yahoo.weather	650
40	de.zalando.iphone	1
41	com.intellectualflame.ledflashlight.washer	1.4.0
42	com.hm.mcom	2.0.2
43	com.liebeapps.LiebeZoo	5
44	com.spotify.client	50900010
45	com.mattnail.step-o-meter	3.0.1
46	com.ea.simpsonssocial.bv2	4.2.1
47	com.gamesforfriends.icomania	1.5
48	de.telekom.Kundencenter	2.0.1.2
49	de.mobile.iphone	2.6.1
50	com.miq.mikey	2.0
51	itube.player.free	1.55
52	com.supercell.magic	3.124
53	com.amazon.AmazonDE	2.3.2
54	de.komoot.berlinbikeapp	5.0.1
55	com.deutschebahn.navigator	2.2.10.5
56	com.facebook.Messenger	136204
57	com.kiloo.subwaysurfers	1.9.0
58	eu.nordeus.TopEleven	21
59	com.getdropbox.Dropbox	2.1.4
60	com.viber	2.3.0.3318
61	com.google.GoogleMobile	3.0.0.18174
62	com.emoji.freemium	4.4
63	com.google.Translate	1.3.1.2978
64	com.pepworks.PEPthedragon	1.2
65	com.ciegames.cartownstreets	3976
66	com.bosch.dustfighter	1.1
67	com.adictiz.spacedogplus	1.2.0
68	com.apple.podcasts	331
69	de.motain.iliga	396
70	com.ea.realracing3.bv	1.1.1
71	com.feelgreatgames.starchart	4.4

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
72	com.ninjafishstudios.makeuptouch2	321
73	de.starfinanz.Finanzstatus	200300
74	com.deutschebahn.DBTickets	11
75	com.sidheinteractive.sif.DR	1.3.1
76	de.immoscout.immoscout	4.0.3
77	lt.manodrabuziai.de	13
78	com.apple.mobileme.fmip1	294
79	com.rtlinteractive.rtlnow	2.1
80	com.booking.BookingApp	5.4.1
81	com.ivanlozano.emoticons	1.0
82	de.gettings.iphone	20010
83	com.supercell.soil	1.4.43.72
84	de.telekom.Fussball-de	2.9
85	com.tap4fun.kingsempire.deluxe	6965
86	com.FDGEntertainment.TentacleWarsiPhone	2.0
87	com.facebook.Facebook	183159
88	de.kaufda.kaufda	4.6.8
89	com.kiss2go.iphone	1.5.2
90	com.autoscout24.lab	3.6.1.001
91	com.sillens.iShape	3.8.2
92	com.sega.sonicdash	13.04.05.16.09.52
93	de.radio.	585
94	com.fingersoft.hillclimbracing	1.8.0
95	com.google.b612	7.0.2
96	com.imangi.templerun2	1.0
97	com.befunky.BeFunkyPhotoEditor	3.5.2
98	de.sky.skysportapp	4.2.1
99	com.gamesforfriends.wheelsforfriends	1.3.1
100	com.itwcalculator.weatherplusfree	2.43
101	com.yourcompany.PPClient	4.6.1
102	com.firsttouch.score	2.0
103	com.mycollage.mycollage	1.1
104	com.apple.mobileme.fmf1	231
105	com.toyopagroup.picaboo	4.0.3
106	com.appturbo.appdestages	1.2
107	com.orangenose.whatiq2.free.sd	4.0
108	com.icandyapps.appiconsfree	1.0

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
109	com.fungames.snipershooter	1.1
110	com.atebits.Tweetie2	5.6.1
111	de.barcoo.iphone	95
112	com.minomonsters.eve	1202
113	com.apple.iBooks	1523
114	com.finderly.shpock	752
115	com.bsb-muenchen.HistorischesBayern	1.1
116	de.rtlinteractive.inside	1.6
117	com.tunein.TuneInRadio	3.3
118	de.spiegel.sponfussball	1.0
119	com.kik.chat	6.3.0.34
120	com.soundcloud.TouchApp	2.5
121	com.funplus.familyfarm	1.5.5
122	com.ImaginationUnlimited.InstaframeFree	1.6.7
123	com.gamevil.doz.free	1.0.2
124	com.alk.copilot9namapviewer	9.4.0.176
125	com.nike.nikeplus-gps	13166
126	de.burgerking.BurgerKing	1400
127	com.wetter.prolite	1.5.1
128	com.limasky.doodlejumpes	1.0.1
129	de.mcdonalds.McDonaldsInfoApp	1.1.2
130	com.apple.itunesu	699
131	com.lufthansa.launcher	2.6.0
132	com.picsart.studio	1.8
133	net.fineseed.Colorful	1.3.1
134	com.naturalmotion.csr4c3	1.2.4
135	com.boyaa.texas-de	2.4.2
136	com.rovio.angrybirdsrio	1.6.2
137	com.biehlsoft.lovecards	1.4
138	com.sandoz.hexal.pollen	20130327.1
139	de.symblcrowd.logoquiz	1.8
140	com.netpowerapps.dai.fvdl	1.5
141	com.wb.Injustice.Brawler2013	1.3
142	de.webfactor.MehrTankenApp	2.8.0
143	com.adobe.Adobe-Reader	74948
144	com.rovio.angrybirdsstarwarsfree	1.1.3
145	de.pixelhouse.chefkoch-iphone	1.3.1

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
146	com.google.Gmail	2.2.0.8921
147	com.appliedvoices.happytalk	6.00
148	de.init.schilder	51
149	de.lieferheld.app	240
150	com.chillingo.dingledangle	669
151	com.magiccubegames.infectionzombies	2.3
152	com.topfreegames.bikeracefree	1.9.4
153	com.airberlin.AirBerlin	2.6.1
154	com.apalon.ringtones	1.3
155	com.expedia.booking	2589
156	com.rtlinteractive.clipfishiphone	1.8
157	qLAppz.Wallpapers4u	1.4
158	com.cardinalblue.PicCollage	3.7.27
159	com.ea.sims3deluxe.ipad.bv	3.5.0
160	com.google.chrome.ios	26.0.1410.53
161	com.navigon.NavigonSelectTmoD	2.3
162	com.vivendime.watchever-de	1.0.6
163	it.ideasolutions.amerigo	9
164	com.equinux.fernsehen	4.0.6
165	com.turner.rua2	1.0.1
166	de.avm.MyFRITZ	113
167	com.runtastic.iphone.roadbike.lite	2.0.1
168	de.cellular.TVSpielfilm	2.5
169	com.maiyo-tv.maiyo	1.0.2
170	com.holidaycheck.hc	1.3.4
171	com.groupon.grouponapp	5477
172	com.netzprofis.tipico-sports	2.3
173	com.ikea.ikeacatalogue2	3.2.2
174	com.webtechies.mp3musicdownloaderfree	5.0.1
175	com.ihandysoft.barcode.free	1.5.1
176	com.ikea.irmw	1.4.0
177	com.fluidfootball	2.2.1
178	com.eifrig.blitzerde	1.4.1
179	com.rtlinteractive.wwm	1.8
180	com.sgiggle.Tango	2.8.42941
181	com.langki.piccollage	1.4
182	com.littleinc.MessageMe	7

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
183	org.leo.org.leo.dict.01	2.0
184	com.tumblr.tumblr	106
185	de.spiegel.spon	1.10.0
186	com.adobe.PSMobile	2.7.2
187	com.outfit7.talkingginger	1.3
188	de.idealo.Idealo	3.1.0
189	com.toursprung.bikemap	1.0
190	com.cervomedia.slots.freeslot1	3.3
191	com.hrs.app	3.1.2
192	de.symblcrowd.logomania	1608
194	com.dtm.iphone-dtm	112
195	com.apple.store.Jolly	2.6
196	com.runtastic.iphone.mountainbike.lite	2.0.1
197	de.mikinimedia.carpooling	2.0.1
198	com.whatisid.bigdaylite	5.5.1
199	de.abnehm-app.caloriecounter	1.12
200	com.toprankapps.PlayTubeFree	1.7
201	com.daserste.daserste	1.4.11
202	com.ihandysoft.carpenter.level	1.62.0
203	ch.mogo.meteode	1.5
204	com.apalon.weatherlivefree	2.0
205	com.mojang.minecraftpe-demo	0.2.1.0
206	com.goodgamestudios.empirefourkingdoms	1.0.2
207	de.olympiaverlag.kicker.kickeronline	2.3
208	com.eyeem.ios	3.4.2
209	com.rovio.angrybirdsfree	1.5.1
210	goatmem.emoji	1.1
211	com.tapsarena.wordmania	1.3
212	com.lego.starwars.battleorders	1.1
213	com.aupeo.aupeo	3.0
214	com.Studio17.drawrider	4.0.3
215	de.sec.lidl	2.3.3
216	com.inditex.zara.iphone	1.5
217	com.ookla.speedtest	3.0.1
218	com.badrobot.actionmoviefx	2.5.1
219	tm.de.ios	1.1.1
220	com.kreeble.freemusicfree	1.60

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
221	com.swoodoo.iSwoodoo	25.0.1
222	de.immowelt.immoweltapp	2.6
223	com.fgol.HungrySharkEvolution	1.6.1
224	com.kupferwerk.sport1de	5.3.2
225	player.tv.dailyme.	292
226	com.badoo.Badoo	2.2.1
227	de.lotumlabs.buddypainting	14643
228	com.iphonesoft3g.roulette	1.1
229	com.weightwatchers.wwmobile.deDE	2.2.20.511
230	es.socialpoint.dragoncity	1304091939
231	com.peterweiss-apps.Slapp	2.4
232	de.triphoenix.fddb	1.1.3
234	de.web.mobilenavigator	2149
235	Quick-Drafts	1.5
236	com.gameloft.IceAge	1.0.9
237	com.igg.poker	1.3.0
238	com.runtastic.iphone.squats.lite	1.3
239	com.outfit7.talkingtom2	3.2
240	com.gamehivecorp.kicktheboss2	1.51
241	com.futurebits.instamessage.free	1.3.1
242	com.aeriamobile.immortalis-de	1.0
243	com.audible.iphone	332
244	com.facebook.PageAdminApp	1.8
245	de.zattoo.app.iphone	22
246	com.kabam.fortress	3.0.0
247	com.niksoftware.snapseedforipad	1.5.2
248	com.chillingo.cuttheropelite	2.2.2
249	com.unitedinternet.mmc.mobile.gmx.iosmailer	2149
250	com.teamlava.jewel	1.1.7
251	com.runtastic.iphone.idealweight.lite	2.1.1
252	com.halfbrick.jetpack	1.4.3
253	com.apple.Remote	599.20
254	net.skyscanner.iphone	3.0
255	com.9gag.ios.mobile	206
256	com.bootstlab.PhotoEditorFX	1.2.0
257	com.amonki.mooniz	1.7.1
258	com.vine.iphone	1.0.3

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
259	de.heinz.CamPlus	1.1
260	com.kicks-apps.republica	2.0.1023
261	com.thebinaryfamily.wissenstraining.free	1.0
262	com.jninteractive.guesstheword	1.3.1
263	de.finanzen100.waehrungen	13864
264	com.eivaagames.3DPoolGame	1.0.3
265	com.vtechnologyjsc.playtubefree	2.1
266	com.tinypiece.PSFotolr	3.0.10
267	com.imangi.templerun	1.6
268	com.halfbrick.FruitNinjaLite	1.8.4
269	com.bigblueclip.picstitch	3.7
270	de.telekom.entertainrc	1.2
271	de.eos-uptrade.fahrinfo.hvv	2.0
272	com.zynga.draw2.ios.free	1.4.12
274	de.hacon.hafas.vbb	2.1.2
275	net.wooga.pocketvillage	1.1.2
276	com.ninjafishstudios.moviestarmakeover	661
277	com.opodo.pxiphone	2.1
278	com.outfit7.talkingtom	2.1
279	de.autobild.sportbildplus	2.9.0.30.63646
280	com.holidaypirates.urlaubspiraten	1.0
281	com.click2mobile.collagegrameFree	2.0.2
282	com.magicsolver.freeappmagic.2012	2.0
283	com.6wunderkinder.wunderlistmobile	2.1.0
284	com.priotecs.MoneyControl	4.1
285	com.gmx.isms	594
286	com.melodis.soundhound.free	5.2.3
287	com.tripadvisor.LocalPicks	7.0.1
288	com.squareenix.minininjasmobile	1.0.1
289	cc.dict.dictcc	32
290	com.tophotapp.bigtruck	1.12
291	com.BLochmann.faktenfree	2.2
292	com.tweakersoft.AroundMe	3967
293	com.orangenose.hard002s.free	1.3.0
294	de.eplus.mappecc.client.ios.alditalk	1.2.19
295	com.ninjafishstudios.dentistoffice	1490
296	de.focus.iphone	12475

Tabelle A.2: Liste der 297 analysierten Anwendungen (Fortsetzung)

position	bundleIdentifier	version
297	de.gelbeseiten.GelbeSeiten	3275
298	com.hotels.HotelsNearMe	1906.2
299	com.gearsprout.chuckthebotpro	1.0.1
300	com.ticktockapps.wallhd-10000	2.6

A.4 Listen der Top 50 Anwendungen aus ausgewählten Kategorien

Tabelle A.3: Education Top 50

position	bundleIdentifier	version
1	com.mobilicos.howtomakeorigami	1.0.22
2	com.apple.itunesu	699
3	de.init.schilder	51
4	com.BLochmann.faktenfree	2.2
5	com.BLochmann.kreuzwortraetsel	1.8
6	cc.dict.dictcc	32
7	com.cerasus-media.AutoBILD	1.4
8	de.fahren-lernen.app	2.3.99
9	fahrschulcard	2.3
10	com.busuu.english.app	3.0.0
11	de.langenscheidt.iqvokabeltrainer.en	170
12	com.brainy.handytimetablelite	1.7.3
13	com.ideoapps.translateDE	7.0
14	com.mobilicos.howtomakepaperairplanes	1.0.6
15	com.babbel.babbelEnglish	3.0.1
16	la.bab.german.deendict	3.0
17	de.phase-6.free-version	1.5.1
18	com.babbel.babbelSpanish	3.0.1
19	de.ifuehrerschein.lite	3.2
20	de.digitales-schwarzes-brett.dsblight	1.4
21	com.tailmind.funforkidsfree	1.0
22	net.devstone.SchlaueSpruecheNachdenken	2.0
23	co.romesoft.toddlers.memory	1.0.1
24	com.dirac.googletranslate	1.3
25	com.busuu.spanish.app	3.0.0
26	ch.swift.iTheorieDeM	2.5.29
27	com.tocaboca.hairsalonxmas	1.0.5
28	com.sofatutor.mobile	1.0.1
29	de.freenet.pocketfahrschulelite	1.4.0
30	com.zy.shiwufree	2.0.0
31	eu.pons.trainer	101
32	com.busuu.french.app	3.0.0
33	com.opensolutionsdev.babytouchandhearlite	2.2

Tabelle A.3: Education Top 50 (Fortsetzung)

position	bundleIdentifier	version
34	com.thebarnofkinderkids.puzzlemagicfree	1.0
35	com.thebarnofkinderkids.peekabookids	1.0
36	de.degener.didi360mobile	1.8
37	com.busuu.italian.app	3.0.0
38	de.ammma.fitfuersabi	20
39	com.klett.AbiLernbox	1.0
40	com.babbel.babbelItalian	3.0.1
41	com.duolingo.DuolingoMobile	1.3
42	com.audioguidia.wegerman	6.2.1
43	com.allesklar.Lehrstellen	1.0.1
44	de.cornelsen.wisopruefer	1.0
45	com.babbel.babbelFrench	3.0.1
46	ch.basic-check.check-app	1.6
47	com.wissensportal.al.merkhilfe.v1	1.1
48	com.mobileminds.schreibHeroR	3.0.0
49	de.langenscheidt.iqvokabeltrainer.es	413
50	com.artelplus.origami	2.2

Tabelle A.4: Entertainment Top 50

position	bundleIdentifier	version
1	com.domingoflamingo.facefusionfull	26
2	com.juicy-fruit.sweettalk	20
3	com.cremagames.InstantButtons	1.3.8
4	com.outfit7.gingersbirthday	1.0
5	com.emoji.freemium	4.4
6	com.apple.podcasts	331
7	de.sky.skysportapp	4.2.1
8	com.peterweiss-apps.Slapp	2.4
9	com.rtlinteractive.rtlnow	2.1
10	com.ImaginationUnlimited.InstaframeFree	1.6.7
11	qLAppz.Wallpapers4u	1.4
12	de.cellular.TVSpielfilm	2.5
13	com.vivendime.watchever-de	1.0.6
14	com.outfit7.talkingginger	1.3
15	com.equinux.fernsehen	4.0.6
16	de.rtl2apps.koeln50667	1.10
17	de.zattoo.app.iphone	22
18	de.rtlinteractive.inside	1.6
19	com.domingoflamingo.friendmix	26
20	de.rtl2apps.berlintn	1.60
21	player.tv.dailyme.	292
22	com.outfit7.talkingtom2	3.2
23	com.outfit7.talkingtom	2.1
24	com.daserste.daserste	1.4.11
25	de.cinemaxx.CinemaxXApp	2.6
26	com.vtechnologyjsc.playtubefree	2.1
27	com.badrobot.actionmoviefx	2.5.1
28	com.9gag.ios.mobile	206
29	com.apple.Remote	599.20
30	com.rtlinteractive.clipfishiphone	1.8
31	de.telekom.entertainrc	1.2
32	de.prosiebensat1digital.myvideotv	1.6
33	com.click2mobile.collagegrameFree	2.0.2
34	com.trizcorp.playmouth	1.2
35	com.doubledipmedia.SexLife	1.6.0
36	stanwood.de.onair	8851
38	com.ea.fifaultimate.bv	1.5.37

Tabelle A.4: Entertainment Top 50 (Fortsetzung)

position	bundleIdentifier	version
39	com.piviandco.fatbooth	4.1
40	IncredibleApp.Wallpapers	3.2
41	de.sevenoneintermedia.prosiebentv	1.9.1
42	com.outfit7.talkingangela	1.1
43	com.videoplayer.videoplayerforyoutubefree	1.32
44	com.22cans.curiosity	3.0
45	com.beyondsoft.freemusicfree	2.0.2
46	de.couchfunk.TV-Programm	201304251345.1.0.5
47	de.maxdome.iosclient	4
48	com.visionmedia.popcorn	1.7.13
49	com.jollydream.camwowretro	1002
50	it.appideas.totaldownloader-free	27

Tabelle A.5: Lifestyle Top 50

position	bundleIdentifier	version
1	com.ebaykleinanzeigen.ebc	1378
2	de.payback.client.iphone	4.3.67
3	com.ebay.iphone	2.8.0
4	com.hm.mcom	2.0.2
5	de.zalando.iphone	1
6	com.amazon.AmazonDE	2.3.2
7	lt.manodrabuziai.de	13
8	com.icandyapps.appiconsfree	1.0
9	com.appturbo.appdestages	1.2
10	de.barcoo.iphone	95
11	com.finderly.shpock	752
12	de.mcdonalds.McDonaldsInfoApp	1.1.2
13	de.gettings.iphone	20010
14	com.ikea.ikeacatalogue2	3.2.2
15	com.apple.store.Jolly	2.6
16	goatmem.emoji	1.1
17	com.groupon.grouponapp	5477
18	de.idealo.Idealo	3.1.0
19	com.inditex.zara.iphone	1.5
20	de.eplus.mappecc.client.ios.alditalk	1.2.19
21	com.ikea.irmw	1.4.0
22	com.tweakersoft.AroundMe	3967
23	br.com.jera.funsounds	20120108
24	jp.spireinc.app.cocoppa	1.8.2
25	com.ticktockapps.wallhd-10000	2.6
26	de.brands4friends.b4f	1.5.4
27	de.sec.lidl	2.3.3
28	com.emoticons.freemium	2.7
29	com.ringtones.freemium	4.8
30	com.magicsolver.freeappmagic.2012	2.0
31	de.otto.shop	1.2.4
32	Co.eVolutive.eMirror	2.2
33	de.douglas.iphone	3.2
34	com.fab.version1	4.4
35	com.icandyapps.coloryourmessages	1.4
36	Stocard	34
37	de.bjoerndemming.aldisued	113

Tabelle A.5: Lifestyle Top 50 (Fortsetzung)

position	bundleIdentifier	version
38	com.amazon.BuyVIP	1.9.2
39	27W7W2CQJT.IconSkinsFREE	32
40	com.valuephone.edekaSHMSale	2.6
41	com.ticktockapps.wallhd-100002	1.2
42	com.coupies.Coupies	3.15
43	com.streetspotr.Streetspotr	4473
44	com.nespresso.nespresso	3.0
45	de.QUOKA.Kleinanzeigen	2.0.8
46	com.mango.2010	2.4.3
47	com.movisolmedia.jokesroulette	2.1
48	de.mybestbrands.iphoneapp	3.0
49	com.tiffany.ringfinder	196
50	de.dkb.cashback	2.02

Tabelle A.6: Books Top 50

position	bundleIdentifier	version
1	com.apple.iBooks	1523
2	com.audible.iphone	332
3	com.amazon.Lassen	1125908748
4	com.bravolang.dictionary.german	7
5	com.bluepantherbooks.FeuchOasenLP	1.2.1
6	de.einszumanderen.unnoetigefaktenfree	1.0
7	de.guentherbruns.fiesewitzefree	1.0
8	com.digitalreplica.germany.lustigestaschenbuch	1.0.1
9	com.goodbeans.JungleBookStoryBook	1.0.2
10	com.google.GoogleBooks	1.5.0.7215
11	com.intelligentipublishing.tipsandtricks.delite	6.1.23.0
12	de.skoobe	0.18.2
13	com.bluepantherbooks.feuchtoasen2lp	1.0.1
14	de.wonderkind.wunderwimmelbuch.zirkus	026
15	com.pointabout.7259	1291.0
16	com.bluepantherbooks.LustSchmerzLP	1.2.2
17	com.bluepantherbooks.KlavierlehrerinInApp	1.2
18	de.wonderkind.wunderwimmelbuch.stadt.free	022
19	com.heubachmedia.wildblumen	1.0.1
20	com.inkstonesoftware.audiobooksfree	1.0
21	com.inkstonesoftware.ebooksearch	1.4
22	com.zeptolab.ctrcomic	1
23	com.bluepantherbooks.WellSexhp	1.0
24	de.bruns.derbeasiwitze	1.2
25	de.guentherbruns.jokesbestode	1.0
26	de.weltbild.ebookapp	2.2.0
27	com.ebooks.ebookreader	3.05
28	com.heubachmedia.voegel	1.1
29	de.guentherbruns.liebesgedichtefree	1.0
30	eu.textunes.Reader	5.0.1
31	de.p-ad.thaliade	1.8
32	com.forecomm.FHM	3.4
33	com.disney.digicomicstest	17200
34	com.bluepantherbooks.AnwaltsHureLP	1.2.1
35	VW001Free	6.3
36	com.goodreads.Goodreads	169
37	tv.bluefire.bluefirereader	1.9.6

Tabelle A.6: Books Top 50 (Fortsetzung)

position	bundleIdentifier	version
38	de.eye-visions.glueckwunschlite	3.1.1
39	de.snippylite	72
40	SpiegelWissenstest.AllgemeinLite	1.4
41	de.einszumanderen.valentin2013free	1.1
42	com.fivemobile.wattpad	3.6.25
43	com.basitsoft.KuranOku	1.3
44	com.bluepantherbooks.anwaltshure2lp	1.2
45	de.guentherbruns.deinemutterfree	1.1
46	com.marvel.MarvelMobileComics	3.2.24055
47	com.streamprotect.abible	2.1.0
48	de.bruns.neuewitzefree	1.0
49	com.etecture.ekz.onleihe.ios.app	2.1
50	com.wayudaorerk.mangastormall	25.3

Tabelle A.7: Business Top 50

position	bundleIdentifier	version
1	com.onavo.protect	1.0.9
2	com.adobe.Adobe-Reader	74948
3	com.facebook.PageAdminApp	1.8
4	com.allesklar.Stellenmarkt	1.1.1
5	de.arbeitsagentur.jobboerse	2953
6	com.avocadohills.Scanner	1.3
7	de.musthaveit.GratisFeiertage	2.2
8	brightcloud.instacollagepro	1.1
9	com.geniussoftware.GeniusScan	3.2
10	com.savysoda.documentsFree	5.8
11	com.reference.stepstone	1.4.8
12	de.mobileeventguide.interzum2013	0.35
13	com.ihandysoft.batterybooster.free	1.0.5
14	com.iphonease.ledflashlight.triple	1.8.0
15	com.monster.mobility.iphone	1.5.3
16	com.flyman.picframepro	1.8
17	com.sam.samcardEuEnLitev1.0	2.6
18	com.yourcompany.iLohnLite	5.5
19	de.audi.konfigurator	1.2
20	com.iosfunny01.PhotoEffect	1.2
21	my.com.pragmatic.My-Apps-Lite	1.4.1
22	de.jobscout24.JobScout24	1.3.5
23	com.intsig.camcard.lite.eng	4.0.0.0.2379
24	com.microsoft.lync2013.iphone	2.0
25	de.bmwi.startapp	1.0.1
26	com.indeed.JobSearch	1.9
27	com.thestojkovic.timesheeter.free	3.6.0
28	de.kimeta.ios.kimetajobs	1.2
29	is24.Finanzplaner	3.0.2
30	com.nuance.Dictation	2.0.28
31	com.bytesquared.office2free	5.2.2
32	com.samsung.smarttvnow	2.0
33	de.fax.easyfreefax	2.1.3
34	com.gls.GLS	1.1.11
35	com.audi.fahrzeugboese	5340
36	com.cisco.anyconnect.gui	3.0.09115
37	com.imesart.AudioMemosFree	3.1.5

Tabelle A.7: Business Top 50 (Fortsetzung)

position	bundleIdentifier	version
38	com.btgs.scannerhdlite	2.5
39	com.cityguideag.berlin	2.0.4
40	de.uvex.uvexdezibel	1.0.5
41	de.immowelt.wohngemeinschaftapp	2.6
42	com.intsig.contacts2excel.lite	1.1.0.1
43	studentenwg	5.4
44	de.audi.myaudimobileassistant	2.0.1
45	com.symantec.snap	1.6.15
46	com.yuyy.ExcelContactsLite	2.5.7
47	com.philips.dictation.recorder	2.6.4
48	com.appOn.KFCBrunei	1.0.2
49	de.lab48.iTie	1.2.0
50	com.ups.m.iphone	2.2.0.11011401

Literaturverzeichnis

[1] 148Apps.biz: *Application Price Distribution*. Online: http://148apps.biz/app-store-metrics/?mpage=appprice, abgerufen am 15. April 2013.

[2] Alkalay, Avi: *iPhone Call History Database*. Online: http://avi.alkalay.net/2011/12/iphone-call-history.html, abgerufen am 16. Mai 2013.

[3] Apple Inc.: *About the iOS Technologies*. Online: http://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/iPhoneOSTechOverview.pdf, abgerufen am 16. Mai 2013.

[4] Apple Inc.: *Apple - iTunes - RSS Generator*. Online: http://itunes.apple.com/rss, abgerufen am 16. Mai 2013.

[5] Apple Inc.: *Apple App Store - Anzahl der Apps*. Online: http://de.statista.com/statistik/daten/studie/20150/, abgerufen am 16. Mai 2013.

[6] Apple Inc.: *iOS Developer Library*. Online: http://developer.apple.com/library/ios/navigation/, abgerufen am 16. Mai 2013.

[7] Apple Inc.: *The iOS Environment*. Online: http://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/iPhoneAppProgrammingGuide.pdf, abgerufen am 16. Mai 2013.

[8] Apple Inc.: *iOS Simulator User Guide*. Online: http://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/iOS_Simulator_Guide.pdf, abgerufen am 16. Mai 2013.

[9] Apple Inc.: *Object-Oriented Programming with Objective-C*. Online: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC.pdf, abgerufen am 16. Mai 2013.

[10] Apple Inc.: *Objective-C Runtime Reference*. Online: https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/ObjCRuntimeRef.pdf, abgerufen am 16. Mai 2013.

[11] Apple Inc.: *Programming with Objective-C*. Online: `http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/ProgrammingWithObjectiveC.pdf`, abgerufen am 16. Mai 2013.

[12] Appthority Inc.: *App Reputation Report - February 2013*. Online: `https://www.appthority.com/appreport.pdf`, abgerufen am 16. Mai 2013.

[13] Appthority Inc.: *App Reputation Report - Juli 2012*. Online: `https://www.appthority.com/reports/Appthority-App_Reputation_Report_July_2012.pdf`, abgerufen am 16. Mai 2013.

[14] Bassen, Nikias, Yiduo David Wang et al.: *Evasi0n Jailbreak*. Online: `http://evasi0n.com`, abgerufen am 16. Mai 2013.

[15] Beck, Kent: *Test-Driven Development: By Example*. Addison-Wesley Professional, 2003.

[16] Bellard, Fabrice: *QEMU, a fast and portable dynamic translator*. In: *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, Seiten 41–41, Berkeley, CA, USA, 2005. USENIX Association.

[17] Bitdefender: *Clueful - Privacy Monitoring App for iPhone*. Online: `http://cluefulapp.com`, abgerufen am 16. Mai 2013.

[18] Bosman, Erik, Asia Slowinska und Herbert Bos: *Minemu: the world's fastest taint tracker*. In: *Proceedings of the 14th international conference on Recent Advances in Intrusion Detection*, RAID'11, Seiten 1–20, Berlin, Heidelberg, 2011. Springer-Verlag.

[19] Chow, Jim, Ben Pfaff, Tal Garfinkel, Kevin Christopher und Mendel Rosenblum: *Understanding data lifetime via whole system simulation*. In: *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, Seiten 22–22, Berkeley, CA, USA, 2004. USENIX Association.

[20] Clause, James, Wanchun Li und Alessandro Orso: *Dytan: a generic dynamic taint analysis framework*. In: *Proceedings of the 2007 international symposium on Software testing and analysis*, ISSTA '07, Seiten 196–206, New York, NY, USA, 2007. ACM.

[21] ComScore: *Anzahl der Smartphone-Nutzer in Deutschland bis 2012*. Online: `http://de.statista.com/statistik/daten/studie/198959/`, abgerufen am 16. Mai 2013.

[22] ComScore: *Smartphone-Betriebssysteme - Marktanteile bis 2013*. Online: `http://de.statista.com/statistik/daten/studie/170408/`, abgerufen am 16. Mai 2013.

[23] Cortesi, Aldo: *mitmproxy - a SSL-capable man-in-the-middle proxy*. Online: `http://mitmproxy.org`, abgerufen am 16. Mai 2013.

[24] Dai Zovi, Dino A.: *Apple iOS 4 Security Evaluation*. Black Hat US, 2011.

[25] Davi, Lucas, Alexandra Dmitrienko, Manuel Egele, Thomas Fischer, Thorsten Holz, Ralf Hund, Stefan Nürnberger und Ahmad-Reza Sadeghi: *MoCFI: A framework to mitigate control-flow attacks on smartphones*. In: *Proceedings of the 19th Annual Symposium on Network and Distributed System Security*, 2012.

[26] Egele, Manuel, Christopher Kruegel, Engin Kirda und Giovanni Vigna: *PiOS: Detecting privacy leaks in iOS applications*. Proceedings of the 18th Annual Network and Distributed System Security Symposium, 2011.

[27] Egele, Manuel, Christopher Kruegel, Engin Kirda, Heng Yin und Dawn Song: *Dynamic spyware analysis*. In: *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, Seiten 18:1–18:14, Berkeley, CA, USA, 2007. USENIX Association.

[28] Enck, William, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel und Anmol N. Sheth: *TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones*. In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, Seiten 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[29] Engler, Justin, Seth Law, Josh Dubik und David Vo: *Automation in iOS Application Assessments*. Black Hat US, 2012.

[30] Freeman, Jay: *MobileSubstrate*. Online: `http://cydia.saurik.com/info/mobilesubstrate/`, abgerufen am 16. Mai 2013.

[31] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 6. Auflage Auflage, 2011.

[32] Gilbert, Peter, Byung-Gon Chun, Landon P Cox und Jaeyeon Jung: *Vision: automated security validation of mobile apps at app markets*. In: *Proceedings of the second international workshop on Mobile cloud computing and services*, Seiten 21–26. ACM, 2011.

[33] Hazelwood, Kim und Artur Klauser: *A dynamic binary instrumentation engine for the ARM architecture*. In: *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, CASES '06, Seiten 261–270, New York, NY, USA, 2006. ACM.

[34] Howett, Dustin: *Theos*. Online: `https://github.com/DHowett/theos/`, abgerufen am 16. Mai 2013.

[35] iPhone Development Wiki: *Mobile Substrate*. Online: `http://iphonedevwiki.net/index.php?title=MobileSubstrate`, abgerufen am 16. Mai 2013.

[36] Jacobson, Van, Craig Leres, Steven McCanne et al.: *Tcpdump*, 1989.

[37] Kurtz, Andreas und Markus Trossbach: *Snoop-it*. Online: `http://code.google.com/p/snoop-it/`, abgerufen am 16. Mai 2013.

[38] Lawton, Kevin P.: *Bochs: A Portable PC Emulator for Unix/X*. Linux J., 1996(29es), September 1996.

[39] Lookout Mobile Security: *App Genome Report*. Online: `https://www.lookout.com/resources/reports/appgenome`, abgerufen am 14. Januar 2013.

[40] Luk, Chi-Keung, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi und Kim Hazelwood: *Pin: building customized program analysis tools with dynamic instrumentation*. In: *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '05, Seiten 190–200, New York, NY, USA, 2005. ACM.

[41] Marquardt, Philip, Arunabh Verma, Henry Carter und Patrick Traynor: *(sp)iPhone: decoding vibrations from nearby keyboards using mobile phone accelerometers*. In: *Proceedings of the 18th ACM conference on Computer and communications security*, CCS '11, Seiten 551–562, New York, NY, USA, 2011. ACM.

[42] McCalpin, John D: *A survey of memory bandwidth and machine balance in current high performance computers*. IEEE TCCA Newsletter, Seiten 19–25, 1995.

[43] Petrich, Ryan: *CaptainHook*. Online: `https://github.com/rpetrich/CaptainHook`, abgerufen am 16. Mai 2013.

[44] Primate Labs Inc.: *Geekbench 2 Benchmarks*. Online: `http://www.primatelabs.com/geekbench/doc/benchmarks.html`, abgerufen am 16. Mai 2013.

[45] Primate Labs Inc.: *Geekbench 2.4.3*. Online: `http://www.primatelabs.com/geekbench/`, abgerufen am 16. Mai 2013.

[46] Qin, Feng, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou und Youfeng Wu: *LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks*. In: *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, Seiten 135–148, Washington, DC, USA, 2006. IEEE Computer Society.

[47] Rastogi, Vaibhav, Yan Chen und William Enck: *AppsPlayground: automatic security analysis of smartphone applications*. In: *Proceedings of the third ACM conference on Data and application security and privacy*, Seiten 209–220. ACM, 2013.

[48] Saxena, Prateek, R Sekar und Varun Puranik: *Efficient fine-grained binary instrumentationwith applications to taint-tracking.* In: *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization*, CGO '08, Seiten 74–83, New York, NY, USA, 2008. ACM.

[49] Seriot, Nicolas: *SpyPhone.* Online: `https://github.com/nst/spyphone/`, abgerufen am 16. Mai 2013.

[50] Seriot, Nicolas: *iPhone Privacy.* Black Hat DC, Seite 30, 2010.

[51] Szydlowski, Martin, Manuel Egele, Christopher Kruegel und Giovanni Vigna: *Challenges for dynamic analysis of iOS applications.* In: *Proceedings of the 2011 IFIP WG 11.4 international conference on Open Problems in Network Security*, iNetSec'11, Seiten 65–77, Berlin, Heidelberg, 2012. Springer-Verlag.

[52] Thampi, Arun: *Path uploads your entire iPhone address book to its servers.* Online: `http://mclov.in/2012/02/08/path-uploads-your-entire-address-book-to-their-servers.html`, abgerufen am 16. Mai 2013.

[53] the iPhone wiki: *UDID.* Online: `http://theiphonewiki.com/w/index.php?title=UDID`, abgerufen am 16. Mai 2013.

[54] Weinlein, Andreas: *Automatisierte Sicherheitsanalyse mobiler Apps.* Unveröffentlichte Masterarbeit, Universität Erlangen-Nürnberg, 2013.

[55] Werthmann, Tim, Ralf Hund, Lucas Davi, Ahmad-Reza Sadeghi und Thorsten Holz: *PSiOS: Bring Your Own Privacy & Security to iOS Devices.* In: *8th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2013)*, 2013.

[56] Zdziarski, Jonathan: *Hacking and Securing iOS Applications.* Oreilly and Associate Series. Oreilly & Associates Incorporated, 2012.

[57] Zhu, Hong, Patrick AV Hall und John HR May: *Software unit test coverage and adequacy.* ACM Computing Surveys (CSUR), 29(4):366–427, 1997.