Michael Spreitzenbarth

Mobile Hacking

Ein kompakter Einstieg ins Penetration Testing mobiler Applikationen – iOS, Android und Windows Mobile

dpunkt.verlag

Dr.-Ing. Michael Spreitzenbarth studierte Wirtschaftsinformatik an der Universität Mannheim mit Schwerpunkt in den Bereichen IT-Security und Digitale Forensik. Zwischen 2010 und 2013 arbeitete er als Doktorand an der Universität Erlangen-Nürnberg. Seine Forschungsthemen lagen in den Bereichen der forensischen Analyse von Smartphones (speziell im Bereich Android) sowie im Bereich der Detektion und automatisierten Analyse von mobilem Schadcode (Malware). Seit April 2013 arbeitet er in einem weltweit operierenden CERT, wo sein Fokus auf der Absicherung mobiler Endgeräte, Incident Handling und der Analyse verdächtiger mobiler Applikationen liegt. In seiner Freizeit arbeitet Michael Spreitzenbarth immer noch im Bereich der Forschung und Entwicklung von Malware-Analyse- und Detektionstechniken sowie digitaler Forensik. Zusätzlich hält er regelmäßig Vorträge und Schulungen zu diesen Themen in der freien Wirtschaft sowie für öffentliche Auftraggeber.

Michael Spreitzenbarth

Mobile Hacking

Ein kompakter Einstieg ins Penetration Testing mobiler Applikationen – iOS, Android und Windows Mobile

dpunkt.verlag

Michael Spreitzenbarth

Lektorat: René Schönfeldt
Lektoratsassistenz: Stefanie Weidner
Projektkoordination: Miriam Metsch
Copy-Editing: Ursula Zimpfer
Satz: Da-TeX, www.da-tex.com
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Print 978-3-86490-348-9
PDF 978-3-96088-124-7
ePub 978-3-96088-125-4
mobi 978-3-96088-126-1

1. Auflage 2017

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Vorwort

Der Marktanteil von Smartphones und Tablets wächst signifikant im Gegensatz zu herkömmlichen PCs und hält auch in immer mehr Unternehmen Einzug. Diese Geräte haben einen enormen Funktionsumfang und werden schon lange nicht mehr nur zum Telefonieren verwendet. Dies bietet jedoch nicht nur neue Möglichkeiten für den Einzelnen sowie viele Firmen, sondern birgt auch ein hohes Maß an Gefahren und Risiken in sich.

Wenn Sie sich schon immer gefragt haben, was die Applikationen auf Ihrem Smartphone so alles im Hintergrund machen und ob die verarbeiteten Daten auch wirklich sicher abgelegt und übertragen werden, dann ist dieses Buch genau das Richtige für Sie!

Im Rahmen des Buches wird Ihnen anhand von ausführlichen Beispielen gezeigt, warum es sinnvoll für Unternehmen – aber auch für Privatpersonen mit einem Bewusstsein für das Schützenswerte – ist, nicht auf die Versprechen der Hersteller mobiler Apps zu hören, sondern selbst zu prüfen, ob Applikationen den eigenen Ansprüchen und Vorgaben entsprechen. Sie erfahren, welche Open-Source-Tools es auf dem Markt gibt und wofür man sie einsetzen kann. Dabei ist immer zu bedenken, dass dies nur eine Auswahl an Tools ist und keinesfalls den Anspruch auf Vollständigkeit erheben soll. Nach der Analyse einiger Apps hat jeder Analyst seine eigene Sammlung an Werkzeugen und Vorgehensweisen, auf die er bei der täglichen Arbeit zurückgreift.

Dabei stehen unter anderem folgende Fragestellungen im Vordergrund:

- Was sind die wichtigsten Komponenten der einzelnen mobilen Betriebssysteme?
- Wie können Sie prüfen, ob Daten einer App auch bei Verlust bzw. Diebstahl eines mobilen Endgerätes weiterhin geschützt sind?
- Wie können Sie feststellen, ob Daten bei der Übertragung ausreichend geschützt sind?
- Welche Schwachstellen sind für Angreifer besonders interessant?

Dieses Buch kann man als Leser auf verschiedene Weisen angehen, die in Abbildung 1 aufgezeigt sind. Hierbei werden in den ersten beiden Kapiteln die Grundlagen geschaffen, um einem Leser, der bisher keine Erfahrungen mit mobilen Be-

triebssystemen oder dem Reversing gesammelt hat, den Einstieg zu ermöglichen. Danach gibt es immer einen »Doppelpack« an Kapiteln, der sich mit der statischen Analyse von Apps einer bestimmten Plattform befasst und sich der anschließenden Suche nach Sicherheitsproblemen widmet. Dies alles wird wieder in Kapitel 9 zusammengeführt, das die Analyse der Datenübertragung als Schwerpunkt hat, die auf allen Systemen nahezu identisch abläuft und deshalb an dieser Stelle auch übergreifend zusammengefasst ist. Den Schluss des Buches bildet ein kurzes Kapitel, das dem Leser Hinweise zu Übungen gibt, die er nun selbst in Angriff nehmen kann, und weitere Werkzeuge und vertiefenden Lesestoff vorschlägt.

Dr. Michael Spreitzenbarth

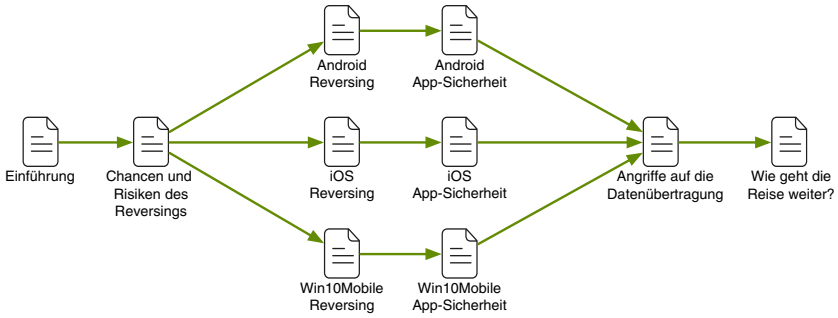

Abb. 1: Mögliche Wege für den Leser durch dieses Buch

Danksagung

Dieses Buchprojekt wäre ohne die Unterstützung anderer nur schwer umsetzbar gewesen. Aus diesem Grund möchte ich einigen dieser Personen auf diesem Wege meinen Dank aussprechen.

Zuerst möchte ich Andreas Kurtz und Siegfried Rasthofer danken, die mir sehr guten Input für dieses Buch gegeben und auf einige hilfreiche Tools und Beispiele aufmerksam gemacht haben.

Ebenso möchte ich aber auch den vielen Autoren und Entwicklern aus der Open-Source-Community danken, die zahllose Stunden in die Verbesserung und Entwicklung neuer Werkzeuge gesteckt haben. Ohne diese Arbeit wären wir heute nicht in der Lage, solche Assessments durchführen zu können.

Mein Dank geht ebenso an den dpunkt.verlag und die anonymen Reviewer sowie an Marko Rogge, die mich gerade in der Schlussphase auf wichtige Verbesserungen am Buch aufmerksam gemacht haben.

Besonderer Dank geht an meine Lebensgefährtin, die mich immer wieder motiviert und unterstützt hat, auch wenn es Zeiten gab, in denen durch dieses Buch und die tägliche Arbeit nahezu keine Freizeit mehr blieb.

Inhaltsverzeichnis

1 Einführung in mobile Betriebssysteme und deren Applikationen

Smartphones und Tablet-PCs sind aus dem täglichen Leben von Privatpersonen ebenso wenig wegzudenken wie aus dem Alltag in Unternehmen. Diese Geräte dienen längst nicht mehr nur dem Zweck der Kommunikation, wie es mit dem Telefon vor einigen Jahren noch der Fall war. Sie werden inzwischen vielmehr dazu benutzt, Zugriff auf sensible Firmen- oder Privatnetze (z. B. per VPN) zu erhalten oder teilweise sehr sensible Daten (wie z. B. Bilder und Angebote) zu verarbeiten.

Betrachtet man die Entwicklung der letzten Jahre, so sieht man, dass diese Geräte immer mehr Fähigkeiten erhalten und in immer mehr Szenarien eine Rolle spielen. Steuerung der kompletten Hauselektronik, Zugangskontrolle zu hochgesicherten Bereichen, Katastrophenfrühwarnung und der Ersatz des Notebooks in Unternehmen sind nur einige Beispiele, in denen diese Geräte und die darauf installierten Applikationen (oder kurz Apps) in Zukunft eine wichtige Rolle spielen werden.

Will man für solch kritische Szenarien ein mobiles Endgerät einsetzen, so landet man nach der initialen Betrachtung der Hardware immer wieder an dem folgenden Punkt: *Wie sicher ist eigentlich die Applikation selbst?*

Um diese Frage zu beantworten, gibt es eigentlich nur zwei mögliche Lösungsansätze: entweder Vertrauen in die Marketingfolien und -versprechen der Hersteller solcher Lösungen oder das Durchführen eigener Tests, um sicherzustellen, dass die ausgesuchte Lösung auch das leistet, was sie verspricht. Einen solchen Test nennt man Penetrationstest (oder kurz Pentest).

Betrachtet man die Schlagzeilen einschlägiger Magazine oder Webseiten (und ebenfalls die Beispiele, die in diesem Buch erwähnt werden), so sieht man sehr schnell, dass die erste Option (blindes Vertrauen in die Marketingversprechen) oft der falsche Ansatz ist, wenn man erwägt, sensible Daten mit einer Applikation zu verwalten oder zu verarbeiten. Aus diesem Grund werden im Rahmen dieses Buches Wege gezeigt, wie man selbst Applikationen überprüfen und deren Schwächen ans Tageslicht bringen kann.

Wir betrachten dazu ausführlich die Systeme Android und iOS, zeigen aber auch erste Einstiegspunkte in Windows Phone, da dieses System in vielen Einsatzszenarien eine immer wichtiger werdende Rolle spielt. Beginnen möchten wir

in diesem Kapitel mit der allgemeinen Einführung in die Systeme und den Aufbau ihrer Applikationen. Außerdem wird das Einrichten der Laborumgebung beschrieben, in der sich die später gezeigten Analysen durchführen lassen.

1.1 Android

Das Android OS, das ursprünglich für die Verwendung auf Smartphones und Tablet-PCs entwickelt wurde, findet in letzter Zeit auch immer mehr Verwendung auf Set-Top-Boxen, TVs oder als Car-Entertainment-System. Die Basis des Betriebssystems wird von der Open Handset Alliance unter der Führung von Google entwickelt und ist vollständig Open Source, lediglich die Anpassungen von Google (eigene Apps wie z. B. Google Maps und zugehörige Bibliotheken) sind nicht in ihrem Quellcode verfügbar. Die Komponenten, die das System ausmachen und auch im weiteren Verlauf des Buches von Interesse sind, werden in den folgenden Abschnitten genauer beleuchtet.

1.1.1 Die Entwicklung der Android-Plattform

Zu Beginn der Einführung in die Android-Plattform wird die bisherige Entwicklung der verschiedenen Android-Versionen aufgezeigt. Die gesamte zeitliche Entwicklung der Plattform, speziell deren Hauptversionen, ist in Tabelle 1–1 zusammen mit ihren Versionsnummern, Codenamen und Erscheinungsdaten dargestellt. Seit der Version 1.5 haben die Hauptversionen immer den Namen von populären US-Süßigkeiten. Dies hat sich erst durch die Kooperation mit Nestlé in Version 4.4 und dem Codenamen *KitKat* geändert, Google blieb zwar bei den Süßigkeiten, wechselte jedoch auf den europäischen Markt.

1.1.2 Die Architektur des Betriebssystems

Die Android-Architektur kann in vier verschiedene Schichten unterteilt werden: Basis der Architektur ist der Linux-Kernel, darüber liegt eine kombinierte Schicht aus Systembibliotheken und der eigentlichen Android-Laufzeitumgebung, dann folgt das Applikationsframework und als oberste Ebene die eigentlichen Applikationen (siehe Abbildung 1–1). Jede dieser vier Schichten stellt spezielle Interfaces und Systemressourcen für die darüberliegende Schicht zur Verfügung, sodass eine Interaktion zwischen den einzelnen Schichten ermöglicht wird.

Der Linux-Kernel

Wie auf der Abbildung 1–1 zu erkennen ist, bildet der Linux-Kernel in Version 2.6.x die Basis des Android-Systems. Dieser wurde um spezielle Module erweitert, um die Hardware der Android-Geräte zu unterstützen. Zusätzlich dazu wird der Kernel für die Verwaltung der laufenden Prozesse sowie des Speichers

Version	Codename	API	Veröffentlichung	Verwendung
1.0	Base	1	09/2008	s
1.5	Cupcake	3	04/2009	s
1.6	Donut	4	09/2009	s
2.0	Eclair	5 & 6	10/2009	s
2.1	Eclair	7	01/2010	s
2.2	Froyo	8	05/2010	s
2.3	Gingerbread	9 & 10	12/2010	s
3.0	Honeycomb	11–13	02/2011	t
4.0	Ice Cream Sandwich	14 & 15	10/2011	s,t
4.1	Jelly Bean	16	06/2012	s,t
4.2	Jelly Bean	17	11/2012	s,t
4.3	Jelly Bean	18	07/2013	s,t
4.4	KitKat	19 & 20	10/2013	s,t
5.0	Lollipop	21	10/2014	s,t
5.1	Lollipop	22	03/2015	s,t
6.0	Marshmallow	23	10/2015	s,t

Tab. 1–1: Liste der Android-OS-Versionen und deren Erscheinungsdatum seit der ersten offiziellen Veröffentlichung in 2008 (s = Smartphone, t = Tablet-PC)

verwendet und stellt einige der Sicherheitsmechanismen bereit, die in Android implementiert sind.

Prozesse mit ihrem zugehörigen Identifier (PID) sind eines der wichtigsten Konzepte des Linux-Kernels. Neben dieser PID speichert der Kernel weitere wichtige Informationen über laufende Prozesse – wie z. B. den Prozessstatus, den Thread, in dem der Prozess läuft, und Angaben darüber, welche Dateien verwendet werden (eine vollständige Liste stellt der Linux-Quellcode [10] bereit). Diese Daten werden in einer gesonderten Struktur – `task_struct` – abgelegt. Die PID ist im weiteren Zusammenhang sehr wichtig, da mit ihrer Hilfe während der dynamischen Analyse die Operationen den entsprechenden Applikationen zugeordnet werden können (mehr dazu in Abschnitt 4.2.1).

Systembibliotheken und Laufzeitumgebung

In der darüberliegenden Schicht befinden sich sie Systembibliotheken und die eigentliche Android-Laufzeitumgebung. Diese Bibliotheken sind in C bzw. C++ geschrieben und werden sowohl vom System selbst als auch von allen installierten

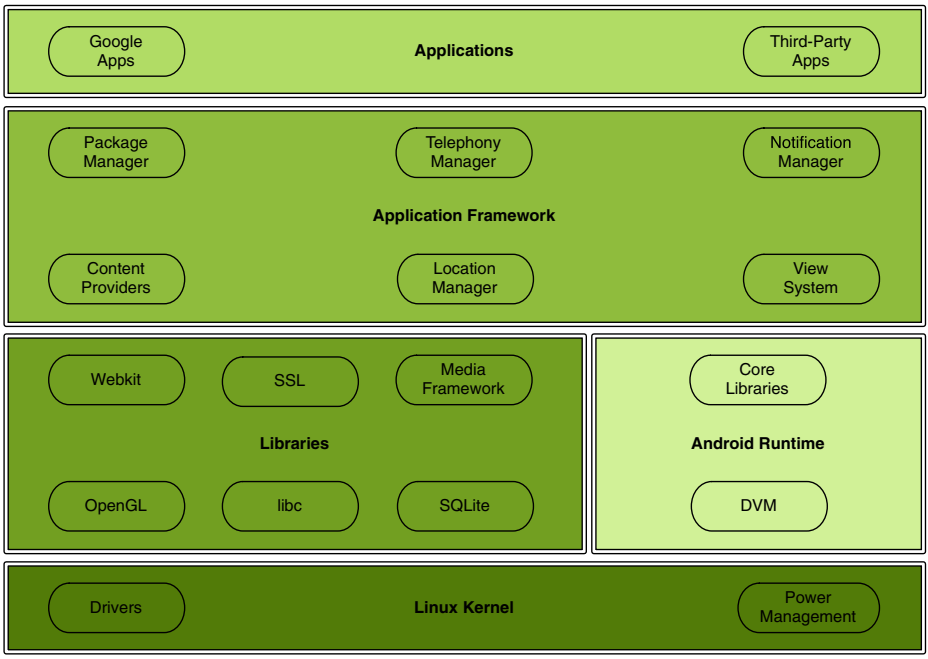

Abb. 1–1: Übersicht über die Architektur des Android-Betriebssystems

Apps verwendet. Die Android-Laufzeitumgebung enthält die *Dalvik Virtual Machine (DVM)* sowie die wichtigsten Java-Bibliotheken. Die DVM- und die Java-Bibliotheken wurden speziell an die Vorgaben eines mobilen Betriebssystems – geringer Stromverbrauch und geringe Rechenleistung – angepasst.

Applikationsframework

Die nächste Ebene ist das sogenannte Applikationsframework, das die *Application Programming Interfaces (API)* bereitstellt. Diese Ebene ist eine Art Übersetzungsschicht: Sie stellt eine hohe Anzahl an geräteabhängigen Schnittstellen zur Verfügung, die es einem Entwickler erlauben, auf alle Features des Endgerätes zuzugreifen, ohne dass er hierfür tiefere Kenntnisse der einzelnen Komponenten benötigt.

Die Applikationen selbst

Darüber liegen schließlich die eigentlichen Applikationen. Jede dieser installierten Applikationen ist zum großen Teil in Java geschrieben (mit optionalen eigenen Bibliotheken) und wird zur Laufzeit in einer eigenen DVM ausgeführt. Aktuell gibt es im offiziellen Google Play Store knapp über 1,6 Millionen dieser Applikationen (Apps) [16].

Android-Applikationen unterstützen auch die Verwendung von nativen Bibliotheken, die in C/C++ geschrieben sind. Sobald man jedoch die Applikation zur Verwendung auf einem Endgerät oder dem Emulator bauen lässt (z. B. durch *Eclipse* oder *Android Studio*), wird aus dem Java-Code ein in der DVM ausführbarer Bytecode, der in einer dex-Datei gespeichert wird. Dieser Bytecode unterscheidet sich an vielen Stellen von herkömmlichem Java-Bytecode, was sich dadurch auch auf die Analyse auswirkt. Zum Wichtigsten einer Android-Applikation gehört das *Android-Manifest*, das alle vom Nutzer abgefragten Berechtigungen sowie die zur Laufzeit nötigen *Intents*, *Listener* und *Receiver* enthält (mehr zu diesen Begriffen später in diesem Abschnitt). Das Android-Manifest wird zusammen mit dem DVM-Bytecode, den externen Bibliotheken und allen anderen Ressourcen, die für die Applikation benötigt werden, in eine Art ZIP- bzw. JAR-Paket zusammengeschnürt. Dieses Paket hat die Dateiendung `.apk` und wird vom Play Store (oder jeder anderen Installationsquelle) auf das Endgerät kopiert und dort installiert.

Im Allgemeinen besteht eine so paketierte Android-Applikation aus den folgenden Dateien und Verzeichnissen:

- **META-INF**: Verzeichnis mit folgenden Dateien:
  - **MANIFEST.MF**: das Manifest in kompilierter Form
  - **CERT.RSA**: das Zertifikat, mit dem die Applikation signiert wurde
  - **CERT.SF**: eine Liste aller Ressourcen und das SHA-1-Digest
- **lib**: Verzeichnis mit speziell für einen bestimmten Prozessor kompiliertem Code
  - **armeabi**: kompilierter ARM-Code
  - **armeabi-v7a**: kompilierter ARMv7-Code
  - **x86**: kompilierter x86-Code
  - **mips**: kompilierter MIPS-Code
- **resources.arsc**: eine Datei mit vorkompilierten Bibliotheken
- **res**: Verzeichnis mit Bibliotheken, die nicht in `resources.arsc` kompiliert wurden
- **AndroidManifest.xml**: ein weiteres Manifest mit wichtigen Metainformationen für die Applikation in codierter Form
- **classes.dex**: der kompilierte Java-Code der Applikation

Nachfolgend ein Beispiel einer realen Applikation, wie sie aus dem Play Store geladen wurde:

```
$: file DEMO.apk

DEMO.apk: Java Jar file data (zip)
```

Codebeispiel 1–1: Ausgabe des `file`-Befehls auf eine Android-Applikation zum Bestimmen des Dateityps

```
// die ersten 4 Byte sind bei jeder Android-Applikation 50 4b 03 04
$: hexdump -C -n 64 DEMO.apk

00000000 504b03040a000008 000039584f410000 |PK........9XOA..|
00000010 0000000000000000 0000250005006173 |..........%...as|
00000020 736574732f436f6e 6669677572617469 |sets/Configurati|
00000030 6f6e732f66646173 666a6b616c616664 |ons/fdasfjkalafd|
```

Codebeispiel 1–2: Anzeige der ersten 64 Byte einer Android-Applikation in Hex

```
$: unzip -l DEMO.apk

Archive:  DEMO.apk
  Length      Date    Time    Name
 --------  ---------- -----   ----
     2632  05-31-14 11:54   res/layout/activity_line.xml
     1820  05-31-14 11:54   res/layout/activity_main.xml
      392  05-31-14 11:54   res/xml/device_admin_sample.xml
     5068  05-31-14 11:54   AndroidManifest.xml
     2760  05-31-14 11:54   resources.arsc
    11566  05-31-14 11:47   res/drawable-hdpi/lineinfo_update.png
    14068  05-24-14 19:21   res/drawable-hdpi/ic_launcher.png
   718628  05-31-14 11:50   classes.dex
      687  05-31-14 11:54   META-INF/MANIFEST.MF
      740  05-31-14 11:54   META-INF/CERT.SF
     1203  05-31-14 11:54   META-INF/CERT.RSA
 --------                   -------
   759564                   11 files
```

Codebeispiel 1–3: Auflistung der Inhalte einer Android-Applikation

Bestandteile einer Android-Applikation

Wie bereits kurz erwähnt, gibt es mehrere essenzielle Bestandteile einer Android-Applikation. Neben dem gerade erwähnten Android-Manifest und den externen

Bibliotheken gibt es auch im eigentlichen Code der Applikation wichtige Bestandteile, die bei der Funktion, aber auch bei der Analyse eine wesentliche Rolle spielen. Hierzu gehören: *Activities*, *Services*, *Broadcast Receiver*, *Shared Preferences*, *Intents* und *Content Provider*. Da diese Komponenten im Folgenden sehr wichtig sind, werden sie an dieser Stelle ausführlicher beschrieben:

Android-Manifest: Jede Android-Applikation besitzt eine besondere Datei, die allgemeine Informationen über die Applikation beinhaltet, wie z. B. den Namen, die SDK-Version, die eigene Versionsnummer, angeforderte Berechtigungen sowie Hard- und Softwareanforderungen. Diese Datei heißt *Android-Manifest.xml* oder kurz *Android-Manifest*. Die codierten Informationen innerhalb dieser Datei werden während der Installation vom System ausgewertet, um so dem Nutzer die Berechtigungen anzuzeigen, die er akzeptieren muss, bevor die Applikation erfolgreich installiert werden kann. Des Weiteren sind in ihr auch sämtliche Activities enthalten, die als Einstiegspunkte in die Applikation dienen können. Die »MAIN«-Activity wird dabei als Einstiegspunkt verwendet, sobald ein Nutzer die App über den Launcher oder den Homescreen startet. Zusätzlich kann man im Android-Manifest erkennen, ob die Applikation auf externe Events wartet, um besondere Aktionen auszuführen (z. B. sobald eine SMS mit einem speziellen Text auf dem Gerät eintrifft, startet die Applikation und zeigt dem Nutzer eine Pop-up-Nachricht an). Wegen dieser Eigenschaften ist das Android-Manifest einer des besten Startpunkte für eine manuelle Analyse einer verdächtigen Applikation.

Activities: Activities stellen eine interaktive grafische Benutzeroberfläche bereit, indem sie Daten und Informationen der Applikation auf dem Display darstellen und Touch-Events des Nutzers an die Applikation zur Verarbeitung weitergeben. Jede dieser Activities muss im Android-Manifest deklariert werden, andernfalls ist eine Ausführung nicht möglich. Wechselt eine Activity in den Hintergrund, z. B. durch das Öffnen einer anderen Applikation, pausiert die Activity und alle Daten, die nicht als persistent deklariert wurden, werden gelöscht. Befindet sich eine Activity in diesem Modus, so kann das Android-System diese löschen, um Ressourcen freizugeben.

Intents: Intents sind ein spezieller Datentyp der Android-Architektur. Sie werden zur Kommunikation zwischen Komponenten einer Applikation oder zwischen Komponenten unterschiedlicher Applikationen verwendet. Jeder Intent besitzt zwei Attribute – Data und Action – und zusätzlich drei optionale Attribute – Types, Categories und Extras. Die Aktion beschreibt, was die Applikation mit den erhaltenen Daten macht. Das optionale Attribut Type ist nur nötig, falls die übermittelten Daten nicht in der *URI-Syntax (Uniform Resource Identifier)* vorliegen. In diesem Fall beschreibt der Type den MIME-Type der übermittelten Daten. Extras werden dann verwendet, wenn die zu übermittelten Daten nicht in das eigentliche Datenfeld passen. Das Attribut Category kann verwendet werden, um die Aktion genauer zu beschreiben.

Intents können in zwei Arten aufgeteilt werden: implizit und explizit. Explizite Intents haben einen spezifischen Receiver, der im Intent selbst definiert ist und meist in unterschiedlichen Komponenten einer Applikation verwendet wird. Im Gegensatz dazu werden implizite Intents an den Paketmanager des Android-Systems weitergegeben, das dann die passende Applikation sucht, die den passenden Broadcast Receiver definiert hat.

Broadcast Receivers: Android-Applikationen können eigene Activities als sogenannte Broadcast Receiver beim System registrieren. Dies hat zur Folge, dass – wenn eine andere Applikation oder das System selbst einen impliziten Intent via Broadcast-Nachricht versendet – der zugehörige Broadcast Receiver benachrichtigt wird und die damit verlinkte Activity gestartet wird.

Content Provider: Dieser Datentyp wird verwendet, um persistente Daten für andere Applikationen zugänglich zu machen. Content Provider sind der einzige Weg, Daten zwischen Applikationen auszutauschen, da es im System keinerlei Speicherbereich gibt, der von allen Applikationen gemeinsam benutzt werden kann. Applikationen verwenden SQL und relationale Datenbankschnittstellen, um die bereitgestellten Daten zu verarbeiten.

Services: Services sind Komponenten, die im Hintergrund und ohne Nutzerinterface ausgeführt werden und ebenfalls im Android-Manifest deklariert sowie dem Service Manager mitgeteilt sein müssen. Bei der Definition dieser Services kann auch festgehalten werden, wie die Applikationen bzw. die Komponenten der eigenen Applikation mit dem Service kommunizieren dürfen. Services können Dateisystemoperationen durchführen, Netzwerkverkehr überwachen oder anderweitige Informationen (wie z. B. den Standort des Gerätes) an andere installierte Applikationen weiterreichen. Verlangt eine Applikation Zugriff auf einen speziellen Service, so wird dies über den Binder IPC und den Service Manager umgesetzt. Dabei prüft der Service Manager die Zugriffsrechte und der Binder IPC agiert als direkte Kommunikationsschnittstelle zwischen Applikation und abgefragtem Service.

Shared Preferences: Diese werden von der Applikation verwendet, um kleinere Datenmengen zu speichern, die zur Funktion benötigt werden (z. B.: Spielstände). Sie liegen meist als XML in einem Verzeichnis namens `shared_prefs`. Sensible Daten sollten hier auf keinen Fall abgelegt werden, da sie an dieser Stelle anfällig für Diebstahl und Offenlegung sind.

1.2 Apple iOS

Apple iOS, das ursprünglich aus Mac OS X entstand und auch heute noch viele Gemeinsamkeiten damit hat, gibt es seit 2007 für iPhones und später auch für iPads und den AppleTV. Im Gegensatz zu Android ist es jedoch Closed Source. Die Komponenten, die das System ausmachen und auch im weiteren Verlauf des

Buches von Interesse sind, werden in den folgenden Abschnitten genauer beleuchtet.

1.2.1 Die Entwicklung der iOS-Plattform

Die Geschichte von iOS begann im Jahr 2005, als bei Apple entschieden wurde, ein Smartphone zu entwickeln, das auf dem vom Mac bekannten OS X beruhte. Die Vorstellung des ersten iPhone und damit auch von iOS war knapp 2 Jahre später im Januar 2007 auf der *MacWorld Conference and Expo*. Zuerst unterstützte das iPhone nur einige wenige Applikationen von Apple selbst und sogenannte Web-Apps, bei denen lediglich im Browser eine GUI angezeigt wird und die eigentliche Technik auf einem Server-Backend läuft. Ab 2008 folgte dann das erste SDK, womit es möglich war, auch native Applikationen als Dritthersteller zu entwickeln. Zeitgleich wurde auch der App Store eingeführt um die Applikationen an die Endkunden zu vertreiben.

Den offiziellen – und bis heute bekannten – Namen *iOS* bekam das System von Apple erst im Jahre 2010. Im selben Jahr wurden auch die Betriebssysteme von iPhones und iPads mit der Veröffentlichung von iOS 4.2.1 vereinheitlicht. Seit 2014 gibt es zusätzlich zu dem bekannten iOS auch noch *watchOS* für die Apple Watch sowie *tvOS* für den AppleTV mit einem Erscheinungsdatum ab 2015 (zuvor besaß der AppleTV ebenfalls iOS als Betriebssystem). Die gesamte zeitliche Entwicklung der Plattform, speziell deren Hauptversionen, ist in Tabelle 1–2 zusammen mit ihren Versionsnummern und Erscheinungsdaten dargestellt.

Wie man an der Tabelle sehr schön erkennen kann, versucht Apple jedes Jahr zu ihrer September-Keynote eine neue Hauptversion von iOS vorzustellen. Dieser Termin ist für alle Entwickler, aber auch Analysten, ein Datum, an dem sie sich die neuen Funktionen der Plattform, aber vor allem auch die Änderungen an bisherigen Funktionen und Sicherheitsmechanismen genau anschauen sollten. Auch viele der in diesem Buch verwendeten Werkzeuge und Techniken sind stark von der iOS-Version der verbundenen Endgeräte abhängig.

1.2.2 Die Architektur des Betriebssystems

Die iOS-Architektur kann in fünf verschiedene Schichten unterteilt werden: Basis der Architektur ist das sogenannte Core OS (Darwin), darüber liegen die Core-Services gefolgt von der Schicht, die für Grafik, Audio und Video zuständig ist. Eine Schicht darüber folgt Cocoa Touch und als oberste Ebene die eigentlichen Applikationen (siehe Abbildung 1–2). Jede dieser fünf Schichten stellt spezielle Interfaces und Systemressourcen für die darüberliegende Schicht zur Verfügung, sodass eine Interaktion zwischen den einzelnen Schichten ermöglicht wird.

Version	Veröffentlichung	Verwendung
1.0	01/2007	s
1.1	09/2007	s
2.0	07/2008	s
2.1	09/2008	s
2.2	11/2008	s
3.0	06/2009	s
3.1	09/2009	s
3.2	04/2010	s,t
4.0	06/2010	s,t,a
4.1	09/2010	s,t,a
4.2	11/2010	s,t,a
4.3	03/2011	s,t,a
5.0	10/2011	s,t,a
5.1	03/2012	s,t,a
6.0	09/2012	s,t,a
6.1	02/2013	s,t,a
7.0	09/2013	s,t,a
7.1	03/2014	s,t,a
8.0	09/2014	s,t,a
8.1	12/2014	s,t,a
8.2	03/2015	s,t,a
8.3	04/2015	s,t,a
8.4	06/2015	s,t,a
9.0	09/2015	s,t
9.1	10/2015	s,t
9.2	12/2015	s,t

Tab. 1–2: Liste der iOS-Versionen und deren Erscheinungsdatum seit der ersten offiziellen Veröffentlichung in 2007 (s = iPhone/iPod Touch, t = iPad, a = AppleTV)

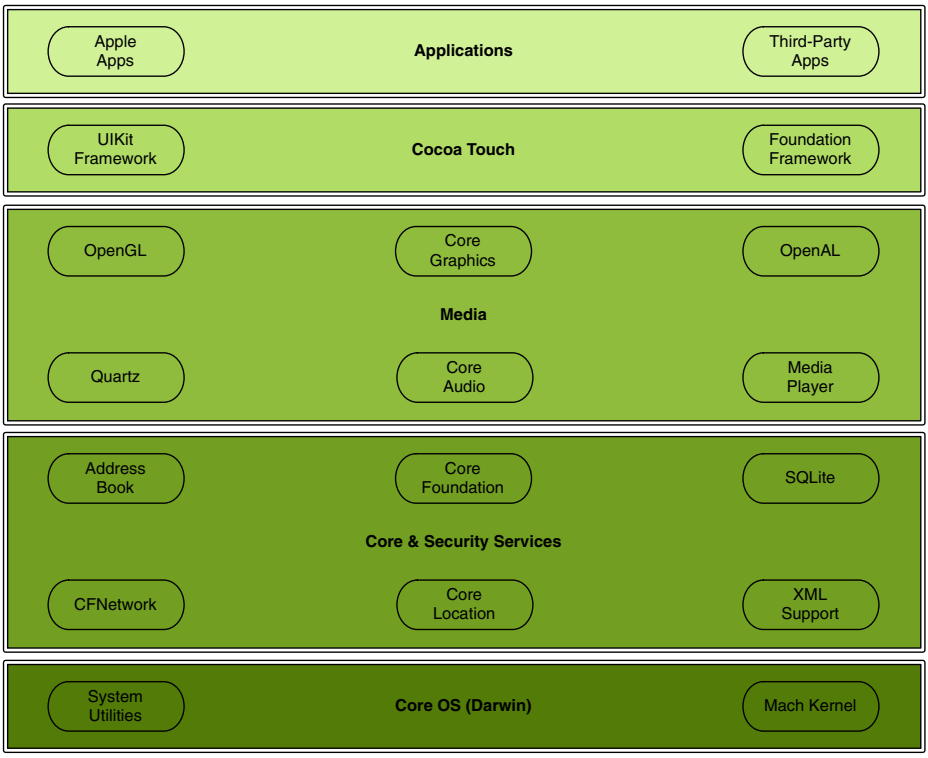

Abb. 1–2: Übersicht der Architektur des iOS-Betriebssystems

Das Core OS

Das Core OS von iOS ist auch unter dem Namen *Darwin* bekannt und ist im Wesentlichen ein vollwertiges UNIX-Betriebssystem, jedoch ohne grafische Oberfläche. Es basiert auf dem Nutzerbereich von FreeBSD und einem Mach-Kernel. Damit bietet es Benutzern und Applikationen eine Schnittstelle zur Kommunikation mit der verbauten Hardware und steuert das Speichermanagement sowie die Zugriffe auf das Dateisystem und die unterste Schicht des Netzwerkstacks. Das Besondere an Darwin ist, dass es sowohl unter Apples älteren PowerPC-Plattform lauffähig ist (und dort auch Verwendung fand) als auch unter der aktuellen Intel- und ARM-Plattform. Somit findet dieser Bereich von iOS auch auf den aktuellen mit Mac OS X betriebenen Systemen seinen Einsatz. Geräte mit iOS 9.x oder Mac OS X el Capitan setzen dabei auf die Version 15.x von Darwin.

Core- und Security-Services

Hier finden sich die Systembibliotheken von iOS, die sowohl vom System selbst als auch von allen installierten Apps verwendet werden. Hierzu gehören u. a. die

API, um mit dem iCloud-Speicherplatz zu interagieren, aber auch die nötigen Schnittstellen, um SQLite-Datenbanken zu lesen und zu schreiben oder um die Kommunikation über das Internet vorzunehmen. Des Weiteren sind hier auch die meisten Sicherheitsmechanismen verankert, wie z. B. Dataprotection (siehe dazu Abschnitt 1.2.3) und Automatic Reference Counting, auf das wir bei den Analysen von iOS-Applikationen noch zu sprechen kommen. In dieser Schicht werden auch *In-App-Käufe* und der Zugriff auf das Adressbuch sowie die *Keychain* (siehe dazu Abschnitt 1.2.3) gesteuert.

Aufrufe von Bibliotheken dieser Schicht sind während des Reversings sehr gut daran zu erkennen, dass ihre Namen mit `CF` beginnen (z. B. `CFReadStream` und `CFNetwork`).

Media

In dieser Schicht liegen alle Bibliotheken, die für die Anzeige von Grafiken jeglicher Art, das Abspielen von Videos, die Wiedergabe sowie Aufzeichnung von Audio und *AirPlay* zuständig sind. *AirPlay* ist im übertragenen Sinn die Streaming-Technologie von Apple. Sie erlaubt es, Video- und Audiodaten an über WLAN verbundene Geräte (z. B. den AppleTV) weiterzugeben (engl. streaming), und stellt diesen Dienst allen Applikationen zur Verfügung.

Cocoa Touch

Cocoa Touch ist vergleichbar mit der *Applikationsframework-Schicht* auf Android und stellt dem Entwickler einfache Schnittstellen zu den darunterliegenden Bibliotheken zur Verfügung. Es besteht aus den folgenden drei Frameworks:

- *Foundation* (relevante Basisklassen der C-basierten Programmierung, wie z. B. Strings und Arrays),
- *UIKit* (besteht aus den folgenden zwei Komponenten: AppKit [alles, was man zur Entwicklung des User-Interface benötigt, also z. B. Menüs und Buttons, aber auch Sprachanbindung] sowie Core-Data zur Erstellung von Objektgraphen und zur Ablage von Daten)

Klassen dieser Frameworks sind während des Reversings sehr gut daran zu erkennen, dass ihre Namen mit `NS` beginnen (z. B. `NSString` und `NSException`).

Die Applikation selbst

Darüber liegen schließlich die eigentlichen Applikationen. Jede dieser installierten Applikationen ist zum großen Teil in Objective-C geschrieben (mit optionalen eigenen Bibliotheken) oder neuerdings in Swift. Aktuell gibt es im offiziellen Apple App Store knapp über 1,5 Millionen dieser Applikationen (Apps) [16].