Sicherheit von Smartphone-Betriebssystemen im Vergleich

Gerhard Klostermeier, Andreas Jansche

Inhaltsverzeichnis

1 Einleitung

1.1 Inhalte und Ziele

Im Rahmen der Vorlesung "Seminar" der Hochschule Aalen wurde das Thema "Sicherheit von Smartphone-Betriebssystemen" bearbeitet. Der Schwerpunkt lag auf den Betriebssystemen an sich und nicht etwa auf den darauf laufenden, sog. "Apps".
Die Ergebnisse wurden in dieser Ausarbeitung in folgender Form dokumentiert:

- Eine Übersicht über sicherheitsrelevante Aspekte der einzelnen Betriebssysteme
- Ein Vergleich der Sicherheit bzw. der sicherheitsrelevanten Features

1.2 Die Auswahl der Betriebssysteme

Die untersuchten Betriebssysteme wurden nach der größten Verbreitung bzw. dem größten Marktanteil in den USA ausgewählt:

- Google Android - 40.1%
- Apple iOS - 26.6%
- RIM + Microsoft + Symbian - 31.2%
- Sonstige - 2.1%

Entnommen wurden die Werte einer Statistik zum Marktanteil von Smartphone Plattformen im Juni 2011, erhoben durch *comScore, Inc.*. Um im Rahmen des Umfangs der Seminararbeit zu bleiben, wurden nur die zwei verbreitetsten Plattformen (Android und iOS) betrachtet und genauer untersucht, welche auch als einzige Plattformen einen wachsenden Marktanteil haben.
[com11, vgl. Smartphone Platform Market Share]

2 Die einzelnen Betriebssysteme

2.1 Android

2.1.1 Das Android Security Program

Schon während der Entwicklung durchläuft Android inklusive seiner fest integrierten Anwendungen das *Android Security Program*. Dieses gliedert sich wie folgt:

- Design Review
  Der Entwurf jedes größeren Features wird vor Einführung auf konzeptionelle Mängel im Bezug auf Sicherheit von entsprechenden Fachleuten im Android-Entwicklerteam geprüft.

- Penetration Testing and Code Review
  Während der Entwicklung werden Penetrationstests durchgeführt und der geschriebene Quellcode wird auf Sicherheitsmängel geprüft. Dies geschieht durch:
  - das *Android Security Team*,
  - Googles *Information Security Engineering Team*
  - und unabhängige Sicherheitsberater.

- Open Source and Community Review
  Das anschließende Veröffentlichen des Quellcodes ermöglicht eine sicherheitskritische Begutachtung durch die Android Community bzw. jeden Interessierten. Über den *Android Market* können zudem Informationen über eine Anwendung von jedem direkt an die Endanwender weitergegeben werden.

- Incident Response
  Sollten trotz aller Vorkehrungen nach dem Veröffentlichen Sicherheitslücken bekannt werden, kümmert sich darum das *Android Security Team*: Wenn etwas über eine vermeindliche Sicherheitslücke bekannt wird, wird überprüft ob sie tatsächlich vorhanden ist. Wenn ja, wird die Lücke schnellstmöglich behoben. Um die Anwender vor der Gefahr zu schützen, können folgende Maßnahmen ergriffen werden:
  - Update der gesamten Android Plattform. (Sog. *over-the-air updates*)
  - Entfernen einer App aus dem *Android Market*.
  - Fernlöschung einer App von den Endgeräten.

  Kritisch bei den *over-the-air updates* ist jedoch, dass der Großteil der Android-Versionen herstellerspezifisch modifiziert ist und von diesen Herstellern gepflegt werden. So dauert es unter Umständen sehr lange bis Updates von Google zu den Herstellern gelangen und diese sie wiederum in ihren Android-Varianten einpflegen und über *over-the-air updates* veröffentlichen.

[and11a, vgl. Security Program Overview]

2.1.2 Die Architektur

Schon durch die Architektur des Android Systems sollen folgende grundlegenden Sicherheitsfeatures gewährleistet sein:

- Sicherheit auf Betriebssystemebene durch den Linuxkernel.
- Ausführung von Anwendungen in einer Sandbox.
- Sichere Interprozesskommunikation.
- Sog. *application signing*.
- Durch Anwendungen definierte und vom Benutzer gewährte Berechtigungen.

[and11a, vgl. Platform Security Architecture]

Die Abbildung 1 stellt den Aufbau der Android Architektur dar.

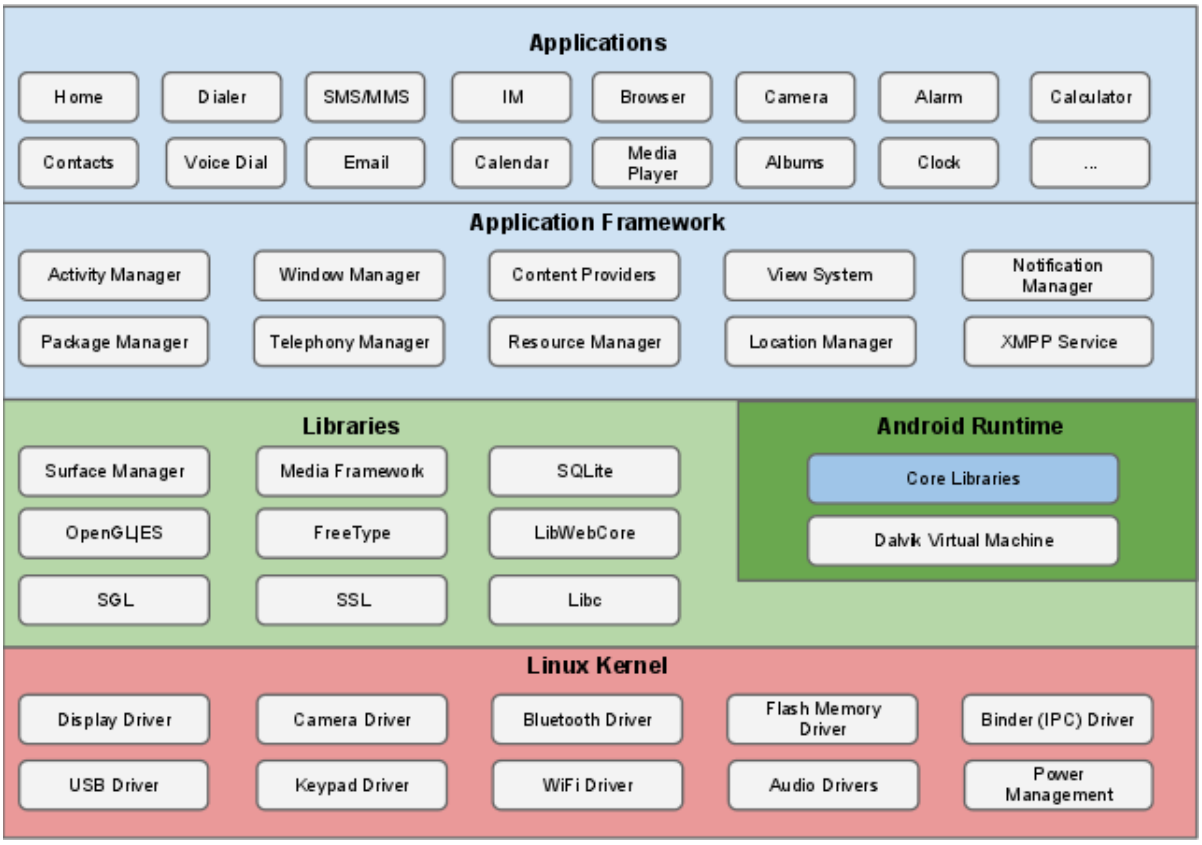

Abbildung 1: Der Android Software Stack (© android.com)
[and11a, Figure 1]

2.1.3 Sicherheit durch den Linuxkernel

Die Androidentwickler vertrauen auf die Sicherheit des Linuxkernels, die sie damit begründen, dass der Linuxkernel seit Jahren weit verbreitet ist und von einer großen Anzahl an Personen auf Sicherheitslücken untersucht wird. Als weitere Vorteile des Linuxkernels nennen die Entwickler das benutzerbasierte Rechtesystem, Prozessisolation, erweiterbare Mechanismen zur sicheren Interprozesskommunikation, sowie die Möglichkeit unnötige und/oder potentiell unsichere Teile des Kernels einfach zu entfernen. Des Weiteren ist Linux als Multi-User-Betriebssystem darauf ausgelegt, die Ressourcen der einzelnen Nutzer vor Zugriff durch andere, nicht berechtigte Benutzer zu schützen. Dies gilt sowohl für Systemressourcen wie Rechenzeit oder Arbeitsspeicher, als auch für die Dateien eines Benutzers. Im Kontext eines Smartphones also auch für Telefonie, GPS, Bluetooth, etc. [and11a, vgl. Linux Security]

2.1.4 Die Application Sandbox

Um die einzelnen Anwendungen/Prozesse voneinander zu isolieren, nutzt Android das Benutzerkonzept von Linux. Dieses wird hierbei jedoch nicht im klassischen Sinne genutzt, bei dem jeder Benutzer eine UID (User Identifier) hat und seine Prozesse im Regelfall mit dieser UID laufen. Android weißt jedem Prozess eine eigene UID zu. Dies könnte man also als Sandbox auf Kernel-Ebene bezeichnen, da über den Linuxkernel sicherstellt wird, dass die verschiedenen Prozesse/Apps nicht ohne Weiteres auf die Daten anderer Prozesse/Apps zugreifen können. Ein großer Vorteil dieser Methode ist, dass das Sandbox-Prinzip auch für nativ ausgeführten Code gilt und nicht nur für die Apps, die in der Dalvik VM[1] laufen. Im Bezug auf Abbildung 1 bedeutet das also, dass die oberen drei Schichten (Applications, Application Framework, Libraries/Android Runtime) dem Sandbox-Prinzip unterworfen sind.

Selbst wenn nun Lücken wie z.B. Buffer Overflows in diesen oberen drei Schichten dazu genutzt werden können Schadcode auszuführen, müsste zusätzlich eine Sicherheitslücke im Kernel vorhanden sein und ausgenutzt werden können, um aus dem Kontext des Prozesses auszubrechen. [and11a, vgl. The Application Sandbox]

Zu erwähnen ist außerdem noch, dass ausschließlich das hier erklärte Prinzip zum Sandboxing genutzt wird. Die Dalvik VM schränkt die Anwendungen nicht im Zugriff auf unterliegende Betriebssystemmittel ein, wie es etwa die Java VM oder die .net-Laufzeitumgebung tun. So ist es Anwendungen z.B. möglich auch nativen Code zu nutzen.
Des Weiteren ist es möglich Code aus externen Quellen zu laden, wovon die Android-Entwickler jedoch abraten, da dies die Gefahr birgt, dass dieser modifiziert wurde und so schädliches Verhalten hervorrufen kann. [and11b, vgl. Using Dalvik Code]

[1] `http://de.wikipedia.org/wiki/Dalvik_Virtual_Machine`

2.1.5 Verschlüsselung des Dateisystems

Ab Version 3.0 von Android gibt es die Möglichkeit, das komplette Dateisystem und somit alle Nutzerdaten zu verschlüsseln. Hierfür kommt *dmcrypt*, ein Modul für den Linuxkernel, zum Einsatz. Als Algorithmen werden AES128 im CBC-Modus[2] zur Verschlüsselung und ESSIV:SHA256 zum Hashing verwendet. Der Schlüssel für das verschlüsselte Dateisystem (*master key / filesystem key*) wird dabei wiederum mit AES128 verschlüsselt. Im Detail läuft die Verschlüsselung des Dateisystems wie folgt ab:

1. Es wird eine 128Bit-Zahl aus /dev/urandom gelesen. (Der *master key / filesystem key*.) Mit diesem wird das Dateisystem verschlüsselt.
2. Es wird ein zweiter Wert aus /dev/urandom zur Verwendung als Salt[3] ausgelesen.
3. Das User-Passwort wird samt Salt (Schritt 2) mit der PBKDF2-Funktion aus der SSL-Library gehasht.
4. Der Wert aus Schritt 1 (*master key*) wird mit dem Hash aus Schritt 3 AES128-verschlüsselt.
5. Der verschlüsselte *master key*, sowie der Salt werden im sog. *crypto footer* gespeichert.
   [and11d, vgl. Enabling encryption on the device]

Hierzu eine Visualisierung:

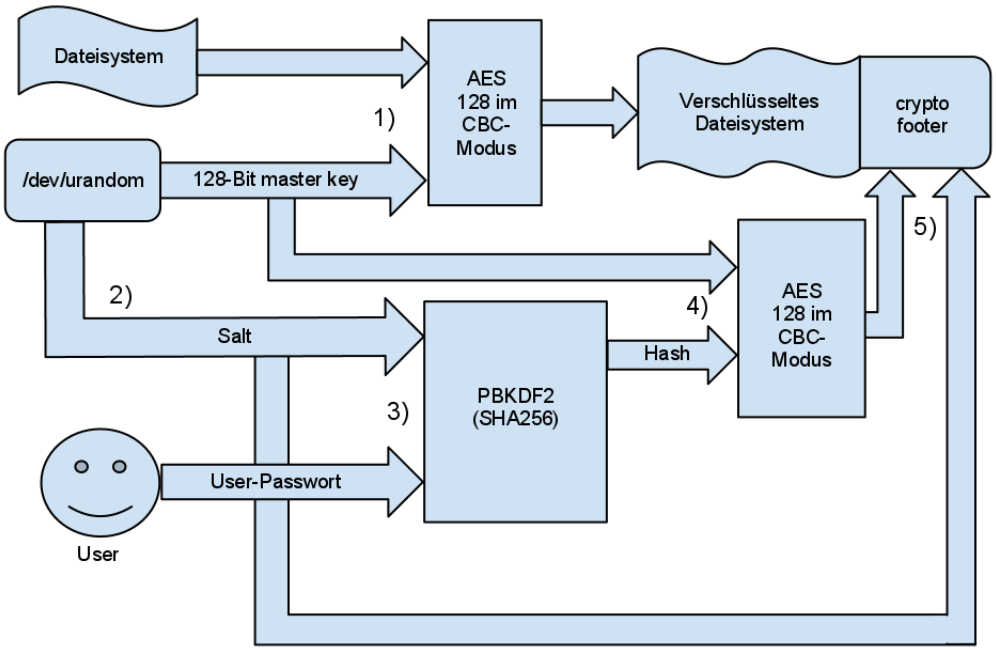

Abbildung 2: Ablauf der Dateisystemverschlüsselung

[2] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Cipher-block_chaining
[3] http://de.wikipedia.org/wiki/Salt_(Kryptologie)

Der Salt wird verwendet, da das User-Passwort alleine zu anfällig für "*rainbow table*"-Angriffe ist. Methoden wie *dictionary attacks* oder "*brute force*"-Angriffe werden zumindest durch den höheren Rechenaufwand verlangsamt.
Zu beachten ist, dass die Dateisystemverschlüsselung nur verwendet werden kann, wenn auch ein User-Passwort verwendet wird. Methoden wie Geräteentsperrung über ein auf dem Bildschirm abzufahrendes Muster werden nicht unterstützt. [and11a, vgl. Filesystem Encryption]

2.1.6 Device Administration API

Im Unternehmensumfeld ist das Prinzip BYOD (*bring your own device*) für die Administratoren der Unternehmen eine problematische Sache, da sie so die Kontrolle über die Geräte und dadurch auch über die Firmendaten verlieren. Um diesem Problem entgegenzuwirken bietet Android ab Version 2.2 die *Device Administration API* an. Sie ermöglicht Administratoren z.B. die Festlegung von Passwortrichtlinien, die Fern-Löschung der Firmendaten von einem Gerät, das Erzwingen einer Dateisystemverschlüsselung (ab Version 3.0), oder etwa das Deaktivieren der Kamera (ab Version 4.0). Die Umsetzung läuft folgendermaßen ab:

- Ein Administrator entwickelt mit Hilfe der *Device Administration API* eine Applikation, welche die Sicherheitsrichtlinien für die Endgeräte erzwingt. Die Sicherheitsrichtlinien sind dabei entweder fest in der Applikation hinterlegt, oder können beispielsweise dynamisch von einem Unternehmensserver nachgeladen werden.

- Diese Applikation muss auf den Geräten der Anwender/Mitarbeiter installiert werden. Dies geschieht wahlweise über den *Android Market* oder direktes Installieren auf dem Gerät über eine Unternehmensinterne Quelle.

- Der Nutzer muss diese Applikation nun aktivieren. Ab nun sind die Sicherheitsrichtlinien aktiv. Durch das Zustimmen/Aktivieren der Richtlinien wird den Mitarbeitern nun Zugriff auf Systeme und Daten des Unternehmens gewährt.

  [and11c, vgl. How does it work?]

Was passiert wenn ein User eine Richtlinie verletzt, ist der entsprechenden "*Device Admin. API*"-App überlassen. Der Standardfall könnte sein, dem User einfach entsprechende Rechte zu entziehen. Wird das vom Admin/Programmierer jedoch nicht konsequent beachtet, könnten wiederum Sicherheitslücken enstehen.

Da die API immer wieder erweitert wird und ältere Versionen nicht alle Policies unterstützen, wurde folgende Regelung getroffen: Versucht ein Gerät mit einer älteren "*Device Admin. API*" sich zu einem Server zu verbinden, der Policies voraussetzt, die die Version auf dem veralteten Gerät nicht unterstützt, erhält das Gerät keinen Zugriff. Der Grund ist, dass die API keine "partielle" Freigabe vorsieht. Hierdurch könnte der Zwiespalt entstehen entweder einige Geräte auszuschließen, oder die Policies entsprechend zu lockern, was wiederum die Sicherheit schwächt. Das in Abschnitt 2.1.1 beschriebene

“Update-Problem“ spielt hier ebenfalls eine Rolle, wenn einige Geräte Updates erst spät oder gar nicht erhalten.

Was passiert nun jedoch, wenn mehrere solcher Anwendungen auf dem Gerät laufen, darunter beispielsweise eine, welche gezielt lockere Richtlinien setzt um die Sicherheit zu schwächen? Als Lösung hierfür wurde einfach festgelegt, dass immer die strikteste Richtlinie gilt. [and11c, vgl. Device Administration]

2.1.7 Sicherheit der Speicherverwaltung

Um das Exploiten von Lücken im Zusammenhang mit *memory corruptions*, wie z.B. Buffer Overflows, zu erschweren bzw. zu verhindern, nutzt Android bzw. die zugehörige SDK samt Compiler folgende Sicherheitsfeatures, wie sie auch von anderen Systemen bekannt sind:

- ASLR - Randomisiert Adressen von z.B. Stack und Heap im Speicher.
- NX-Bit - Verhindert das Ausführen von Code in dafür nicht vorgesehenen Speicherbereichen.
- ProPolice - Compilererweiterung, die Stack Overflows verhindert.
- etc.

[and11a, vgl. Memory Management Security Enhancements]

2.1.8 Das “Rooten“

Im Regelfall laufen unter Android nur der Kernel und wenige Kernanwendungen mit root-Rechten. Diese haben somit Zugriff auf alle Daten auf dem Gerät. Für alle anderen Anwendungen sind root-Rechte nicht vorgesehen. Da manche Geräte es jedoch erlauben den Bootloader zu entsperren und so die Installation eines modifizierten Betriebssystems ermöglichen, ist es für Benutzer möglich Anwendungen mit root-Rechten laufen zu lassen. Wenn eine solche Anwendung nun Sicherheitslücken hat, ist die Sicherheit des ganzen Systems gefährdet. Um die Nutzerdaten zu schützen, löscht der Bootloader beim Entsperren selbige. Wenn man das Gerät nicht “rootet“ indem man ein modifiertes Betriebssystem aufspielt sondern eine Lücke im Kernel ausnützt, findet diese Löschung jedoch nicht statt. Um seine Daten, beispielsweise bei Verlust des Gerätes, zu schützen, könnte die Dateisystemverschlüsselung genutzt werden, da zum Entschlüsseln das Nutzerpasswort nötig ist, welches nicht auf dem Gerät hinterlegt ist. [and11a, vgl. Rooting of Devices]

2.1.9 Weiteres

- Die Systempartition
  Die Systempartition, welche den Kernel, Bibliotheken, die Laufzeitumgebung und die Anwendungen enthält ist *read-only* gesetzt, was die Manipulation selbiger verhindern soll. [and11a, vgl. System Partition and Safe Mode]

- Safe Mode
  Android kann in den sogenannten *Safe Mode* booten in dem nur Android Kernanwendungen vorhanden sind und sichergestellt ist, dass keine Software von Drittanbietern vorhanden ist bzw. läuft. [and11a, vgl. System Partition and Safe Mode]

- User Passwort
  Android ermöglicht das Festlegen eines Nutzerpasswortes zum Entsperren des Gerätes. Dieses Passwort wird ebenfalls für die Dateisystemverschlüsselung (siehe 2.1.5) genutzt. Vom Geräteadministrator können hierzu Regeln, z.B. zur Komplexität des Passwortes, festgelegt werden. [and11a, vgl.Password Protection]

- Application Signing
  Damit Anwendungen auf einem Android-System ausgeführt werden können, müssen sie signiert sein. Es wird jedoch ausschließlich vom Autor der Software signiert, nicht etwa von einer zentralen Prüfstelle, welche die Anwendungen zuvor untersucht. Des Weiteren werden die Signaturen noch verwendet, um z.B. Applikationen vom selben Autor, wenn dieser das explizit so angibt, eine gemeinsame UID zu geben [4] oder um sog. *Permissions* so zu schützen (protectionLevel[5]), dass nur Anwendungen vom gleichen Autor sie bekommen. [and11e, vgl. Application Signing]

[4] `http://developer.android.com/guide/topics/manifest/manifest-element.html#uid`
[5] `http://developer.android.com/guide/topics/manifest/permission-element.html#plevel`

2.2 iOS

2.2.1 Allgemein

Im Folgenden wird das Sicherheitskonzept des, von der Firma Apple entwickelten, Betriebssystems iOS näher betrachtet.

iOS ist ausschließlich für den Gebrauch mit einem iPhone, iPad, iPod Touch, oder Apple TV gedacht. Das Smartphone iPhone wird, wie das zugehörige Betriebsystem, von Apple entwickelt und vermarktet. Obwohl nur iPhones auf demy Handy-Markt mit iOS ausgestattet sind, schafft es das Handy dennoch auf Platz 2 der meist genutzten Smartphone-Betriebssysteme in den USA. [com11, vgl. Smartphone Platform Market Share]

Abbildung 3 zeigt die allgemeine iOS Architektur in Layer unterteilt. Die eigentlichen Sicherheits-APIs sind im *Core Services* Layer angesiedelt. Anwendungen die die APIs benutzen wollen, rufen diese direkt auf, ohne durch den *Cocoa Touch* oder *Media* Layer zu gehen. Programme die sichere Netzwerkfunktionen nutzen wollen, können dies per Zugriff auf die *CFNetwork* API realisieren. [Inc11b, vgl. iOS]

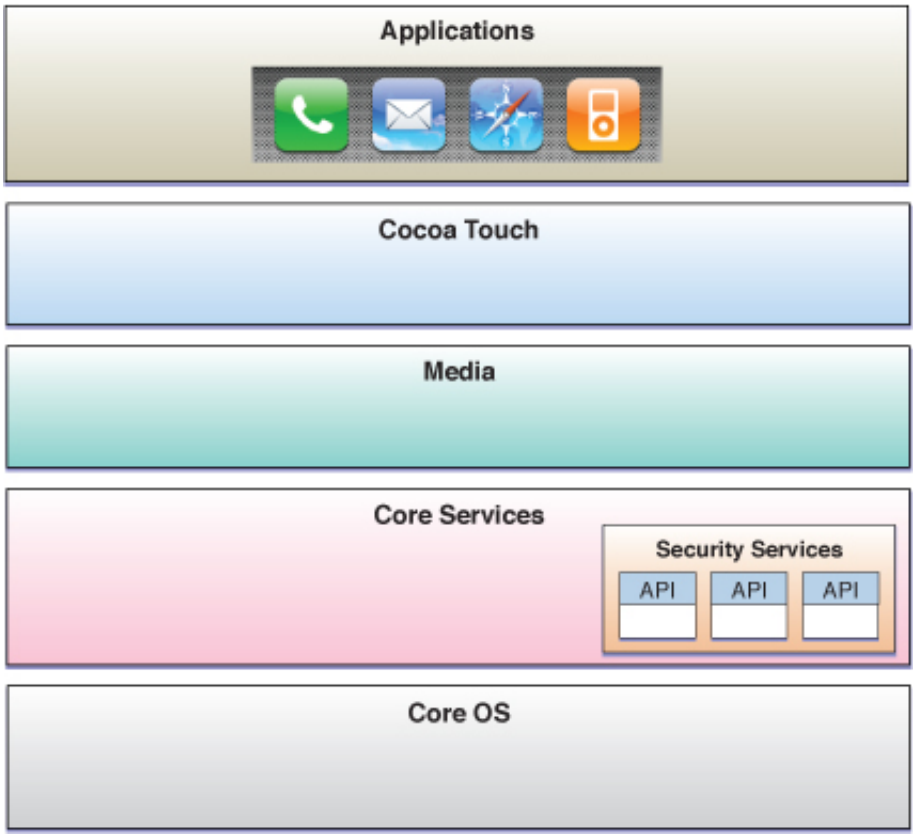

Abbildung 3: iOS Architektur im Überblick

2.2.2 Security Server

Weiterer wichtiger Baustein des Sicherheitskonzepts ist der *Security Server Daemon*. Dieser implementiert mehrere Sicherheitsprotokolle, sowie das *root certificate trust management* und die Geheimnisverwaltung (*Keychain*). Des Weiteren ist er für die *Authorization Services* verantwortlich. Wenn eine Anwendung gestartet wird, die bestimmte Rechte einfordert, so schlägt der Server in einer Datenbank mit Richtlinien nach, ob dem Programm die entsprechenden Rechte zugesagt wurden. Eine klassische Authorisierung (wie beispielsweise auf einem Linux-Heimcomputer) durch den Benutzer (z.B. Ändern der Leserechte einer Datei) sind nicht möglich. Anwendungen können lediglich vor dem Start einen PIN einfordern. [Inc11c, vgl. Authorization Services] Der *Security Server* hat allerdings keine öffentlichen APIs. Programme nutzen stattdessen die *Keychain Services API* und die *Certificate, Key, and Trust services API*. Diese kommunizieren dann wiederum mit dem*Security Server*. [Inc11b, vgl. Security Server Daemon]

2.2.3 iOS Sicherheits APIs

Die sicherheitsrelevanten APIs des iOS Betriebssystems teilen sich in drei Haupt-APIs auf. Zusätzlich wird eine weitere high-level API zur Verfügung gestellt. All diese APIs nutzen letztendlich die *Common Crypto* Bibliothek der Systembibliothek des *Core OS (Kernel)* Layers (*siehe Abbildung 4*). [Inc11b, vgl. iOS Security APIs]

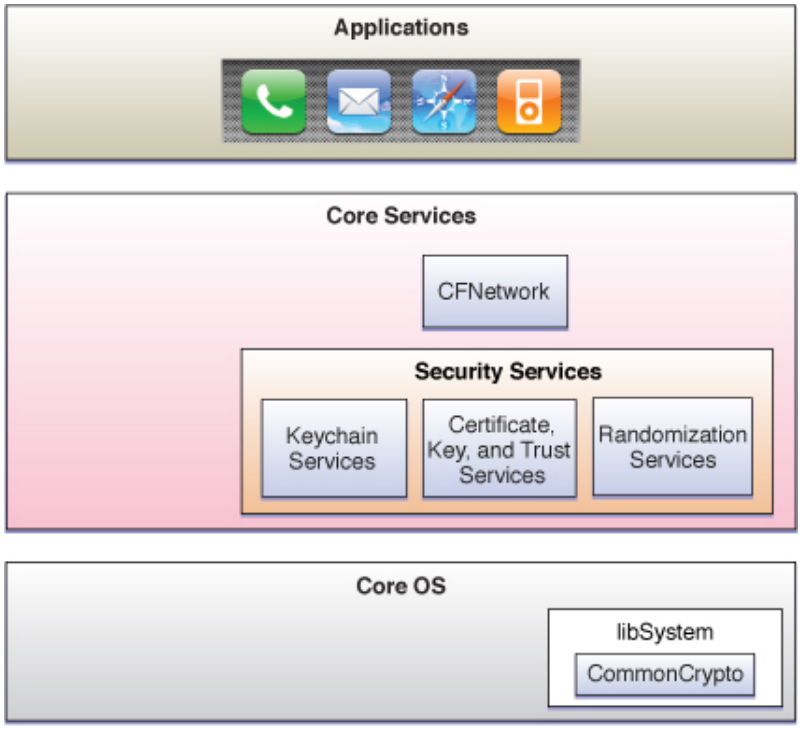

Abbildung 4: iOS Sicherheits-APIs

- CFNetwork
  CFNetwork ist eine high-level API, die es Programmen ermöglicht, sicher Datenströme zu verwalten und Authentisierungsinformationen an Nachrichten anzuhängen. Sie nutzt dafür die unter ihr liegenden Sicherheitsservices.
  [Inc11b, vgl. CFNetwork]

- Keychain Services
  Als *Keychain* bezeichnet Apple ihre Systeme zum Verwalten von Geheimnissen. Dies umfasst Passwörter, Schlüssel, Zertifikate, etc. Für diese Features wird auf die *Common Crypto* Bibliothek ,sowie auf Datenspeicherfunktionen zurückgegriffen.
  [Inc11b, vgl. Keychain]

- Certificate, Key, and Trust Services
  Die *Certificate, Key, and Trust Services* API beinhaltet folgende Funktionen:
  - Erstellen, Lesen und Verwalten von Zertifikaten.
  - Zertifikate zur Geheimnisverwaltung (*Keychain*) hinzufügen.
  - Verschlüsselungsschlüssel erstellen.
  - Daten verschlüsseln und entschlüsseln.
  - Daten signieren und Signaturen verifizieren.
  - Vertrauensrichtlinien (*trust policies*) verwalten.

  [Inc11b, vgl. Certificate, Key, and Trust Services]

- Randomization Services
  Diese API wird genutzt um kryptographisch sichere Zufallszahlen zu generieren. Bei den Zahlen handelt es sich um sogenannte Pseudozufallszahlen, da Computer keine echten Zufallszahlen generieren können. Der Algorithmus ist dabei aber so konstruiert, dass aus den generierten Zahlenfolgen keine Rückschlüsse auf den Algorithmus möglich sind.
  [Inc11b, vgl. Randomization Services]

2.2.4 Sandboxing

Anders wie bei seinem Vorreiter (*Mac OS X*). von dem das iOS Betriebssystem abstammt [Inc11a, vgl. Powerful Foundation], ist es hier Standard jede Anwendung in eine Sandbox zu installieren. Das grundlegende Prinzip der Sandbox ist wie bei Android auch, jede Anwendung zu isolieren. Ein Programm das gestartet wird, kann somit nur auf seine eigenen Daten und Einstellungen zugreifen. Auch in der Geheimnisverwaltung (*Keychain*) kann die App nur ihre eigenen Einträge nutzen. [Inc11c, vgl. Restrictions On Code Execution] Will sie jedoch erweiterte Rechte (z.B. Zugriff auf Kontakte), so muss eine Anfrage an den *Security Server* (siehe Kapitel Security Server) gestellt werden.

Eine andere Quelle kritisiert das System, da beispeilsweise für jede App die gleiche User ID genutzt wird. Dies kann zu Problemen führen wenn doch einmal die Sandbox überwunden wurde. Zusätzlich zu dem Problem der User ID kommt, dass technisch nicht

verhindert wird, dass Apps außerhalb ihres Verzeichnisses lesen können. In manchen Fällen müssen zwar zuerst die Dateinamen bekannt sein, in vielen anderen allerdings nicht. Das von Apple genannte Feature, dass Apps nur ihr Verzeichnis lesen können, wird durch die Kontrolle vor der Veröffentlichung der App im App Store realisiert. Liest eine Anwendung außerhalb des Verzeichnisses (ohne logische Begründung), wird sie einfach nicht zugelassen. [Hei11a, vgl.]

2.2.5 Code Signing

Beim *Code Signing* handelt es sich um klassisches Signieren, wobei die zu signierenden Dokumente Programme sind.

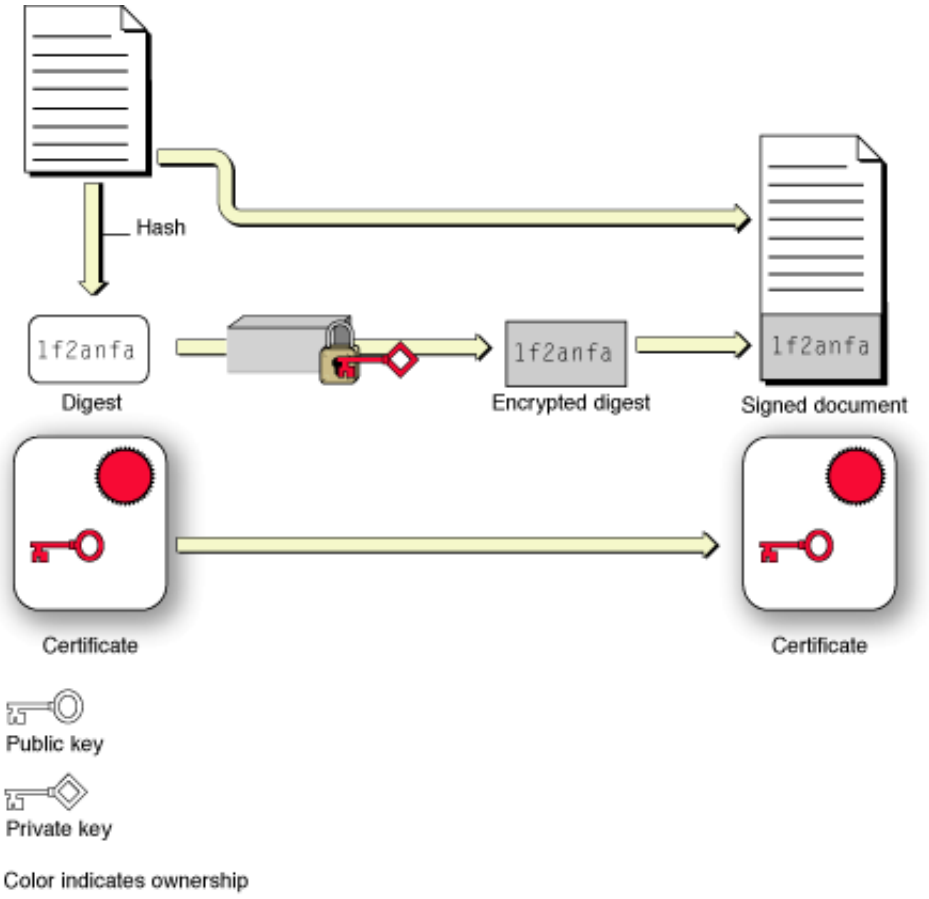

Abbildung 5: Digitales Signieren von Dokumenten

Durch das Verfahren wird gewährleistet, dass nur von Apple signierte Anwendungen auf dem Smartphone zur Ausführung kommen. Es wird also vor dem Programmstart die Anwendung geprüft (siehe Abbildung 6). Hierbei stellt sich nicht nur herraus, ob es sich um eine von Apple unterschriebene Datei handelt, sondern auch ob die Datei in irgendeiner Form manipuliert wurde. Veränderte Programme ohne gültige Signatur werden ebenfalls nicht vom iPhone ausgeführt.

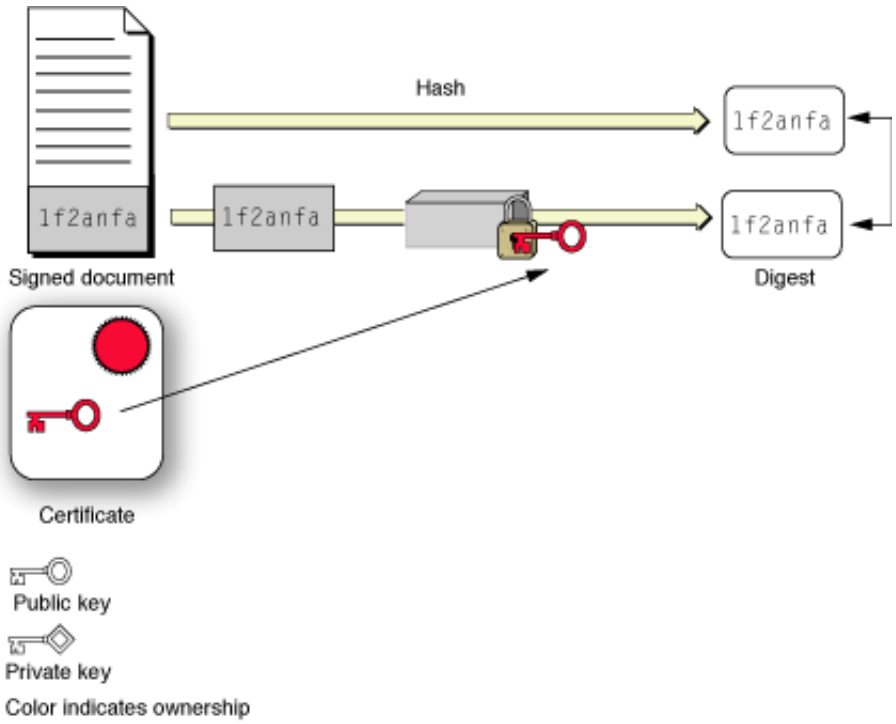

Abbildung 6: Verifizierung digital signierter Dokumente

Damit dieses System in sich schlüssig ist und Anwendungen weltweit vertrieben werden können, bietet Apple eine Download-Plattform für die Apps - den sogenannten *App Store*. Wenn ein Entwickler ein Programm geschrieben hat, reicht er die App bei Apple ein. Dann wird diese anhand vieler Kriterien (nicht ausschließlich Sicherheit) überprüft. Erfüllt sie den Katalog an Bedingungen, wird die Anwendung digital signiert (siehe Abbildung 5) und im App Store veröffentlicht. [Sta10, vgl.]

Der Kontrollpozess bevor eine App veröffentlicht wird, wird von vielen als größter Sicherheitsgewinn im Umgang mit iOS empfunden, obwohl er nicht transparent ist. Allgemein ist festzustellen, dass Apps meistens die Überprüfung bestehen und zugelassen werden. [wik11, vgl. Abschnitt: Entwicklung von Apps für den Store] Der US-Amerikanische Sicherheitsexperte Charlie Miller stellte das System auf die Probe, in dem er eine harmlos aussehende App einreichte. Keiner entdeckte die Funktion Code nachzuladen und auszuführen, selbst wenn dieser unsigniert ist. Die Anwendung wurde wie von Miller geplant, im App Store veröffentlicht. Inzwischen ist diese natürlich nicht mehr abrufbar. [Gre11, vgl.]

2.2.6 Jailbreak

Eine weitere Möglichkeit die Sicherheitsmechanismen von Apple zu umgehen, ist ein *Jailbreak*. Ein Jailbreak erlaubt das Benutzen des Smartphones ohne die Nutzungseinschränkungen. So können beispielsweise nach einem Jailbreak, Apps von anderen Quellen als dem App Store installiert werden. Da nach einem Jailbreak der Nutzer *root*-Rechte hat, besitzt er allgemein alle Freiheiten, ist aber auch alleinig für die Sicherheit verantwortlich. Ein durchschnittlicher Smartphone-Besitzer verliert hierbei schnell die Übersicht und installiert unter Umständen auch getarnte Malware. [And09, vgl.] [New09, vgl.]

3 Vergleich der Sicherheitsfeatures

	Android	iOS
Sandboxing	 Alles im Userspace, also auch für nativen Code	
Verschlüsselung des Dateisystems	 ab v3.0 AES128,SHA256	
read-only Systempartition		 (siehe: [Mor10])
Sicheres Booten (ohne Drittanbieter-Software)		 Vermutlich nicht, da andere Konzepte dies überflüssig machen
Sicherheitskonzept für BYOD	 Dev. Admin. API ab v2.2	Kaum von Apple, hauptsächlich über Drittanbieter möglich [Hei11b]
Nur signierter Code ausführbar	 Code Signing (nur durch Entwickler)	 Code Signing (durch Entwickler und Apple)
Kontrollierte App-Distribution	Keine Kontrolle durch Google	 App Store
Maßnahmen gegen *memory curruption*		 (siehe: [Con11])

Tabelle 1: Sicherheitsfeatures im Vergleich

4 Quellen

Literatur

[And09] Brigid Andersen. Australian admits creating first iphone virus. Webseite, nov 2009. Online verfügbar unter http://www.abc.net.au/news/2009-11-09/australian-admits-creating-first-iphone-virus/1135474; abgerufen im Dezember 2011.

[and11a] android.com. Android security overview. Website, 2011. Online verfügbar unter http://source.android.com/tech/security/index.html; abgerufen im Dezember 2011.

[and11b] android.com. Designing for security. Website, 2011. Online verfügbar unter http://developer.android.com/guide/practices/security.html; abgerufen im Dezember 2011.

[and11c] android.com. Device administration. Website, 2011. Online verfügbar unter http://developer.android.com/guide/topics/admin/device-admin.html; abgerufen im Dezember 2011.

[and11d] android.com. Notes on the implementation of encryption in android 3.0. Website, 2011. Online verfügbar unter http://source.android.com/tech/encryption/android_crypto_implementation.html; abgerufen im Dezember 2011.

[and11e] android.com. Security and permissions. Website, 2011. Online verfügbar unter http://developer.android.com/guide/topics/security/security.html; abgerufen im Dezember 2011.

[com11] comScore. Mobile subscriber market share. Website, jun 2011. Online verfügbar unter http://www.comscore.com/Press_Events/Press_Releases/2011/8/comScore_Reports_June_2011_U.S._Mobile_Subscriber_Market_Share; abgerufen im Dezember 2011.

[Con11] Lucian Constantin. ios gets native aslr. Webseite, mar 2011. Online verfügbar unter http://news.softpedia.com/news/iOS-Finally-Gets-Native-ASLR-188762.shtml; abgerufen im Dezember 2011.

[Gre11] Andy Greenberg. iphone security bug lets innocent-looking apps go bad. Website, jul 2011. Online verfügbar unter http://www.forbes.com/sites/andygreenberg/2011/11/07/iphone-security-bug-lets-innocent-looking-apps-go-bad/; abgerufen im Dezember 2011.

[Hei11a] Jens Heider. Gegenüberstellung der technischen iphone und android sicherheit. PDF (Folien), mar 2011. Online verfügbar unter http://www.dfn.de/fileadmin/3Beratung/Betriebstagungen/bt54/forum-mobileit-heider.pdf; abgerufen im Dezember 2011.

[Hei11b] Jens Heider. Smartphone-sicherheit im betrieblichen einsatz. PDF (Folien), mar 2011. Online verfügbar unter http://www.dfn.de/fileadmin/3Beratung/Betriebstagungen/bt54/forum-mobileit-heiderEnterprise_Smartphones.pdf; abgerufen im Dezember 2011.

[Inc11a] Apple Inc. Develop for ios. Website, dec 2011. Online verfügbar unter https://developer.apple.com/technologies/ios/; abgerufen im Dezember 2011.

[Inc11b] Apple Inc. Security overview: Security architecture. Website, dec 2011. Online verfügbar unter https://developer.apple.com/library/ios/#documentation/Security/Conceptual/Security_Overview/Architecture/Architecture.html#//apple_ref/doc/uid/TP30000976-CH202-SW3; abgerufen im Dezember 2011.

[Inc11c] Apple Inc. Security overview: Security services. Website, dec 2011. Online verfügbar unter https://developer.apple.com/library/ios/#documentation/Security/Conceptual/Security_Overview/Security_Services/Security_Services.htm; abgerufen im Dezember 2011.

[Mor10] Sean Morrissey. Ios forensic analysis. Book, eBook, 2010. Teils Online verfügbar unter http://books.google.de/books?id=XYHMwVYZ3lsC&lpg=PA41&dq=systempartition%20ios&hl=de&pg=PA41#v=onepage&q=systempartition%20ios&f=false; abgerufen im Dezember 2011.

[New09] BBC News. New iphone worm can act like botnet say experts. Webseite, nov 2009. Online verfügbar unter http://news.bbc.co.uk/2/hi/technology/8373739.stm; abgerufen im Dezember 2011.

[Sta10] AppleInsider Staff. Apple issues review guidelines for mac app store. Website, oct 2010. Online verfügbar unter http://www.appleinsider.com/articles/10/10/20/apple_issues_review_guidelines_for_mac_app_store.html; abgerufen im Dezember 2011.

[wik11] wikipedia.de. Wikipedia: App store. Website, dec 2011. Online verfügbar unter https://de.wikipedia.org/w/index.php?title=App_Store&oldid=96931169#Entwicklung_von_Apps_f.C3.BCr_den_Store; abgerufen im Dezember 2011.