# Entwurf und Implementierung eines Konzepts zur Absicherung mobiler Geräte am Beispiel von Apple iOS

## Bachelorarbeit

Studiengang Medizinische Informatik

29. Februar 2012

UNIVERSITÄT
HEIDELBERG

HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK


| | |
|---|---|
| vorgelegt von: | Markus Troßbach |
| Matrikelnummer: | 172087 |
| Referent: | Prof. Dr. Dirk Heuzeroth |
| Koreferent: | Prof. Dr. Martin Haag |
| Betreuer: | Dipl.-Inform. Med. Andreas Kurtz |



Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

# Zusammenfassung

Die vorliegende Bachelorarbeit befasst sich mit der Absicherung mobiler Geräte am Beispiel des Betriebssystems „iOS“ der Firma Apple. Das Ziel dieser Arbeit ist es, ein Konzept zur Absicherung des iOS-Betriebssystems zu entwerfen und zu implementieren.

Geräte mit dem Betriebssystem iOS bieten die Möglichkeit, Applikation von Drittanbietern zu installieren und auszuführen. Dies stellt ein potentielles Sicherheitsrisiko für den Nutzer und die auf dem Gerät gespeicherten persönlichen Daten dar. So wurden in Vergangenheit bereits einige Fälle öffentlich bekannt, in denen eine Applikation unbemerkt private Daten des Nutzers (wie beispielsweise dessen Adressbuch) ausgelesen und an Dritte versendet hat. Durch diese Vorfälle motiviert, wird durch das hier vorgelegte Sicherheitskonzept eine Lösung geschaffen, über die der Nutzer den Zugriff auf persönliche Daten kontrollieren kann. Das unbemerkte Auslesen persönlicher Daten wird durch geeignete Maßnahmen auf dem Endgerät verhindert.

Zur Erreichung des Ziels dieser Arbeit wird zunächst eine Anforderungsanalyse an eine solche Lösung durchgeführt. Dabei werden auch auf Ergebnisse bereits veröffentlichter Arbeiten und auf Sicherheitslösungen für andere mobile Betriebssysteme zurückgegriffen. Die Analyse umfasst Anforderungen zum Schutz der Privatsphäre des Nutzers, zur Systemhärtung, zur Benutzerfreundlichkeit der Lösung und zur Erweiterbarkeit einer Lösung. Auf Grundlage der aufgestellten Anforderungen wird ein Konzept für eine Sicherheitsapplikation entwickelt. Es werden Quellen für private Daten auf den Geräten ermittelt und verschiedene Ansätze zur Lösung bewertet. Zu jeder Anforderungen werden jeweils konkrete Maßnahmen zur Umsetzung beschrieben. Aus dem entwickelten Konzept wird eine Sicherheitsapplikation implementiert und in dieser Arbeit vorgestellt. Zur Realisierung der Kontrolle der installierten Applikation kommt die Technik der „Dynamic Library Injection“ zur Anwendung.

Im abschließenden Teil werden die Ergebnisse der Arbeit zusammengefasst und eingeordnet. Aus den Grenzen des Konzepts und der implementierten Sicherheitsapplikationen werden Vorschläge für weitergehende Untersuchungen und Entwicklungen abgeleitet.

# Inhaltsverzeichnis

**Zusammenfassung** **I**

**Abbildungsverzeichnis** **V**

**Tabellenverzeichnis** **VI**

**Verzeichnis der Listings** **VII**

**1. Einleitung** **1**
- 1.1. Einführung . . . 1
- 1.2. Motivation . . . 2
- 1.3. Ziele der Arbeit . . . 3
- 1.4. Aufbau der Arbeit . . . 4

**2. Grundlagen** **5**
- 2.1. iOS Sicherheitskonzept . . . 5
- 2.2. Jailbreak . . . 7
- 2.3. Dynamic Library Injection . . . 8
  - 2.3.1. Funktionsweise . . . 8
  - 2.3.2. MobileSubstrate . . . 9
- 2.4. Gorilla . . . 11

**3. Anforderungsanalyse** **12**
- 3.1. Schutz der Privatsphäre des Nutzers . . . 12
- 3.2. Schutz des Systems vor dem Einschleusen von Schadcode . . . 13
- 3.3. Einfache Konfiguration . . . 14
- 3.4. Einfache Erweiterbarkeit . . . 15

**4. Verwandte Arbeiten** **16**
- 4.1. WhisperCore . . . 16
- 4.2. AppFence . . . 17
- 4.3. App Permission Watcher . . . 18
- 4.4. Zusammenfassung und Bezug . . . 19

**5. Konzept** **20**
5.1. Methoden . . . 20
5.2. Schutz der Privatsphäre des Nutzers . . . 25
5.2.1. Unique Device Identifier und WLAN MAC-Adresse . . . 26
5.2.2. Adressbuch . . . 27
5.2.3. Kalender . . . 28
5.2.4. Keyboard-Cache . . . 29
5.2.5. Freier Speicherplatz . . . 30
5.3. Einschleusen von Schadcode verhindern . . . 30
5.3.1. Änderung des User-Agents . . . 31
5.3.2. Hinweis auf unsicherer Systemeinstellungen . . . 31
5.4. Einfache Konfiguration . . . 32
5.4.1. Konfiguration über Profile . . . 32
5.4.2. Erinnerungsmodus . . . 33
5.5. Erweiterbarkeit durch erweiterbare Konfigurationsdateien . . . 33

**6. Implementierung** **35**
6.1. Benutzeroberfläche . . . 35
6.1.1. Bereich „Home“ . . . 36
6.1.2. Bereich „System“ . . . 37
6.1.3. Bereich „Apps“ . . . 37
6.1.4. Bereich „Help“ . . . 40
6.2. Architektur . . . 41
6.2.1. Berechtigungsprofile und Applikationskonfigurationen . . . 42
6.2.2. Übersichtsbildschirm . . . 46
6.2.3. Überwachung der Dateisystemzugriffe . . . 47

**7. Diskussion und Ausblick** **49**
7.1. Diskussion . . . 49
7.2. Ausblick . . . 50
7.2.1. Schutz weiterer Daten . . . 50
7.2.2. Überwachung der Dateisystemzugriffe . . . 51
7.2.3. Verbesserung der Systemhärtung . . . 51
7.2.4. Webservice für Profile und Einstellungen . . . 51
7.2.5. Verbesserung der Methoden . . . 52
7.3. Fazit . . . 52

**Literaturverzeichnis** **54**

**A. Anhang** **i**

A.1. Datenquellen für private Daten in iOS . . . . . i

A.1.1. Geräteinformationen . . . . . i

A.1.2. SIM-Informationen . . . . . i

A.1.3. Telefonbuch . . . . . ii

A.1.4. Kalender . . . . . ii

A.1.5. Ortungsdienst . . . . . ii

A.1.6. Fotos und Videos . . . . . iii

A.1.7. Tastatur . . . . . iii

A.1.8. WLAN . . . . . iii

A.1.9. Email . . . . . iv

**Eidesstattliche Erklärung** **v**

# Abbildungsverzeichnis

1.1. App Store Review Prozess . . . . . 2

2.1. Dynamic Linker Model in iOS . . . . . 8
2.2. Inline Function Hooking . . . . . 10

5.1. iOS System Architektur . . . . . 21
5.2. Ersetzung von Systemmethoden . . . . . 24

6.1. Home-Bildschirm . . . . . 36
6.2. System-Bildschirm . . . . . 37
6.3. Einstellungen für Apps vornehmen . . . . . 38
6.4. Ausnahmen für einzelne Einstellungen erstellen . . . . . 39
6.5. Neues Berechtigungsprofil anlegen . . . . . 40
6.6. Help-Bildschirm . . . . . 41
6.7. Architektur der Sicherheitsapplikation . . . . . 41
6.8. Flussdiagramm vom Laden der Konfiguration im Tweak . . . . . 45
6.9. Vereinfachtes UML-Klassendiagramm des SecurityCenters . . . . . 46
6.10. Beispiel für eine symbolische Verknüpfung . . . . . 47

# Tabellenverzeichnis

A.1. Datenquellen für Geräteinformationen . . . . . i
A.2. Datenquellen für SIM-Informationen . . . . . i
A.3. Datenquellen für Kontaktdaten und Telefonummern . . . . . ii
A.4. Datenquellen für Termindaten . . . . . ii
A.5. Datenquellen für Positionsdaten . . . . . iii
A.6. Datenquellen für Foto- und Videodaten . . . . . iii
A.7. Datenquellen für Tastaturdaten . . . . . iii
A.8. Datenquellen für WLAN-Daten . . . . . iii
A.9. Datenquellen für Email-Daten . . . . . iv

# Verzeichnis der Listings

6.1. Beispiel für ein Berechtigungsprofil . . . . . 43

# 1. Einleitung

## 1.1. Einführung

Ausgelöst durch die rasante Weiterentwicklung der Mobiltelefone in den letzten Jahren hat sich ein neuer Markt der so genannten *post-pc-devices* gebildet. Zu dieser Klasse von Geräten gehören in erster Linie die neuste Generation der Smartphones und der Tablet-PCs. Laut dem Marktforschungsunternehmen Gartner [Gartner, Inc. 2011] wurden allein im Jahr 2010 über 296 Mio. Smartphones und über 17 Mio. Tablets weltweit verkauft. Die Geräte haben sich vor allem durch ihre direkte und einfache Bedienbarkeit, zu einer Kommunikations- und Unterhaltungsplattform entwickelt. Auch in Unternehmen kommen sie mittlerweile immer mehr zur Anwendung.

Einen großen Teil dieser Anziehungskraft ist neben den neuen Bedienungskonzepten vor allen Dingen den Marktplätzen für Applikationen zuzuschreiben. Auf diesen Marktplätzen können oft kostenlos oder für einen geringen Betrag Applikationen von Drittanbietern erworben, heruntergeladen und direkt auf dem Gerät installiert werden. Nach offiziellen Angaben in [Apple, Inc. 2011a] waren Anfang Juli 2011, nur drei Jahre nach dessen Eröffnung, bereits über 425.000 Applikationen im *App Store* von Apple verfügbar.

Die Möglichkeit Applikationen, meist einfach nur *Apps* genannt, von Drittanbietern installieren und ausführen zu können, stellt ein potentielles Sicherheitsrisiko für den Benutzer dar. Auf den Geräten sammeln sich eine Menge privater Daten. Diese können unter anderem im Telefonbuch, im Kalender, in den Metadaten der Fotos und im Verlaufsspeicher des Webbrowsers gefunden werden. Auch der Ortungsdienst des Gerätes kann verwendet werden. Zusätzlich verwenden viele dieser Apps Analysebibliotheken von Drittanbietern, bei denen das Nutzungsverhalten der App aufgezeichnet und zur Weiterverarbeitung an einen Analysedienst übermittelt wird. Jedes Gerät besitzt eine eindeutige ID, die Unique Device ID (UDID). Diese wird häufig zur Benutzeridentifikation verwendet. Da dabei wenige große Anbieter fast den kompletten Markt abdecken, können unter Umständen auch Nutzungsprofile einzelner Nutzer über mehrere Apps hinweg erstellt werden.

## 1.2. Motivation

Apple versucht auf seiner Plattform *iOS* durch vorherige Kontrolle der Apps einen gewissen Grad an Sicherheit herzustellen. Jede App, die auf dem Gerät installiert werden soll, muss aus dem von Apple kontrollierten App Store stammen. Um in den App Store aufgenommen zu werden, muss die App einen Review Prozess (siehe Abbildung 1.1) durchlaufen, bei dem unter anderem auch Stabilitäts- und Sicherheitsaspekte untersucht werden. Die App muss bei der Einreichung vom Entwickler digital signiert sein. Um die App auf einem iOS Gerät ausführen zu können wird jedoch auch die digitale Signatur von Apple benötigt. Diese wird nach erfolgreichem Abschluss des Review Prozesses gewährt. Da der Prüfungsprozess jedoch nicht öffentlich beschrieben ist, kann nicht allein auf dessen Ergebnis vertraut werden.

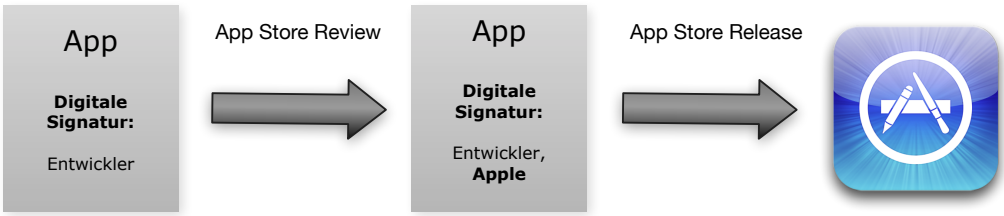


Abbildung 1.1.: App Store Review Prozess

Denn bereits in der Vergangenheit wurden Apps im App Store zugelassen, welche unbemerkt die Telefonnummern der Nutzer versendet und diese dann zu Werbezwecken verwendet haben. In der Quelle [The Register 2009] ist dieser Fall beschrieben. Der Benutzer selbst kann, mit Ausnahme der Ortungsfunktionalität, den Apps nicht den Zugriff auf einzelne Bereiche verbieten sondern muss dem Selektionsprozess von Apple und dem Anbieter der App vertrauen.

Des Weiteren wurde bereits gezeigt, dass durch Fehler in Systemprogrammen das Einschleusen und Ausführen von Programmcode möglich sind ([ZDNet 2011]). Diese Sicherheitslücken können dazu verwendet werden, unbemerkt Programme auf einem Gerät zu installieren, die nicht aus dem offiziellen App Store stammen und somit keiner Kontrolle unterzogen wurden. Diese Schwachstellen werden aber teilweise auch durch das Systemverhalten begünstigt. Es werden z.B. PDF und Microsoft Office Dokumente im Email Client automatisch geöffnet und angezeigt wenn diese einer Email anhängen. Gleichzeitig existieren in den Komponenten zur Anzeige von PDF und

Office-Dateien aber zahlreiche Schwachstellen (CVE-2011-3261[1], CVE-2011-3260[2], CVE-2011-0208[3], CVE-2011-0184[4], etc.), die sich für einen Angriff auf das System ausnutzen lassen. Somit könnte ein Angriff völlig unbemerkt alleine durch die automatische Vorschau einer Email erfolgen.

Im Rahmen einer statischen Analyse [Egele u. a. 2011] von Apps wurden bereits zahlreiche Verstöße gegen die Privatsphäre aufgedeckt. Dabei wurden rund 1.400 Apps automatisiert dahingehend untersucht, ob personenbezogene Daten ohne Zustimmung des Nutzers übertragen werden. Die Ergebnisse zeigten, dass 14,57 % der Apps direkt auf die personenbezogenen Daten Zugriff nehmen. Weiterhin wurde gezeigt, dass 55 % der getesteten Apps zusätzlich eine Analysebibliothek eines Drittanbieters einsetzen. Diese Analysebibliotheken nehmen ebenfalls Zugriff auf Daten, die zur Identifizierung des Geräts und somit des Nutzers verwendet werden können.

## 1.3. Ziele der Arbeit

In der Bachelorarbeit soll eine Sicherheitsarchitektur für das Betriebssystem iOS der Firma Apple entwickelt und implementiert werden. Dieses Konzept soll es ermöglichen, Apps beim Abruf von privaten Daten zu kontrollieren und das ungewollte Auslesen durch geeignete Maßnahmen zu verhindern.

Der Benutzer des Gerätes soll in die Lage versetzt werden, Zugriffsberechtigung auf privaten Daten für einzelne Apps selektiv zu vergeben. Die übergreifende Identifizierbarkeit durch Analyse-Tools soll unterbunden werden und der Benutzer soll in die Lage versetzt werden, das Aufzeichnen des Benutzerverhaltens durch die Applikationen deaktivieren zu können. Außerdem soll der Nutzer auf möglicherweise unsichere Einstellungen seines Gerätes (z.B. durch das Verwenden von Standard-Passwörtern) hingewiesen werden.

Die Umsetzung soll durch die Entwicklung einer Applikation für das Betriebssystem iOS erfolgen. Da die Applikation dabei tiefer in das System eingreifen muss als Apple dies durch Nutzungsbeschränkungen gestattet, wird die Applikation nur auf Geräten mit aktiven *Jailbreak* lauffähig sein. Als Jailbreak wird das Überwinden

[1] Weitere Informationen: `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3261`
[2] Weitere Informationen: `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3260`
[3] Weitere Informationen: `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0208`
[4] Weitere Informationen: `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-0184`

dieser Nutzungsbeschränkungen durch bestimmte Werkzeuge bezeichnet, die es erlauben Systemkomponenten zu manipulieren und Apps direkt zu installieren (siehe Abschnitt 2.2).

## 1.4. Aufbau der Arbeit

Die Arbeit gliedert sich in sieben Teile, die im Folgenden kurz vorgestellt werden. Im ersten Kapitel wird die Motivation und die Ziele der Arbeit vorgestellt. Im Kapitel 2 werden die Grundlagen vermittelt, die für das Verständnis der Arbeit benötigt werden. Das Kapitel 3 arbeitet die Anforderungen heraus, die an ein Lösungskonzept gestellt werden müssen. Im Kapitel 4 werden verwandte Arbeiten, die ein ähnliches Ziel verfolgen, kurz vorgestellt. Das Konzept zur Implementierung einer Lösung wird im Kapitel 5 beschrieben. Dabei werden verschiedene Lösungsansätze abgewogen. Im Kapitel 6 wird die Implementierung vorgestellt. Das Kapitel 7 gibt eine Zusammenfassung der Arbeit und einen Ausblick auf Themen für weitere Arbeiten und Forschungen.

# 2. Grundlagen

Diese Kapitel befasst sich mit den Grundlagen, die zum Verständnis dieser Arbeit benötigt werden. Hierzu wird in Abschnitt 2.1 zunächst das Sicherheitskonzept des Apple iOS Betriebssystems erläutert. Im darauf folgenden Abschnitt werden die Grundlagen eines *Jailbreaks* detailliert beschrieben. In Abschnitt 2.3 wird die Technik der Dynamic Library Injection erläutert. Dieses Verfahren wird im Rahmen dieser Arbeit verwendet, um die Funktionalität von Apps um Sicherheitsroutinen zu erweitern. Am Ende dieses Kapitels wird eine bereits existierende Sicherheitslösung vorgestellt und diskutiert. Der Schwerpunkt der weiteren Arbeit beschäftigt sich mit der Erweiterung dieser vorhandenen Technologie.

## 2.1. iOS Sicherheitskonzept

Das Betriebssystem iOS verfügt bereits über einige im System integrierte Sicherheitsmerkmale. Im Folgenden werden die wichtigsten, für die Ausarbeitung relevanten Sicherheitsmechanismen beschrieben.

Das erste Sicherheitsmerkmal, das iOS den Nutzern bietet, ist der App Store selbst. Dieser Marktplatz ist die einzige Quelle zum Installieren neuer Apps auf dem Gerät. Andere offizielle Vertriebswege sind, mit Ausnahme einer unternehmensinternen Verteilung unternehmensspezifischer Apps, nicht erlaubt und werden durch die Überprüfung der digitalen Signatur der App verhindert. Es können nur Apps, die von Apple signiert wurden auf dem Gerät ausgeführt werden. Um als Entwickler eine App in den App Store einstellen zu können, muss diese zuvor durch Apple im sogenannten *App Store Review* geprüft werden. Welche Tests dabei genau vorgenommen werden ist öffentlich nicht bekannt. Es wurde lediglich ein Leitfaden[5] für Entwickler veröffentlicht, in dem einige Gründe für eine Ablehnung der App genannt werden. Aufgrund der Ausführungen im Leitfaden ist es sehr wahrscheinlich, dass sowohl statische als auch dynamische Analysen stattfinden. Die meisten dieser Prüfungen beziehen sich jedoch auf die Anwendungsstabilität, rechtliche Aspekte und

[5] App Store Review Guidelines: `https://developer.apple.com/appstore/resources/approval/guidelines.html`

der Konformität zu Konventionen der Plattform. Innerhalb des Leitfadens werden kaum Aspekte im Bereich des Datenschutzes oder der Systemsicherheit genannt. Da jedoch der genaue Testablauf und Umfang nicht bekannt ist, kann die dadurch entstehende Sicherheit durch Außenstehende nicht bewertet werden. Wie bereits in der Einleitung in Kapitel 1.2 auf Seite 2 erwähnt, gab es auch schon einige Fälle in dem dieses System versagt hat.

Das nächste Sicherheitsmerkmal betrifft das Ausführen der Programme auf dem Gerät. Das System unterscheidet zwischen zwei verschiedenen Typen von Apps. Den System-Apps und den User-Apps. Die von Apple vorinstallierten Apps sind System-Apps. Sie unterliegen kaum Einschränkungen und können auch Root-Rechte erlangen. Die vom Benutzer über den App Store nachinstallierte Programm werden als User-Apps eingestuft und werden in einer *Sandbox* ausgeführt. Mit Hilfe dieses Sandbox-Mechanismus können Apps voneinander abgeschottet werden, indem man Dateisystemzugriffe nur einem abgeschlossenen Bereich erlaubt. Änderungen am System oder an anderen Apps werden unterbunden. Um jedoch Funktionen wie beispielsweise den Zugriff auf das Adressbuch ermöglichen zu können, gibt es auch gemeinsame Bereiche, auf die alle Apps Zugriff haben.

Ein weiteres Sicherheitsmerkmal betrifft den Schutz der Positionsdaten des Nutzers. Wenn eine App zum ersten Mal den Ortungsdienst, also das GPS-Modul, die Ortung über WLAN Access Points oder die Ortung durch Mobilfunksendemasten, des Gerätes verwendet, informiert das System zuvor den Nutzer durch ein Dialogfenster auf dem Bildschirm. Der Nutzer hat die Möglichkeit dieser App den Zugriff zum Ortungsdienst zu verweigern oder ihn zuzulassen. Die Apps haben keine Möglichkeit diese Abfrage zu umgehen, zu verhindern oder die Antwort vorwegzunehmen. Über das Einstellungsmenü des Gerätes kann diese Erlaubnis jederzeit vom Nutzer wieder entzogen oder erst nachträglich erteilt werden.

Um das Gerät vor unberechtigter Nutzung zu schützen gibt es die Möglichkeit einen *Passcode* einzurichten. Wenn ein Passcode aktiviert ist muss jedes Mal, wenn das Gerät aus dem Standby-Modus geweckt wird ein geheimer Code zur Entsperrung eingegeben werden. Das Betriebssystem bietet die Möglichkeit einen aus vier Ziffern bestehenden Code einzurichten. Außerdem wird auch noch die Option eines erweiterten Code geboten, der eine beliebige Länge haben kann und aus beliebigen, über die Tastatur eingebbaren Zeichen bestehen kann. Weiterhin kann eine automatische Sperrung aktiviert werden, die das Gerät nach einer vordefinierten Zeit der Inaktivität sperrt und in den Standby-Modus versetzt.

## 2.2. Jailbreak

Als *Jailbreak* wird das Überwinden der Nutzungseinschränkungen auf Geräten mit dem Betriebssystem iOS bezeichnet. Es existiert schon seit einigen Jahren eine Community, die es sich zum Ziel gemacht hat Werkzeuge zu entwickeln, mit denen die Geräte von ihren Nutzungsbeschränkungen befreit werden können.

Bei der Entwicklung eines Jailbreak-Werkzeugs werden zunächst Schwachstellen im Betriebssystem oder der Firmware des Gerätes gesucht, durch die eine Manipulation des Systems vorgenommen werden kann. Durch Ausnutzung der gefundenen Schwachstelle wird dann die Überprüfung der digitalen Signatur für das Betriebssystem und der ausführbaren Apps ausgehebelt und einige Basistools zur Erlangung von Root-Rechten installiert. Damit ist dann ein vollständiger Zugriff auf das System realisierbar. Dadurch wird die Möglichkeit geschaffen, Apps auch ohne digitale Signatur von Apple auf dem Gerät zu installieren und auszuführen. Weiterhin wird es auch möglich Dienste wie eine *Secure Shell* zu installieren oder tiefergehende Einstellungen am System vorzunehmen.

Ein Jailbreak ist durch die von der Community bereitgestellten Werkzeuge sehr einfach und praktisch von jedem Nutzer auch ohne Vorkenntnisse durchführbar. Ein Jailbreak-Vorgang wird meist noch mit der Installation des Cydia Paketmanagers abgeschlossen. Cydia ist ähnlich wie der original App Store ein Markplatz, auf denen Apps für gejailbreakte Geräte heruntergeladen und installiert werden können.

Die über Cydia installierten Apps unterstehen keiner Kontrolle durch Apple und können vollen Zugriff auf das System herstellen. Sie erlauben jedoch auch viele Möglichkeiten, die normale Apps aus dem offiziellen App Store aufgrund der Einschränkungen durch Apple nicht bieten können. Beliebt sind dabei Apps, die das Aussehen der Oberfläche anpassen oder Funktionen der Systemprogramme wie z.B. des SMS-Apps erweitern oder ersetzen können.

Die Implementierung, des in dieser Arbeit vorgestellten Konzeptes hat einen Jailbreak des Gerätes zur Voraussetzung, da Methoden verwendet werden müssen, die durch die Nutzungseinschränkungen von Apple auf den Geräten nicht erlaubt sind.

## 2.3. Dynamic Library Injection

Um die in Kapitel 1.3 beschriebenen Ziele zu realisieren, ist es erforderlich, tiefgreifend in das iOS Betriebssystem einzugreifen. Im Folgenden werden die dazu erforderlichen Mechanismen näher beschrieben.

### 2.3.1. Funktionsweise

Die *Dynamic Library Injection* ist eine Technik, mit der ausführbarer Code in den Adressraum eines anderen Prozesses eingeschleust und dieser Code zur Ausführung gebracht werden kann. Dem eingeschleusten Code ist es möglich die Abläufe des eigentlichen Prozesses zu verändern, in dem er z. B. Aufrufe an das Betriebssystem abfängt und verändert. Je nach Betriebssystem ist die Methode der Dynamic Library Injection unterschiedlich.

In iOS erfolgt die Library Injection über den *Dynamic Linker*. Der Dynamic Linker ist dafür verantwortlich, dynamische Bibliotheken, die von der jeweiligen App benötigt werden, in deren Adressraum hinzuzuladen. Wie aus Abbildung 2.1 hervor geht, wird der Dynamic Linker vom Systemkernel vor dem Start eines neuen Prozesses in dessen Adressraum geladen und ausgeführt. Der ursprüngliche Applikationscode wird dann mit den dazugeladenen Bibliotheken gebunden.

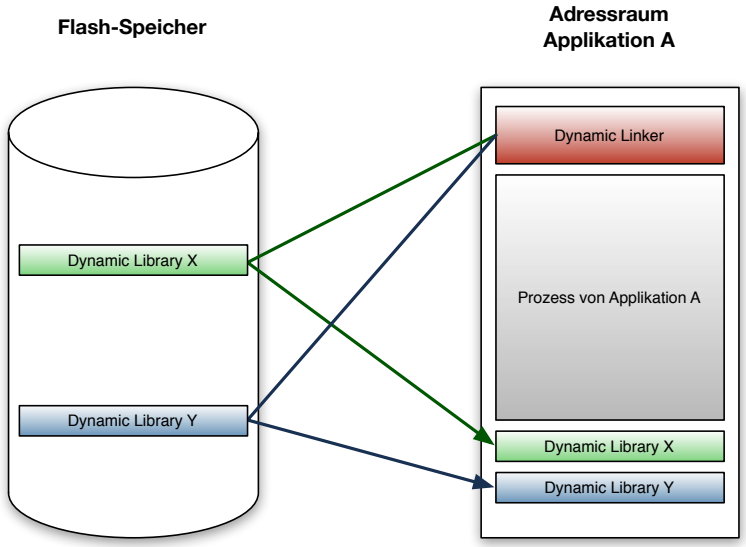


Abbildung 2.1.: Dynamic Linker Model in iOS

Während dieses Vorgangs kann die App auch mit dem Linker interagieren und ihn anweisen zusätzliche Bibliotheken in den Adressraum zu laden. Dies ist sogar noch später während der Laufzeit der eigentlichen App möglich.

Das Verhalten des Dynamic Linkers lässt sich aber auch von außerhalb der App beeinflussen. Dies geschieht durch Umgebungsvariablen, die den Dynamic Linker anweisen, bestimmte Aktionen zusätzlich oder verändert durchzuführen. Die Umgebungsvariablen werden oft benutzt um beispielsweise den Suchpfad für gemeinsam verwendete Bibliotheken zu verändern. Für die Dynamic Library Injection ist besonders die Umgebungsvariable `DYLD_INSERT_LIBRARIES` von Bedeutung. Diese Variable ermöglicht es, den Dynamic Linker von außerhalb der App anzuweisen, bestimmt Bibliotheken zusätzlich in den Adressraum zu laden. Durch das Setzen dieser Variable kann eine eigene Bibliothek in einem fremden Prozess eingeschleust werden. Über diese zusätzliche Bibliothek wird dann der Programmcode eingeschleust, der die zusätzlichen Sicherheitsroutinen bereitstellt.

### 2.3.2. MobileSubstrate

*MobileSubstrate*[6] ist ein Hooking-Framework für iOS das es mit einfachen Mitteln erlaubt eigene Bibliotheken in fremde Apps einzubringen und die Möglichkeit bietet, Methoden in der ursprünglichen App durch eigenen Methoden zu überschreiben. Es wurde von Jay Freeman (saurik) entwickelt und gehört zu den quasi Standard Jailbreak-Bibliotheken, die auf fast allen gejailbreakten Geräten installiert sind. Das Framework besteht aus zwei Hauptkomponenten, dem *MobileHooker* und dem *MobileLoader*.

Der MobileHooker ist für das Austauschen von vorhandenen Methodenimplementierungen durch eine eigene Implementierung verantwortlich. Die meisten Apps auf iOS sind in der Programmiersprache Objective-C implementiert. Zum Austauschen der Methodenimplementierungen verwendet der MobileHooker die dynamischen Laufzeitfähigkeiten der Sprache[7]. Die Methode `method_setImplementation` kann die Implementierung einer vorhandenen Methode zur Laufzeit ersetzen.

Außerdem wird auch das Austauschen von C/C++ Funktionen ermöglicht. Dazu wird die Technik des *Inline Function Hookings* verwendet, die in Abbildung 2.2 dargestellt ist. Am Funktionseinstieg der Ursprungsfunktion, dem Prolog, wird ein Sprungbefehl zur ausgetauschte Funktion aus der dynamischen Bibliothek eingefügt.

[6] Weitere Informationen: `http://svn.saurik.com/repos/menes/trunk/mobilesubstrate/`

[7] Objective-C Runtime Reference: `http://developer.apple.com/library/ios/#documentation/Cocoa/Reference/ObjCRuntimeRef/Reference/reference.html`

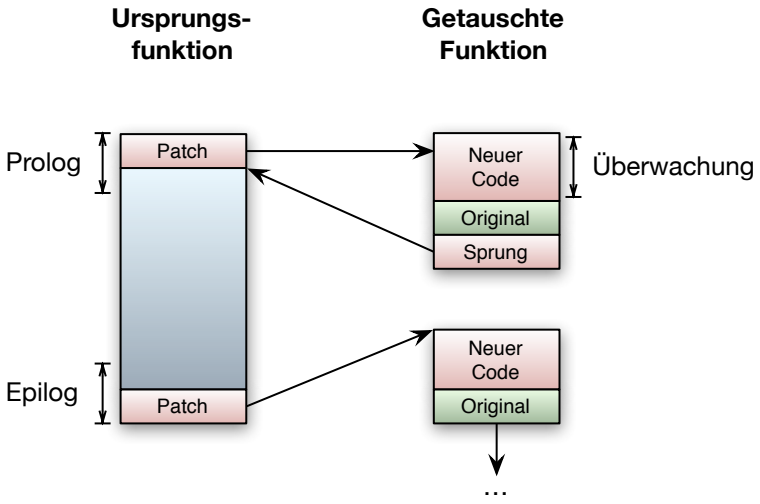


Abbildung 2.2.: Inline Function Hooking

Der dabei überschriebene Anfang der Ursprungsfunktion wird zuvor an eine andere Stelle des Hauptspeichers ausgelagert. Im Rahmen dieser Arbeit übernimmt die ausgetauschte Funktion die Überwachung des Ausrufs der Ursprungsfunktion um damit eine Zugriffskontrolle zu realisieren. Dabei kann entschieden werden, ob die Ursprungsfunktion ausgeführt werden soll oder der Aufruf an dieser Stelle beendet wird. Soll zur Ursprungsfunktion zurückgesprungen werden, so muss zuvor der ausgelagerte Anfang der Funktion ausgeführt werden. Am Ende der Ursprungsfunktion wird zur getauschten Funktion zurückgesprungen. Um den Schreibzugriff auf ausführbare Codestellen zum Einbringen des Sprungbefehls in die Ursprungsfunktion zu ermöglichen, muss der Betriebssystemkern verändert werden. Diese Änderung wird jedoch bereits vom Jailbreak, der Voraussetzung für MobileSubstrate ist, ausgeführt.

Der MobileLoader wird durch die in Kapitel 2.3.1 erwähnte Dynamic Linker Umgebungsvariable `DYLD_INSERT_LIBRARIES` in den Adressraum der Apps geladen. Dort lädt er alle in einem definierten Verzeichnis liegenden dynamischen Bibliotheken ebenfalls in den Adressraum und führt bei jeder dieser Bibliotheken einen speziellen Konstruktor aus, um sie vor dem Start den eigentlichen App zu initialisieren.

## 2.4. Gorilla

*Gorilla* ist eine von NESO Security Labs GmbH entwickelte *proof-of-concept* Sicherheitsapplikation für iOS. Es bietet bereits einige Sicherheitsfunktionen, wie einen Zugriffsschutz für das Adressbuch, das sichere Löschen von freiem Speicherplatz sowie das Fälschen der *Unique Device ID*. Auch eine Passwortprüfung der Benutzeraccounts des Systems, eine Änderung der Browser User-Agents sowie Sicherheitswarnungen beim Öffnen von PDF- und Microsoft Office Dokumente sind möglich.

Diese Einstellung kann dann auf ausgewählte Apps angewendet werden. Eine selektive Berechtigungsvergabe pro App ist jedoch nicht möglich. Aktuelle Versionen des iOS-Betriebssystems werden nicht unterstützt.

Im Rahmen dieser Arbeit wird die Gorilla-App erweitert, um diese Defizite zu beheben.

# 3. Anforderungsanalyse

Die Anforderungsanalyse dient dazu, Anforderungen an die zu implementierende Lösung zu ermitteln und somit später einen umfassenden Schutz des Nutzers zu ermöglichen. Dazu werden die Anforderungen anhand verschiedener Aspekte beschrieben. Der Abschnitt 3.1 geht dabei auf Aspekt des Schutzes der Privatsphäre des Benutzers ein. Im Abschnitt 3.2 werden Anforderungen zum Schutz vor dem Einschleusen von Schadcode beschrieben. Die Anforderungen an die Einfachheit der Konfiguration werden im Abschnitt 3.3 aufgestellt. Im letzten Abschnitt werden die Anforderungen im Hinblick auf die einfache Erweiterbarkeit um weitere Funktionen erläutert.

## 3.1. Schutz der Privatsphäre des Nutzers

Im Zentrum der Anforderungsanalyse steht der Schutz der Privatsphäre des Nutzers. Die zentrale Forderung ist hier: private Daten sollen nicht an Dritte weitergegeben werden können. Der effektivste Schutz der Daten ist, wenn die Apps auch wirklich nur auf die Daten zugreifen können, die sie benötigen. Bei allen anderen Daten soll der Zugriff verhindert werden. Zu den privaten Daten gehören verschiedene Bereiche, die individuell berücksichtigt werden müssen.

a. Zum einen gibt ein Gerät verschiedenste Daten preis, die zum Zweck der Identifizierung oder Wiedererkennung der Geräte und Nutzer verwendet werden können. Vorangegangene Untersuchungen ([Egele u. a. 2011] und [Smith 2010]) haben gezeigt, dass viele Apps diese gerätespezifischen Daten versenden. Oft werden sie zum Erstellen von Nutzungsstatistiken und zur Optimierung der Werbeanzeigen verwendet. Sowohl die Nutzungsstatistiken, als auch das Schalten der Werbung, wird in den aller meisten Fällen von Werbe- bzw. Tracking-Diensten übernommen und nicht vom Anbieter der Apps selbst. Dabei beherrschen eine nur sehr kleine Zahl von Dienstanbietern fast den kompletten Markt für mobile Werbung und mobile Nutzungsstatistiken. Somit besteht die Möglichkeit, dass diese Anbieter den Nutzer applikationsübergreifend wiedererkennen können. Dadurch kann ohne deren Wissen ein vollständiges Nutzungsprofil

der Anwender erstellt werden. Das Auslesen der Daten zur Identifizierung oder Wiedererkennung des Gerätes durch eine App muss also durch geeignete Maßnahmen verhindert werden.

b. Des Weiteren befinden sich auf den Geräten eine Menge personenbezogener und persönlicher Daten, die der Nutzer eingegeben hat. Dies können beispielsweise Adressen und Telefonnummer der Kontakte, Termine und Veranstaltungen aus dem Kalender oder SMS und Emails sein. Viele Apps benötigen diese Daten jedoch nicht um ordnungsgemäß zu funktionieren. Um das Risiko der Herausgabe dieser Daten zu minimieren sollte der Zugriff auf diese Daten verhindert werden können.

c. Schließlich entstehen auch personenbezogene und persönliche Daten, deren der Nutzer sich nicht bewusst ist. Die virtuelle Tastatur, die dem Nutzer auf dem Bildschirm zur Texteingabe geboten wird, hat die Funktionalität falsch geschriebene Wörter automatisch zu korrigieren. Dazu wird ein Wörterbuch genutzt. Dieses Wörterbuch wird aber auch durch neu eingegebene Wörter automatisch erweitert. Die neu hinzugefügten Wörter können private Informationen über den Nutzer enthalten, die er aber nicht öffentlich preisgeben will. Im Allgemeinen sollten daher die unbemerkt im Hintergrund entstehenden Daten ebenso geschützt werden, wie die vom Nutzer absichtlich in das System eingetragenen Daten.

## 3.2. Schutz des Systems vor dem Einschleusen von Schadcode

Trotz der Sicherheitsvorkehrungen, die das System wie in Kapitel 2.1 auf Seite 5 beschrieben bietet, werden immer wieder Möglichkeiten gefunden, Schadcode auf dem System einzuschleusen. Durch eine Ausnutzung von Sicherheitslücken der System-Apps kann der Angreifer sogar die Kontrolle über das System übernehmen, da diese Apps ohne Sandbox ausgeführt werden. Dies macht eine weitere Härtung des Systems nötig. Mitte 2011 war es durch eine Sicherheitslücke beim Öffnen von PDF Dateien sogar möglich, einen Jailbreak im Browser durchzuführen ([ZDNet 2011]). Dabei musst lediglich eine Webseite mit eingebetteter PDF-Datei aufgerufen werden.

Diese Schwachstelle hätte aber auch mit böswilligen Zielen ausgenutzt werden können. Solche sogenannte *Drive-By-Angriffe* werden durch das Verhalten des System

begünstigt. Viele Dokumente wie z. B. PDF- und Microsoft Office Dokumente werden im Browser und Email-Client direkt ohne weiter Nachfrage geöffnet und angezeigt. Ähnlich verhält es sich hier mit Video- und Audiodateien. Der Nutzer soll durch geeignete Maßnahmen vor diesen Gefahren geschützt werden. Gleichzeitig soll er aber nicht in seinen Möglichkeiten, die App zu benutzt zu sehr eingeschränkt werden.

Das Betriebssystem iOS bringt in der aktuellen Version auch einige interne Funktionalitäten zur Optimierung der Sicherheit des Systems mit. Diese Einstellungsmöglichkeiten sind dem Nutzer jedoch nicht immer gleich ersichtlich und werden zum Teil auch in den Tiefen des Einstellungsmenüs einfach nicht gefunden. Um die Sicherheit des Systems weiter zu optimieren, sollte der Nutzer auf unsichere Einstellungen hingewiesen werden. Außerdem sollte ihm ein einfacher Weg zur Lösung der Probleme aufgezeichnet werden.

## 3.3. Einfache Konfiguration

Alle diese Vorkehrungen sind für den Nutzer nur sinnvoll, wenn er diese auch mit angemessenem Aufwand selbst konfigurieren kann. Dafür sind folgende Anforderungen an die Benutzerfreundlichkeit und die Komplexität erforderlich.

Dem Nutzer soll auf übersichtliche Weise der Zustand seines System aufgezeigt werden. Dabei soll einfach zu erkennen sein, ob das System gerade sicher konfiguriert. Ist eine Handlung erforderlich, so soll der Benutzer darauf hingewiesen werden und bei der Durchführung unterstützt werden.

Das Konfigurieren der Sicherheitseinstellung soll transparent vonstatten gehen. Dem Nutzer soll klar sein welche Einstellung er für welche App getroffen hat. Die meisten der installierten Apps lassen sich mit wenigen einheitlichen Konfigurationen betreiben. Daher soll es dem Nutzer einfach möglich sein, ein Berechtigungsprofil anzulegen und dieses auf mehrere Apps anzuwenden. Dennoch kommt es vor, dass einzelne Apps unter Umständen eine leicht veränderte Konfiguration benötigen. Der Benutzer möchte z. B. einer VoIP-App den Zugriff auf das Adressbuch und somit den Telefonnummern erlauben, aber für den Terminkalender die Regeln wie für andere Apps anwenden. Zu diesem Zweck soll es dem Nutzer durch eine einfache und übersichtliche Darstellungsweise möglich sein, Ausnahmen für einzelne Konfigurationselemente zu definieren.

Nach der ersten Installation und Konfiguration des Sicherheitssystems kann bei den Nutzern das Sicherheitsinteresse wieder in den Hintergrund geraten, da das funktionierende System nur wenige Eingriffe des Benutzers erfordert. Beim Installieren einer neuen App kann das Konfigurieren der Sicherheitseinstellungen daher für einige Zeit vergessen werden. Der Nutzer muss hierbei weiter unterstützt werden. Er sollte einen Hinweis konfigurieren können, die ihm vor dem ersten Start einer neuen App an das Konfigurieren der Einstellungen erinnert.

Viele Nutzer haben zudem eine Vielzahl von Apps auf ihrem Gerät installiert. Zu diesem Zweck sollte der Zugriff auf die Konfiguration einer App durch eine Suchfunktion erleichtert werden.

## 3.4. Einfache Erweiterbarkeit

Neben den Anforderungen, die an die Funktionalität und die Benutzerfreundlichkeit gestellt werden, gibt es auch einige Anforderungen an die Architektur zu berücksichtigen.

Das Betriebssystem und dessen Funktionen erweitern und verändern sich mit jeder neuen Version. Durch das Einbringen und Verändern der Funktionalitäten werden in Zukunft weiter Aspekte in die Sicherheitsapplikation mit einbezogen werden müssen. Auch die Methoden der Angreifer und die auftretenden Sicherheitsprobleme in den Apps verändern sich mit der Zeit.

Daher soll eine einfache Erweiterbarkeit der Sicherheitsapplikation erreicht werden. Diese einfache Erweiterbarkeit soll neben der Erweiterung der Funktionalität auch die Erweiterbarkeit der Konfigurationsoberfläche betreffen, da der Nutzer die neuen Funktionen auch konfigurieren können muss. Das schließt auch der in Kapitel 3.2 auf Seite 13 geforderte Hinweis auf unsichere Systemeinstellungen mit ein.

# 4. Verwandte Arbeiten

Es existieren bereits einige Sicherheitsapplikationen auf mobilen Plattformen, die ein ähnliches Ziel verfolgen, wie das in dieser Arbeit entworfene Konzept. Einige dieser Sicherheitsapplikationen werden in den Abschnitten 4.1, 4.2 und 4.3 genauer vorgestellt. Das Kapitel wird in Abschnitt 4.4 von einer kurzen Zusammenfassung abgerundet.

## 4.1. WhisperCore

WhisperCore[8] ist eine von WhisperSystems entwickelte Sicherheitsplattform für das Betriebssystem Android. WhisperCore beinhaltet dabei eine Vielzahl von Sicherheitsfunktionen. Bis vor Kurzem konnte es kostenlos von der Homepage des Herstellers heruntergeladen werden. Seit des Aufkaufs des Herstellers durch eine andere Firma Ende des Jahres 2011 ist die Zukunft des Produkts jedoch ungewiss.

Android enthält, im Gegensatz zu iOS, schon im Betriebssystem ein Berechtigungssystem für Applikationen. Allerdings erlaubt es dieses System nur, die geforderten Berechtigungen einer Applikation anzusehen. Auf der Grundlage dieser Informationen kann den Benutzer dann entscheiden, ob er diese Applikation installieren und ausführen möchte oder nicht. Eine Zulassung einiger Berechtigungen bei gleichzeitiger Verweigerung anderer Berechtigungen ist nicht möglich.

Mit der Hilfe von WhisperCore ist es jedoch möglich genau diese selektive Auswahl dieser Berechtigungen zu erreichen. Das gelingt WhisperCore, in dem es einen eigenen Betriebssystemkern mitbringt. Der normalerweise installierte Betriebssystemkern muss durch diesen ersetzt werden. Die Modifikation ermöglicht Systemzugriffe, die im Berechtigungssystem einer Berechtigung zugewiesen sind, abzufangen und zu blockieren.

Durch die Anpassung des Betriebssystemkerns ist es möglich, eine Reihe weiterer Funktionen anzubieten. WhisperCore erlaubt die Datenpartition und die eingelegte

[8] Weitere Informationen: `http://www.whispersys.com/whispercore.html`

Speicherkarte des Gerätes zu verschlüsseln. Die Entschlüsselung der Daten erfolgt dabei transparent mit dem beim Start des Gerätes einzugebenden Passwort.

Weiterhin ist in WhisperCore eine App-Firewall integriert. Damit können die Netzwerkverbindungen der Apps überwacht und durch vom Nutzer festgelegte Regeln unterbunden oder erlaubt werden. Auch ein Backup-Programm, mit dem sich verschlüsselte Sicherungen des Systems auf Servern im Internet anlegen lassen, ist integriert.

Dieser tiefe Eingriff in das System hat gleichwohl auch Nachteile. Beim Einschalten eines Android-Gerätes wird zunächst der Bootloader aktiv. Dieser ist ein fest integrierter Bestandteil des Gerätes und ist dafür zuständig das eigentliche Android-Betriebssystem zu starten. Zuvor wird jedoch mit Hilfe einer digitalen Signatur überprüft, ob ein vom Hersteller freigegebenes Betriebssystem verfügbar ist. Da WhisperCore jedoch den Betriebssystemkern modifizieren muss und dieser somit einen anderen digitalen Fingerabdruck aufweist als der normalerweise installierte Kern, kann WhisperCore nicht ohne weiteres installiert werden. Die meisten Hersteller erlauben das Starten von selbst installierten Betriebssystemen nicht. Das Verändern und Entsperren des Bootloaders zieht jedoch auch einen Verlust der Herstellergarantie für das Gerät mit sich. Darüber hinaus ist das Gerät damit von offiziellen Updates des Betriebssystems ausgeschlossen bis ebenfalls für WhisperCore eine modifizierte Version des neuen Betriebssystemkerns verfügbar wird. Wegen des verzögerten Updates ist das System eventuell länger für Sicherheitslücken anfällig, die durch das Updaten behoben werden könnten.

Die Installation wird aufgrund der Komplexität von einem Installationsassistenten begleitet. Offiziell werden dabei jedoch nur zwei Android Geräte (Google Nexus One und Google Nexus S) unterstützt. Eine Funktion, den alten Systemzustand vor der Installation wiederherzustellen findet sich nicht.

## 4.2. AppFence

AppFence[9] ist ebenfalls eine Sicherheitsplattform für das Betriebssystem Android. Es wurde von Forschern der University of Washington und Microsoft Research entwickelt ([Hornyack u. a. 2011]).

Auch AppFence versucht die Privatsphäre des Nutzers durch Beschränkung der Rechte der Apps zu schützen. Der Ansatz von AppFence geht dabei jedoch weit

[9] Weitere Informationen: `http://appfence.org/`

über das Berechtigungssystem der Android-Plattform hinaus und ist daher auch dem Berechtigungssystem von WhisperCore überlegen. AppFence bietet fein granulare Möglichkeiten die Berechtigungen der Applikationen zu steuern. Einzelne Elemente wie z. B. den Zugriff auf die Geräte-Seriennummer oder die Telefonnummer können einzeln konfiguriert werden. Darüber hinaus können auch falsche Angaben an die Applikation durchgereicht werden. Anstatt der Seriennummer des Geräts kann z. B. eine zufällig generierte Nummer an die Applikation weitergegeben werden. Im Gegenzug zur kompletten Verweigerung des Zugriffs auf die Information erhält die Applikation bei dieser Methode eine gültige Antwort zurück. Die Methode verhält sich also aus Sicht der Applikation wie erwartet. Dass es sich bei den Daten um gefälschte Informationen handelt ist für die Applikation nicht erkennbar.

Genau wie WhisperCore setzt auch AppFence eine modifizierte Version des Betriebssystemkerns voraus um die Funktionalitäten umzusetzen. Daher gelten die gleichen Probleme und Einschränkungen die auch für WhisperCore in Kapitel 4.1 auf Seite 16 beschrieben wurden.

Die Installation von AppFence ist ebenso kompliziert wie die von WhisperCore. Sie wird jedoch nicht von einem Installationsassistent begleitet, so dass der Nutzer sich selbst das dazu nötige Wissen aneignen muss. AppFence unterstützt dabei nur die Installation auf dem Gerät Google Nexus One.

## 4.3. App Permission Watcher

App Permission Watcher[10] ist eine Sicherheitsapplikation für das Betriebssystem Android. Sie wurde im Rahmen einer Abschlussarbeit an der Ruhr-Universität Bochum und der Universität Duisburg-Essen entwickelt.

Im Gegensatz zu den bisher vorgestellten Applikationen greift diese Applikation nicht in den Ablauf anderer Applikationen ein. Es werden lediglich die Berechtigungen systemfremder Apps analysiert und verdächtige Berechtigungskombinationen erkannt. Der Nutzer erhält eine Warnung vor diesen Applikationen.

Der Nutzer wird also nicht vor den Applikationen geschützt. Er wird lediglich auf eine verdächtige Applikation hingewiesen. Die Applikation überprüft jedoch nur, ob eine andere Applikation grundsätzlich bestimmte Berechtigungen besitzt. Ob die Applikation diese Berechtigung wirklich verwendet wird nicht überprüft.

[10] Weitere Informationen: `http://www.apewatch.de/`

Da diese Applikation nicht in andere Applikation eingreift wird auch kein modifizierter Betriebssystemkern benötigt. Die Applikation kann über den offiziellen Android Market heruntergeladen und auf allen Android-Geräten installiert werden.

## 4.4. Zusammenfassung und Bezug

Die dargestellten verwandten Arbeiten haben alle das Ziel, die Sicherheit des Geräts oder die Privatsphäre des Nutzers besser zu schützen. Alle hier aufgeführten verwandten Arbeiten sind für das Betriebssystem Android entwickelt worden. Für iOS sind keine verwandten Arbeiten oder Lösungen bekannt. Die in dieser Arbeit entwickelte Implementierung ist daher die erste umfassende Lösung für dieses Betriebssystem.

WhisperCore erweitert das Berechtigungssystem von Android um die Flexibilität, die Berechtigungen selektiv zu erteilen. Zusätzlich bietet es die Möglichkeit ein- und ausgehende Netzwerkverbindungen zu überwachen und das Dateisystem zu verschlüsseln. Die Implementierung setzt auf der Ebene des Systemkernels an. Aufgrund des fehlenden Berechtigungssystems in iOS kann WhisperCore nicht als Vorbild für diese Arbeit dienen. Auch die Implementierung auf der Ebene des Systemkernels gestaltet, wie später in Kapitel 5.1 auf Seite 20 beschrieben wird, schwierig. Die weiteren Funktionen von WhisperCore sind außerhalb des Rahmens dieser Arbeit.

AppFence bietet noch feingranularere Möglichkeiten den Zugriff der Applikation zu steuern. Es bringt ein eigenes Berechtigungssystem mit, bei dem jedes einzelne Element, wie beispielsweise der Zugriff auf die Geräte-Seriennummer oder die Telefonnummer einzeln konfigurierbar sind. Auch AppFence setzt bei der Implementierung auf der Ebene des Systemkernels an. Das eigene Berechtigungssystem kann analog in der Lösung dieser Arbeit umgesetzt werden. Die Implementierung auf Ebene des Systemkernels gestaltet sich in iOS jedoch schwierig.

App Permission Watcher geht einen komplett anderen Weg. Die Applikation greift nicht in den Ablauf anderer Applikationen ein. Stattdessen werden die angeforderten Berechtigungen anderer Apps analysiert und verdächtige Kombinationen an den Nutzer gemeldet. Für App Permssion Watcher wird keine Modifikation im Systemkernel benötigt. Aufgrund des fehlenden Berechtigungssystems kann diese Form der Analyse in iOS nicht umgesetzt werden. Die Ziele dieser Arbeit können außerdem nicht mit den Methoden des App Permission Watchers erreicht werden.

# 5. Konzept

Wie in den Grundlagen bereits erörtert wurde, ist es erforderlich, bestimmte Methodenaufrufe abzufangen. Außerdem ist es nötig Dateizugriffe abzufangen oder umzuleiten. Dazu werden in Abschnitt 5.1 mehrere unterschiedliche Ansätze beschrieben und bewertet. In Abschnitt 5.2 wird das Konzept zum Schutz der Privatsphäre des Nutzers anhand der verschiedenen Datenquellen präsentiert. Anschließend ist in Abschnitt 5.3 das Konzept zur Verhinderung des Einschleusens von Schadcode beschrieben. Der Abschnitt 5.4 erläutert das Konzept, das eine einfache Konfiguration ermöglicht. Das Kapitel wird in Abschnitt 5.5 von der Beschreibung des Konzepts zur einfachen Erweiterbarkeit abgerundet.

## 5.1. Methoden

Um Methodenaufrufe von Apps (beispielsweise zum Auslesen des Adressbuchs) abzufangen, existieren unterschiedliche Ansätze. Die Auswahl des Ansatzes muss sich jedoch an den Gegebenheiten der Plattform orientieren.

Die Systemarchitektur des Betriebssystems iOS besteht aus mehreren Sichten. Die Abbildung 5.1 auf Seite 21 stellt diese Schichten dar. Auf der obersten Schicht befinden sich die Apps, mit denen der Nutzer interagiert. Diese Apps bauen auf den Funktionalitäten auf, die von den Gerätebibliotheken und den Systembibliotheken bereitgestellt werden. Dafür sind einige Schnittstellen definiert, die die Apps verwenden können. Diese Schnittstellen werden als *Public-API* bezeichnet. Daneben existieren noch weitere Schnittstellen. Diese dürfen, laut den Vorgaben von Apple, in Apps, die über den App Store angeboten werden, nicht verwendet werden. Diese Schnittstellen werden als *Private-API* bezeichnet. Möchte eine App z. B. auf das Adressbuch des Gerätes zugreifen, können die entsprechenden Public-API Methoden aus den Gerätebibliotheken verwendet werden. Die verwendete Gerätebibliothek öffnet dazu die Datei im Dateisystem, in der die Adressbuchdaten abgespeichert sind. Dazu verwendete die Gerätebibliothek Dateizugriffsmethoden, die durch die Systembibliotheken bereitgestellt werden. Die Systembibliotheken bilden die tiefste Schicht

der Architektur vor dem Betriebssystemkernel. Um ihre Aufgaben zu erfüllen, kommunizieren die Systembibliotheken über die sogenannten *System Calls* direkt mit dem Kernel. Im Adressbuch-Beispiel verarbeitet der Kernel den Dateisystemzugriff und steuert dazu die Speicherhardware des Gerätes. Sind die erforderlichen Daten geladen, werden diese an die Systembibliotheken zurückgegeben. Die Systembibliothek meldet der Gerätebibliothek, dass die erforderlichen Daten geladen wurden. Die Gerätebibliothek wertet die Daten anschließend aus und liefert der App die geforderten Adressbucheinträge zurück.

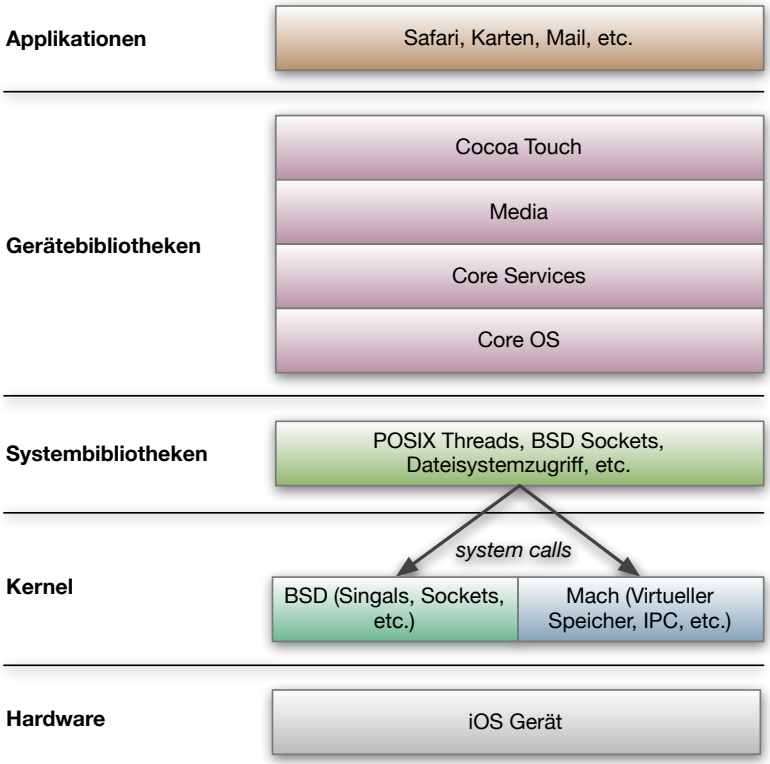


Abbildung 5.1.: iOS System Architektur

Im Folgenden werden mehrere unterschiedliche Ansätze zum Abfangen von Methodenaufrufen und Dateisystemzugriffen beschrieben und bewertet.

**Kernel Patching**

Eine Möglichkeit besteht darin die Aufrufe auf Kernel-Ebene abzufangen. Diese Methode wird häufig in Sicherheitsanwendungen wie Virenscanner auf Desktop Computern verwendet. Zu diesem Zweck ist es notwendig, ein Programm in den Kernel einzuschleusen. Dieses Programm greift dann an die entsprechenden Stellen innerhalb des Kernels in den Ablauf ein. Somit kann der Ablauf eines Aufrufs verändert

werden. Die Sicherheitsroutinen werden dabei meist über einen Treiber in den Betriebssystemkern geladen. Falls der Quellcode des Betriebssystems vorliegt können die Sicherheitsroutinen auch direkt in den Betriebssystemkern integriert werden. Die im Kapitel 4.1 und 4.2 beschriebenen Arbeiten WhisperCore und AppFence verwenden diesen Ansatz. Dieser Ansatz erfordert allerdings Kenntnisse über die Schnittstellen und Abläufe innerhalb des Kernels. In Android kann dies leicht umgesetzt werden, da eine modifizierte Version von *Linux* als Betriebssystemkern verwendet wird und somit der Quellcode frei zugänglich ist. Das Betriebssystem iOS hingegen baut auf das Desktop Betriebssystem Mac OS X auf. Obwohl ein großer Teil des Kernels von Mac OS X als Quellcode veröffentlicht wird, sind die Modifikationen, die für iOS durchgeführt wurden nicht öffentlich zugänglich. Ebenso existiert keine Treiberdokumentation oder sonstige Dokumente für iOS, die Rückschlüsse auf die Veränderungen zu Mac OS X ziehen lassen. Es ist jedoch zu vermuten, dass neben der Portierung auf eine andere Prozessorarchitektur, ähnlich wie bei Android noch viele weitere Veränderungen vorgenommen werden mussten um die Eigenheiten der neuen Geräteklasse berücksichtigen zu können. Um die benötigten Informationen zu erhalten ist eine größere *reverse-engineering* Untersuchung des iOS-Kernels erforderlich. Diese würde den Rahmen dieser Arbeit sprengen. Die nichtöffentliche Schnittstellendokumentation führt aber auch zu einem weiteren Problem. Apple hat die volle Kontrolle über die im iOS-Kernel verwendeten Treiber und Programme. Diese können nur von Apple selbst oder mit Hilfe von Mitarbeitern von Apple entwickelt werden. Dadurch wird es für Apple einfach, die nichtöffentlichen Schnittstellen innerhalb des Betriebssystems zu verändern oder zu optimieren. Bei jedem Update könnten sich also die Schnittstellen ändern und die reverse-engineering Untersuchung müsste erneut stattfinden. Aufgrund der genannten Probleme ist die Vorgehensweise für diese Arbeit nicht geeignet.

**System Call Hooking**

Eine weitere Möglichkeit besteht darin die Übergänge zwischen den Bibliotheken und dem Kernel zu überwachen. Systembibliotheken können mit dem Kernel über sogenannte *System Calls* kommunizieren. Die System Calls bieten die Möglichkeit, die im Kernel bereitgestellten Funktionalitäten aufzurufen. Durch das Überwachen der System Calls ist es möglich, bestimmte Aufrufe zu identifizieren und diese abzufangen. Dazu ist jedoch eine Manipulation des Kernels oder der aufrufenden App nötig. Eine Manipulation des Kernels wird durch die fehlende Dokumentation des Kernels erschwert. Die Anzahl der verschiedenen System Call Funktionen ist gering, die genaue Auswirkung eines Aufrufs definiert sich dabei mehr über die übergebenen Parameterdaten. Die Überwachung ist daher sehr aufwendig, da erst aufgrund der

übergebenen Parameter auf die volle Semantik des Aufrufs geschlossen werden kann. Gleichzeitig ist die Aufrufhäufigkeit von System Calls sehr hoch, da Hintergrundprozesse des Betriebssystems selbst über System Calls mit dem Betriebssystemkern interagieren. Geräten auf denen iOS als Betriebssystem eingesetzt wird sind jedoch mobile Geräte die nur über begrenzte Systemressourcen und eine begrenzte Stromversorgung verfügen. Eine Überwachung der System Calls scheint daher im Hinblick auf die Akkulaufzeit und die Rechenleistung nicht sehr effizient zu sein. Daher ist diese Vorgehensweise ebenfalls nicht geeignet.

**Binary Patching**

Eine dritte Möglichkeit besteht darin, die installierten Apps zu modifizieren. Dazu müssen die installierten Apps automatisch auf den Aufruf bestimmter Methoden untersucht werden. Die gefunden Aufrufe müssen dann durch andere Aufrufe, die Funktionen der Sicherheitsapplikation verwenden, ersetzt werden. Durch diese Vorgehensweise kann der Aufruf direkt beim Aufrufer abgefangen werden. Es ist besser möglich, nur auf spezielle Aufrufe zu reagieren und so Systemressourcen und Akkuleistung zu sparen. Ein Problem stellt dabei die Signaturprüfung dar. Das System überprüft vor jedem Start einer App deren digitale Signatur. Eine Änderung des Codebereichs der App zieht natürlich auch eine Änderung der Prüfsumme der App mit sich. Die Signatur ist damit ungültig. Mit Hilfe des *pseudo-signings* könnte, laut [Freeman 2008], die Signaturprüfung jedoch auch nach dem Patchen erfolgreich durchgeführt werden. Allerdings geht bei einem App-Update die Schutzfunktion verloren, da die App mit einer neuen Version überschrieben wird. Der Analyse- und Änderungsprozess muss danach erneut ausgeführt werden. Die Updatefrequenz von Apps ist allerdings so hoch, dass sich auch diese Vorgehensweise nicht eignet.

**Dynamic Library Injection**

Der in dieser Arbeit favorisierte Ansatz basiert auf der in Abschnitt 2.3 auf Seite 8 vorgestellten *Dynamic Library Injection*. Die relevanten Aufrufe werden von einer zur Laufzeit in die App eingebrachten Bibliothek abgefangen und gegebenenfalls abgeblockt. Dabei werden Zugriffe auf offizielle API-Methoden untersucht. Gegenüber der Implementierung auf Kernel-Ebene hat diese Methode den Vorteil, dass diese *Public-API*-Methoden über die Zeit konstant bleiben. Die überwachte App liegt unverändert im Dateisystem, so dass die Signaturprüfung ebenfalls kein Problem darstellt. Diese Herangehensweise birgt jedoch das Risiko, dass andere Apps, die durch den Jailbreak mit Root-Rechten ausgeführt werden, die Sicherheitskomponenten umgehen oder deaktivieren können. Daher ist ein größeres Augenmerk auf die Härtung des Systems zu legen. Außerdem wird versucht die relevanten Aufrufe an der tiefsten Stelle in der Applikationsarchitektur abzufangen. Dadurch kann die

größtmögliche Abdeckung der relevanten Aufrufe erreicht werden. Die Ansatzpunkte befinden sich daher auf Höhe der Gerätebibliotheken und Systembibliotheken wie in Abbildung 5.1 auf Seite 21 dargestellt. Es ist die effizienteste Methode, um die im folgenden Konzept beschriebene Vorgehensweise umzusetzen.

Für die Umsetzung wird zunächst eine Liste mit relevanten Systemmethoden erstellt. Auf Grundlage dieser Liste wird eine dynamische Bibliothek implementiert. Diese Bibliothek besitzt für jede relevante Systemmethode eine kompatible Methode, die die gleichen Argumente annehmen und die gleichen Rückgabewerte liefern kann. Beim Einbringen der Bibliothek in einen Prozess werden dann die vorhandenen Systemmethoden durch die in der Bibliothek definierten Methoden ersetzt.

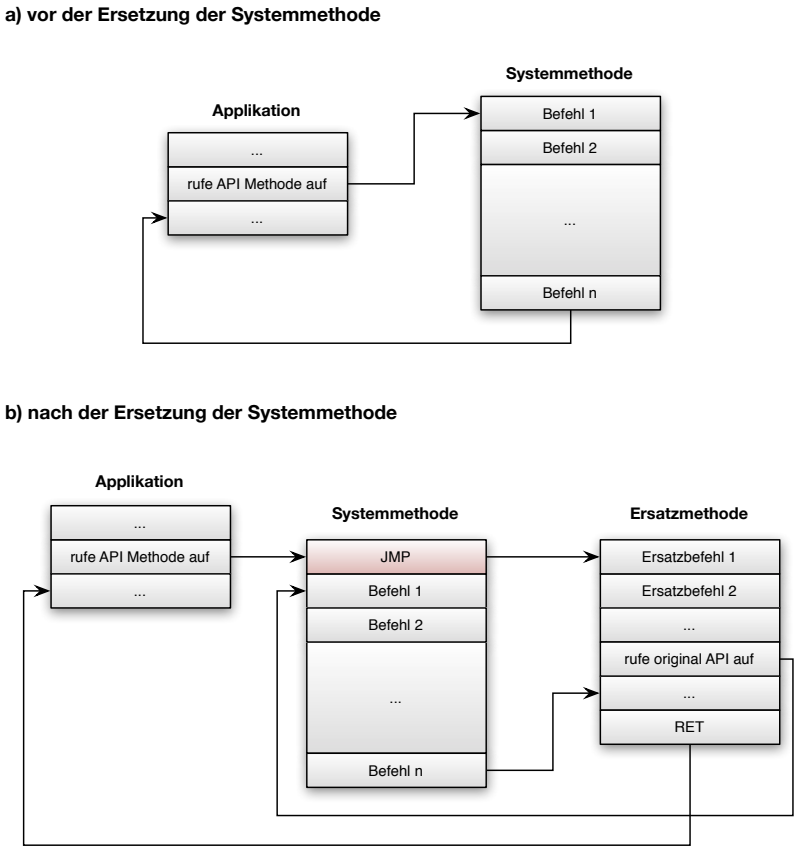


Abbildung 5.2.: Ersetzung von Systemmethoden

In Abbildung 5.2 wird dieser Vorgang beschrieben. In Teil a) der Abbildung ist einer normaler Aufruf einer Systemmethode dargestellt. Die App ruft die Systemmethode auf. Die Methode wird ausgeführt und anschließend wird zur App zurückgesprungen. Durch das Ersetzen der Systemmethode in Teil b) der Abbildung wird beim Aufruf

dieser Systemmethoden durch die App nicht die originale Systemmethode verwendet. Stattdessen wird der Aufruf an die in der Bibliothek definierten Ersatzmethode umgeleitet. Diese Methode kann dann entscheiden, ob die originale Systemmethode dennoch aufgerufen werden soll. Vor dem Aufruf der originalen Systemmethode können die Argumente verändert werden und nach dem Aufruf die Rückgabewerte angepasst werden. Dadurch ist es möglich zu schützende Daten vor dem Rücksprung zum ursprünglichen Aufrufer verändern oder aus den Datenstrukturen entfernen zu können.

Zur Überwachung der Dateisystemzugriffe wird ebenfalls zuerst eine Liste mit den relevanten Dateien und Pfaden erstellt. Das Abfangen der Dateisystemzugriffe kann durch Überwachen des Aufrufs der Funktion `open` realisiert werden. Diese Funktion ist in der *Standard C Library* definiert und bildet den tiefsten Einstiegspunkt für Zugriffe auf Dateien. Durch das Abfangen dieser Methodenaufrufe können somit auch Dateisystemzugriffe von den System-Frameworks überwacht werden. Beim Eintritt in die Ersatzmethode wird zunächst der angegebene Dateiname überprüft. Falls es sich bei der Datei um eine zu schützende Datei handelt, wird der Aufruf abgebrochen oder auf eine anonymisierte Version der Datei (eine Schattenkopie) umgeleitet. Falls es sich nicht um eine zu schützende Datei handelt wird der Aufruf unverändert an die originale Systemmethode weiter gegeben und ausgeführt. Ein Abfangen ermöglicht somit die Überwachung aller Dateisystemzugriffe einer App.

## 5.2. Schutz der Privatsphäre des Nutzers

Um den Schutz der Privatsphäre des Nutzers sicherzustellen, müssen zuerst die Quellen dieser Daten ermittelt werden. Dabei muss auch entschieden werden, auf welche Weise die Daten zu schützen sind. Auf manche Daten kann der Zugriff verweigert werden. Andere Daten werden jedoch auf Grund der Funktionsweise der Apps benötigt, müssen jedoch nicht die echten Daten des Nutzers sein. In diesen Fällen werden simulierte Datenstrukturen (Schattenkopien) zurückgeliefert, die zwar im Aufbau dem erwarteten Rückgabewert entsprechen, aber keine persönlichen Nutzerdaten enthalten (beispielsweise eine leere SQLite-Datenbank mit Kontakten).

Die Veröffentlichung [Seriot 2010] beschreibt die Gefahr für die privaten Daten, die von den installierten Apps ausgeht. In der Untersuchung wurden auch bereits einige Quellen privater Daten identifiziert und verschiedene Szenarien zur bösartigen Ausnutzung dieser Daten beschrieben. Auf der Grundlage dieser Untersuchungen

werden die wichtigsten zu schützenden Daten ausgewählt. Da die Untersuchung bereits im Jahr 2010 stattgefunden hat, müssen an einigen Stellen noch weitere Quellen privater Daten ergänzt werden. Andere, wie z. B. das Auslesen der Telefonnummer des Geräts, sind in der Zwischenzeit schon von Apple selbst verbessert worden. Eine vollständigere Liste der möglichen Quellen für private Daten befindet sich im Anhang der Arbeit (Anhang A.1). Die folgende Quellen wurden für diese Arbeit ausgewählt.

### 5.2.1. Unique Device Identifier und WLAN MAC-Adresse

Der Unique Device Identifier (UDID) kann zur Identifizierung und Wiedererkennung des Gerätes verwendet werden. Zahlreiche Studien ([Smith 2010], [Egele u. a. 2011]) belegen bereits, dass dieses Gerätemerkmal oft an Dritte übertragen wird. Der Zugriff auf die UDID erfolgt mit der Methode `uniqueIdentifier` der Klasse `UIDevice`, die Apple in der offiziellen API bereitstellt. Mit der Aktualisierung auf iOS Version 5.0 im Oktober 2011 wurde Zugriff auf die UDID von Apple als unerwünscht erklärt und soll in einer späteren Version des Betriebssystem nicht mehr verfügbar sein ([Apple, Inc. 2011b]). Da Apple jedoch mittlerweile eine ganze Reihe an Methoden für unerwünscht erklärt und diese noch nicht entfernt hat, ist davon auszugehen, dass auch die Methode zum Zugriff auf die UDID noch eine ganz Weile im System verfügbar bleibt. Dennoch wurden als Reaktion die ersten Bibliotheken veröffentlicht, die ebenfalls zum Zwecke der Identifizierung und Wiedererkennung alternative Gerätemerkmale (wie beispielsweise die MAC-Adresse der Wireless LAN-Karte) verwenden[11]. Weitere Wege zur Ermittlung der UDID sind nicht bekannt.

Zur Ermittlung der Zugriffswege auf die MAC-Adresse wurden verschiedenen Apps, die im App Store zur Anzeige der MAC-Adresse angeboten werden untersucht. Dabei wurde eine selbst entwickelte Bibliothek, die mit Hilfe des in Kapitel 2.3.1 auf Seite 8 vorgestellten Verfahrens der Dynamic Library Injection in die Apps eingeschleust wurde, benutzt. Das Betriebssystem iOS baut im Kern auf Komponenten des Desktop-Betriebssystems Mac OS X auf. Daher wurden einige Systemmethoden, die auch im Desktop-Betriebssystem Mac OS X zum Auslesen der MAC-Adresse verwendet werden, ersetzt und das Verhalten der Apps analysiert.

Die Untersuchung konnte für den Zugriff auf die MAC-Adresse des WLAN-Gerätes drei Wege über Betriebssystemzugriffe durch Systembibliotheken identifizieren. Bei den betreffenden Funktionen handelt es sich um `ioctl`, `sysctl` und `getifaddrs`, die dabei jeweils mit speziellen Parametern aufgerufen werden. Diese Funktionen

[11] Vgl. `https://github.com/gekitz/UIDevice-with-UniqueIdentifier-for-iOS-5/`

werden auch in den höheren Schichten der Betriebssystembibliotheken, wie dem *CFNetwork Framework* verwendet. Außerdem wurde eine Datei gefunden, in der die MAC-Adresse und weitere Netzwerkeinstellungen zwischengespeichert sind. Diese Datei befindet sich im Dateisystem des Gerätes und kann von jeder App, trotz der App-Sandbox, lesend erreicht werden und ist somit ebenfalls als Quelle für die MAC-Adresse geeignet. Für alle getesteten Apps konnte mit dieser Methode der Zugriff auf die MAC-Adresse nachgewiesen werden. Da für den Betriebssystemkern von iOS keine Dokumentation vorliegt, ist es jedoch nicht völlig auszuschließen, dass es weitere Zugriffsmöglichkeiten gibt.

Auch die Art der Zugriffskontrolle auf diese Daten gestaltet sich etwas komplizierter. Es müssen mehrere Fälle berücksichtigt werden. Einige Apps benötigen den Zugriff auf den UDID oder die MAC-Adresse. Der Zugriff kann daher nicht einfach verweigert werden, sondern muss speziell gesteuert sein. Apps wie z. B. WhatsApp[12], ein sehr weit verbreitetes Sofortnachrichtenprogramm, verwendet die UDID zur Authentifizierung auf deren Servern. Daher werden zwei Zugriffsmodi auf den UDID und die WLAN MAC-Adresse implementiert. Der erste Modus erzeugt bei jedem Zugriff auf die Information einen zufälligen Wert, der zu dem Format der angeforderten Information passend ist. Dieser Modus kann für die meisten Apps verwendet werden. Eine App, die den UDID oder WLAN MAC-Adresse zur Authentifizierung verwendet, sollte mit dem zweiten Modus betrieben werden. Dabei wird einmalig eine zufällige Version der Information erstellt und diese zukünftig verwendet. Die Apps können damit weiter wie bisher verwendet werden. Der Benutzer kann zwar nach wie vor wiedererkannt werden, das applikationsübergreifende Tracking des Benutzers ist jedoch mit dieser Methode nicht mehr möglich.

### 5.2.2. Adressbuch

Das Adressbuch enthält viele vom Benutzer eingegebene private Daten und Informationen. Der Zugriff der Apps erfolgt über ein spezielles Framework, welches vom Betriebssystem zur Verfügung gestellt wird. Das *Address Book Framework* stellt Methoden zur Verfügung, mit der eine App lesend und schreiben auf das Adressbuch zugreifen kann. Das Framework kann von jeder App verwendet werden. Es findet keine Zugriffskontrolle durch das System statt. Das vollständige Auslesen oder Ändern des Adressbuchs ist somit ohne weiteres möglich.

In einem besonders aktuellem Fall, im Februar des Jahres 2012, ist einem Entwickler eine App aufgefallen, die den kompletten Inhalt des Adressbuchs an einen Server

[12] Weitere Informationen: `http://www.whatsapp.com/`

überträgt ([Thampi 2012]). Laut den Ausführungen in [AllThingsD 2012] hat Apple daraufhin angekündigt, in einer zukünftigen Version des Betriebssystems das Adressbuch, ähnlich wie bereits den Ortungsdienst, vor unbefugtem Zugriff zu schützen. In welcher Version des Betriebssystems dies geschehen soll ist nicht bekannt. Daher bleibt der Schutz des Adressbuchs weiterhin Gegenstand dieser Arbeit.

Durch Untersuchungen mit der Bibliothek, die auch zur Ermittlung der Zugriffswege auf die MAC-Adresse verwendet wurde, konnte die Datei im Dateisystem ausgemacht werden, in der das Adressbuch gespeichert wird (`/private/var/mobile/Library/AddressBook/AddressBook.sqlitedb`). Es handelt sich dabei um eine SQLite-Datenbank. Die Datei wird bei den Zugriffen über das Address Book Framework direkt geöffnet und bearbeitet. Allerdings können Apps auch selbst Zugriff auf die Datei nehmen und diese zum Lesen und Schreiben öffnen. Dieser Weg ist in der offiziellen Dokumentation nicht beschrieben. Dennoch ist es möglich, dass eine App diesen Weg direkt verwendet um die Daten auszulesen. Der Weg über das offizielle System-Framework ist also nicht unbedingt erforderlich. Der Schutz des Adressbuchs muss daher auf Dateiebene ansetzen. Damit können beide Zugriffswege abgefangen werden. Die Kontrolle des Zugriffs über das Framework reicht nicht aus.

Die Art der Zugriffskontrolle gestaltet sich in diesem Fall jedoch einfacher. Anders als bei der UDID oder der MAC-Adresse kann ein Adressbuch auch unter normalen Umständen keinerlei Daten enthalten. Einige der getesteten Apps weisen jedoch Defizite bei der Ausnahmenbehandlung von verweigerten Dateizugriffen auf. Die Apps stürzen beim Verweigern des Zugriffs auf die Adressbuch-Datenbank ab. Um die Daten zu schützen wird daher der Zugriff auf eine leere Schattenkopie der Datenbank mit identischer Struktur umgeleitet.

### 5.2.3. Kalender

Der Kalender enthält ebenfalls viele vom Benutzer eingegebene private Daten und Informationen. Ähnlich wie beim Zugriff auf das Adressbuch bietet das System ebenfalls ein spezielles Framework zum Zugriff auf den Kalender. Das betreffende Framework heißt *Event Kit Framework* und stellt Methoden zur Verfügung, durch deren Hilfe eine App lesend und schreibend auf den Kalender des Benutzers Zugriff erhalten kann. Auch bei diese Framework gibt es keine Zugriffskontrolle durch das System.

Genau wie im Falle des Adressbuch wurden auch Untersuchungen zur Verwendung des Kalenders angestellt. Auch hier konnte die Datei ermittelt werden, in der die

Kalenderdaten abgespeichert werden (`/private/var/mobile/Library/Calendar/Calendar.sqlitedb`). Dabei handelt es sich ebenfalls um eine SQLite-Datenbank. Die Datei wird bei Zugriffen über das Event Kit Framework direkt geöffnet und bearbeitet. Der direkte Zugriff auf die Datei ist, wie beim Adressbuch, lesend und schreibend möglich aber nicht in den offiziellen Dokumentation beschrieben. Daher muss auch beim Kalender der Schutz der Daten auf Dateiebene ansetzen.

Die Art der Zugriffskontrolle ist auf Grund der Gemeinsamkeiten zur Vorgehens beim Adressbuch analog. Das Umleiten des Dateizugriffs auf eine Schattenkopie der Kalender-Datenbank ist eine geeignete Methoden die darin gespeicherten Daten zu schützen.

### 5.2.4. Keyboard-Cache

Der *Keyboard-Cache* bezeichnet den Speicher der virtuellen Tastatur, der zur Abspeicherung neuer Wörter verwendet wird. Wie schon in der Anforderungsanalyse in Kapitel 3.1 auf Seite 12 beschrieben, wird der Speicher dazu verwendet, um bei der automatischen Fehlerkorrektur dem Nutzer angepasste Korrekturvorschläge anbieten zu können. Der Keyboard-Cache wird in iOS aber zusätzlich auch dazu verwendet, die Häufigkeit der vom Nutzer verwendeten Wörter zu analysieren, um so bessere Korrekturvorschläge für schon bekannt Wörter anbieten zu können. Im Keyboard-Cache befinden sich also alle vom Nutzer in einem gewissen Zeitraum eingegebenen Wörter. Der Keyboard-Cache wird automatisch bei der Eingabe über die Tastatur erweitert. Nur Passwort-Felder bilden hier eine Ausnahme. Die dort eingegebenen Informationen werden nicht im Keyboard-Cache abgespeichert.

Durch eine Analyse der im Keyboard-Cache gespeicherten Wortlisten könnten Rückschlüsse auf das Kommunikationsverhalten des Nutzers gezogen werden. Der Keyboard-Cache kann vom Benutzer geleert werden. Die dazu notwendigen Konfigurationselemente befinden sich jedoch erst in der dritten Ebene des Einstellungsmenüs.

Auch in diesem Fall wurden wieder Untersuchungen durchgeführt und der Speicherort des Keyboard-Caches lokalisiert. Der Keyboard-Cache ist in mehrere Dateien aufgeteilt. Es findet ebenfalls eine Unterteilung in Sprachen statt, da eine deutsche Tastatur z. B. nur deutsche Korrekturvorschläge enthalten soll. Alle zum Keyboard-Cache gehörenden Dateien befinden sich in einem gemeinsamen Ordner im Dateisystem (`/private/var/mobile/Library/Keyboard/`). Tests haben gezeigt, dass Apps auch hier direkt Zugriff auf die Dateien nehmen und diese zum Lesen öffnen können.

Um den Zugriff auf den Keyboard-Cache vollständig zu hindern ist es also notwendig den Zugriff auf den ganzen Ordner des Keyboard-Caches zu blockieren.

Die Art der Zugriffskontrolle gestaltet sich also auch hier einfach. Die komplette Verweigerung oder Erlaubnis zum Zugriff auf den Keyboard-Cache ist eine geeignete Methode die darin enthalten Daten vor unbefugtem Zugriff durch die Apps zu schützen. Der größte Nachteil, der durch die Verweigerung des Zugriffs entstehen kann, ist der Verlust der selbstlernenden Korrektur der Tastatur. Die normale, nicht dem Nutzungsverhalten angepasste Fehlerkorrektur bleibt erhalten.

### 5.2.5. Freier Speicherplatz

Ein Problem ganz anderer Art besteht mit dem freien Speicherplatz auf dem Gerät. Beim Löschen von Daten auf einem Datenträger werden meist nicht die Daten selbst gelöscht. Lediglich die Einträge in den Meta-Daten des Dateisystems werden entfernt. Die Datei taucht damit nicht mehr im Dateisystem auf, die ursprünglichen Dateiinhalte sind jedoch immer noch auf dem Datenträger gespeichert. Wird ein Geräte in fremde Hände (beispielsweise zur Reparatur) gegeben oder anderweitig mit einem geeigneten Analysewerkzeug untersucht können diese gelöschten Daten wiederhergestellt werden.

Das sichere Löschen von Daten ist nur durch ein anschließendes Überschreiben der betreffenden Sektoren auf dem Datenträger möglich. Zu diesem Zweck beinhaltet die Sicherheitsapplikation die Funktionalität, den kompletten freien Speicherplatz zu überschreiben. Eine Wiederherstellung von gelöschten Daten ist damit lediglich noch unter Einsatz spezieller, sehr kostspieliger Spezialhardware möglich. Eine Wiederherstellung von Dateien mittels Softwarelösungen wird durch diese Vorgehensweise effektiv verhindert. Da in iOS-Geräten ausschließlich Flash-Speicher verbaut werden ist ein einmaliges Überschreiben des freien Speicherplatzes ausreichend.

## 5.3. Einschleusen von Schadcode verhindern

Um den Schutz vor dem Einschleusen von Schadcode sicherzustellen wurden in der Anforderungsanalyse mehrere Forderungen aufgestellt.

Der Schutz gegen die sogenannten Drive-By-Angriffe (siehe Abschnitt 3.2) durch PDF- und Microsoft Office Dokumente kann durch das Abfangen der Methodenaufrufe zur Initialisierung des Parsing-Vorgangs verbessert werden. Zur Initialisierung wird bei PDF-Dokumenten die C-Funktion `CGPDFDocumentCreateWithURL`

oder `CGPDFDocumentCreateWithProvider` verwendet. Das Parsing von Microsoft Office-Dokumenten wird durch den Aufruf der Methode `computePreview` der Klasse `QLBasePreviewParts` eingeleitet. Statt den ursprünglichen Methodenaufruf fortzusetzen, wird dem Nutzer eine Dialogbox präsentiert. Diese Dialogbox weist ihn darauf hin, dass die App gerade das Öffnen eines PDF bzw. Office Dokuments veranlasst. Der Nutzer hat die Möglichkeit dies fortzusetzen oder abzubrechen. Falls der Nutzer den Vorgang fortsetzt, wird auch der ursprüngliche Methodenaufruf weiter ausgeführt. Falls der Nutzer den Vorgang abbricht, wird der ursprüngliche Methodenaufruf nicht mehr ausgeführt. Damit wird der Parsing-Vorgang abgebrochen und ein möglicherweise schadhaftes Dokument kann das System nicht infizieren. Durch das Anzeigen der Dialogbox ist es dem Nutzer möglich, auch versteckt eingebunden Dokumente (beispielsweise als unsichtbarer iFrame innerhalb einer HTML-Seite) zu erkennen und den Zugriff darauf zu verweigern.

Dieses Verhalten sollte in einer späteren Version der Sicherheitsapplikation noch auf Mediendateien wie MP3-Dateien oder QuickTime-Filme ausgeweitet werden. Von diesen Dateien kann eine ähnliche Gefahr für das System ausgehen. Dazu sind allerdings weitere Untersuchung zum Systemverhalten beim Öffnen dieser Dateien erforderlich.

### 5.3.1. Änderung des User-Agents

Damit eine Drive-By-Angriff erfolgreich durchgeführt werden kann ist eine genaue Kenntnis über die verwendete Betriebssystem- und Browserversion erforderlich. Diese Informationen können über den HTTP User-Agent erlangt werden. Der Standardbrowser in iOS versendet im HTTP User-Agent sehr viele Daten mit genauen Versionsnummern zu Betriebssystem und Browser sowie die Modellnummer der Hardware. Der Angreifer kann damit seinen Angriff genau abstimmen und Schadcode speziell für verwundbare Betriebssystemversionen ausliefern. Die Sicherheitsapplikation beinhaltet daher die Möglichkeit, den HTTP User-Agent der Browserkomponente zu ändern, um dem Angreifer die Informationsbeschaffung zu erschweren.

### 5.3.2. Hinweis auf unsicherer Systemeinstellungen

Um den Nutzer auf unsichere Systemeinstellungen hinzuweisen wird eine Übersicht implementiert. In dieser Übersicht werden die Probleme mit der Konfiguration aufgelistet. Die Liste besteht aus zwei Bereichen. Der erste Bereich zeigt kritische Probleme an, die sofort behoben werden sollten. Der zweite Bereich zeigt Vorschläge an,

die Systemeinstellungen weiter zu optimieren. Diese Vorschläge haben jedoch eine geringe Dringlichkeit. Zu jedem dieser Vorschläge wird noch ein Beschreibungstext angezeigt, der das Problem weiter erläutert. Der Nutzer kann sich dadurch sehr einfach eine Übersicht über den Sicherheitsstatus seines Systems verschaffen und bei Problemen die richtigen Maßnahmen ergreifen.

## 5.4. Einfache Konfiguration

Auf eine einfache Konfiguration der Funktionalitäten wird Wert gelegt. Die installierten Apps werden in zwei Kategorien eingeteilt. Zum einen in Apps, die Root-Rechte erhalten können und zum anderen in Apps, die keine Root-Rechte erhalten können. Zur ersten Gruppe gehören die vorinstallierten System-Apps sowie die nachträglich installierten Jailbreak-Apps, die beispielsweise aus Cydia stammen. Zur zweiten Gruppe gehören die über offizielle Wege nachinstallierten Apps. Für jede dieser Applikationskategorien wird eine Liste mit den dazugehörigen Apps dargestellt. Zu jeder der dargestellten Apps wird neben dem Applikationsnamen auch das Applikationssymbol angezeigt. Die Symbole erhöhen den Wiedererkennungswert durch den Nutzer und helfen ihm eine bestimme App zu finden. Außerdem wird eine Suchfunktion implementiert, um den Nutzer bei sehr langen Applikationslisten besser unterstützen zu können.

### 5.4.1. Konfiguration über Profile

Jeder App, die überwacht werden soll, muss ein Berechtigungsprofil zugewiesen werden. Das Berechtigungsprofil enthält alle möglichen Einstellungen, die zu einer App gemacht werden können. Berechtigungsprofile können frei erstellt, bearbeiten und einer oder mehreren Apps zugewiesen werden. Es werden zwei vorkonfigurierte Berechtigungsprofile mit ausgeliefert. Jeder App ist zunächst eines der beiden vorkonfigurierten Berechtigungsprofile zugeordnet.

Das erste der vorkonfigurierten Berechtigungsprofile soll ein Standardprofil darstellen, das ohne Änderungen auf die meisten Apps aus dem offiziellen App Store angewendet werden kann. Da diese Apps von Drittanbietern stammen, enthält das Profil eine sehr restriktive Konfiguration. Der Zugriff auf Adressbuch, Kalender und Keyboard-Cache wird verweigert. Um die Funktion der Apps nicht zu sehr zu beschränken wird in diesem Profil der zweite Zugriffsmodus auf den UDID und die

WLAN MAC-Adresse verwendet. Dabei wird einmalig eine zufällige Version der Informationen erstellt und zukünftig verwendet.

Das zweite Berechtigungsprofil soll ein Standardprofil für alle von Apple entwickelte Apps darstellen. Da diese Apps aus einer vertrauenswürdigeren Quelle stammen, kann diesen ein größerer Freiraum gewährt werden, um die Integrationsfunktionen des Adressbuchs oder des Kalenders in den vorinstallierten Anwendungen nicht zu verlieren. Daher wird diesen Apps der Zugriff auf das Adressbuch, den Kalender und den Keyboard-Cache erlaubt. Auch auf den UDID und die WLAN MAC-Adresse dürfen diese Apps zugreifen.

Ist einer App ein Berechtigungsprofil zugewiesen, werden die daraus übernommen Einstellungen im Übersichtsbildschirm für diese App angezeigt. Wenn der Nutzer eine Ausnahme für eine bestimmte Einstellung definieren möchte, kann er dies durch die Verwendung eines einfachen Wischens über die Einstellung tun. Es ist somit möglich ein Profil für eine App zu verwenden aber gleichzeitig für einzelne Einstellungselemente Ausnahmen zu definieren.

### 5.4.2. Erinnerungsmodus

Nach dem Installieren neuer App kann das Konfigurieren eines Berechtigungsprofils vom Nutzer vergessen werden. Daher wird die Möglichkeit geboten, beim Starten einer neuen oder nicht konfigurierten App, ein Sicherheitshinweis auszugeben und den Nutzer aufzufordern die Sicherheitseinstellung für diese App vorzunehmen. Dieses Verhalten kann auf Wunsch des Nutzers auch deaktiviert werden. Der Nutzer wird beim ersten Starten der Sicherheitsapplikation gefragt, ob er diesen automatischen Erinnerungsmodus aktivieren möchte.

## 5.5. Erweiterbarkeit durch erweiterbare Konfigurationsdateien

Eine einfache Erweiterbarkeit wird durch die Verwendung von Konfigurationsdateien erreicht. Alle Einstellungen die in der Sicherheitsapplikation für Apps vorgenommen werden können, sind in XML-Dateien definiert. Die XML-Dateien verwenden das *Property List* XML-Schema von Apple.

In der *Definitionsdatei* werden die einzelnen Elemente definiert, die der Benutzer für Apps konfigurieren kann. Eines dieser Elemente stellt beispielsweise den Zugriffsschutz für das Adressbuch oder Zugriffsmodus auf den UDID dar. Zu jedem Element kann ein Name und ein Standardwert angegeben werden. Auch weitere Einstellungen, die die Darstellung auf der Konfigurationsoberfläche betreffen, sind möglich. So kann z. B. angegeben werden, ob der Einstellungspunkt als An/Aus-Schalter oder als Auswahlliste auf der Konfigurationsoberfläche erscheinen soll. Auch Hilfetexte, die den Benutzer bei der Konfiguration unterstützen sollen, können angegeben werden. Die Konfigurationsoberfläche zum Erstellen und Bearbeiten von Profilen wird aus den Definitionen in dieser Datei erstellt.

Zur Abspeicherung der Konfiguration für eine App wird für jede installierte App eine separate *App-Konfigurationsdatei* angelegt. In dieser Datei sind die konkreten Einstellungen, die der Benutzer für diese App vornimmt, abgespeichert. Das kann beispielsweise die Aktivierung des Zugriffsschutzes auf das Adressbuch sein. Das genaue Schema dieser Datei richtet sich nach den in der Definitionsdatei beschriebenen Einstellungselementen. Das Schema wird dynamisch an die Definitionsdatei angepasst.

Um den Benutzer über den Status seines System zu informieren wird die in Kapitel 5.3 auf Seite 30 beschriebene Übersicht implementiert. Dazu werden im Hintergrund verschiedene Systemeinstellungen, die Einfluss auf die Sicherheit des Gerätes haben, überprüft und ausgewertet. Die in der Auswertung gefundenen Probleme werden dem Benutzer auf dieser Übersichtsseite präsentiert. Um die Erweiterbarkeit der Systemüberprüfung und Auswertung zu gewährleisten, werden die Überprüfung der Systemeinstellungen in einzelne Module aufgeteilt. Jedes dieser Module ist für die Überprüfung eines bestimmten Aspekts der Systemeinstellungen zuständig. Die Überprüfung kann dadurch noch nachträglich um weitere Aspekte erweitert werden.

# 6. Implementierung

Die Implementierung des Konzeptes erfolgt als App für das Betriebssystem iOS. Es wird die Version 5.0.1 des Betriebssystems unterstützt. Auf älteren Versionen ist die Sicherheitsapplikation nicht lauffähig. Die Unterstützung von zukünftigen Versionen hängt davon ab, ob Apple durch ein Update des Betriebssystems die von der Sicherheitsapplikation verwendeten Funktionalitäten verändert. Da einige Funktionalitäten auf *Private APIs* zugreifen oder ein bestimmtes internes Verhalten des Betriebssystems voraussetzen, muss die App auf zukünftigen Versionen von iOS neu getestet und gegebenenfalls angepasst werden. Als Private APIs werden die nicht öffentlichen Schnittstellen des Betriebssystem bezeichnet.

Im Folgenden wird die Implementierung genauer beschrieben. Dazu wird in Abschnitt 6.1 zunächst die Benutzeroberfläche der Sicherheitsapplikation vorgestellt. Anschließend wird die Architektur im Allgemeinen umrissen und auf ausgewählte Punkte genauer eingegangen.

## 6.1. Benutzeroberfläche

Die Benutzeroberfläche ist in einem iOS typischen Muster aufgebaut. Am oberen Ende des Bildschirms, gleich unter der Statuszeile des Betriebssystems, befindet sich die Titelzeile. Diese dient dazu dem Benutzer anzuzeigen auf welcher Bildschirmseite er sich gerade befindet. Das Systemverhalten gibt außerdem vor, dass sich bei einer mehrstufigen Hierarchie die Zurück-Schaltflächen zur nächst höheren Hierarchiestufe ebenfalls in der Titelzeile befinden. Die Titelzeile wird durchgehend auf allen Bildschirmseiten verwendet.

In der Mitte des Bildschirms befindet sich der eigentliche Inhalt der Benutzeroberfläche. Der Benutzer interagiert hauptsächlich mit diesem Teil der Oberfläche.

Am unteren Ende des Bildschirm befindet sich eine Registernavigation, die sogenannte *Tabbar*. Die Tabbar trennt die Funktionen der App in vier Bereiche auf, die im Folgenden genauer vorgestellt werden. Durch das Berühren des Symbols eines anderen Bereichs kann in diesen Bereich gewechselt werden. Die Tabbar ist ebenfalls

auf den meisten Bildschirmseiten sichtbar. Sie wird lediglich in tieferen Hierarchieebenen zur Platzeinsparung ausgeblendet.

### 6.1.1. Bereich „Home“

Den Einstieg in die App bildet der *Home*-Bereich. Im Home-Bereich befindet sich der Übersichtsbildschirm, der dem Benutzer einen Überblick über den Zustand des Systems gibt. Dazu wird im oberen Bereich des Übersichtsbildschirms eine Schnellinformationszeile angezeigt, die den Komplettzustand des Systems zusammenfasst.

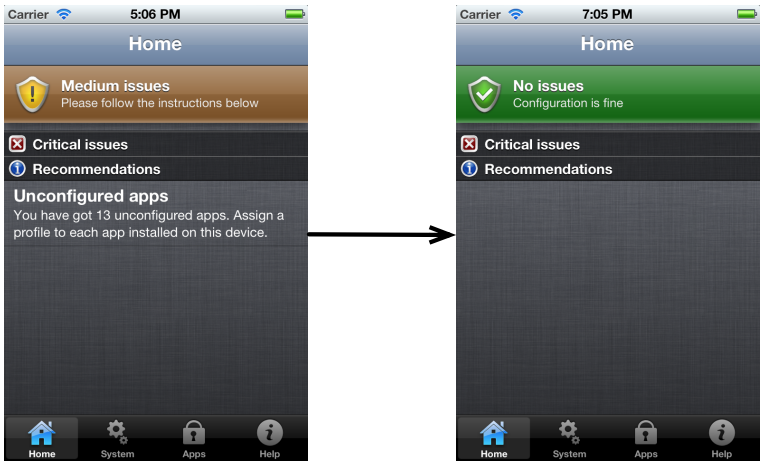


Abbildung 6.1.: Home-Bildschirm

Die Anzeige des Komplettzustands verwendet dazu ein dreistufige Zustandskonzept. Die niedrigste Sicherheitsstufe gibt an, dass es kritische Konfigurationsprobleme gibt, die unbedingt schnell behoben werden müssen. Die mittlere Sicherheitsstufe gibt an, dass es zwar weitere Konfigurationsvorschläge gibt, diese jedoch nicht kritisch für die Sicherheit des Systems sind. Die höchste Sicherheitsstufe gibt an, dass keine Probleme in der Konfiguration festgestellt wurden. Die Systemkonfiguration ist aus Sicherheitssicht optimal.

Falls Sicherheitsprobleme in der Systemkonfiguration gefunden wurden, werden diese in der Tabelle unter der Schnellinformationszeile angezeigt. Durch das Antippen der einzelnen Vorschlagszeilen wird der Nutzer auf die relevanten Einstellungsseiten weitergeleitet oder es werden ihm weitere Informationen zur Behebung des Problems angezeigt. Beseitigt der Benutzer das Konfigurationsproblem taucht es beim nächsten Anzeigen des Übersichtsbildschirms nicht mehr in der Tabelle auf.

Dieser Zusammenhang wird in der Abbildung 6.1 verdeutlicht. Der linke Teil der Abbildung zeigt den Übersichtsbildschirm mit der mittleren Sicherheitsstufe. Die Sicherheitsapplikation weist den Benutzer darauf hin, dass nicht allen installierten Apps ein Berechtigungsprofil zugewiesen wurde. Der rechte Teil der Abbildung zeigt den Übersichtsbildschirm nach der Behebung des Sicherheitsproblems. Es wird die höchste Sicherheitsstufe angezeigt.

### 6.1.2. Bereich „System“

Im Bereich *System* befinden sich Funktionen, die eine systemweite Auswirkung besitzen und sich nicht auf eine App beschränken. Die Funktionen werden in einer Listenform dargestellt. Die Abbildung 6.2 zeigt diese Listenform. Durch das Tippen auf eine Zeile der Liste wird die dazugehörige Funktionalität aufgerufen. Zu den Funktionen gehört zu diesem Zeitpunkt nur das sichere Löschen des freien Speicherplatzes des Geräts. Die App kann jedoch in Zukunft um weitere systemweite Funktionen erweitert werden.

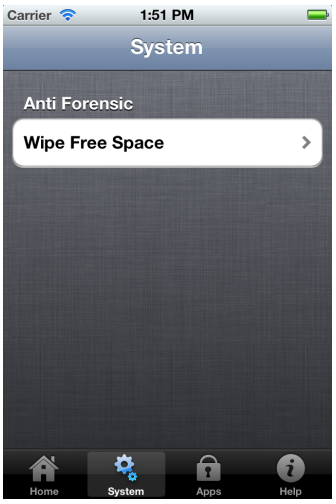


Abbildung 6.2.: System-Bildschirm

### 6.1.3. Bereich „Apps“

Im Bereich *Apps* befinden sich Funktionen, die Auswirkungen auf die installierten Apps besitzen. Die verschiedenen Funktionen werden ebenfalls in einer Listenform dargestellt. Auf dieser Bildschirmseite befinden sich die Funktionalitäten um Einstellungen für Apps zu ändern, um Berechtigungsprofile zu verwalten und um das

Standardverhalten bei neuen Apps zu verändern. Durch das Tippen auf eine Zeile der Liste wird die dazugehörige Funktionalität aufgerufen.

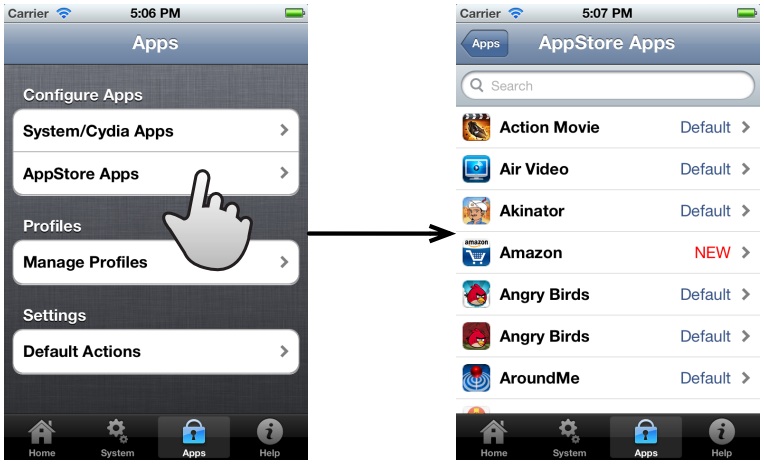


Abbildung 6.3.: Einstellungen für Apps vornehmen

**Sektion „Configure Apps“**

Die Einstellungen für die Apps sind auf zwei getrennte Unterbereiche aufgeteilt. Im ersten Unterbereich werden die System- und Jailbreak-Apps verwaltet. Im zweiten Unterbereich werden die normalen, über den offiziellen App Store installierten Apps verwaltet. In Abbildung 6.3 wird im linken Teil der Abbildung die Anfangsbildschirmseite des Apps-Bereichs gezeigt. Durch das Antippen der Funktionalität *AppStore Apps* gelangt man zu der Applikationsliste der installierten Apps, die aus dem App Store stammen. Die Applikationsliste ist auf der rechten Seite der Abbildung dargestellt. Alle installierten Apps aus dem App Store werden mit Name und Applikationssymbol in einer Liste dargestellt. Außerdem wird zu jeder der App das jeweils zugewiesene Berechtigungsprofil auf der rechten Seite der Zelle angezeigt. Falls einer App noch kein Berechtigungsprofil zuwiesen wurde befindet sich dort stattdessen der Vermerk *NEW*. Zwischen der Applikationsliste und der Titelzeile befindet sich die Suchleiste. Dort kann anhand eines Suchbegriffs die Liste durchsucht werden.

Durch das Antippen einer Zeile der Liste wird der Einstellungsbildschirm für diese Apps aufgerufen. Der Einstellungsbildschirm für Apps zeigt eine Liste mit der konfigurierbaren Einstellungen für Apps an. An oberster Stelle kann das Berechtigungsprofil für diese App eingestellt werden. Durch das Antippen der Zeile *Profile* wird eine Liste mit verfügbaren Profile eingeblendet aus denen gewählt werden kann. Die restlichen Einstellungen sind zunächst deaktiviert. Diese werden aus dem Berechtigungsprofil übernommen.

Abbildung 6.4.: Ausnahmen für einzelne Einstellungen erstellen

Um Ausnahmen für einzelne Einstellungen zu erstellen muss der Benutzer eine Geste ausführen. Die Abbildung 6.4 stellt dies genauer dar. Zunächst wischt der Benutzer mit dem Finger über die Einstellung, für die er eine Ausnahme erstellen möchte. Diese Geste wird häufig an verschiedenen Stellen des Systems verwendet und ist den Nutzern der Plattform geläufig. Im Anschluss an das Wischen werden zwei Schaltflächen anzeigt. Der Benutzer kann nun wählen, ob er für diese Einstellung eine Ausnahme erstellen möchte. Alternativ kann er den Vorgang abbrechen. Wenn der Benutzer fortsetzt, wird für diese Einstellung eine Ausnahme erstellt. Die Listenzeile wird zur Modifikation freigeschaltet. Das Zurücksetzen auf die Berechtigungsprofileinstellungen geschieht analog dazu auf einer freigeschalteten Zeile.

**Sektion „Profiles“**

Die Verwaltung der Berechtigungsprofile kann durch die Funktion *Manage Profiles* im Bereich Apps aufgerufen werden. Dem Benutzer wird eine Liste der verfügbaren Profile angezeigt. Zusätzlich wird ihm die Möglichkeit geboten, Profile anzulegen, zu löschen und zu bearbeiten.

Zum Ändern vorhandener Berechtigungsprofile müssen diese in der Liste der verfügbaren Profile angetippt werden. Dann wird ein Änderungsbildschirm für dieses Profil angezeigt.

Zum Löschen der Profile muss über die Zeile des gewünschten Profils gewischt werden. Dadurch wird auf dieser Zeile eine Löschen-Schaltfläche angezeigt. Dies ist normales Systemverhalten und den Benutzern von iOS-Geräten geläufig.

Abbildung 6.5.: Neues Berechtigungsprofil anlegen

Die Abbildung 6.5 zeigt die Erstellung eines neuen Berechtigungsprofils. Auf der linken Seite der Abbildung ist die Bildschirmseite mit der Liste der verfügbaren Profilen dargestellt. Durch das Tippen auf die Hinzufügen-Schaltfläche in der Titelleiste wird ein neues Berechtigungsprofil angelegt. Die rechte Seite der Abbildung zeigt den Anlegebildschirm. Dort kann der Name des Berechtigungsprofils sowie dessen Einstellungen angegeben werden.

**Sektion „Settings“**
In der Funktion *Default Actions* im Bereich Apps, die in Abbildung 6.3 auf Seite 38 als letzte Funktion in der Liste angezeigt wird, kann das Standardverhalten beim Starten nicht konfigurierten App eingestellt werden. Der Nutzer kann mit Hilfe eines Schalters wählen, ob er beim Starten einer nicht konfigurierten App zur Konfiguration eines Profils aufgefordert werden will.

### 6.1.4. Bereich „Help“

Der Bereich *Help* beinhaltet eine Liste mit Hilfequellen, die dem Benutzer zur Verfügung stehen. Die verschiedenen Quellen werden in einer Listenform dargestellt, wie in Abbildung 6.6 gezeigt.

Dem Benutzer steht dabei eine Liste mit häufig gestellten Fragen und deren Antworten zur Verfügung. Außerdem kann der Benutzer die Website der App besuchen um direkten Kontakt zum Entwickler herzustellen.

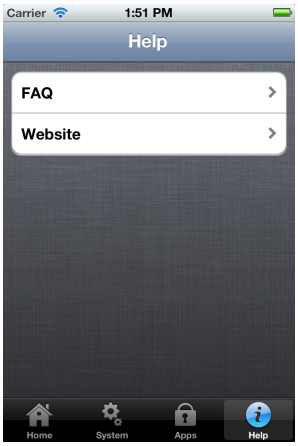


Abbildung 6.6.: Help-Bildschirm

## 6.2. Architektur

Die Funktionalitäten der App sind in zwei getrennten Teilkomponenten implementiert. Die Abbildung 6.7 stellt die Architektur und den Informationsfluss innerhalb der Architektur dar.

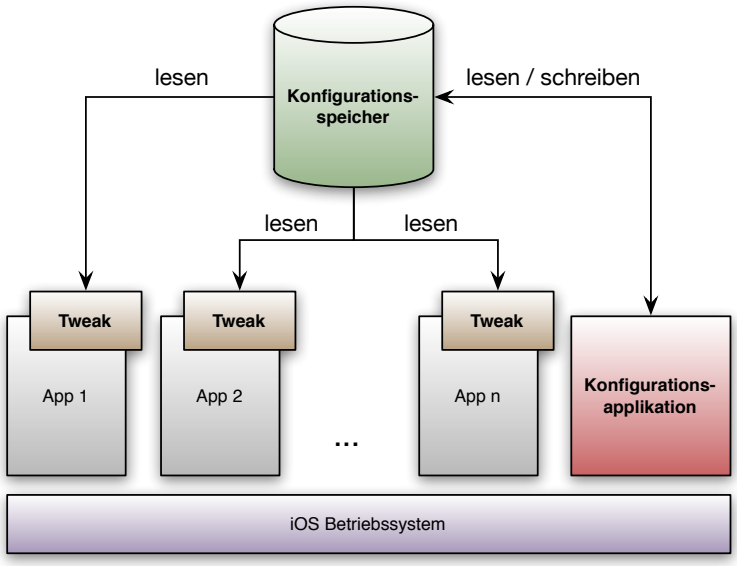


Abbildung 6.7.: Architektur der Sicherheitsapplikation

**Konfigurationsapplikation**
Die Konfigurationsapplikation ist die App, mit der der Benutzer interagiert. Diese ist als Systemapplikation auf dem Gerät installiert und kann über die normale App-Übersicht des Betriebssystems gestartet werden. Die App besitzt die in Abschnitt 6.1 beschriebene Benutzeroberfläche. Die hier konfigurierten Berechtigungsprofile und App-Einstellungen werden als XML-Dateien im Konfigurationsspeicher gespeichert. Der Konfigurationsspeicher ist ein bestimmter Ordner im Dateisystem (`/private/var/mobile/Documents`). Auf diesen Ordner können nur System-Apps schreibend zugreifen. Allen anderen Apps ist nur ein lesender Zugriff möglich.

Bei der Implementierung wurde das Model-View-Controller Design Pattern um gesetzt und somit die Oberfläche, die Steuerung der Oberfläche und die Datenhaltung voneinander getrennt. Dadurch kann die Oberfläche später relativ einfach auch den Bildschirm des Tablet-PCs *iPad* angepasst werden.

**Tweak**
Die zweite Teilkomponente ist der sogenannte *Tweak*. Der Tweak ist die dynamische Bibliothek, die zur Überwachung und Kontrolle anderer Apps in deren Adressraum geladen wird. Dazu wird das im Kapitel 2.3 auf Seite 8 beschriebene Verfahren der Dynamic Library Injection verwendet. Die Bibliothek lädt nach dem Einbringen in eine App die im Konfigurationsspeicher abgelegten Konfigurationsdateien und initialisiert danach die konfigurierten Funktionalitäten. Der Ablauf beim Laden der Konfiguration ist im Abschnitt 6.2.1 genauer beschrieben. Die Bibliothek ist für den Nutzer unsichtbar.

### 6.2.1. Berechtigungsprofile und Applikationskonfigurationen

Die Berechtigungsprofile und die Applikationskonfigurationen werden in der Konfigurationsapplikation erstellt. Als Dateiformat wird eine XML-Datei im Property List Format[13] von Apple verwendet. Das XML-Format zusammen mit der Property List Format Definition von Apple bietet die einfachste Möglichkeit Konfigurationen abzuspeichern. Die Unterstützung dieses Formats ist bereits in die Frameworks des Systems integriert. Dadurch wird das Anlegen und Bearbeiten dieser Dateien erleichtert.

Für jedes Berechtigungsprofil wird eine eigene Konfigurationsdatei erstellt. In dieser Datei werden die Einstellungen für das Profil sowie dessen Name abgespeichert.

[13] Weitere Informationen: `https://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/PropertyLists/Introduction/Introduction.html`

Beim Erstellen des Berechtigungsprofils wird ein pseudozufälliger Universally Unqiue Identifier (UUID) generiert und ebenfalls in der Konfigurationsdatei gespeichert. Der UUID wird zur Identifizierung des Profils verwendet und bleibt während der ganzen Lebenszeit des Profils unverändert.

Listing 6.1: Beispiel für ein Berechtigungsprofil

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
    PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>DisplayName</key>
   <string>Restriktiv</string>
   <key>FileSchemaVersion</key>
   <integer>1</integer>
   <key>Identifier</key>
   <string>EF3CC188-A4A1-4941-ACE6-D36D5F2F35EA</string>
   <key>Type</key>
   <integer>2</integer>
   <key>Settings</key>
   <dict>
      <key>BLOCK_ADRESSBOOK</key>
      <true/>
      <key>BLOCK_CALENDAR</key>
      <true/>
      <key>BLOCK_KEYBOARDCACHE</key>
      <true/>
      <key>FAKE_MAC</key>
      <true/>
      <key>FAKE_UDID</key>
      <true/>
      <key>UDID_FAKE_METHOD</key>
      <integer>0</integer>
      <key>WARN_OFFICE</key>
      <true/>
      <key>WARN_PDF</key>
      <true/>
      <key>FAKE_USER_AGENT</key>
      <string>Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)</string>
   </dict>
</dict>
</plist>
```

Das Listing 6.1 veranschaulicht den Aufbau eines Berechtigungsprofils an einem Beispiel. Das Property List Format definiert schon einige Datentypen. Ein Element mit

dem Namen `dict` stellt dabei ein *Data Dictionary* dar, das Schlüssel-Wert Daten aufnehmen kann. Der Inhalt wird als Folge von Schlüsselelementen (mit dem Elementnamen `key`) und Wertelementen (Elementname je nach Datentyp der gespeicherten Information verschieden) beschrieben. Ein Dictionary kann weitere Dictionaries enthalten. Das Beispielprofil in Listing 6.1 wird in der Oberfläche mit dem Namen *Restriktiv* angezeigt. Dieser Namen ist unter dem Schlüssel `DisplayName` definiert. Der Wert des Schlüssels `FileSchemaVersion` enthält die Information über die Version des Dateischemas. Da Änderungen am Dateischema in späteren Version nicht ausgeschlossen werden können, kann dieses Feld zur Identifizierung der Schemaversion der Datei verwendet werden. Unter dem Schlüssel `Identifier` ist der UUID gespeichert, der für dieses Berechtigungsprofil generiert wurde. Der Wert des Schlüssels `Type` wird dazu verwendet, den Typ der Datei anzugeben um die Datei von eine App-Konfigurationsdatei besser unterscheiden zu können. Der Wert `1` bedeutet hierbei, dass es sich um ein Berechtigungsprofil handelt. Unter dem Schlüssel `Settings` wird die Konfiguration des Berechtigungsprofils als Dictionary abgespeichert. Dabei entspricht jedem Schlüssel-Wert Eintrag einem Konfigurationselement.

Für jede App wird eine eigene App-Konfigurationsdatei erstellt. In iOS hat jede App eine eindeutigen *Bundle Identifier*. Daher wird dieser zur Identifikation der Konfigurationsdatei und zur Zuordnung der Konfiguration zu einer App verwendet und in der Konfigurationsdatei abgespeichert. Die Konfigurationsdatei enthält außerdem einen Verweis auf ein Berechtigungsprofil. Dazu wird der UUID des Berechtigungsprofils in die Datei geschrieben. Außerdem ist eine Liste mit Ausnahmen, bei der die Konfiguration für diese App von Berechtigungsprofil abweichen soll, integriert. Das Dateischema entspricht größtenteils dem des Berechtigungsprofils. Zusätzlich enthält die Datei noch den Schlüssel `ParentProfileIdentifier`, der ein Berechtiungsprofil über dessen Identifier referenziert. Unter dem Schlüssel `Settings` werden dann lediglich die Ausnahmen vom Berechtiungsprofil definiert. Außerdem werden die generierten Schattenkopien des UDID und der WLAN MAC-Adresse zwischengespeichert. Dazu werden die Schlüssel `ApplicationUDID` bzw. `ApplicationWiFiMAC` verwendet.

Zur Verdeutlichung des Zusammenhangs ist in Abbildung 6.8 auf Seite 45 der Ablauf beim Laden der Konfiguration dargestellt. Dieser Ablauf findet beim Starten des Tweaks statt. Der Tweak ermittelt zunächst den Bundle Identifier der App, in deren Adressraum er geladen wurde. Danach wird eine passende App-Konfigurationsdatei gesucht. Falls keine Konfigurationsdatei gefunden wurde, wird die Applikation als nicht konfiguriert eingestuft. Es wird überprüft, ob sich der Nutzer beim Start einer nicht konfigurierten App an die Konfiguration der App erinnern lassen will. Falls

dies der Fall ist, wird der Erinnerungsbildschirm angezeigt. Falls dies nicht der Fall ist, wird der Tweak für den restlichen Ablaufs des Programms deaktiviert.

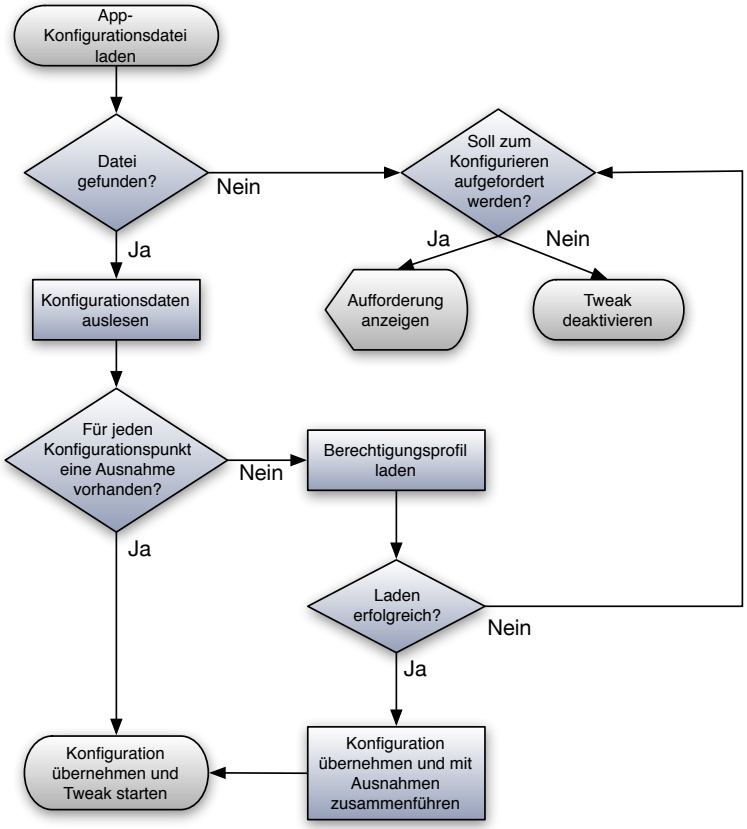


Abbildung 6.8.: Flussdiagramm vom Laden der Konfiguration im Tweak

Wenn die Konfigurationsdatei erfolgreich gefunden und geöffnet werden konnte, werden die Daten ausgelesen. Falls dabei für jeden Einstellungspunkte eine Ausnahme in der App-Konfigurationsdatei vorhanden ist, werden diese geladen und der Tweak aktiviert. In den meisten Fällen werden jedoch keine Ausnahme definiert oder die Ausnahmen betreffen nur einzelne Einstellungselemente. In diesem Fall wird auch das Berechtigungsprofil geladen. Falls dieses nicht gefunden oder geöffnet werden kann, stuft der Tweak die App als nicht konfiguriert ein. Falls das Berechtigungsprofil geladen werden konnte, werden die Einstellungen des Berechtigungsprofils übernommen. Dabei werden die Ausnahmen, die in der App-Konfigurationsdatei definiert wurden, berücksichtigt. Anschließend wird der Tweak aktiviert.

### 6.2.2. Übersichtsbildschirm

Der Home-Bildschirm ist der Startbildschirm der Konfigurationsapplikation. Der Nutzer kann sich hier einen Überblick über den Sicherheitsstatus seines Systems verschaffen. Im Hintergrund ist das Model-View-Controller Design Pattern umgesetzt.

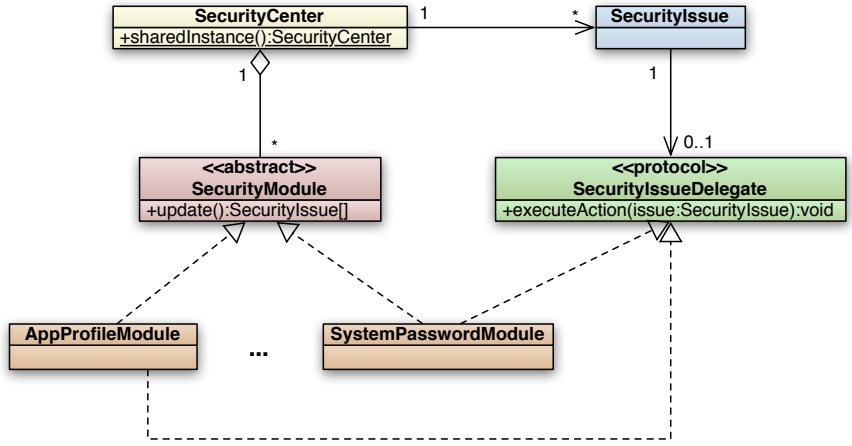


Abbildung 6.9.: Vereinfachtes UML-Klassendiagramm des SecurityCenters

Für die Verwaltung des Sicherheitszustandes des Systems ist Klasse `SecurityCenter` zuständig. Sie dient als zentrale Informationsquelle und ist als *Singleton*, wie in [Gamma u. a. 1994] beschrieben, realisiert. Von dem Singletonobjekt dieser Klasse kann der allgemeine Systemzustand sowie die konkreten Sicherheitsprobleme und Empfehlungen abgefragt werden.

Zur Ermittlung des Systemzustands, also der Sicherheitsprobleme und Empfehlungen, wird das `SecurityCenter` durch unabhängige Module erweitert. Jedes dieser Module ist für die Überprüfung eines Sicherheitsaspektes zuständig. Die Module ermitteln dabei das konkrete Probleme und bewerten die Relevanz und die Dringlichkeit des Problems. Die Information wird in ein Objekt des Typs `SecurityIssue` gespeichert und im `SecurityCenter` gesammelt. Es ist zusätzlich möglich einen Action-Handler für das `SecurityIssue` anzugeben. Dieses kann zur Ausführungen der Interaktionen des Nutzers benutzt werden. Anschließend wird die Dringlichkeitsinformation in den `SecurityIssue`-Objekten ausgewertet und daraus eine allgemeine Bewertung des Systemzustands ermittelt. Zum Schluss wird eine Update-Benachrichtigung über das Nachrichtensystem des Betriebssystems versendet.

Die Update-Benachrichtigungen werden dann von den *ViewControllern*, die für den Übersichtsbildschirm zuständig sind, empfangen und verarbeitet. In der Schnellinformationszeile des Übersichtsbildschirms wird die allgemeine Bewertung des Systemzustand anzeigt. Der Controller, der für die Liste der Sicherheitsprobleme und Empfehlungen zuständig ist, greift direkt auf die `SecurityIssue`-Objekte zu und zeigt diese Informationen an. Wenn der Nutzer auf einen Eintrag in der Liste tippt, wird der Action-Handler, der in der `SecurityIssue` registriert ist, aufgerufen falls dieser angegeben wurde.

### 6.2.3. Überwachung der Dateisystemzugriffe

Die Überwachung der Dateisystemzugriffe findet ebenfalls im Tweak statt. Die Überwachung wird benötigt, um beispielsweise Zugriffe auf das Adressbuch oder den Kalender feststellen zu können. Die relevanten Dateien werden anhand des Dateinamens erkannt. Dies alleine reicht allerdings noch nicht aus um wirklich alle Zugriffe auf bestimmte Daten registrieren zu können. Es müssen weitere Aspekte des Systems berücksichtigt werden, die im Folgenden näher beschrieben werden.

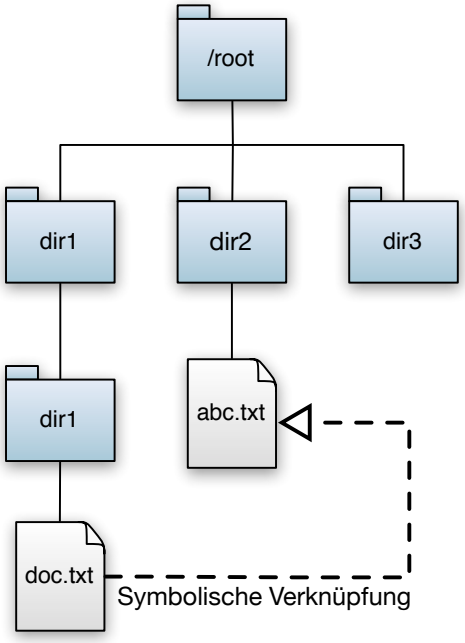


Abbildung 6.10.: Beispiel für eine symbolische Verknüpfung

Das Dateisystem HFS+[14], das auf Geräten mit iOS verwendet wird, hat die Fähigkeit *symbolische Verknüpfungen* anzulegen. Eine symbolische Verknüpfung ist eine Verknüpfung im Dateisystem, die eine andere Datei oder einen anderen Ordner anhand deren Pfad referenziert. Symbolische Verknüpfungen sind für die meisten Dateioperationen transparent. Lese- und Schreiboperationen auf die symbolische Verknüpfungen werden zur Zieldatei umgeleitet.

Symbolische Verknüpfungen können von jeder App angelegt werden. Um trotz der Dateisystemzugriffsüberwachung beispielsweise auf das Adressbuch zugreifen zu können, könnte eine bösartige App symbolische Verknüpfungen verwenden, um den Dateinamen zu verfälschen. Im iOS-System sind bereits symbolische Verknüpfungen zu verschiedenen Zielen vorhanden, die für das Umgehen der Dateisystemzugriffsüberwachung ausgenutzt werden könnten. Daher müssen die Verknüpfungen bei der Überwachung der Dateisystemzugriffe berücksichtigt werden. Zu diesem Zweck ist eine rekursive Verknüpfungsauflösung implementiert. Diese verfolgt ein Verknüpfungen solang zu ihrem Ziele zurück bis der korrekte Pfad gefunden wurde. Falls der korrekte Pfad nach eine bestimmen maximalen Anzahl an Rekursionen nicht gefunden werden konnte, wird der Zugriff auf die Datei komplett verweigert um Überläufe durch zyklische Verknüpfungsbeziehungen auszuschließen.

Eine weitere Form von Verknüpfungen stellen die sogenannten *harten Verknüpfungen* dar. Harte Verknüpfungen sind keine Referenzen auf eine andere Datei oder Ordner. Stattdessen referenzieren sie direkt die betroffenen Daten auf dem Datenträger und stellen somit einen gleichwertige Referenz auf die Daten wie die Original-Datei dar. Harte Verknüpfungen können in iOS nur mit Root-Rechten angelegt werden. Daher ist es Apps, die über die den offiziellen App Store heruntergeladen wurden, nicht möglich harte Verknüpfungen anzulegen.

Die implementierte Lösung berücksichtig diese Form der Verknüpfungen daher nicht. Im Gegensatz zu den symbolischen Verknüpfungen werden die harten Verknüpfungen im iOS-Betriebssystem kaum verwendet. In der Standardinstallation finden sich keine harten Verknüpfungen, die zum Zugriff auf private Daten verwendet werden können. Daher wurde vorerst auf die Implementierung der Überwachung von harten Verknüpfungen verzichtet. In zukünftigen Versionen der Sicherheitsapplikation sollten diese jedoch berücksichtigt werden.

[14] Weitere Informationen: `https://developer.apple.com/legacy/mac/library/#technotes/tn/tn1150.html`

# 7. Diskussion und Ausblick

Zum Abschluss der Arbeit wird das Erreichte in Abschnitt 7.1 zusammengefasst. Dabei werden in Abschnitt 7.2 auch die Grenzen der implementierten Lösung und der daraus resultierende Ausblick für weitere Arbeiten und Forschungen auf diesem Gebiet beschrieben. Abgerundet wird das Kapitel in Abschnitt 7.3 mit dem Fazit, der nochmals die neuen Anteile, die durch die Arbeit erreicht wurden verdeutlicht.

## 7.1. Diskussion

Im Rahmen dieser Arbeit wurde eine Sicherheitsapplikation für das Betriebssystem iOS der Firma Apple entworfen und implementiert. Die Sicherheitsapplikation ermöglicht es, andere Apps beim Abruf von privaten Daten zu kontrollieren und das unbemerkte Auslesen persönlicher Informationen zu verhindern. Die Sicherheitsapplikation verwendet dazu das Verfahren der Dynamic Library Injection, um die Kontrolle über die relevanten Systemzugriffe der Apps zu erhalten. Private Daten können damit vor einer App verborgen werden.

Zum Schutz vor der Identifizierbarkeit des Gerätes enthält die Sicherheitsapplikation die Möglichkeit, den Unqiue Device Identifier (UDID) sowie die MAC-Adresse des WLAN-Gerätes für die Apps zu fälschen. Dadurch erhalten die Apps anstatt der original Daten des Gerätes zufällige Werte zurückgeliefert. Da einige Apps den UDID zur Authentifizierung benutzen, wurden zwei Modi dieser Funktion implementiert. Zum Einen kann der Wert für den UDID und die WLAN MAC-Adresse kann bei jedem Zugriff der App neu aus Zufallszahlen erstellt werden. Zum Anderen ist es möglich für jede App einen festen UDID und WLAN MAC-Adresse festzulegen. Die zweite Möglichkeit gestattet es den Apps, die z. B. den UDID zur Authentifizierung verwenden wie gewohnt zu funktionieren. Gleichzeitig hat der Nutzer den Vorteil, dass ein applikationsübergreifendes Tracking durch den UDID verhindert wird.

Weiterhin ist ein Zugriffsschutz auf die Daten des Adressbuchs, des Kalenders und des Keyboard-Caches implementiert. Durch das Standardverhalten des Systems werden PDF- und Microsoft Office Dokumente, die an Emails anhängen oder in Web-

seiten eingebunden sind, automatisch verarbeitet und angezeigt. Durch Ausnutzung von Sicherheitslücken kann dadurch Drive-By-Angriffe Schadcode in das System eingeschleust werden. Daher ist Härtung des Systems durch eine Warnmeldung beim Öffnen und Anzeigen von PDF- und Microsoft Office Dokumenten verbessert worden. Durch eine Änderung des HTTP User-Agents kann die Identifizierbarkeit der genauen Betriebssystem- und Browserversion für Angreifer erschwert werden.

Auf einer Übersichtsseite kann der Benutzer sich über den Zustand seines Systems informieren. Falls unsichere Einstellungen im System vorliegen wird der Benutzer darauf hingewiesen und bei der Lösung des Problems unterstützt.

Dem Benutzer ist es durch das Konfigurationsprogramm möglich, die Zugriffsrechte auf private Daten für einzelne Apps selektiv zu vergeben. Dazu werden Berechtigungsprofile verwendet. Diese Berechtigungsprofile speichern die Einstellungen des Benutzers und können auf verschiedenen Apps angewendet werden. Zusätzlich wird die Möglichkeit geboten, für einzelne Apps in einzelnen Einstellungselementen Ausnahmen vom Berechtigungsprofil zu definieren. Der Benutzer kann damit von einer grob granularen Einstellungsmöglichkeit durch die Profile profitieren. Gleichzeitig bietet ihm die Sicherheitsapplikation aber auch die Möglichkeit fein granulare Einstellungen für einzelne Apps und Einstellungspunkte vorzunehmen.

Die Sicherheitsapplikation ist durch die Verwendung XML-Konfigurationsdateien einfach erweiterbar, um zukünftige Funktionalitäten miteinbeziehen zu können.

## 7.2. Ausblick

In der Arbeit wurden an einigen Stellen bereits Vorschläge für weitergehende Untersuchungen und Implementierungen gemacht um die Grenzen der App zu erweitern. Die Vorschläge sollen an dieser Stelle noch einmal zusammengefasst und erweitert werden.

### 7.2.1. Schutz weiterer Daten

Der Schutz der privaten Daten, der durch die in dieser Arbeit entwickelten Lösung besteht, ist noch nicht vollständig. Es gibt weitere Datenquellen, die noch nicht berücksichtigt wurden. Eine Liste weitere ermittelter Datenquellen befindet sich im Anhang dieser Arbeit (Anhang A.1).

### 7.2.2. Überwachung der Dateisystemzugriffe

Die in dieser Arbeit entwickelten Implementierung überwacht Dateisystemzugriffe auf Grundlage des Dateinamens der angeforderten Datei. Dabei werden auch Dateisystemzugriffe, die über symbolische Verknüpfungen erfolgen, zum eigentlich Ziel der Verknüpfung verfolgt. Bei harten Verknüpfungen ist das nicht möglich. Diese können somit zum Täuschen des Dateinamenfilters verwendet werden. Daher sollte eine Methode entwickelt werden, um auch harte Verknüpfungen überwachen zu können.

### 7.2.3. Verbesserung der Systemhärtung

In der in dieser Arbeit entwickelten Implementierung wird die Härtung des Systems vor allem durch das Überprüfen der Systemeinstellungen und die Warnung beim Öffnen von PDF- und Microsoft Office-Dokument realisiert. Die Probleme, die beim Einlesen und Verarbeiten von PDF- oder Microsoft Office-Dokumente auftreten sind jedoch nicht ausschließlich auf diese Formate beschränkt. Grundsätzlich können diese Probleme bei jeder Datei, die eine etwas kompliziertes Dateiformat aufweist auftreten. Daher sollte, außer bei PDF- und Office-Dokumente, noch beim Öffnen weiterer Dateitypen eine Warnmeldung erscheinen. Dies betrifft vor allem Mediendateitypen wie MP3 oder QuickTime, da diese oft Verwendung finden. Um das Öffnen und Verarbeiten dieser Datei abfangen zu können sind weitere Untersuchungen erforderlich.

### 7.2.4. Webservice für Profile und Einstellungen

Die Konfiguration der Sicherheitseinstellungen für Apps kann für unerfahrene Benutzer trotz der einfachen Benutzeroberfläche schwierig werden. Die Benutzer haben nicht das erforderliche technische Wissen die Gefahren, die von einer App ausgehen, einzuschätzen und eine korrekte Konfiguration einzustellen.

Abhilfe könnte ein Webservice schaffen, der Standardkonfigurationen für die am häufigsten verwendeten Apps vorhält. Diese können dann von der Sicherheitsapplikation auf dem Gerät des Benutzers bei Bedarf heruntergeladen und aktiviert werden. In diesem Zusammenhang könnte aktuelle Sicherheitsprobleme, die im Zusammenhang mit bestimmten Apps auftreten, reagiert werden. Davon würden auch technisch erfahrene Benutzer profitieren. Um die Anzahl der unterstützten Apps zu erhöhen könnte auch ein community-getriebenes Modell verfolgt werden.

### 7.2.5. Verbesserung der Methoden

Die in dieser Arbeit entwickelten Implementierung verwendet das Verfahren der Dynamic Library Injection um die Kontrolle über die App zu erhalten. Der Schutz setzt damit auf Applikationsebene an. Durch eine gezielte App-Programmierung, die genau auf die Sicherheitsapplikation abgestimmt ist, könnten die Schutzmechanismen ausgehebelt werden. Eine weitere Gefahr geht von anderen Jailbreak-Apps aus. Diese Apps können die gleichen Berechtigungen wie die hier implementierte Lösung erlangen. Dadurch wird es ihnen möglich die Sicherheitsapplikation komplett zu deaktivieren oder zu löschen.

Daher wäre es sinnvoll, die Lösung auf Ebene des Systemkernels anzusetzen. Da jedoch keine Dokumentation über den genauen Aufbau des Kernels vorliegt ist dies nicht ohne weiteres möglich. Um diesen Ansatz verfolgen zu können müsste Apple die benötigten Informationen veröffentlichen. Dies erscheint jedoch zum momentanen Zeitpunkt als sehr unwahrscheinlich. Daher müsste stattdessen eine aufwendige reverse-engineering Untersuchung des Systemkernels stattfinden um die relevanten Informationen zu erhalten.

Zusätzlich sollten aber auf jeden Fall noch einige Selbstschutzmechanismen in die Sicherheitsapplikation integriert werden. Damit kann zwar kein vollständiger Schutz hergestellt werden, jedoch kann das Deaktivieren der Sicherheitsapplikation erschwert werden.

## 7.3. Fazit

Die entwickelte Lösung stellt eine deutliche Erweiterung zur proof-of-conecpt Grundlage in Gorilla dar. Die Benutzeroberfläche wurde enorm verbessert und vereinfacht. Der Schutz der privaten Daten wurde verbessert und auf weitere wichtige Daten ausgeweitet. Es wurde ein neues Konzept zum Schutz der UDID erstellt und auf die WLAN MAC-Adresse erweitert. Die Berechtigungen können nun pro App selektiv vergeben werden. Es besteht die Möglichkeit Berechtigungsprofile zu benutzen und auf verschiedene Apps anzuwenden. Unsichere Systemeinstellungen werden von der Sicherheitsapplikation erkennt und der Benutzer wird darüber informiert. Der Benutzer kann sich nun auf einen Blick eine Übersicht über den Sicherheitsstatus seines Systems verschaffen.

Mit der entwickelten Sicherheitsapplikation ist es dem Benutzer möglich, die Sicherheit seines iOS-Systems zu verbessern. Gleichzeitig kann er seine Privatsphäre besser

schützen. Zum momentanen Zeitpunkt sind keine vergleichbaren Sicherheitsapplikation unter iOS bekannt.

# Literaturverzeichnis

**AllThingsD 2012**

AllThingsD: *Apple: App Access to Contact Data Will Require Explicit User Permission.* http://allthingsd.com/20120215/apple-app-access-to-contact-data-will-require-explicit-user-permission/. Version: February 2012, Abruf: 26.02.2012 5.2.2

**Apple, Inc. 2011a**

Apple, Inc.: *Apple's App Store Downloads Top 15 Billion.* http://www.apple.com/pr/library/2011/07/07Apples-App-Store-Downloads-Top-15-Billion.html. Version: Juli 2011, Abruf: 17.01.2012 1.1

**Apple, Inc. 2011b**

Apple, Inc.: *iOS Developer Library Deprecated UIDevice Methods.* https://developer.apple.com/library/ios/#DOCUMENTATION/UIKit/Reference/UIDevice_Class/DeprecationAppendix/AppendixADeprecatedAPI.html#/apple_ref/occ/instp/UIDevice/uniqueIdentifier. Version: Oktober 2011, Abruf: 25.01.2012 5.2.1

**Egele u. a. 2011**

Egele, M. ; Kruegel, C. ; Kirda, E. ; Vigna, G.: PiOS: Detecting Privacy Leaks in iOS Applications. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS).* San Diego, CA, Februar 2011 1.2, a, 5.2.1

**Freeman 2008**

Freeman, Jay: *Bypassing iPhone Code Signatures.* http://www.saurik.com/id/8. Version: 2008, Abruf: 30.01.2012 5.1

**Gamma u. a. 1994**

Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software.* 1. Addison-Wesley Professional, 1994 http://www.worldcat.org/isbn/0201633612. – ISBN 0201633612 6.2.2

**Gartner, Inc. 2011**

Gartner, Inc.: *Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010; Smartphone Sales Grew 72 Percent in 2010.* http://www.gartner.com/it/page.jsp?id=1543014. Version: Februar 2011, Abruf: 17.01.2012 1.1

**Hornyack u. a. 2011**

Hornyack, Peter ; Han, Seungyeop ; Jung, Jaeyeon ; Schechter, Stuart ; Wetherall, David: These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In: *Proceedings of the 18th ACM conference on Computer and communications security.* New York, NY, USA : ACM, 2011 (CCS '11). – ISBN 978–1–4503–0948–6, 639–652 4.2

**Seriot 2010**

Seriot, Nicolas: iPhone Privacy Who am I ? In: *Applied Sciences* (2010) 5.2

**Smith 2010**

Smith, Eric: iPhone Applications & Privacy Issues An Analysis of Application Transmission of iPhone Unique Device Identifiers ( UDIDs ). In: *New York* (2010), 1–19. http://www.kompatscher.biz/phocadownload/iPhone-Applications-Privacy-Issues.pdf a, 5.2.1

**Thampi 2012**

Thampi, Arun: *Path uploads your entire iPhone address book to its servers.* http://mclov.in/2012/02/08/path-uploads-your-entire-address-book-to-their-servers.html. Version: February 2012, Abruf: 26.02.2012 5.2.2

**The Register 2009**

The Register: *iPhone app grabs your mobile number.* http://www.theregister.co.uk/2009/09/30/iphone_security/. Version: September 2009, Abruf: 17.01.2012 1.2

**ZDNet 2011**

ZDNet: *Apple beseitigt kritische PDF-Lücke in iOS 4.* http://www.zdnet.de/news/41554944/apple-beseitigt-kritische-pdf-luecke-in-ios-4.htm. Version: Juli 2011, Abruf: 17.01.2012 1.2, 3.2

# A. Anhang

## A.1. Datenquellen für private Daten in iOS

### A.1.1. Geräteinformationen

| **Datum** | **Art** | **Zugriff** |
|---|---|---|
| UDID | API | `[UIDevice uniqueIdentifier];` |
| IMEI | API | `[NetworkContoller IMEI];` (Private-API) |
| WLAN MAC | API | `ioctl, getifaddrs, sysctl` |
| | Datei | /private/var/preferences/SystemConfiguration/ NetworkInterfaces.plist |
| Bluetooth MAC | Datei | /private/var/preferences/SystemConfiguration/ NetworkInterfaces.plist |

Tabelle A.1.: Datenquellen für Geräteinformationen

### A.1.2. SIM-Informationen

| **Datum** | **Art** | **Zugriff** |
|---|---|---|
| Telefonnummer | Datei | /private/var/wireless/Library/Preferences/ com.apple.commcenter.plist, Key: `SBFormattedPhoneNumber` |
| SIM-ICCID | Datei | /private/var/wireless/Library/Preferences/ com.apple.commcenter.plist, Key: `ICCID` |
| Netzbetreiber | API | `CTTelephonyNetworkInfo` |

Tabelle A.2.: Datenquellen für SIM-Informationen

### A.1.3. Telefonbuch

| Datum | Art | Zugriff |
|---|---|---|
| Telefonbuch | API | `AddressBook.framework` |
| | Datei | /private/var/mobile/Library/AddressBook/ AddressBook.sqlitedb |
| Zuletzt gewählter Kontakt | Datei | /var/mobile/Library/Preferences/ com.apple.mobilephone.plist, Key: `AddressBookLastDialedUid` |
| Zuletzt gewählte Nummer | Datei | /var/mobile/Library/Preferences/ com.apple.mobilephone.plist, Key: `DailerSavedNumber` |
| Weiterleitungs-Rufnummer | Datei | /var/mobile/Library/Preferences/ com.apple.mobilephone.plist, Key: `call-forwarding-number` |

Tabelle A.3.: Datenquellen für Kontaktdaten und Telefonummern

### A.1.4. Kalender

| Datum | Art | Zugriff |
|---|---|---|
| Telefonbuch | API | `EventKit.framework` |
| | Datei | /private/var/mobile/Library/Calendar/ Calendar.sqlitedb |

Tabelle A.4.: Datenquellen für Termindaten

### A.1.5. Ortungsdienst

| Datum | Art | Zugriff |
|---|---|---|
| Aktuelle Position | API | `CoreLocation.framework`, `MapKit.framework` |
| Wetter-Städte | Datei | /var/mobile/Library/Preferences/ com.apple.weather.plist, Key: `Cities` |
| Letzte Positionsdaten aus Maps | Datei | /var/mobile/Library/Preferences/ com.apple.Maps.plist, Keys: `LastViewedLatitude`, `LastViewedLongitude` |

| | | |
|---|---|---|
| Letzte Routendaten aus Maps | Datei | /var/mobile/Library/Preferences/ com.apple.Maps.plist, Keys: `RouteStartString`, `RouteEndString` |

Tabelle A.5.: Datenquellen für Positionsdaten

### A.1.6. Fotos und Videos

| **Datum** | **Art** | **Zugriff** |
|---|---|---|
| Fotos und Videos | Datei | /User/Media/DCIM/* |

Tabelle A.6.: Datenquellen für Foto- und Videodaten

### A.1.7. Tastatur

| **Datum** | **Art** | **Zugriff** |
|---|---|---|
| Keyboard-Cache | Datei | /var/mobile/Library/Keyboard/* |

Tabelle A.7.: Datenquellen für Tastaturdaten

### A.1.8. WLAN

| **Datum** | **Art** | **Zugriff** |
|---|---|---|
| Liste bekannter Netzwerke | Datei | /Library/Preferences/SystemConfiguration/ com.apple.wifi.plist |

Tabelle A.8.: Datenquellen für WLAN-Daten

### A.1.9. Email

| **Datum** | **Art** | **Zugriff** |
|---|---|---|
| Liste der eingerichteten Email-Accounts (ohne Passwort) | Datei | /var/mobile/Library/Preferences/ com.apple.accountsettings.plist |

Tabelle A.9.: Datenquellen für Email-Daten

# Eidesstattliche Erklärung

Ich, Markus Troßbach, Matrikel-Nr. 172087, versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema

> *Entwurf und Implementierung eines Konzepts zur Absicherung mobiler Geräte am Beispiel von Apple iOS*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Heilbronn, den 29. Februar 2012

Markus Troßbach