SNOOP-IT: Dynamische Analyse und Manipulation von Apple iOS Apps

Andreas Kurtz Markus Troßbach Felix C. Freiling

Department Informatik
Friedrich-Alexander-Universität
Martensstr. 3, 91054 Erlangen

{andreas.kurtz,felix.freiling}@cs.fau.de, markus@trossbach.net

Abstract: Applikationen für mobile Endgeräte (Apps) verarbeiten vielfach Daten, die unsere Privatsphäre betreffen. Bislang existieren aber kaum Werkzeuge, mit denen auf einfache Weise geprüft werden kann, wie Apps mit diesen Daten umgehen und ob dabei möglicherweise unsere Privatsphäre beeinträchtigt wird. Wir stellen das Werkzeug SNOOP-IT vor, mit dem bestehende Apps für die Apple iOS Plattform zur Laufzeit untersucht werden können. Am Beispiel einer populären App aus dem Apple App Store zeigen wir, wie unser Werkzeug dynamische Sicherheitsanalysen erleichtert und dabei unterstützt, Verletzungen der Privatsphäre effizient ausfindig zu machen.

1 Einleitung

In den vergangenen Jahren sind mobile Endgeräte wie Smartphones oder Tablet PCs ein fester Bestandteil unserer Lebenswelt geworden. Ein Grund für diesen Erfolg sind mitunter die vielfältigen Einsatzmöglichkeiten, die sich aus der Erweiterbarkeit durch Apps ergeben. Häufig sind sich App-Nutzer dabei allerdings der Risiken nicht bewusst, die sich beispielsweise aus dem Verlust ihres mobilen Endgeräts oder aus der unachtsamen Installation von Drittanbieter-Apps ergeben [BR13]. Dies gilt insbesondere für die Apple iOS Plattform, deren Quelltext im Gegensatz etwa zu Android, nicht öffentlich verfügbar ist. Insbesondere für diese Plattform existieren deshalb bislang kaum Werkzeuge, die es Sicherheitsfachleuten erlauben, iOS Apps effizient zu analysieren. Defizite innerhalb von Apps bleiben daher vielfach unentdeckt oder werden eher zufällig bzw. unter hohem Zeitaufwand gefunden [Tha, Ley, Bil].

Eine vergleichbare Ausgangssituation bestand bis vor wenigen Jahren auch bei der Analyse von Schadsoftware für Computer mit dem Windows-Betriebssystem. Dort haben sich aber mittlerweile dynamische Analysewerkzeuge etabliert, bei denen das zu untersuchende Programm unter kontrollierten Bedingungen ausgeführt wird und im Hintergrund sämtliche Aktivitäten, wie beispielsweise Dateisystem- oder Netzwerkzugriffe, zur Laufzeit überwacht werden [EGSF12, WHF07]. Obwohl sich die dynamische Analyse dabei als besonders effizient erwiesen hat, gibt es bislang kaum Werkzeuge zur Anwendung dieser Verfahren für mobile Apps.

1.1 Verwandte Arbeiten

In der Literatur gibt es verschiedene Arbeiten, die Ansätze zur Analyse mobiler Apps vorstellen. Die Forschung hat sich dabei in den vergangenen Jahren allerdings hauptsächlich mit der quelloffenen Android Plattform beschäftigt. Dort existieren Systeme wie Taintdroid [EGgC+10], Andrubis [Int13] oder Mobile Sandbox [SFE+13], die bereits jetzt eine automatisierte, dynamische Analyse von Android Apps erlauben.

Für Apples iOS Plattform ist das Angebot an Werkzeugen hingegen überschaubar. Neben ersten Ansätzen zur automatisierten dynamischen Analyse [SEKV12, JM12], existieren letztendlich kaum Werkzeuge, um Experten bei der zielgerichteten, manuellen Sicherheitsanalyse zu unterstützen. Die Werkzeuge Introspy [iSE] und iAuditor [MDS] verwenden API Hooking Techniken, um sicherheitsrelevante Vorgänge einer App aufzuzeichnen. Beide Werkzeuge beschränken sich dabei aber auf die Protokollierung grundlegender API-Aufrufe und bieten beispielsweise keine Möglichkeit zum Methoden-Tracing oder um mit der App zur Laufzeit zu interagieren. Darüber hinaus werden die von Introspy aufgezeichneten Daten nicht in Echtzeit ausgegeben und die von iAuditor registrierten API-Aufrufe werden lediglich in eine Datei protokolliert.

Das Werkzeug iNalyzer [Lab] beschränkt sich weitestgehend darauf, vorhandene App-Methoden aufzulisten und ermöglicht eine Ausführung dieser App-Methoden über eine Weboberfläche. Die Werkzeuge Aspective-C [Frea] und Subjective-C [Ken] ermöglichen ausschließlich eine Überwachung von Objective-C-Methodenaufrufe, beschränken sich dabei aber nicht auf App-eigene Methoden bzw. wichtige iOS API-Methoden, was ihre Nützlichkeit im Rahmen der App-Aanalyse einschränkt.

1.2 Resultate

In diesem Beitrag stellen wir das Werkzeug SNOOP-IT vor, das erste dynamische Analysewerkzeug zur zielgerichteten Analyse von Apps der Apple iOS Plattform, das Untersuchungsergebnisse in Echtzeit präsentiert. SNOOP-IT erweitert bestehende Apps mittels *Library Injection* zur Laufzeit um zusätzliche Funktionen zur Sicherheitsanalyse und kann somit Zugriffe einer App auf sicherheits- und privatsphärerelevante Methoden der iOS Plattform mittels *API Hooking* überwachen. Unser Werkzeug stellt dabei eine webbasierte Oberfläche zur Verfügung, um mit der überwachten App in Echtzeit zu interagieren. Durch eine XML-RPC-Schnittstelle ist es zudem möglich, automatisiert auf die aufgezeichneten Daten sowie interne App-Zustände zuzugreifen. Anhand der Analyse einer populären App zur Verwaltung von Passwörtern zeigen wir, wie SNOOP-IT zur zielgerichteten Analyse von sicherheitsrelevanter App-Funktionalität eingesetzt werden kann.

1.3 Ausblick

In Abschnitt 2 werden zunächst technische Hintergründe zur iOS Plattform und zum verwendeten API Hooking erläutert. Anschließend werden in Abschnitt 3 die einzelnen Komponenten von SNOOP-IT sowie deren Funktionsweise beschrieben. Im Rahmen der anschließenden Evaluation (Abschnitt 4) dokumentieren wir die Analyse der genannten App. Abschnitt 5 fasst die Ergebnisse zusammen.

SNOOP-IT ist unter der BSD Lizenz frei verfügbar und kann über die Projektseite `https://code.google.com/p/snoop-it/` heruntergeladen und ausprobiert werden.

2 Technische Hintergründe

In diesem Abschnitt werden technische Hintergründe zur iOS Plattform sowie Details zum eingesetzten API Hooking erläutert.

2.1 Objective-C

Alle Apps für das Apple iOS Betriebssystem werden in Objective-C entwickelt. Dabei handelt es sich um eine objektorientierte Erweiterung der Programmiersprache C. Darüber hinaus bietet Objective-C umfassende Möglichkeiten, um mit der zugrundeliegenden Ausführungsumgebung zu interagieren: Ein in Objective-C entwickeltes Programm kann beispielsweise sämtliche Werte und Methoden eigener Objekte zur Laufzeit einsehen oder auch verändern. Diese als Reflection bzw. Introspection bekannten Konzepte werden von der Objective-C Laufzeitumgebung zur Verfügung gestellt [Appb]. Die Funktionen der Laufzeitumgebung werden dabei beim Erstellen einer App über eine dedizierte Programm-Bibliothek (*libobjc.A.dylib*) zur App hinzu gelinkt.

Eine der wichtigsten Aufgaben dieser Laufzeitumgebung ist die Vermittlung von Nachrichten zwischen bestehenden Objekten (*Message Passing*). Beim Aufruf einer Methode wird eine Nachricht an das jeweilige Zielobjekt gesendet. Die Nachricht enthält dabei den Namen der aufzurufenden Methode sowie eine Liste der zugehörigen Parameterwerte. Um die Nachrichten zu vermitteln, stellt die Laufzeitumgebung über die Funktion `objc_msgSend` einen zentralen Dispatcher zur Verfügung. Der Aufruf einer App-Methode bzw. einer Methode der iOS-API hat folglich immer mindestens einen Aufruf dieses zentralen Dispatchers zur Folge.

2.2 Jailbreak

Mittels eines Jailbreaks ist es unter Ausnutzung von Software-Schwachstellen in der iOS-Plattform möglich, die Nutzungsbeschränkungen des iOS-Betriebssystems zu entfernen und administrative Rechte auf einem mobilen Endgerät zu erlangen. Darüber hinaus ermöglicht ein Jailbreak die Installation beliebiger Software (Analysewerkzeuge), die nicht von Apple signiert bzw. freigegeben wurde und bietet einen direkten Zugang zur Objective-C Laufzeitumgebung. Aus diesen Gründen erfolgt eine App-Analyse in der Praxis typischerweise auf Geräten mit durchgeführtem Jailbreak. Auch SNOOP-IT setzt zum Betrieb ein iOS-Gerät mit durchgeführtem Jailbreak voraus.

2.3 API Hooking

Um eine iOS App zur Laufzeit analysieren oder manipulieren zu können, muss zunächst der bestehende App-Programmcode um zusätzliche Funktionalität erweitert werden. Dies geschieht durch das Einbringen einer zusätzlichen Bibliothek (*Library Injection*). Dabei wird beim Start einer App der Dynamic Linker des iOS Betriebssystems über die Umgebungsvariable `DYLD_INSERT_LIBRARIES` angewiesen, den Code der angegebenen Bibliothek in den Adressraum der App zu laden. Anschließend muss bei der Initialisierung der Bibliothek sichergestellt werden, dass der eingeschleuste Programmcode auch tatsächlich aufgerufen wird (vgl. Abbildung 1). Dazu werden bestehende Methoden- und Funktionsaufrufe bzw. Zeiger auf die entsprechenden Codebereiche zur Laufzeit überschrieben, so dass anschließend anstatt der jeweiligen API Methode der zuvor über die Bibliothek eingeschleuste Code ausgeführt wird (*Runtime Patching*). Zur Unterstützung des API Hooking wird in SNOOP-IT das Mobile Substrate Framework verwendet [Freb].

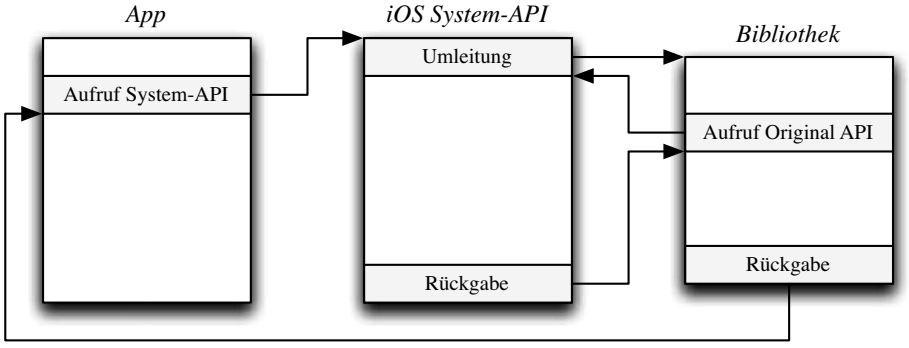

Abbildung 1: Überschreiben von iOS Methoden- bzw. Funktionsaufrufen zur Laufzeit mittels API Hooking.

3 Aufbau und Funktionsweise von SNOOP-IT

3.1 Grundprinzip

SNOOP-IT verwendet *Library Injection* und API Hooking zur Laufzeit, um zusätzliche Funktionalität zur Sicherheitsanalyse in eine iOS App einzubringen: Beim Start einer App wird diese durch das Einschleusen einer Bibliothek nachträglich um Analysefunktionen erweitert. Während der anschließenden Ausführung und Bedienung der App werden im Hintergrund zentrale sicherheits- und privatsphärerelevanten Vorgänge aufgezeichnet.

Um auf die aufgezeichneten Daten zuzugreifen und mit der App zu interagieren, wird innerhalb der App zusätzlich ein Webserver gestartet. Dieser Webserver stellt eine XML-RPC-Schnittstelle zur Verfügung, über die in Echtzeit auf die aufgezeichneten Ergebnisse und die Analysefunktionen zugegriffen werden kann. Zur einfachen Bedienung wird von dem Webserver zusätzlich eine webbasierte Benutzeroberfläche auf Basis des Google Web Toolkits [Sto12, gwt] bereitgestellt. Diese Benutzeroberfläche wiederum interagiert mittels AJAX mit der von der Bibliothek bereitgestellten Webservice-Schnittstelle und ermöglicht darüber einen einfachen Zugriff auf die SNOOP-IT-Funktionen.

Zur Auswahl der zu untersuchenden Apps stellt SNOOP-IT selbst eine eigene Konfigurations-App zur Verfügung. Darin werden sämtliche auf dem Analysegerät befindlichen Apps aufgelistet und können zur späteren Analyse markiert werden. Darüber hinaus können über die Konfigurations-App auch grundlegende Einstellungen wie beispielsweise der Netzwerkport des Webservers oder eine Authentifizierung zum Zugriff auf die Webservice-Schnittstelle vorgenommen werden.

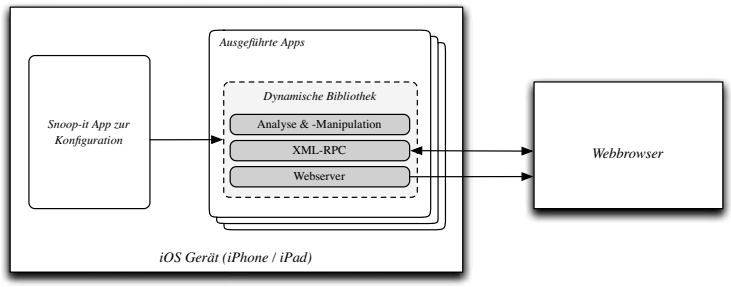

Abbildung 2: Durch das Einbringen einer dynamischen Bibliothek werden Apps zur Laufzeit um Funktionen zur Sicherheitsanalyse erweitert. Analyseergebnisse können in Echtzeit über einen Webbrowser eingesehen werden.

Abbildung 2 gibt einen Überblick über die in SNOOP-IT integrierten Komponenten: Nach dem Anpassen der Konfiguration, muss lediglich die jeweils zu untersuchende App gestartet und anschließend normal bedient werden. Während des Startvorgangs wird die SNOOP-IT-Programmbibliothek in die App integriert und die Überwachung gemäß der zuvor getroffenen Einstellungen begonnen. Der anschließende Zugriff auf die Analyseergebnisse

sowie die Interaktion mit der App zur Laufzeit erfolgt durch Aufruf der von SNOOP-IT bereitgestellten Weboberfläche im Browser.

Alternativ zum Zugriff über die Weboberfläche können die von dem Webservice bereitgestellten Funktionen auch von anderen Client-Anwendungen bzw. Werkzeugen direkt aufgerufen werden. Dazu findet sich auf der Projektseite von SNOOP-IT eine ausführliche Dokumentation der XML-RPC-Schnittstelle.

3.2 Bereitgestellte Funktionalität

In diesem Abschnitt werden die wichtigsten von SNOOP-IT bereitgestellten Funktionen erläutert.

3.2.1 Überwachung von API- und Systemaufrufen

Wird eine App um die Analysefunktionen erweitert, so werden anschließend zentrale App-Vorgänge überwacht und protokolliert. Dabei werden unter anderem sämtliche Zugriffe auf das Dateisystem und die iOS Keychain [Appa] registriert. In diesem Zusammenhang wird unter anderem auch ermittelt, ob eine App die vom iOS Betriebssystem bereitgestellten Verschlüsselungsmechanismen nutzt. Zahlreiche Filter innerhalb der grafischen Benutzeroberfläche erleichtern anschließend die Suche nach sensitiven App-Daten, die möglicherweise unverschlüsselt gespeichert werden und bei Verlust eines Geräts ein Risiko darstellen können.

Darüber hinaus überwacht SNOOP-IT den Netzwerkverkehr einer App sowie Aufrufe sämtlicher API-Methoden, über die eine App auf persönliche Informationen des Nutzers zugreifen kann (Kontakte, Fotos, Mikrofon, Kamera, Aufenthaltsort, Geräte-IDs etc.). Mit Hilfe dieser Überwachung kann relativ einfach nachvollzogen werden, ob eine App beispielsweise auf persönliche Daten oder sensitive Gerätesensoren zugreift.

3.2.2 Verfolgung des Kontrollflusses

Eine weitere Funktion in SNOOP-IT ermöglicht detaillierte Einblicke in den Kontrollfluss einer App. Dazu werden sämtliche Methodenaufrufe und zugehörige Parameterwerte protokolliert (*Tracing*). Auf diese Weise können beispielsweise interne Abläufe eingesehen werden, ohne dass dazu der eigentliche App-Quelltext benötigt wird.

Um Methodenaufrufe und zugehörige Parameterwerte aufzuzeichnen, werden sämtliche Nachrichten an den zentralen Dispatcher der Objective-C Laufzeitumgebung abgefangen. Mittels Hooking werden dabei Aufrufe der Dispatcher-Funktion `objc_msgSend` (siehe Abschnitt 2.1) überwacht. Dabei sind insbesondere die ARM CPU-Register $r0$ und $r1$ von Interesse, da diese Register beim Aufruf des Dispatchers jeweils einen Zeiger auf das Zielobjekt ($r0$) und einen Zeiger auf den Namen der auszuführenden Methode ($r1$) vorhalten. Da ein Protokollieren dieser Werte (beispielsweise in eine Datei) die Registerinhalte aber verändert würde, müssen die Zeiger vor dem Aufruf der Protokollierungsroutine

entsprechend gesichert werden. Dazu wird im Heap-Speicher eine Stack-Datenstruktur angelegt, um die original Registerwerte auf Instruktionsebene auf diesem alternativen Stack zu sichern. Nach der Protokollierung der Zeiger können die ursprünglichen CPU-Registerinhalte wiederhergestellt werden, bevor anschließend die normale Ausführung der App fortgesetzt wird.

Ein Nachteil dieses Ansatzes ist, dass dabei auch sämtliche Hintergrundaktivitäten der Laufzeitumgebung erfasst werden (beispielsweise Methodenaufrufe bei der Berührung der Displayoberfläche). Um dieses Funktionsrauschen auszublenden, wird in SNOOP-IT die Ausgabe des Tracings auf App-eigene Methoden und wichtige Funktionen der iOS Plattform beschränkt. Dazu werden zunächst über die Objective-C Laufzeitumgebung sämtliche von einer App bereitgestellten Klassen und Methoden ermittelt. Bei der Protokollierung der Nachrichtenaufrufe werden anschließend lediglich solche Aufrufe berücksichtigt, die von der App selbst oder in erster Instanz von der iOS API bereitgestellt werden.

3.2.3 App-Manipulation zur Laufzeit

Neben diesen Analysefunktionen stellt SNOOP-IT auch diverse Möglichkeiten bereit, um Apps zur Laufzeit zu manipulieren. So können über die Weboberfläche beispielsweise Hardware-Merkmale wie die Geräte-ID, die WLAN MAC-Adresse, der Gerätetyp oder der aktuelle Aufenthaltsort einfach vorgetäuscht werden. Darüber hinaus ermittelt SNOOP-IT aus dem Programmcode einer App alle verfügbaren Klassen und Methodennamen und stellt diese als Baumstruktur dar. Dabei werden auch Initialisierungen von Klassen überwacht und jeweils eine Referenz auf die neu angelegten Instanzen zwischengespeichert. Dadurch können anschließend vorhandene Instanzen bzw. Klassen über die Weboberfläche ausgewählt und deren Methoden beliebig aufgerufen werden.

4 Evaluation

Die Möglichkeiten, die sich aus der zielgerichteten, dynamischen Analyse von Apps mittels SNOOP-IT ergeben, sind vielfältig. So kann zur Laufzeit beispielsweise sehr einfach festgestellt werden, auf welche persönliche Daten oder sensitive Gerätesensoren eine App zugreift. Diese Informationen können anschließend verwendet werden, um daraus wiederum potentielle Privatsphäreverletzungen abzuleiten. Im Gegensatz zu automatisierten Ansätzen [SEKV12, SFE+13] versucht SNOOP-IT dabei nicht, schadhaftes Verhalten automatisiert aufzudecken und zu bewerten. Vielmehr ermöglicht es unser Werkzeug einem Experten auf sehr effiziente Weise, interne Abläufe einer App zu untersuchen und daraus Rückschlüsse auf mögliche Schadfunktionen zu ziehen.

Ferner unterstützt SNOOP-IT dabei, technische Defizite innerhalb der Implementierung einer App aufzudecken, die ihrerseits wiederum Auswirkungen auf die Privatsphäre haben können. Um beispielsweise Integrität und Vertraulichkeit lokal gespeicherter Daten auch bei Verlust eines mobilen Endgeräts weiterhin gewährleisten zu können, werden sensitive App-Inhalte meist verschlüsselt. In diesem Zusammenhang wird bei Prüfungen häufig der

Fragestellung nachgegangen, ob die hinterlegten Daten durch das eingesetzte Verschlüsselungsverfahren effektiv geschützt werden. Am Beispiel einer populären App zur Verwaltung von PINs und Passwörtern demonstrieren wir im Folgenden, wie SNOOP-IT dazu verwendet werden kann, solchen Fragestellungen nachzugehen und sicherheitsrelevante Implementierungsfehler effizient aufzudecken.

Fallbeispiel: App zur sicheren PIN- und Passwortverwaltung

Die im Apple App Store verfügbare App *iPIN* wirbt damit, sämtliche "Daten (..) mit dem Advanced Encryption Standard und einer Schlüssellänge von 256 Bit" zu verschlüsseln [IBI]. Der Zugriff auf die verschlüsselten Daten soll erst dann möglich werden, wenn über die integrierte "Sensor-Tastatur" ein zuvor aufgezeichnetes Entsperrmuster eingegeben wurde.

Durch die in SNOOP-IT integrierte Dateisystemüberwachung (vgl. Abbildung 3) wurde festgestellt, dass die App direkt beim Start auf die Datei `iPinSecurityModel.dat` zugreift. Diese Datei wird nicht über die Verschlüsselungs-API des iOS Betriebssystems geschützt (*NSFileProtectionNone*), weshalb die darin enthaltenen Daten bei Verlust eines Geräts ausgelesen werden könnten. Ein Zugriff auf die Datei zeigte allerdings, dass deren Inhalte binär bzw. kodiert vorliegen und nicht unmittelbar einsehbar sind.

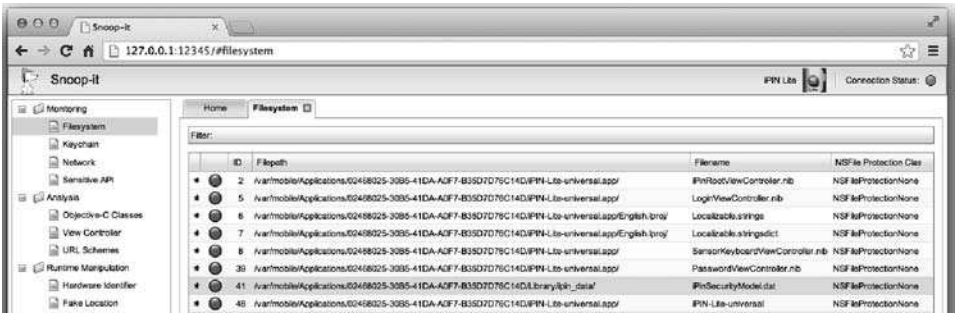

Abbildung 3: Überwachung von Dateisystemzugriffen

Mit Hilfe der in SNOOP-IT integrierten Funktion zur Verfolgung von Methodenaufrufen wurde anschließend untersucht, wie diese Datei nach dem Einlesen weiterverarbeitet wird. Listing 1 zeigt, dass direkt nach dem Einlesen der Datei mittels Key Stretching (*PBKDF2*) aus einer im App-Programmcode statisch hinterlegten Zeichenkette ein Schlüssel abgeleitet wird (vgl. Zeile 3). Unter Verwendung dieses Schlüssels werden die Daten der Datei anschließend entschlüsselt und die entschlüsselten Werte im Objekt *iPINModel* zwischengespeichert (vgl. Zeile 4). Wie sich der Ausgabe entnehmen lässt, befindet sich in der Datei unter anderem der MD5-Hashwert des erwarteten Entsperrmusters (vgl. *sensorHash* in Zeile 7).

```
+ [iPinModel(0x90f68) initFromFile]
+ [iPinModel(0x90f68) securityModelFilePath]
+ [PBKDF2(0x9124c) getKeyForPassphrase:], args: <__NSCFConstantString 0x92160:
    [initForWritingWithMutableData]>
+ [iPinModel(0x90f68) initSharedModelWithUnarchiver:withObjectKey:], args: <0
    x2002aef0>, <__NSCFConstantString 0x92150: iPINModel>
+ [iPinModel(0x90f68) sharedModel]
- [iPinModel(0x200e2130) initWithCoder:], args: <0x2002aef0>
- [iPinModel(0x200e2130) setSensorHash:], args: <__NSCFString 0x2002a630:
    8CF37F50FB1A7943FBA8EAA20FFF1E56>
```

Listing 1: Methodenaufrufe beim Start der App zeigen die Verwendung eines im Programmcode der App statisch hinterlegten Schlüssels

Weitere Analysen mithilfe der Tracing-Funktion ergaben, dass die von der Sensor-Tastatur bereitgestellten Tasten durchnummeriert sind (vgl. Abbildung 4). Bei jedem Tastendruck wird der aktuelle Wert mit den zuvor gedrückten Tastenwerten zusammengeführt und davon ein MD5-Hashwert errechnet. Dieser wird mit dem zuvor aus der Datei ermittelten Hashwert (*sensorHash*) verglichen.

Da das erwartete Entsperrmuster in Form des MD5-Hashwerts auf dem Gerät selbst gespeichert wird und der zugehörige Wertebereich relativ klein ist, werden dadurch Brute-Force-Angriffe begünstigt. Ein beliebiges Muster bestehend aus neun gedrückten Sensortasten ließ sich in Praxistests über ein Python-Script in nur wenigen Sekunden ermitteln. Der eingegebene Sensor-Code wird von der App anschließend verwendet, um daraus erneut einen Schlüssel abzuleiten und damit die eigentlichen App-Daten zu entschlüsseln.

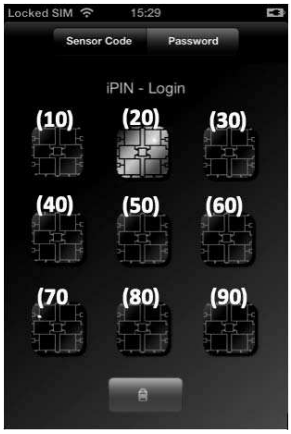

Abbildung 4: Sensor-Tastatur zur Eingabe eines Entsperrmusters

Da durch diesen Implementierungsfehler die Authentifizierung und App-eigene Verschlüsselung umgangen werden kann, können Integrität und Vertraulichkeit der durch die App verarbeiten Daten bei Verlust eines mobilen Endgeräts nicht sichergestellt werden. Direkt nach dem Entdecken der Schwachstellen wurde daher der Hersteller der App darüber informiert. Seit Anfang Juli 2013 steht eine aktualisierte Version der App zur Verfügung, in der die Befunde durch Überarbeitung des Verschlüsselungsverfahrens adressiert wurden.

5 Zusammenfassung

Die dynamische Analyse von iOS Apps zur Laufzeit gewährt detaillierte Einblicke in deren Innenleben. Verletzungen der Privatsphäre können dabei häufig ebenso einfach identifiziert werden, wie sicherheitsrelevante Implementierungsfehler. Mit SNOOP-IT stellen wir ein einfach zu bedienendes Werkzeug zur Verfügung, um solche Defizite zukünftig effizienter ermitteln zu können. Neben diesen technischen Analysen, ermöglicht die Interaktion zur Laufzeit aber auch völlig neue Angriffswege und zwar immer dann, wenn Sicherheitsmaßnahmen innerhalb einer App umgesetzt werden. Apps sollten daher verstärkt unter dem Bewusstsein entwickelt werden, dass sicherheitsrelevante Vorgänge auf Client-seite (wie statische Verschlüsselungsschlüssel, hartkodierte Zugangsdaten, vorgelagerte Anmeldemasken etc.) beliebig eingesehen, manipuliert und umgangen werden können.

Literatur

[Appa] Apple. Keychain Services Programming Guide. `https://developer.apple.com/library/ios/documentation/security/conceptual/keychainServConcepts/01introduction/introduction.html`.

[Appb] Apple. Objective-C Runtime Reference. `https://developer.apple.com/library/mac/documentation/cocoa/reference/objcruntimeref/Reference/reference.html`.

[Bil] Nick Bilton. Apple Loophole Gives Developers Access to Photos. `http://bits.blogs.nytimes.com/2012/02/28/tk-ios-gives-developers-access-to-photos-videos-location/`.

[BR13] Zinaida Benenson und Lena Reinfelder. Should the Users be Informed? On Differences in Risk Perception between Android and iPhone Users. In *Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[EGgC+10] William Enck, Peter Gilbert, Byung gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel und Anmol Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In Remzi H. Arpaci-Dusseau und Brad Chen, Hrsg., *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*, Seiten 393–407. USENIX Association, 2010.

[EGSF12] Markus Engelberth, Jan Göbel, Christian Schönbein und Felix C Freiling. PyBox-A Python Sandbox. In *Sicherheit*, Seiten 137–148, 2012.

[Frea] Jay Freeman. Aspective-C. `http://svn.saurik.com/repos/menes/trunk/aspectivec/AspectiveC.mm`.

[Freb] Jay Freeman. Mobile Substrate. `http://www.cydiasubstrate.com/`.

[gwt] Google Web Toolkit. `http://www.gwtproject.org/`.

[IBI] IBILITIES, INC. iPIN - Secure PIN & Passwort Safe App. `https://itunes.apple.com/de/app/ipin-secure-pin-passwort-safe/id379865087`.

[Int13] International Secure Systems Lab. Anubis: Analyzing Unknown Binaries. online http://anubis.iseclab.org/, November 2013.

[iSE] iSECPartners. Introspy. https://github.com/iSECPartners/introspy/.

[JM12] Mona Erfani Joorabchi und Ali Mesbah. Reverse engineering iOS mobile applications. In *Reverse Engineering (WCRE), 2012 19th Working Conference on*, Seiten 177–186. IEEE, 2012.

[Ken] KennyTM˙ Subjective-C. http://networkpx.blogspot.de/2009/09/introducing-subjective-c.html.

[Lab] AppSec Labs. iNalyzer. https://appsec-labs.com/iNalyzer.

[Ley] John Leyden. D'OH! Use Tumblr on iPhone or iPad, give your password to the WORLD. http://www.theregister.co.uk/2013/07/17/tumblr_ios_uncryption/.

[MDS] MDSec. iAuditor. https://github.com/mdsecresearch.

[SEKV12] Martin Szydlowski, Manuel Egele, Christopher Kruegel und Giovanni Vigna. Challenges for dynamic analysis of iOS applications. In *Open Problems in Network Security*, Seiten 65–77. Springer, 2012.

[SFE+13] Michael Spreitzenbarth, Felix C. Freiling, Florian Echtler, Thomas Schreck und Johannes Hoffmann. Mobile-sandbox: having a deeper look into android applications. In Sung Y. Shin und José Carlos Maldonado, Hrsg., *SAC*, Seiten 1808–1815. ACM, 2013.

[Sto12] Sebastian Stocker. Entwurf einer grafischen Oberfläche zur Laufzeitmanipulation von iOS Apps. Bachelor Thesis, Universität Heidelberg, Hochschule Heilbronn, Medizinische Informatik, 2012.

[Tha] Arun Thampi. 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.

[WHF07] Carsten Willems, Thorsten Holz und Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *Security & Privacy, IEEE*, 5(2):32–39, 2007.

Sämtliche angegebenen Online-Quellen wurden zuzletzt am 26.01.2014 erfolgreich abgerufen.