30. Dezember 2015 Studie · Mobile Sicherheit · 23 Min. Lesezeit

Analyse aktueller Frameworks zur Entwicklung von mobilen Cross-Plattform-Applikationen

Archiviert. Dieser Inhalt wird nicht mehr aktualisiert. Er bleibt zur Information verfügbar.

Quelle https://technology2.a-sit.at/veroeffentlichungen/frameworks-fuer-mobile-cross-plattform-apps/

In dieser Studie wird analysiert, welche Sicherheitsmechanismen in beliebten Cross-Plattform-Frameworks zur Verfügung stehen. Betrachtet werden die beiden am häufigsten verwendeten Cross-Plattform-Frameworks, Apache Cordova und Xamarin, sowie Alpha Anywhere. Alpha Anywhere wurde zusätzlich ausgewählt, weil es explizit mit Sicherheitsfeatures wirbt. Die ausgesuchten Frameworks decken die beiden populären Ansätze zu Erstellung von Cross-Plattform-Applikationen ab. Apache Cordova und Alpha Anywhere erzeugen interpretierte Applikationen, während Xamarin hybride Applikationen erzeugt.

Um die Frameworks vergleichen zu können, wurde ein Satz an Sicherheitsmechanismen definiert und analysiert, ob und wie diese Sicherheitsmechanismen in den jeweiligen Frameworks verwendet werden können. Dabei zeigten sich große Unterschiede zwischen den beiden Ansätzen zur Erstellung von Cross-Plattform-Applikationen.

1. Einleitung

Der Markt für Mobilgeräte ist sehr fragmentiert. 82% der im 2. Quartal 2015 verkauften Smartphones basieren auf dem Betriebssystem Android (wobei eine Vielzahl an Versionen im Umlauf ist), 14% laufen mit Apple iOS und 2,6% mit Windows Phone. Um mobile Applikationen für mehrere mobile Plattformen anbieten zu können, müssen Applikationen separat für jede zu unterstützende Plattform implementiert werden. Die Plattformen unterscheiden sich jedoch signifikant in Bezug auf unterstützte Programmiersprachen, Frameworks und Build-Tools. Applikationsentwicklerinnen und -entwickler müssen sich dazu die Architekturkonzepte und das Wissen über Frameworks mehrerer Plattformen aneignen. Die Entwicklung für mehrere Plattformen gestaltet sich daher zeitaufwendig und kostspielig.

Um den Entwicklungsprozess für mehrere mobile Plattformen zu vereinfachen, wurden in den letzten Jahren mehrere Ansätze zur sogenannten *Cross-Plattform-Entwicklung* gefunden. Cross-Plattform-Frameworks verfolgen das Ziel, Quellcode für andere Plattformen wiederverwenden zu können. Generell können zwei populäre Ansätze zur Cross-Plattform-Entwicklung unterschieden werden. Einerseits die Entwicklung von

hybriden Applikationen, wobei Applikationscode in einem in der Applikation eingebetteten Browser ausgeführt wird. Andererseits die Entwicklung von interpretierten Applikationen, wobei Applikationen eine eigene Laufzeitumgebung inkludieren und Applikationscode zur Laufzeit interpretiert wird.

Apache Cordova [1] ist das bekannteste Framework zur Entwicklung von hybriden Applikationen. Mehrere andere Cross-Plattform-Frameworks basieren auf Apache Cordova, beispielsweise Alpha Anywhere [2] und Chrome Apps for Mobile [3]. Beispiele für den interpretierten Ansatz sind Appcelerator Titanium [4], Xamarin [5], Adobe Air [6] und Unity 3D [7] mit speziellem Fokus auf Spieleentwicklung. Laut einer von VisionMobile [8] durchgeführten Studie stellt Apache Cordova das am meisten verwendete Cross-Plattform-Framework dar, gefolgt von Xamarin und Unity 3D.

Aktuell ist wenig über Sicherheitsaspekte von Applikationen bekannt, die mithilfe von Cross-Plattform-Frameworks entwickelt wurden. Zwar behandeln einige Publikationen Sicherheitsprobleme der integrierten WebView-Komponenten und betreffen somit lediglich hybride Frameworks, jedoch sind keine ganzheitlichen Sicherheitsanalysen von mittels Cross-Plattform-Frameworks entwickelter Applikationen bekannt. Bei der Betrachtung der Sicherheit von Cross-Plattform-Applikationen ist besonders die Verbindung der Framework APIs zu den APIs der darunterliegenden Plattform von Interesse. Falls das Framework Generalisierungen der Plattform APIs z.B. zur sicheren Speicherung von Zugangsdaten anbietet, sind die Umsetzungen auf den einzelnen Plattformen von Relevanz. Generell enthalten die offiziellen Dokumentationen der verschiedenen Cross-Plattform-Frameworks wenig Informationen zu Sicherheitsthemen.

Bei zunehmender Verwendung von Frameworks zur Entwicklung von Cross-Plattform-Applikationen steigt somit auch das Sicherheitsrisiko, das von derartigen Frameworks ausgeht. Sicherheitslücken in einem Framework können somit Angriffsvektoren für eine große Anzahl an Applikationen darstellen. Ziel dieser Studie ist es, mehrere Cross-Plattform-Frameworks bezüglich verfügbarer Sicherheitsfunktionen zu analysieren. Dazu werden im nachfolgenden Abschnitt mehrere Sicherheitsanforderungen definiert. Anhand dieser Anforderungen werden die Frameworks Apache Cordova, Xamarin und Alpha Anywhere analysiert. Apache Cordova und Xamarin wurden gewählt, da es sich hierbei um die beiden beliebtesten hybriden bzw. interpretierten Frameworks handelt. Zusätzlich wurde eine Kurzanalyse von Alpha Anywhere durchgeführt, da dieses Framework explizit mit Sicherheitsfunktionen wirbt.

2. Anforderungen

Um Sicherheitsaspekte von Cross-Plattform-Frameworks zu analysieren, wurde ein Satz an Sicherheitsmechanismen extrahiert.

1. Zugriff auf Schlüsselspeicher oder anderweitig geschützte Speicher 2) Zugriff auf kryptographische Funktionen 3) Verfügbarkeit von Root Checks 4) Verfügbarkeit von Tools zur Code Obfuskierung 5) Schutz vor Repacking und Modifikation von Applikationspaketen 6) Sicherheitsaspekte SSL/TLS a. Unterstützte Algorithmen b. Verwendung eigener Truststores c. Verfügbarkeit von Certificate Pinning d. Unterstützung für selbst-signierte Zertifikate 7) Anfälligkeit für Code Injection 8) Verfügbarkeit von Per-App-VPN Services Die nachfolgenden Abschnitte beschreiben die untersuchten Frameworks.

3. Xamarin

Mittels Xamarin [5] können Entwicklerinnen und Entwickler native Applikationen für Android, iOS und Windows Phone entwickeln. Die Applikationen werden in C# geschrieben und können auf alle nativen APIs zugreifen. Zusätzlich stellt Xamarin native Bedienelemente bereit. Die erzeugten nativen Applikationen können über die offiziellen Applikationsstores der jeweiligen Plattform vertrieben werden. Die Applikation muss für jede Plattform extra kompiliert werden. Der C# Quellcode wird in die Common Intermediate Language (CIL) kompiliert. Der entstandene IL Code wird dann von einer Laufzeitumgebung ausgeführt. Für Android und iOS wird die Mono Laufzeitumgebung genützt. Bei Mono handelt es sich um eine Open-Source-Implementierung von Microsofts .NET Framework. Wegen Apples Sicherheitsanforderungen wird für iOS ein Ahead-oftime-Compiler verwendet.

3.1. Analyse

Dieser Abschnitt beschreibt, wie die geforderten Sicherheitsmechanismen umgesetzt werden

1) Zugriff auf Schlüsselspeicher oder anderweitigen geschützten Speicher Xamarin stellt plattformspezifische C# APIs für den Zugriff auf die KeyStores bzw. KeyChains bereit. So gibt es für Android beispielsweise die Klasse *Android.Security.KeyChain* und für iOS und MacOS die Klasse *Security.SecKeyChain.* Beide Klassen erlauben den Zugriff auf die KeyChain der jeweiligen Plattform. Der dadurch entwickeltet Quelltext ist plattformspezifisch. **2) Zugriff auf kryptographische Funktionen** Es können die nativen kryptographischen APIs oder Bibliotheken der jeweiligen Plattform verwendet werden. Alternativ kann auch eine portable Klassenbibliothek, wie beispielsweise Bouncy Castle PCL, verwendet werden. **3) Verfügbarkeit von Root Checks** Xamarin stellt keine APIs speziell für Root-Überprüfungen bereit. Über die nativen Methoden der jeweiligen Plattform können die Checks aber durchgeführt werden. Zudem können existierende native Bibliotheken wie *RootTools* [1] oder *DTTJailbreakDetection [2]* verwendet werden. **4) Verfügbarkeit von Tools zur Code Obfuskierung** Die offizielle IDE für Xamarin, Xamarin Studio, unterstützt ProGuard [3] . Allerdings muss ProGuard manuell aktiviert werden und kann nur Android-Applikationen obfuskieren. iOS-Code-Obfuskierung ist nicht relevant, da iOS-Code zu nativen ARM-Code kompiliert wird, wodurch Rückentwickeln erschwert wird. **5) Schutz vor Repacking und Modifikation von Applikationspaketen** Es sind keine Schutzfunktionen von Xamarin bekannt, die die Modifikation von Applikationen verhindern. Es können aber die plattformspezifischen Ansätze verwendet werden. **6) Sicherheitsaspekte SSL/TLS** Mono unterstützt derzeit nur TLS 1.0 [9]. Android und iOS Entwickler und Entwicklerinnen können auf den *ModernHttpClient* [10] zurückgreifen. *ModernHttpClient* verwendet auf iOS

1 https://code.google.com/p/roottools/wiki/Usage 2 https://github.com/thii/DTTJailbreakDetection 3 https://developer.xamarin.com/guides/android/deployment,_testing,_and_metrics/publishing_an_application/part_1_-_preparing_an_application_for_release/ *NSURLSession* und auf Android OkHttp, beide Implementierungen unterstützen aktuelle TLS Versionen.

a. Unterstützte Algorithmen Prinzipiell werden die von Mono unterstützen Algorithmen auch von Xamarin unterstützt. Wird auf plattformspezifische Implementierungen zurückgegriffen, hängen die unterstützen Algorithmen von der jeweiligen Plattform ab. **b. Verwendung eigener Truststores** Es können selbstdefinierte Truststores verwendet werden. **c. Verfügbarkeit von Certificate Pinning** Certificate Pinning kann über native APIs [4] erreicht werden **d. Unterstützung für selbst-signierte Zertifikate** Die Verwendung eigener Truststores erlaubt die Benutzung selbst-signierter Zertifikate. **7) Anfälligkeit für**

Code Injection Da es sich bei Xamarin-Applikationen nicht um Webapplikationen handelt, ist Code Injektion generell kein Problem. **8) Verfügbarkeit von Per-App-VPN Services** Über native APIs können die plattformspezifischen VPN-Funktionen benutzt werden.

3.2. Demo-Applikation

Die von Xamarin bereitgestellte Beispielapplikation „Tasky“ wurde als Ausgangsbasis verwendet. Die Applikation wurde um Funktionen erweitert, die das Speichern von Daten im Android-KeyStore erlauben. Derzeit wurde nur eine Version für Android umgesetzt. Auf die Verwendung anderer Sicherheitsmechanismen wurde verzichtet, da diese durch die Verwendung nativer APIs unabhängig von Xamarin sind.

3.3. Applikationsanalyse

Die APK-Datei enthält zusätzlich zu den typischen Dateien noch einige native Bibliotheken sowie DLLs. Die nativen Android-Bibliotheken stellen die Monolaufzeitumgebung für unterschiedliche Architekturen bereit. Die Laufzeitumgebung führt dann den in den DLLs enthaltenen IL-Code aus. Für die Analyse sind die DLLs von besonderem Interesse, da sie üblicherweise den Großteil der Programmlogik enthalten. Abbildung 1 zeigt den dekompilierten Code zur Speicherung von Schlüsselmaterial im Android-KeyStore. Wie aus dem Code ersichtlich ist, wird nicht der übergebene Schlüssel im Android-KeyStore gespeichert. Stattdessen wird ein neuer Schlüssel im Android-KeyStore generiert. Mittels dieses generierten Schlüssels wird dann der übergebene Schlüssel verschlüsselt. Der verschlüsselte Schlüssel wird dann im Dateisystem abgelegt. Dieser Umweg ist notwendig, da bis Android 6 nur asymmetrische Schlüssel im Android-Keystore abgelegt werden konnten.

4 http://stackoverflow.com/questions/26466115/xamarin-cross-platform-certificate-pinning

public static void **StoreInKeystore** (string alias, string key, Activity activity){ Calendar instance = Calendar.Instance; Calendar instance2 = Calendar.Instance; instance2. **Add** (CalendarField.Year, 3); KeyPairGeneratorSpec param = **new** KeyPairGeneratorSpec. **Builder** (activity.ApplicationContext). **SetAlias** (alias). **SetSubject** (**new X500Principal** (string. **Format** (“CN=%s, OU=%s”, alias, activity.ApplicationContext.PackageName))). **SetSerialNumber** (BigInteger.One). **SetStartDate** (instance.Time). **SetEndDate** (instance2.Time). **Build** (); KeyPairGenerator instance3 = KeyPairGenerator. **GetInstance** (“RSA”, “Android-KeyStore”); instance3. **Initialize** (param); KeyPair keyPair = instance3. **GenerateKeyPair** (); Cipher instance4 = Cipher. **GetInstance** (“RSA/ECB/PKCS1Padding”); instance4. **Init** (CipherMode.EncryptMode, keyPair.Public); **byte** [] bytes = Encoding.UTF8. **GetBytes** (key); **byte** [] bytes2 = instance4. **DoFinal** (bytes); string folderPath = Environment. **GetFolderPath** (Environment.SpecialFolder.MyDocuments); string path = Path. **Combine** (folderPath, alias); File. **WriteAllBytes** (path, bytes2); }

Abbildung 1: Dekompilierter Code - Speichern von Schlüsselmaterial

3.4. Zusammenfassung

Xamarin ermöglicht die Verwendung aller plattformspezifischen APIs, daher gibt es im Vergleich zu nativen Applikationen keine Einschränkungen. Es können alle Sicherheitsmechanismen genutzt werden. Der Nachteil der nativen APIs ist, dass der entstehende Code nicht plattformunabhängig ist.

4. Apache Cordova

Das Apache Cordova-Framework [1] ermöglicht die Entwicklung von Cross-Plattform-Applikationen mithilfe von Webtechnologien. Eine Apache Cordova-Applikation wird im Wesentlichen als Webanwendung realisiert, d.h. die Benutzeroberfläche wird mit HTML und CSS und die Logik mithilfe von JavaScript umgesetzt. Die HTML-, JavaScript- und CSS-Dateien werden über die Apache Cordova CLI als native Applikation verpackt und können somit über die offiziellen Applikationsstores der jeweiligen Plattform vertrieben werden. Cordova-Applikationen werden in einer sogenannten *WebView*, einem in der Applikation integrierten Browser Fenster, ausgeführt. Dabei werden die mit der Applikation ausgelieferten HTML-Dateien angezeigt. Abbildung 2 illustriert den grundsätzlichen Aufbau einer Apache Cordova-Applikation.

Abbildung 2: Komponenten einer Apache Cordova-Applikation (Quelle: [11])

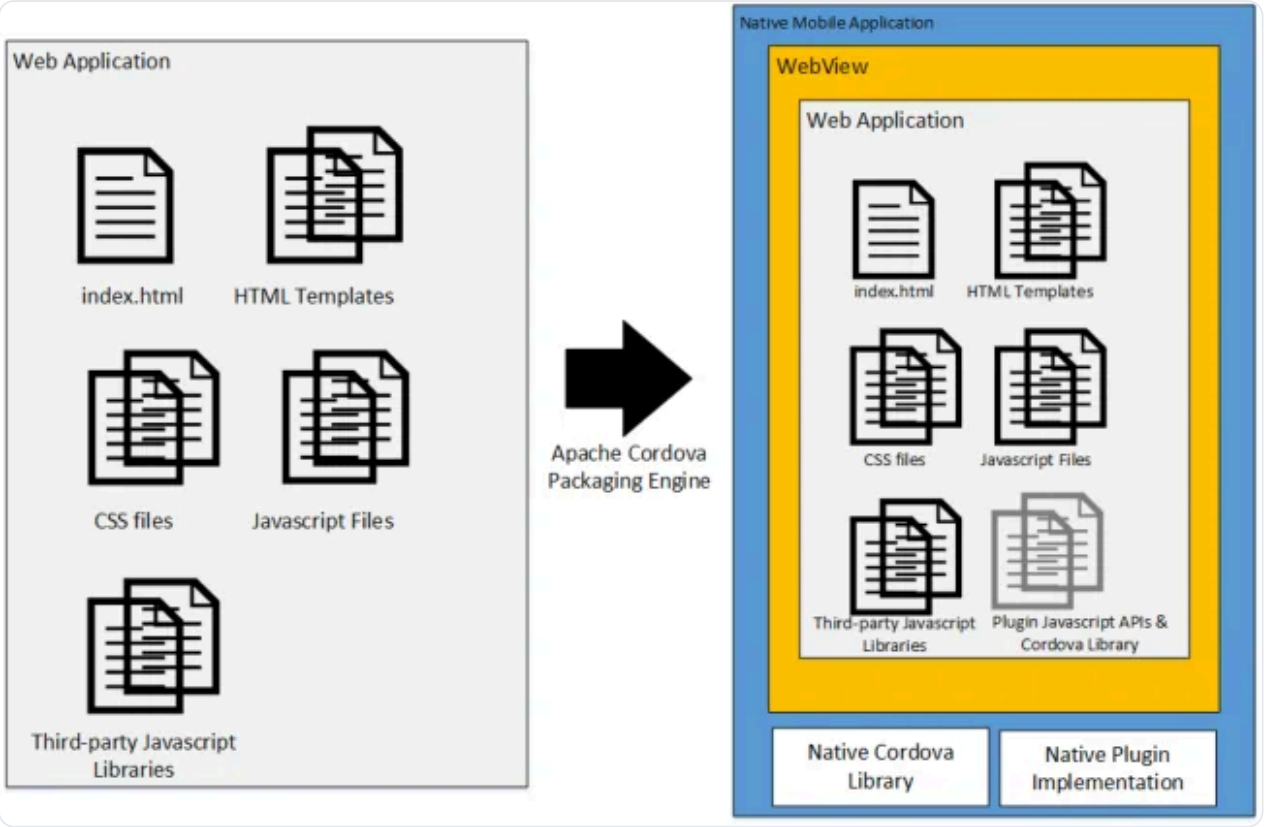


Abbildung 3: Apache Cordova Plugin Konzept (Quelle [11]) Um Sicherheitsaspekte von Apache Cordova-Applikationen zu analysieren, wird nachfolgend die Umsetzung der in Abschnitt 2 definierten Sicherheitsanforderungen diskutiert.

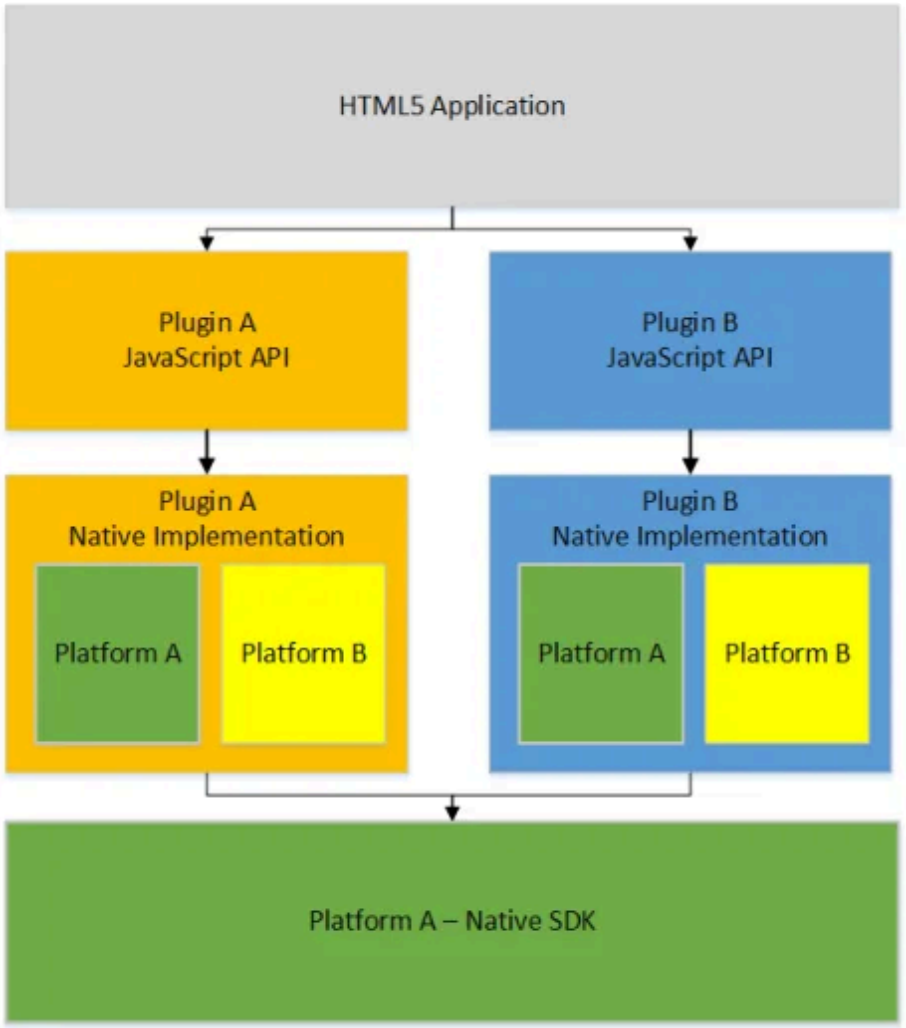
Prinzipiell beschränkt sich der Funktionsumfang einer Webanwendung auf die vom Browser zur Verfügung gestellten APIs. Browser APIs bieten jedoch keinen Zugriff auf z.B. das Dateisystem der darunterliegenden mobilen Plattform, Sensoren oder APIs, die die sichere Speicherung von Zugangsdaten oder anderen sensiblen Daten ermöglichen. Um nun den Zugriff auf APIs der darunterliegenden Plattform zu ermöglichen, können sogenannte Plugins verwendet werden. Jedes Cordova-Plugin besteht aus einer JavaScript-Schnittstelle und den darunterliegenden Plattformspezifischen Implementierungen in der nativen Entwicklungssprache der jeweiligen Plattform. Abbildung 3 stellt das Plugin-Konzept grafisch dar. Ein Cordova-Plugin kann als JavaScript-Bibliothek, die die im Browser verfügbaren Funktionen ergänzt, angesehen werden. Die JavaScript-Schnittstelle des Plugins wird wie eine herkömmliche JavaScript-Bibliothek in die Applikation inkludiert. Die Bibliothek übernimmt die Kommunikation mit dem nativen Code

des Plugins und retourniert das Resultat an die Anwendung. Des Weiteren verfügt jede Apache Cordova-Applikation über eine Konfigurationsdatei (config.xml), die generelle Konfigurationen, wie beispielsweise den Pfad zur Startseite, benötigte Berechtigungen etc. enthält.

4.1. Analyse

Dieser Abschnitt beschreibt, wie die geforderten Sicherheitsmechanismen umgesetzt werden

1) Zugriff auf Schlüsselspeicher oder anderweitigen geschützten Speicher Apache Cordova ermöglicht den Zugriff auf native plattform-spezifische Funktionen über Plugins. Es existieren Plugins, die eine Integration der iOS *Keychain* bzw. des *Android-Keystore* versprechen. Das *KeychainPlugin* [12], beispielsweise, unterstützt Android und iOS Geräte. Unter iOS erfolgt die Ablage in der *Keychain* . Jedoch ist es Entwicklerinnen und Entwicklern nicht möglich, die zu verwendende *Protection Class* zu setzen, weswegen die Standardwerte der jeweiligen iOS Version übernommen werden. Das Plugin „cordova-plugin-secure-storage“ [13] ist ebenso für Android und iOS verfügbar. Unter Android wird ein simpler Wrapping-Mechanismus angewandt, d.h. der zu speichernde Datenstring (z.B. ein kryptographischer Schlüssel) wird mit einem zufällig generierten 256 Bit AES-Schlüssel im GCM Modus verschlüsselt. Der AES-Schlüssel wird mit einem im Android *Keystore* abgelegten RSA-Schlüssel verschlüsselt. Sowohl der AES-Schlüssel als auch der RSA-Schlüssel werden für jeden Eintrag neu generiert. Unter iOS kommt die *Keychain* zur



Anwendung. Entwicklerinnen und Entwickler können über Konfigurationsparameter die gewünschte *Protection Class* setzen. Diese ist für alle von der Applikation getätigten *Keychain* Einträge gültig und kann daher nicht für jeden Eintrag separat gesetzt werden. **2) Zugriff auf kryptographische Funktionen** Zugriff auf kryptographische Funktionen kann über JavaScript-Bibliotheken wie beispielsweise SJCL [5] oder Crypto-JS [6] oder über die im Browser implementierte Web Crypto API [7] erfolgen. Des Weiteren können kryptographische Funktionen über Plugins angesprochen werden und daher wird auch die Verwendung von Bibliotheken in der jeweiligen Programmiersprache unterstützt. Von A-SIT wurde etwa prototypisch ein Kryptographie-Plugin [14] für Android umgesetzt, das die Funktionen der Web Crypto API implementiert. **3) Verfügbarkeit von Root Checks** Apache Cordova bietet standardmäßig keine Methoden zur Durchführung von Root bzw. Jailbreak Checks. Diese Checks können jedoch über Plugins durch Zuhilfenahme der Plattformspezifischen APIs umgesetzt werden. Es sind derartige Plugins für Android [15] und iOS [16] verfügbar. Beide Plugins bieten jedoch nur eher primitive Checks an, die von einem potenziellen Angreifer bzw. einer Angreiferin mit Rootzugriff auf das Gerät relativ einfach umgangen werden können (z.B. durch das Umbenennen der Dateien die überprüft werden). **4) Verfügbarkeit von Tools zur Code Obfuskierung** Apache Cordova-Applikationen bestehen aus HTML-, JavaScript- und CSS-Dateien, die als „Assets" in eine native Applikation inkludiert werden, d.h. die Quelltextdateien sind nach Entpacken eines Applikationspakets unverändert sichtbar. Zum Schutz des geistigen Eigentums der Entwicklerin bzw. des Entwicklers ist es daher ratsam, den Quelltext zu obfuskieren. Zu Testzwecken wurde unsere Demo-Applikation mit dem Obfuskator *jscrambler* [8] obfuskiert. Die Obfuskierung wurde auf die JavaScript-Dateien der Demo-Applikation angewandt. Nicht inkludiert in der Obfuskierung sind JavaScript-Dateien von Cordova-Plugins. Diese sind daher für eine Angreiferin oder einen Angreifer ohne weitere Bearbeitungsschritte sichtbar. **5) Schutz vor Repacking und Modifikation von Applikationspaketen** Es sind keine Plugins bekannt, die vor Repacking schützen bzw. die Integrität der Applikation sicherstellen sollen. Die Umsetzung eines solchen Plugins wäre auch sehr schwer bis unmöglich, da eine Angreiferin bzw. ein Angreifer den Schutzcode nur suchen und entfernen müsste. Man kann nur den Aufwand für einen Angreifer bzw. eine Angreiferin erhöhen. Auch die Obfuskierung hilft nicht gegen Repacking, sie erschwert allerdings die gezielte Manipulation der Applikation. **6) Sicherheitsaspekte SSL/TLS a. Unterstützte Algorithmen** Die unterstützten Algorithmen hängen von der jeweiligen mobilen Plattform bzw. vom eingesetzten Browser, der für die *WebView* verwendet wird, ab. **b. Verwendung eigener Truststores** Die *WebView* verwendet den Truststore des Browsers auf der jeweiligen mobilen Plattform. Android erlaubt das Hinzufügen von Zertifikaten zum systemweiten Truststore, dies kann manuell durch die Benutzerin bzw. den Benutzer erfolgen. Jedoch kann die verwendete *WebView* nicht umkonfiguriert werden, um einen eigenen Trustmanager zu nutzen. Dies gilt jedoch nicht für Cordova-Plugins. Diese können beim Durchführen von Requests auf der nativen Seite sehr wohl einen eigenen Truststore verwenden. Auch unter iOS kann der Benutzer bzw. die Benutzerin manuell Zertifikate zum systemweiten Truststore hinzufügen. Über native APIs kann auch definiert werden, welchen Zertifikaten vertraut werden soll. **c. Verfügbarkeit von Certificate Pinning** Für Apache Cordova existiert ein Plugin zur Durchführung von Certificate Pinning [17]. Das Plugin unterstützt Android und iOS, weist jedoch mehrere Einschränkungen auf. Das Plugin bietet eine JavaScript-Schnittstelle zu *get(...)* und *post(...)* Methoden, die für Android nativ in Java umgesetzt sind und Certificate Pinning implementieren.

5 https://crypto.stanford.edu/sjcl/ 6 https://code.google.com/p/crypto-js/ 7 http://www.w3.org/TR/WebCryptoAPI/ 8 https://jscrambler.com/en/ Da die Aufrufe in Java umgesetzt werden, kann somit auch ein modifizierter Zertifikats-Truststore zur Anwendung kommen und daher der vom Browser verwendete Truststore umgangen werden. Als weitere Einschränkung ist anzumerken, dass im

Vergleich zu ansonsten verwendeten XMLHttpRequests kein Zugriff auf retournierte http-Header besteht und kein automatisches Handling von Cookies angeboten wird. Zusätzlich werden die Whitelisting-Mechanismen von Apache Cordova nur auf Requests angewendet, die von der WebView abgesetzt wurden, d.h. über Plugins abgesetzte Requests sind vom Whitelisting-Mechanismus ausgenommen. **d. Unterstützung für selbst-signierte Zertifikate** Die Angabe des Parameters „android:debuggable=true“ in der Android Manifest-Datei resultiert darin, dass SSL/TLS Fehler ignoriert werden und damit auch selbstsignierte Zertifikate akzeptiert werden. Des Weiteren existiert auch ein Plugin [18], welches die WebView umkonfiguriert, um auch nicht vertrauenswürdige Zertifikate verwenden zu können. Jedoch kann die Entwicklerin bzw. der Entwickler hierbei keine Zertifikate angeben, d.h. die Validierung erfolgt fortan positiv für alle nicht vertrauenswürdigen Zertifikate. Für iOS-Geräte besteht die Möglichkeit, die Validierung durch Überschreiben eines „NSURLRequests“ zu umgehen [9] . **7) Anfälligkeit für Code Injection** Die Sicherheit herkömmlicher Webanwendungen basiert zu einem großen Teil auf dem Sandbox-Mechanismus des Browsers und der Same-Origin-Policy. Diese Mechanismen hindern Applikationen daran, auf Funktionen und Daten des darunterliegenden Systems und anderer Webseiten zuzugreifen. Die Same-Origin-Policy im Speziellen ermöglicht Zugriff lediglich auf Inhalte und Funktionen, die über denselben Hostnamen, Port und dasselbe Protokoll ausgeliefert wurden. Bei hybriden Applikationen ist eine WebView in eine native Applikation eingebettet. Zugriff auf Gerätefunktionen wird anhand des Berechtigungsmodells der jeweiligen Plattform erteilt und dieser Zugriff ist prinzipiell für Inhalte jeglicher Domain möglich. Um nun die Applikation vor Schadcode aus nicht von der Entwicklerin bzw. vom Entwickler kontrollierten Domänen zu schützen, verfügt Apache Cordova über einen Whitelisting-Mechanismus. Dabei können Entwicklerinnen und Entwickler unterschiedliche Whitelists setzen und somit die Server, mit denen eine Applikation kommuniziert, einschränken. Zum einen können damit Ziele von Netzwerkanfragen (z.B. XMLHttpRequest und `<script>` Tags) und Benutzernavigation (URLs die in der Applikation geladen werden dürfen) eingeschränkt werden, zum anderen können verfügbare Intents beschränkt werden. Damit kann die Entwicklerin bzw. der Entwickler einschränken, welche Intents von der Applikation an das Betriebssystem gesendet werden können, um vom System oder einer anderen Applikation behandelt zu werden. Angemerkt sei, dass diese Whitelisting-Mechanismen lediglich Requests bzw. Intents innerhalb der WebView beeinflussen. Cordova-Plugins können daher genutzt werden, um diese Whitelists zu umgehen. Um Angreiferinnen und Angreifer daran zu hindern, über diverse Kanäle Schadcode in die Applikation einzuschleusen, ist die Entwicklerin bzw. der Entwickler gefordert, alle Eingaben und Daten vor Ausführung bzw. Darstellung in der WebView zu validieren. Zu Kanälen, die potenziell Schadcode enthalten können, gehören unter anderem Push Notifications, QR-Codes, Dateinamen oder Kontaktinformationen, falls diese in der Applikation dargestellt werden, und jegliche von der Benutzerin bzw. vom Benutzer getätigte Eingaben. **8) Verfügbarkeit von Per-App-VPN Services** Apache Cordova-Applikationen können VPN-Funktionalität über ein Plugin integrieren. Das Plugin [19] steht für iOS und Android zur Verfügung.

4.2. Demo-Applikation

Im Zuge der Analyse von Apache Cordova wurde eine beispielhafte Demo-Applikation, die einige der geforderten Sicherheitsmechanismen inkludiert, erstellt. Dabei wurde folgende Funktionalität umgesetzt:

9 http://stackoverflow.com/questions/8858674/allow-unverified-ssl-certificate-in-uiwebview

- Certificate Pinning mit Cordova-Plugin
- Verwendung der beiden beschriebenen *KeyChain* Plugins

- Root Check über Plugin Die Demo-Applikation ist auf Android und iOS Geräten lauffähig.

4.3. Applikationsanalyse

Es wurde die Android-Version der Applikation analysiert. Der Fokus der Analyse lag auf den Plugins. Es wurden daher die beiden KeyChain Plugins, das Root Check Plugin und das Certificate-Pinning Plugin analysiert.

4.3.1. KeyChainPlugin

Das *KeyChainPlugin* [12] Plugin verzichtet unter Android auf die Verwendung des hardwaregestützten Schlüsselspeichers und legt stattdessen einen Software- *KeyStore* an. Dessen Passwort und Ablageort muss die Entwicklerin bzw. der Entwickler in der Java-Quelltextdatei setzen. Die dadurch erreichte Sicherheit ist gering, da das Passwort leicht aus der Anwendung extrahiert werden kann. Bei diesem Plugin handelt es sich um eine Abspaltung des Plugins „ *Keychain Plugin for Apache Cordova* “ [20].

4.3.2. cordova-plugin-secure-storage Plugin

Das „ *cordova-plugin-secure-storage* “ [13] Plugin verschlüsselt das übergebene Geheimnis mit einem zufälligen generierten AES-Schlüssel. Dieser Schlüssel wird dann mittels eines vom Android-KeyStore generierten RSA-Schlüssel verschlüsselt. Das verschlüsselte Geheimnis und der verschlüsselte AES-Schlüssel werden lokal gespeichert. Der Android-KeyStore verwendet, sofern vorhanden, einen hardwarebasierten Schlüsselspeicher. Der AES-Schlüssel kann nur auf dem Gerät entschlüsselt werden.

4.3.3. Root Check Plugin

Das verwendete Root Check Plugin [15] führt drei gängige Root-Prüfungen durch. Es wird überprüft, dass die *Build-Tags* nicht den String „ *test-keys* “ enthalten, dass die Datei *„/system/app/Superuser.apk* “ nicht existiert und dass das *su* -Programm nicht existiert. Nach der *su* Datei wir in folgenden Verzeichnissen gesucht: " */sbin/su* “, " */system/bin/su* “, " */system/xbin/su* “, " */data/local/xbin/su* “, " */data/local/bin/su* “, " */system/sd/xbin/su* “, " */system/bin/failsafe/su* " und " */data/local/su* “. Alle drei Checks lassen sich mit root-Zugriff relativ leicht, durch Umbenennen der jeweiligen Datei bzw. des *Build-Tags*, umgehen.

4.3.4. Certificate Pinning Plugin

Das *Certificate Pinning Plugin* [17] fügt, wenn Pinning aktiviert wird, alle Zertifikate aus dem „ *Assets* “ Ordner und dem " *www/certificates* " Ordner als vertrauenswürdig zur verwendeten *SSLSocketFactory* hinzu. HTTP-Verbindungen werden dann generell blockiert, HTTPS-Verbindungen können nur zu gepinnten Servern aufgebaut werden.

4.4. Zusammenfassung

Für einen Großteil der geforderten Funktionen steht bereits ein Plugin zur Verfügung. Jedoch muss bei der Verwendung von Plugins beachtet werden, dass diese größtenteils von externen Entwicklerinnen und Entwicklern als Open-Source-Projekte zur Verfügung gestellt werden. Plugins bieten oftmals nur einen

eingeschränkten Funktionsumfang, außerdem gibt es keine Qualitätsanforderungen an Plugin-Entwicklerinnen und -Entwickler. Jede bzw. jeder kann Plugins veröffentlichen und im offiziellen Plugin Register [10] indizieren lassen. Teilweise bieten die Plugins nicht die gewünschte Sicherheit.

5. Alpha Anywhere

Alpha Anywhere stellt eine Umgebung zur Entwicklung von mobilen Applikationen und Webapplikationen bereit. Native mobile Applikationen können für die Android, die iOS und die Windows Phone Plattform erstellt werden. Hierfür setzt Alpha Anywhere auf PhoneGap. Alpha 10 https://cordova.apache.org/plugins/ Anywhere wirbt zudem mit einem integrierten Sicherheits-Framework und Apperian Integration. Zu den beworbenen Fähigkeiten von Apperian gehören unter anderem [21]:

- VPN auf Applikationsebene
- Data-at-Rest Verschlüsselung
- Copy/Paste Schutz
- Fernsteuerung
- Jailbreak und Root Erkennung
- Applikationen mit Ablaufdatum
- Integritätsprüfung zur Laufzeit
- Selbstaktualisierende Applikationen
- Datenlöschung
- Melden von Fehlern Diese Sicherheitsfeatures werden durch App-Wrapping hinzugefügt. Beim App-Wrapping wird die eigentliche Applikation in eine zweite Applikation eingepackt. Diese zweite Applikation kann unabhängig vom Gerät mittels Regeln gemanagt werden. Nachdem Alpha Anywhere PhoneGap verwendet und PhoneGap auf Cordova aufbaut, müssen die für Cordova analysierten Plugins nicht nochmal analysiert werden. Stattdessen konzentriert sich dieses Kapitel auf Plugins, die von Alpha Anywhere zur Auswahl bereitgestellt werden und auf Sicherheitsfeatures von Apperian. Für die Analyse wurde ein Demokonto verwendet. Die Analyse beschränkt sich damit auf die in der Demoversion enthaltenen Sicherheitsfeatures. Von diesen Sicherheitsfeatures ist besonders die Integritätsprüfung zur Laufzeit relevant für diese Studie. Die Vollversion bietet einige zusätzliche Sicherheitsfeatures. Dazu gehören unter anderem VPN auf Applikationsebene, Data-at-Rest Verschlüsselung sowie Jailbreak- und Root-Erkennung. Alpha Anywhere bietet zudem eine Reihe von Plugins zur Auswahl an. Für die Analyse sind die Plugins „ *Key Chain Access* “ und „ *SSL Certificate Checker* “ relevant.

5.1. Analyse

Dieser Abschnitt beschreibt, wie die geforderten Sicherheitsmechanismen umgesetzt werden

1) Zugriff auf Schlüsselspeicher oder anderweitigen geschützten Speicher Alpha Anywhere stellt nur für iOS ein Plugin bereit, welches den Zugriff auf die KeyChain erlaubt. **2) Zugriff auf kryptographische Funktionen** Es stehen verschiedene kryptographische Funktionen zur Auswahl bereit, die meisten werden jedoch am Server ausgeführt. Über die Apperian-Integration steht Data-At-Rest (DAR) Verschlüsselung bereit. **3) Verfügbarkeit von Root Checks** Apperian stellt eine Jailbreak- und Root-Schutzregel bereit, die die Ausführung der App auf gerooteten Geräten verhindern soll. **4) Verfügbarkeit von Tools zur Code Obfuskierung** Es wurden keine Tools zur Quelltext-Obfuskierung speziell für Alpha Anywhere gefunden. **5) Schutz vor Repacking und Modifikation von Applikationspaketen** Apperian stellt eine „ *Run-time Integrity Check* “ Regel bereit. Diese soll die Ausführung von manipulierten Applikationen verhindern. **6) Sicherheitsaspekte SSL/TLS** Alpha Anywhere bietet ein Plugin zur Zertifikatsprüfung an [22]. Ansonsten gilt das in Abschnitt 4.1 Gesagte, da Alpha Anywhere auf Cordova aufbaut. Das heißt, die unterstützten Algorithmen hängen von der verwendeten Plattform und vom verwendeten Browser ab. Die *WebView* verwendet den Truststore des Browsers. **7) Anfälligkeit für Code Injection** Auch hier gelten die Aussagen von Abschnitt 4.1. Entwickler und Entwicklerinnen sollten alle Eingaben und Daten vor Ausführung bzw. Darstellung in der *WebView* validieren. Dies gilt besonders wenn nicht von Alpha Anywhere bereitgestellte Funktionen verwendet werden. **8) Verfügbarkeit von Per-App-VPN Services** Apperian stellt App-Level-VPN bereit. Dadurch wird eine sichere Verbindung von der Applikation zum Firmennetz gewährleistet, ohne dem gesamten Gerät Zugriff auf das interne Netz zu geben.

5.2. Demo-Applikation

Es wurde eine Demo-Applikation für Android und Windows Phone umgesetzt. Die Applikation verwendet die Plugins „ *Key Chain Access* “ und „ *SSL Certificate Checker* “, sowie die Apperian Regel „ *Run-time Integrity Check* “.

5.3. Applikations-Analyse

Es wurde die Android-Version der Demo-Applikation analysiert.

5.3.1. Key Chain Access

Alpha Anywhere liefert das „ *Keychain Plugin for Apache Cordova* “ [20] Plugin mit aus. Dieses ist nur für iOS verfügbar. Ein Nachteil dieses Plugins ist, dass die Entwicklerin oder der Entwickler nicht die zu verwendende *Protection Class* setzen kann, wodurch die Standardwerte der jeweiligen iOS Version genommen werden.

5.3.2. SSL Certificate Checker

Alpha Anywhere liefert das „ *PhoneGap SSL Certificate Checker plugin* “ [22] mit aus. Dieses Plugin stellt eine Prüfmethode bereit, die die vom Server zurück gelieferte Zertifikatskette bzw. deren Zertifikatsfingerabdrücke gegen erlaubte Fingerabdrücke prüft. Die Prüfmethode liefert dann „ *CONNECTION_SECURE* “ oder „ *CONNECTION_NOT_SECURE* “ zurück. Das Problem bei diesem Plugin ist, dass danach wieder eine Verbindung zu dem Server aufgebaut werden muss, um die eigentlichen Daten zu schicken oder zu empfangen. Eine Angreiferin bzw. ein Angreifer könnte die Schutzfunktion dieses Plugins leicht umgehen, indem die erste Verbindung nicht angegriffen wird, d.h. die Prüfung des Plugins „

CONNECTION_SECURE " zurück liefert und erst die zweite Verbindung mittels MITM-Angriff belauscht wird. Die Angreiferin oder der Angreifer hat dadurch keinen Nachteil, da erst über die zweite Verbindung die interessanten Daten übertragen werden.

5.3.3. Run-time Integrity Check

Es wird der SHA-256 Fingerabdruck der APK-Datei berechnet und mit dem am Server gespeichertem Wert und einem lokal gespeicherten Wert verglichen.

5.3.4. Root Check

Obwohl das Apperian-Feature „Jailbreak und Root Erkennung" nicht aktivierbar war, fiel bei der Analyse der dazugehörige Code auf. Es wird der Wert des *Build-Tags* überprüft, es wird überprüft ob die Programmdatei „ *su* ", „ *.su* ", „ *busybox* ", „ *mu* " oder „ *su-backup* " existiert und ob die Datei „ */system/app/Superuser.apk* " exisitiert. Zusätzlich wird noch eine native Überprüfung durchgeführt und überprüft ob eine der folgenden Applikationen installiert ist: " *com.noshufou.android.su* ", " *com.thirdparty.superuser* ", " *eu.chainfire.supersu* ", " *com.koushikdutta.superuser* ", " *com.zachspong.temprootremovejb* ", " *com.ramdroid.appquarantine* ". Die native Methode wurde nicht analysiert.

5.4. Zusammenfassung

Alpha Anywhere und Apperian bieten einige interessante Sicherheitsfeatures an. Im Vergleich zu den anderen evaluierten Lösungen gibt es mehr vorbereitete Sicherheitsfeatures, die auf einfache Art und Weise verwendet werden können.

6. Schlussfolgerung

Alle betrachteten Frameworks haben im Kontext dieser Studie ihre Vor- und Nachteile. Xamarin hat den Vorteil, dass der Entwickler bzw. die Entwicklerin alle Features der jeweiligen Plattform nutzen kann, dass die Bedienoberfläche native Elemente nutzen kann und dass nur eine Programmiersprache beherrscht werden muss. Der Nachteil ist, dass der Entwickler bzw. die Entwicklerin alle zu unterstützenden Plattformen und deren APIs kennen muss. Apache Cordova hat den Vorteil, dass bestehende HTML5 Seiten relativ leicht in Apps konvertiert werden können und dass auf Plattform-Features zugegriffen werden kann. Der Nachteil ist, dass der Entwickler bzw. die Entwicklerin die Programmiersprachen aller zu unterstützenden Plattformen beherrschen muss, um Funktionen, die über die Möglichkeiten der WebView hinausgehen, verwenden zu können, sofern es hierfür noch kein Plugin gibt. Gibt es für die benötigten Plattformfeatures bereits ein Plugin, kommt der Entwickler bzw. die Entwicklerin mit HTML5 Technologien aus, muss aber fremdem Code vertrauen. Außerdem kommen Applikationen, die mittels Apache Cordova entwickelt wurden, meistens nicht an das Look-and-feel von nativen Applikationen heran. Für die Benutzerin bzw. den Benutzer ist in den meisten Fällen ein klarer Stilbruch erkennbar. Für Alpha Anywhere ergibt sich ein ähnliches Bild wie für Apache Cordova, mit der Ausnahme, dass es einige vorbereitete Funktionen und Sicherheitsfeatures gibt, die eine schnellere Entwicklung einer App ermöglichen sollen.

7. Referenzen

[1] The Apache Software Foundation, „Apache Cordova," [Online]. Available: https://cordova.apache.org/. [Zugriff am 18 12 2015].

[2] Alpha Software Corporation, „Alpha Anywhere,“ [Online]. Available: http://www.alphasoftware.com/. [Zugriff am 18 12 2015].

[3] M. C. Apps, „https://github.com/MobileChromeApps/mobile-chrome-apps,“ [Online]. Available: https://github.com/MobileChromeApps/mobile-chrome-apps . [Zugriff am 18 12 2015].

[4] Appcelerator Inc, „The Appcelerator Platform,“ [Online]. Available: http://www.appcelerator.com/. [Zugriff am 18 12 2015].

[5] Xamarin Inc., „Xamarin,“ [Online]. Available: https://xamarin.com. [Zugriff am 18 12 2015].

[6] Adobe Systems Incorporated, „Adobe AIR,“ [Online]. Available: http://www.adobe.com/products/air.html . [Zugriff am 18 12 2015].

[7] Unity Technologies, „Unity - Game Engine,“ [Online]. Available: https://unity3d.com/ . [Zugriff am 18 12 2015].

[8] VisionMobile, „Cross-Platform Tools 2015,“ [Online]. Available: http://www.visionmobile.com/product/cross-platform-tools-2015/. [Zugriff am 18 12 2015].

[9] M. d. Icaza, „State of TLS in Mono,“ [Online]. Available: http://tirania.org/blog//archive/2015/Aug.html. [Zugriff am 28 12 2015].

[10] P. Betts, „ModernHttpClient,“ [Online]. Available:

https://github.com/paulcbetts/ModernHttpClient. [Zugriff am 28 12 2015].

[11] S. Kreuzhuber, „A Flexible Cross-Platform Framework for Integrating Multi-Factor Authentication,“ [Online]. Available: https://online.tugraz.at/tug_online/wbAbs.showThesis?pThesisNr=60228&pOrgNr=37&pPer sNr=54583. [Zugriff am 18 12 2015].

[12] Wymsee, „Keychain Plugin for Apache Cordova,“ [Online]. Available:

https://github.com/wymsee/KeychainPlugin. [Zugriff am 18 12 2015].

[13] Crypho AS, „SecureStorage plugin for iOS & Android,“ [Online]. Available:

https://github.com/Crypho/cordova-plugin-secure-storage. [Zugriff am 18 12 2015].

[14] A-SIT, „Apache Cordova Kryptographie Plugin,“ [Online]. Available: https://demo.a sit.at/apache-cordova-kryptographie-plugin/ . [Zugriff am 18 12 2015].

[15] trykovyura, „Root Detection Plugin for Apache Cordova,“ 18 12 2015. [Online]. Available:

https://github.com/trykovyura/cordova-plugin-root-detection.

[16] L. Crossley, „Cordova Jailbreak Detection Plugin,“ [Online]. Available:

https://github.com/leecrossley/cordova-plugin-jailbreak-detection. [Zugriff am 18 12 2015].

[17] Wymsee, „cordovaHTTP,“ [Online]. Available: https://github.com/wymsee/cordova-HTTP.

[Zugriff am 18 12 2015].

[18] M. Reinhardt, „Certificate Plugin for Apache Cordova,“ [Online]. Available:

https://github.com/hypery2k/cordova-certificate-plugin. [Zugriff am 18 12 2015].

[19] Aquto, „cordova-plugin-vpn,“ [Online]. Available: https://github.com/aquto/cordova-plugin vpn . [Zugriff am 18 12 2015].

[20] A. Shazron, „Keychain Plugin for Apache Cordova,“ [Online]. Available:

https://github.com/shazron/KeychainPlugin. [Zugriff am 18 12 2015].

[21] Apperian, Inc., „Managing Application Policies - EASE - Apperian,“ [Online]. Available:

https://help.apperian.com/display/pub/Managing+Application+Policies. [Zugriff am 18 12 2015].

[22] E. Verbruggen, „PhoneGap SSL Certificate Checker plugin (iOS, Android, Windows Universal),“ [Online]. Available: https://github.com/EddyVerbruggen/SSLCertificateChecker-PhoneGap-Plugin. [Zugriff am 21 12 2015].