Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-SIT

Seidlgasse 22 / 9, 1030 Wien
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

Inffeldgasse 16a, 8010 Graz
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

https://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

AUTOMATISCHE ERKENNUNG UND BEHEBUNG VON CRYPTO-API-MISUSE IN ANDROID-APPLIKATIONEN

Version 1.0 vom 20.08.2021
Florian Draschbacher – florian.draschbacher@iaik.tugraz.at

Auch knapp ein Jahrzehnt nach der ersten wissenschaftlichen Erforschung bleibt Crypto-API-Misuse eine der häufigsten Ursachen für Sicherheitslücken in Android-Anwendungen. Durch falsche Parametrisierung von kryptografischen Primitiven oder von Funktionalität für den Aufbau von TLS-Verbindungen, können Angreifer in vielen Fällen unkompliziert sensible Nutzerdaten abschöpfen. Da die verschiedenen Bemühungen vom Plattform-Anbieter Google in den letzten Jahren zu keiner nennenswerten Veränderung der Situation führen konnten, schlagen wir in diesem Forschungsprojekt eine neuartige Lösung vor, die diese weitverbreitete Bedrohung für Endnutzer abwenden soll. Unser Ansatz sieht einen Hintergrundprozess vor, der auf unmodifizierten Android-Systemen automatisiert alle installierten Anwendungen um einen Patch zur Überwachung und Korrektur von Aufrufen kryptografischer APIs ergänzt. Wir demonstrieren die Effektivität dieser Lösung anhand zweier Case-Studies populärer Android-Anwendungen von Google Play.

Inhaltsverzeichnis

1. Einleitung

Als Folge der wachsenden Popularität von Mobilgeräten für immer mehr Anwendungsfälle sind diese Plattformen in den letzten Jahren zum beliebten Gegenstand von Forschungsprojekten geworden.

Wissenschaftler haben sich zum Ziel gesetzt, zu untersuchen, ob die Entwickler von mobilen Anwendungen mit den ihnen anvertrauten sensiblen Daten entsprechend verantwortungsbewusst umgehen. Dabei haben sie mehrere kritische Sicherheitslücken in einer Vielzahl von Applikationen entdeckt, die aus dem falschen Umgang mit kryptografischen Programmierschnittstellen (APIs) resultieren [1] [2]. Durch fehlerhafte Parametrisierung von Funktionalität zum Aufbau geschützter Kommunikationskanäle, der Ver- oder Entschlüsselung von Daten, der Ableitung von kryptografischen Schlüsseln aus Passwörtern oder zur Erzeugung von Zufallszahlen erhalten Angreifer unkompliziert Zugriff auf sensible Daten, während den Nutzern ein falsches Gefühl von Sicherheit vermittelt wird. In diesem Projekt haben wir uns dieser Problematik in Bezug auf das populäre Betriebssystem Android gewidmet.

Obwohl im vergangenen Jahrzehnt von wissenschaftlicher Seite große Bemühungen unternommen wurden, um das Ausmaß dieses Problems regelmäßig zu quantifizieren, gab es nur sehr wenige Beiträge, die die entstandenen Sicherheitslücken gezielt bekämpfen. Die einzigen beiden Publikationen beschränken sich jeweils nur auf einen Teilbereich der problematischen APIs und setzen entweder auf langsame statische Analyse [3] oder benötigen ein gerootetes Gerät [4]. Beide Anforderungen schränken die praktische Anwendbarkeit der Lösungen drastisch ein.

Plattform-Anbieter Google veröffentlichte mit erheblichem Aufwand verbesserte Programmierwerkzeuge und Entwicklerdokumentationen, doch konnten diese keine deutliche Verbesserung der Lage erbringen, wie aus aktuellen Publikationen hervorgeht [5]. Laut [6] und [7] haben Entwickler oft zu wenig Erfahrung, um Best Practices zur Absicherung ihrer Anwendungen umzusetzen.

Weil das Problem ein Jahrzehnt nach der ersten diesbezüglichen Veröffentlichung noch immer eine ernsthafte Gefahr für Nutzer vieler Android-Applikationen darstellt, wurde im Rahmen dieses Projekts eine Lösung erarbeitet, die falsche Parametrisierung von kryptografischen APIs in kompilierten Android-Applikationen automatisiert erkennen und beheben kann. Zur Steigerung des praktischen Mehrwerts läuft diese Lösung eigenständig auf dem Mobilgerät selbst und benötigt keine Modifikationen des Betriebssystems (zB „rooten“).

2. Hintergrund

Android-Applikationen werden in Java (oder kompatiblen Programmiersprachen) entwickelt und vor der Installation in ein spezielles „Bytecode“-Format übersetzt. Dieses wird dann am Android-Gerät von einer Runtime ausgeführt. Dieses Vorgehen soll sicherstellen, dass Android-Programme auf allen verfügbaren Geräten gleichermaßen funktionieren, unabhängig von der jeweiligen Prozessorarchitektur und anderen Hardware-Merkmalen.

Als Teil der Java Cryptography Architecture (JCA) stehen Entwicklern von Android-Anwendungen auch verschiedene Schnittstellen für kryptografische Operationen zur Verfügung. Der folgende Abschnitt soll einen Überblick über besonders wichtige Schnittstellen liefern, die häufig Gegenstand fehlerhafter Parametrisierung sind.

2.1. TLS/SSL

Transport Layer Security (TLS) bzw. das Vorgängerprotokoll Secure Socket Layer (SSL) dienen zur Absicherung von Kommunikationskanälen. Ein häufiger Anwendungsfall ist das HTTPS-Protokoll, wo TLS dazu dient, die Identität von Webservern eindeutig sicherzustellen und die Verbindung gegen Angreifer abzusichern.

Die JCA implementiert TLS/SSL in der `SSLSocket`-Schnittstelle. Obwohl in der Standard-Konfiguration die Überprüfung der Zertifikatskette und prinzipielle Gültigkeit des Server-Zertifikats schon umgesetzt ist (`TrustManager`-Objekt), bleibt es die Verantwortung des Entwicklers, auch zu überprüfen, ob das präsentierte Server-Zertifikat auch für den konkret kontaktierten Host ausgestellt wurde (`HostnameVerifier`-Objekt). Diese Kontrolle wird von vielen Anwendungen ausgespart. Ein anderes Problem sind Programme, die eigene `TrustManager` implementieren, um selbst-signierte

Zertifikate zu unterstützen. In vielen Fällen führen auch diese Änderungen dazu, dass Anwendungen anfällig für Man-In-The-Middle-Angriffe (MITM) werden, in denen der Netzwerkverkehr zwischen Server und Client nicht nur abgehört, sondern auch entschlüsselt werden kann (SSL Stripping).

2.2. Chiffren

Chiffren sind kryptografische Primitive, die dazu dienen, Daten zu Ver- bzw. Entschlüsseln. Es wird unterschieden zwischen Blockchiffren, die Daten in Blöcken einer fixen Größe verarbeiten, sowie Stromchiffren, die Daten beliebiger Länge verarbeiten. Um Blockchiffren für beliebige Datenmengen nutzbar zu machen, gibt es eine Reihe von Betriebsmodi, die spezifizieren, wie die unstrukturierten Klartext-Daten mithilfe des Chiffren-Primitivs (und des Schlüssels) zu Geheimtext-Blöcken transformiert werden. Die einfachste Variante ist hier der Electronic-Code-Book-Modus (ECB), der den Klartext direkt in Blöcke teilt, die unabhängig voneinander verschlüsselt werden. Dieses Verfahren ist allerdings in den meisten Anwendungsfällen unsicher, da Ähnlichkeiten zwischen Klartext-Blöcken auch nach dem Verschlüsseln noch sichtbar bleiben. Um dies zu verhindern, verwenden sichere Betriebsmodi einen zufälligen Initialisierungs-Vektor (IV) und integrieren in verschiedener Form Abhängigkeiten zwischen den Geheimtexten aufeinanderfolgender Blöcke.

Eine orthogonale Dimension zur Klassifikation von Chiffren ist die Schlüssel-Symmetrie. Sie beschreibt, ob zur Entschlüsselung der gleiche Schlüssel wie für die Verschlüsselung benötigt wird („symmetrisch“), oder ob zwei verschiedene (aber mathematisch verknüpfte) Schlüssel verwendet werden („asymmetrisch“).

Android-Entwickler können über die `Cipher`-Schnittstelle der JCA auf Implementierungen all dieser Verfahren zurückgreifen. Ein häufiger Fehler ist hier die Verwendung des ECB-Modus‘ für Anwendungen, in denen mehrere Datenblöcke mit dem gleichen Schlüssel chiffriert werden. In vielen Fällen geschieht dies, weil die JCA den ECB-Modus als Standard nutzt, sofern vom Entwickler kein anderer Modus gewünscht wird. Ein anderes Problem ergibt sich regelmäßig durch Applikationen, die für jede Chiffren-Operation denselben IV benutzen. Ähnlich wie beim ECB-Modus können Angreifer hier aus Ähnlichkeiten zwischen den Geheimtexten Rückschlüsse auf die entsprechenden Klartexte ziehen.

2.3. Passwort-basierte Verschlüsselung

In einigen Anwendungsfällen sollen Daten mit einem Passwort verschlüsselt werden, das vom Nutzer eingegeben werden kann. Um sicherzustellen, dass hier das eingegebene Passwort die Anforderungen in Bezug auf Entropie erfüllt, die von den Chiffren-Algorithmen an Schlüssel gerichtet werden, muss eine Schlüsselableitungsfunktion genutzt werden. Diese erzeugt aus dem Passwort und einem zufälligen Salt-Wert den Schlüssel, der dann für die Chiffren-Operation genutzt werden kann. Um Angriffe mittels Brute-Force-Verfahren zu erschweren, bestehen Schlüsselableitungsfunktionen aus einer Vielzahl von zeitaufwändigen Iterationen, in denen jeweils ein neuer Zwischenwert aus dem vorherigen Zwischenwert abgeleitet wird. Auch die Anzahl dieser Iterationen muss vom Entwickler vorgegeben werden.

Die JCA verfügt über eine Unterstützung für Passwort-basierte Verschlüsselung, auf die über die `SecretKeyGenerator`-Schnittstelle zugegriffen werden kann. Entwickler sind hier verantwortlich dafür, das Passwort, einen Salt-Wert und die Anzahl der Iterationen zu übergeben. Leider kommt es immer wieder vor, dass Anwendungen Salt-Werte wiederverwenden oder zu wenige Iterationen spezifizieren. Beide Fehler führen dazu, dass Brute-Force-Angriffe für Angreifer erheblich erleichtert werden.

2.4. Zufallszahlen-Generatoren

Wo in kryptografischen Systemen Zufallszahlen nötig sind, muss häufig ein kryptografisch sicherer Pseudozufallszahlengenerator verwendet werden. Dieser wird mit Entropie (etwa zufällige Messungen der Geräte-Umgebung durch Sensoren) befüllt und erzeugt ausgehend von diesem Startwert deterministisch eine Serie von gleichverteilten Zahlen aus einem definierten Bereich.

Die JCA stellt für diese Funktionalität die `SecureRandom`-Schnittstelle zur Verfügung. Während es bei alten Android-Versionen noch entscheidend war, dass Entwickler Instanzen dieser Klasse vor der Benutzung mit ausreichend Entropie befüllen, erledigen dies neuere Versionen des Betriebssystems schon automatisch. Ein häufiges Problem waren hier in der Vergangenheit Anwendungen, die den Pseudozufallszahlengenerator wiederholt mit dem gleichen Startwert initialisiert haben. Als Abhilfe verwenden neue Android-Versionen vom Entwickler spezifizierte Startwerte nur noch als Ergänzung der systemeigenen Entropie, nicht mehr als Ersatz. Dennoch kann die Verwendung eines konstanten Startwerts zumindest als grobe Missachtung der gängigen Sicherheitsstandards für die sichere Softwareentwicklung im Allgemeinen gelten.

3. System-Aufbau

Weil die Erfahrung der letzten Jahre gezeigt hat, dass sich Nutzer bei der Absicherung der von Android-Anwendungen verarbeiteten Daten weder auf die App-Entwickler, noch auf die Plattform selbst verlassen können, schlagen wir eine Drittanbieter-Lösung vor. Diese soll es Anwendern ermöglichen, Applikationen zu verwenden, die Crypto-API-Misuse enthalten, ohne dabei Opfer von entstehenden Sicherheitslücken werden zu können.

Zu diesem Zweck soll im Rahmen dieses Projekts ein Hintergrundprozess verwirklicht werden, der am Android-Gerät alle neu installierten Applikationen automatisiert um ein Modul erweitert, das Aufrufe von kryptografischen Funktionen protokolliert und bei Bedarf fehlerhafte Parametrisierungen selbständig korrigieren kann. Besonderer Wert wird hier auf Kompatibilität mit einer großen Anzahl an Android-Geräten gelegt, was etwa bedeutet, dass die Lösung ohne Root-Berechtigung auskommen soll. Die Lösung soll außerdem zusätzlich zum automatischen Schutz für unerfahrene Nutzer auch Monitoring-Funktionalität für erfahrene Anwender bieten. Letztere sollen so Einblicke in sicherheitstechnische Details von Applikationen erlangen können, ohne langwierige statische Analysen abwarten zu müssen.

Die Kernbestandteile des Systems sind zusammenfassend:

- **Der Patch**
  Ein Modul, das in bestehende Android-Applikationspakete injiziert wird, um dort alle Aufrufe zu kryptografischen Funktionen des System-Frameworks abzufangen, zu protokollieren und gegebenenfalls mit korrigierten Parametern an die ursprünglich adressierten Komponenten weiterzuleiten.

- **Die Applikation**
  Eine Android-Anwendung, die Komponenten für zwei Funktionsstränge vereint: Ein Hintergrundprozess überwacht permanent die am Gerät installierten Programme und erzeugt für jede neu erkannte Applikation eine Version, die um den Patch erweitert („gepatcht“) wurde. Zusätzlich soll eine übersichtliche Benutzeroberfläche dazu dienen, die gesammelten Monitoring-Informationen anzuzeigen, sowie verschiedene Verwaltungsaufgaben für den Nutzer zu vereinfachen. Dazu zählen etwa die Deinstallation von Applikationen, die Einsicht in laufende Patching-Vorgänge oder (in Sonderfällen) die Deaktivierung der Misuse-Korrekturen.

4. Patch

Wie oben erwähnt, dient der Patch dazu, Aufrufe von relevanten kryptografischen APIs abzufangen, um sie zu protokollieren und bei Bedarf unsichere Parametrisierungen zu korrigieren. In den folgenden Abschnitten soll diese Komponente unseres Systems näher beleuchtet werden. Der Fokus liegt hier insbesondere in den Prozeduren, die erarbeitet wurden, um die verschiedenen Fehler im Umgang mit kryptografischen APIs zu beheben. Zusätzlich zu den im Einzelnen beschriebenen Mitigations-Strategien werden alle relevanten Parameter auch gesammelt und zur Anzeige in der Benutzeroberfläche der Applikation (5.2) weitergeleitet.

4.1. TLS/SSL

Wie in 2.1 erläutert, sind die häufigsten Probleme im Umgang mit den Schnittstellen für TLS/SSL die Verwendung unsicherer `TrustManager` und `HostnameVerifier`. Beide Fehler tragen zu einer signifikant höheren Anfälligkeit für MITM-Attacken bei.

Um dieses Angriffsszenario zu verhindern, verfolgt unser Patch einen dynamischen Ansatz, der TLS-Verbindungen mittels Certificate Pinning absichert. Üblicherweise wird dieses Verfahren von Entwicklern direkt in ihre Apps integriert, wo nach dem Aufbau einer Verbindung zum Server dessen Zertifikat mit einer Liste von hinterlegten Zertifikaten verglichen wird. Nur wenn eine Übereinstimmung gefunden wird, wird die Verbindung als sicher eingestuft. MITM-Attacken werden damit effektiv verhindert, da der sonst anfälligen PKI-Infrastruktur nicht vertraut werden muss.

Damit Certificate Pinning nachträglich in eine kompilierte App eingefügt werden kann, verwendet unsere Lösung einen Web-Service, über den das legitime Zertifikat eines bestimmten Ziel-Hosts abgefragt werden kann. Wir gehen hier davon aus, dass der Web-Service zu diesem Zweck über eine Leitung zum Ziel-Host verfügt, die zu keinem Zeitpunkt manipuliert werden kann[1]. Zusätzlich ist das System nach dem Prinzip Fail-Safe-Default so konzipiert, dass ein Angreifer auf die Verbindung zwischen Patch und Web-Service maximal einen Denial of Service bewirken kann, nicht aber die Manipulation der übermittelten Daten.

Der Patch fängt alle Aufrufe des `SSLSocketFactory`-Interfaces ab, um neu erzeugte `SSLSocket`-Instanzen mit einem Wrapper-Objekt zu ummanteln. Dieses kann den oben skizzierten Zertifikats-Check durchführen, kurz bevor die Ziel-Anwendung den Austausch von Daten auf Applikationsebene beginnt. Wird zu diesem Zeitpunkt ein Wiederspruch zwischen dem direkt vom Ziel-Host gezeigten Zertifikat und dem legitimen Zertifikat (das über den Web Service ermittelt wurde) festgestellt, kann der Patch daraus schließen, dass ein MITM-Angriff vorliegt und die untersuchte Applikation auf diesen anfällig ist. Der Patch unterbricht in diesem Fall die Verbindung und kann so den Abfluss von sensiblen Nutzerdaten verhindern.

Wir möchten hier kurz darauf hinweisen, dass sich aus dieser simplen Implementierung ein Privacy-Problem ergeben könnte, weil der Betreiber des Web-Service Einblick hat, welche Hosts ein Nutzer kontaktiert. Da es sich bei der hier umgesetzten Lösung aber nur um einen Prototyp handelt, erachten wir dieses Problem hier als out-of-scope. Bei einer Umsetzung für den Produktivbetrieb ließen sich beispielsweise Elemente von Mix-Netzwerken integrieren.

4.2. Chiffren

Kernaufgabe des Patches für die `Cipher`-API ist es, sicherzustellen, dass keine unsicheren Chiffren (bzw. Betriebsmodi) verwendet werden. Zu diesem Zweck wird ein ähnlicher Ansatz verfolgt, wie oben schon für TLS geschildert. Bei der Erstellung neuer `Cipher`-Objekte wird ein Wrapper-Objekt erzeugt und an den Applikations-Code übergeben. Die so eingefügte Logik ist in der Lage, die übergebenen Parameter zu überprüfen und bei Bedarf zu adaptieren.

Wird zum Beispiel erkannt, dass die Applikation eine symmetrische Blockchiffre im ECB-Modus zu betreiben versucht, kann der Patch automatisch ein Upgrade auf den sichereren Cipher-Block-Chaining-Modus (CBC) durchführen. Weil die Sicherheit dieses Modus‘ auf der Wahl eines zufälligen Initialization Vectors (IV) basiert, muss ein solcher Zufallswert nicht nur für die Verschlüsselung erzeugt werden, sondern auch bei der korrespondierenden Entschlüsselung zur Verfügung stehen, obwohl ein solcher Transport von den ursprünglichen App-Entwicklern nie vorgesehen wurde. Dasselbe Problem tritt auch auf, wenn der Patch die wiederholte Verwendung desselben IVs bemerkt hat und daher ein anderer Wert verwendet werden muss als zunächst vom Applikations-Code übergeben.

[1] Wir gehen hier also davon aus, dass der MITM-Angreifer näher am Endgerät positioniert ist, als am Ziel-Server. Dies ist zum Beispiel der Fall, wenn Nutzer eines unsicheren Wifi-Hotspot attackiert werden.

Aus diesem Grund wurde eine Zustands-Maschine integriert, die in der Lage ist, den IV im Zuge der Verschlüsselung in den Geheimtext zu integrieren (als Präfix) und während des Entschlüsselns zu extrahieren. Dazu wird die zugrunde liegende Chiffre erst initialisiert, wenn dem Wrapper-Objekt ein ausreichend großes Stück des Geheimtexts übergeben wurde, sodass entschieden werden konnte, ob ein nachträglich eingefügter IV vorliegt, oder der ursprünglich übergebene Wert genutzt werden soll.

Um die wiederholte Verwendung eines IVs erkennen zu können, baut der Patch eine Datenbank aus allen Werten auf, die in der Vergangenheit als IVs für die Verschlüsselung genutzt wurden. Wird hier zu einem späteren Zeitpunkt eine Übereinstimmung erkannt, kann eine Warnmeldung erzeugt und ein sicherer Alternativwert verwendet werden.

4.3. Password-basierte Verschlüsselung

Die Probleme, die hier eventuell behoben werden müssen, sind die Spezifikation eines zu niedrigen Iterations-Wertes oder die wiederholte Nutzung eines (statischen) Salt-Wertes.

Auch hier wird wieder in die Erzeugung von Objekten eingegriffen um Mantelobjekte einzufügen, diesmal sind es solche der `SecretKeyFactory`-Klasse. Zu diesem Zweck werden Aufrufe der entsprechenden Factory-Methode abgefangen, der die beiden potentiell fehlerhaften Parameter übergeben werden. Der Iteration-Count-Parameter kann ab hier trivial korrigiert werden, indem er bei Bedarf durch einen sicheren Wert ersetzt wird, bevor er an die ummantelte `SecretKeyFactory`-Instanz weitergegeben wird.

Die Korrektur des Salt-Wertes wird von unserer Lösung aus technischen Gründen nicht unterstützt. Zwar haben wir ähnlich wie für Chiffren-IVs eine Erkennung für die wiederholte Nutzung desselben Wertes implementiert. Allerdings bietet die API hier keine Möglichkeit, ein Ersatz-Salt ohne Kooperation des Anwendungs-Codes für spätere Nutzungen zu hinterlegen.

4.4. Zufallszahlen-Generatoren

Wie schon in 2.4 erwähnt, war die `SecureRandom`-API in der Vergangenheit vor allem mit Problemen durch fehlerhafte Startwerte konfrontiert. Diese Fälle erkennt unsere Lösung durch eine Datenbank aller bereits genutzten Werte. Bei Bedarf kann dann ein vom Applikations-Code bereitgestellter unsicherer Wert entfernt werden und stattdessen die Standard-Implementierung des Frameworks zur Entropie-Gewinnung herangezogen werden.

5. Applikation

Die Applikation ist innerhalb unseres Gesamtkonzepts jenes Element, das der Lösung eine hohe Praktikabilität verleiht. Die Applikation wird direkt auf einem beliebigen Android-Gerät installiert, das durch unsere Lösung geschützt werden soll. Hervorzuheben sind hier insbesondere die Fähigkeiten, automatisiert bestehende Applikationspakete um den Patch (4) zu erweitern und das resultierende gepatchte Programmpaket ohne Nutzereingabe zu installieren. Diese beiden Funktionen, sowie die Benutzerschnittstelle zur Verwaltung der Funktionalität, werden im Folgenden beschrieben.

5.1. Hintergrundprozess

Im Stile von Antiviren-Programmen, wie sie von Desktop-Systemen bekannt sind, startet unsere Lösung einen Hintergrundprozess direkt auf dem Android-Gerät, das geschützt werden soll. Sobald der Prozess vom System über die Installation oder die Aktualisierung von Software informiert wird, wird auf deren Basis ein neues Programmpaket erzeugt, das um den Patch ergänzt wurde. Zu diesem Zwecke wurde eine eigene Patching-Pipeline geschaffen, die speziell für größtmögliche Effizienz im Umgang mit den stark beschränkten Ressourcen von Mobilgeräten entwickelt wurde.

Während des Patchens wird nicht nur der Patch in das Programmpaket eingefügt, sondern auch der Package-Name des Pakets geändert, sowie die Authority-Identifikation aller `ContentProvider` adaptiert. Dies ist notwendig, um die modifizierte Version der Software parallel zur ursprünglichen Version installieren zu können. Letztere wird deaktiviert, bleibt aber am Gerät installiert. Auf diese Weise können Nutzer weiterhin wie gewohnt Updates über den Google Play Store beziehen. Wird ein solches Update installiert, wird auch von unserer Lösung automatisiert eine aktualisierte gepatchte Version erzeugt.

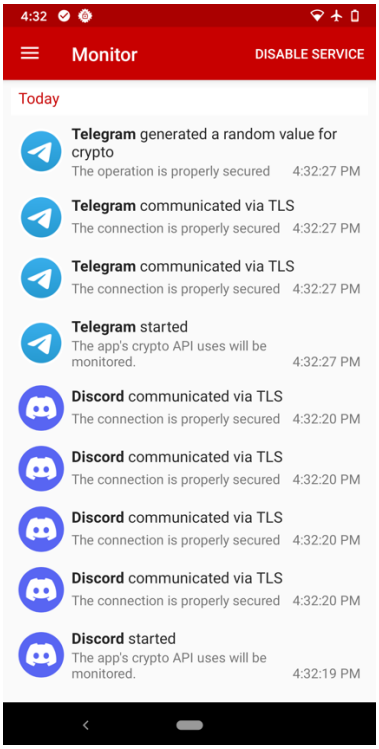
Um am Android-Gerät die Berechtigung zu erhalten, ohne zusätzliche Nutzerinteraktion Programmpakete installieren oder deaktivieren zu können, nutzt unsere Lösung die Android Device Administration API[2], die üblicherweise dazu verwendet wird, größere Flotten von Geräten zum Beispiel innerhalb von Firmen zu administrieren.

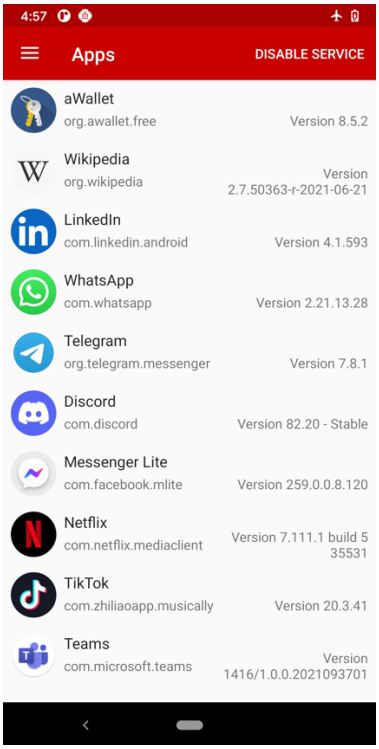
5.2. Benutzeroberfläche

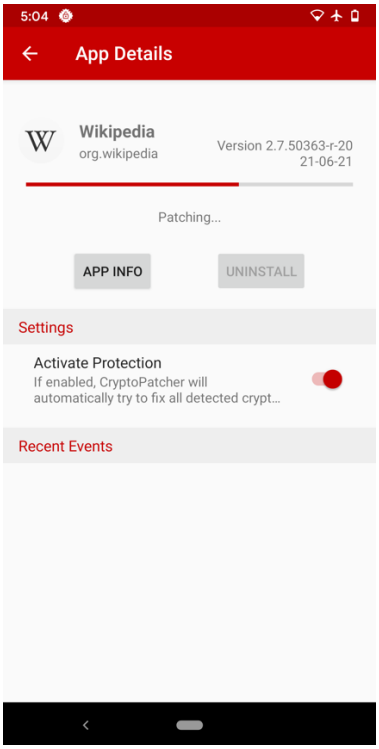
Über den Hauptbildschirm unserer Anwendung (Abbildung 1a) können erfahrene Nutzer Informationen einholen über die verschiedenen kryptografischen APIs, die von den überwachten Programmen benutzt werden. Dazu werden für relevante Aufrufe nicht nur die genaue Schnittstelle und übergebenen Parameter protokolliert, sondern auch erweiterte Informationen, die von den Mantelobjekten aus anderen Funktionsaufrufen gesammelt wurden.

Der Überblicks-Bildschirm über alle aufgezeichneten Ereignisse enthält auch kurze Beschreibungen, die neugierigen, aber technisch nicht versierten Nutzern Aufschluss geben können. Zusätzlich werden verschiedene Gefahrenstufen mittels Farbkodierung übersichtlich dargestellt.

Auf einem weiteren Bildschirm (Abbildung 1b) erhalten Nutzer eine Liste aller überwachten Applikationen, die am Gerät installiert sind. Die Auswahl eines Eintrags führt zu einem ausführlichen Detailbildschirm (Abbildung 1c). Neben Informationen zum Patching-Fortschritt laufender Operationen gibt es hier auch die Möglichkeit, bei Bedarf die Parameter-Korrekturen für einzelne Apps zu deaktivieren. Diese Option macht vor allem Sinn für Forscher, die wissen möchten, wie Apps im Originalzustand funktionierten.







[2] Device Administrator API: https://developer.android.com/guide/topics/admin/device-admin

Abbildung 1: Die Benutzeroberfläche unserer Lösung – (a) Protokoll der Aufrufe von kryptografischen APIs, (b) Liste von Anwendungen, (c) Anwendungs-Details mit Patch-Fortschritt

6. Evaluierung

Um zu demonstrieren, wie unsere Lösung wirksam Sicherheitslücken schließen kann, die durch Fehler im Umgang mit kryptografischen Programmierschnittstellen entstehen, werden im Folgenden zwei verschiedene populäre Android-Applikationen vorgestellt, die aufgrund von Crypto-API-Misuse den Diebstahl sensibler Nutzerdaten riskieren. Wir zeigen, dass unsere Lösung diese Angriffe wirksam verhindert.

6.1. Amaze File Manager

Wie schon der Name suggeriert, handelt es sich bei dieser Software um einen Dateiverwalter. Das Open-Source-Projekt hat zusammen mit verschiedenen Derivaten in Summe allein von Google Play mehr als 10 Millionen Downloads angesammelt. Zusätzlich zu Funktionen, die für diese Programmkategorie üblich sind, bietet *Amaze* auch die Möglichkeit, Dateien zu verschlüsseln.

Wird dieses Angebot auf einem Gerät genutzt, das mit unserer Lösung ausgestattet ist, schlägt diese Alarm: Die wiederholte Benutzung desselben IVs wird gemeldet. Die Nachschau im Quellcode von *Amaze File Manager*[3] bestätigt diese Warnung:

```
package com.amaze.filemanager.filesystem.files;

...

public class CryptUtil {
    // TODO: Generate a random IV every time, and keep track of it (in database against
    // encrypted files)
    private static final String IV = BuildConfig.CRYPTO_IV; // "LxbHiJhhUXcj"

    private void aesEncrypt(BufferedInputStream inputStream, BufferedOutputStream outputStream){
        Cipher ciph = Cipher.getInstance(ALGO_AES);
        GCMParameterSpec paramSpec = GCMParameterSpec(128, IV.getBytes());
        ciph.init(Cipher.ENCRYPT_MODE, getSecretKey(), paramSpec);
        ...
    }
}
```

Listing 1: Exzerpt der Datei-Verschlüsselungs-Funktion von *Amaze File Manager*

Wie kritisch das Sicherheitsproblem ist, das aus dieser Nachlässigkeit resultiert, möchten wir anhand des folgenden Beispiels demonstrieren. Für die kryptografischen Eigenschaften des AES-Algorithmus im GCM-Modus ist es entscheidend, dass die Kombination aus Schlüssel und IV nur einmal verwendet wird. Andernfalls kann aus zwei Geheimtexten C_1 und C_2 (mit demselben Schlüssel und IV chiffriert) und einem der beiden Klartexte P_1 der zweite Klartext P_2 berechnet werden als $P_2 = C_1 \oplus C_2 \oplus P_1$. Da Dateien, die vom Nutzer mittels *Amaze File Explorer* verschlüsselt wurden, im externen (öffentlichen) Speicher abgelegt werden müssen, kann jede andere App, die mit entsprechenden Zugriffsberechtigungen ausgestattet ist, alle gleich verschlüsselten Dateien nur durch Erraten eines einzigen Klartextes entschlüsseln. Ein solcher Angriff wird durch den Patch unserer Lösung zuverlässig verhindert. Durch Einschleusen eines frischen IVs ist die oben geschilderte Formel zum Erlangen von P_2 nicht mehr gültig.

Dieser Fall zeigt, dass Entwicklern teils sogar bewusst ist, dass sie die kryptografischen APIs des Android-Frameworks nicht korrekt verwenden. Aufgrund mangelnder Fachkenntnisse und Erfahrung sind sie allerdings nicht in der Lage, adäquate Lösungen zu implementieren. Das Beispiel

[3] Amaze File Manager: https://github.com/TeamAmaze/AmazeFileManager/

unterstreicht damit den Bedarf für eine Drittanbieter-Lösung, die Android-Apps ohne Mithilfe ihrer Entwickler absichern kann.

6.2. Facebook Messenger Lite

Facebook Messenger ist eine Internet-Kommunikationsplatform, auf der sich Nutzer via Textnachrichten und Videotelefonie austauschen können. Die hier untersuchte Lite-Version der Anwendung wurde laut Google Play über 500 Millionen Mal heruntergeladen. Sie verfügt über einen leicht eingeschränkten Funktionsumfang, um die Kompatibilität mit besonders leistungsschwachen Android-Geräte der Einsteigerklasse sicherzustellen.

Die Anwendung verwendet allerdings eine unsichere Network Security Configuration (NSC)[4]. Das NSC-System wurde in Android 7.0 eingeführt, um Entwicklern die statische Konfiguration von vertrauenswürdigen Zertifikaten zu vereinfachen. Anstelle von fehleranfälligen eigenen Implementierungen von `TrustManager` und `HostnameVerifier` kann hierzu eine XML-Datei benutzt werden.

Facebook Messenger Lite bestimmt in der NSC-Konfiguration, dass auch Zertifikaten vertraut werden soll, die vom Nutzer im System-Zertifikatsspeicher hinterlegt wurden. Üblicherweise wird eine solche Regelung nur in Entwicklungsversionen von Anwendungen aktiviert und vor der Freigabe an die Öffentlichkeit entfernt.

Dass die Software auch in der öffentlichen Version Nutzer-installierten Zertifikaten vertraut, führt dazu, dass ein MITM-Angreifer Anwender nur dazu bewegen muss, ein Zertifikat für das MITM-Relay zu installieren. Dies kann zum Beispiel auch mit einer Clickjacking-Attacke [8] relativ einfach erledigt werden. Wie unsere Experimente zeigen, kann ein Angreifer ab diesem Zeitpunkt zum Beispiel die Anmeldedaten eines Nutzers von *Facebook Messenger Lite* auslesen, was zur Übernahme des Benutzerprofils führt.

Auch dieser Angriff kann mit unserer im Rahmen dieses Projekts entwickelten Lösung erfolgreich verhindert werden. Das Patch-Modul erkennt die Abweichung zwischen dem vom vermeintlichen Facebook-Server gelieferten Zertifikat und dem bekannten legitimen Zertifikat, wie es über den Web Service abgefragt werden kann. Als Reaktion auf den MITM-Angriff wird daher die Kommunikation eingestellt, noch bevor Nutzerdaten (wie etwa die Anmeldedaten) übermittelt wurden.

7. Diskussion

In diesem Abschnitt gehen wir auf die bekannten Einschränkungen und mögliche Verbesserungen unserer Lösung ein.

7.1. Kompatibilität

Durch die Veränderung des Paket-Namens und des Authority-Identifikators von `ContentProvider`n ergibt sich auch der Bedarf, Aufrufe von verschiedenen Funktionen zu überwachen, denen der Anwendungscode möglicherweise einen statisch hinterlegten ursprünglichen Identifikator übergibt. Um in diesen Fällen die korrekte Operation der Anwendung sicherzustellen, müssen diese Parameter korrigiert werden, bevor sie an die ursprünglich aufgerufene Funktion weitergereicht werden. Obwohl hier große Anstrengungen unternommen wurden, gibt es immer noch einzelne Anwendungen, die nicht korrekt funktionieren. Von den 100 beliebtesten Programmen aus 31 Kategorien auf Google Play waren 4 betroffen.

7.2. Signatur-Überprüfungen

Durch die Modifikation von Programmpaketen ändert sich auch deren Signatur. Dies hat zur Folge, dass Anwendungen erkennen können, ob sie verändert wurden und in diesem Fall zum Beispiel den

[4] Network Security Configuration: https://developer.android.com/training/articles/security-config

Betrieb verweigern. Dieses Vorgehen wird besonders von Entwicklern gewählt, die die böswillige Manipulation ihrer Software etwa zum Zwecke der Verbreitung von Viren unterbinden möchten. Obwohl eine Umgehung technisch in vielen Fällen möglich wäre, haben wir uns bewusst dazu entschieden, die Intention hinter Signatur-Überprüfungen zu respektieren. Unseren Analysen zufolge betrifft diese Einschränkung nur eine kleine Minderheit der verfügbaren Android-Anwendungen. Von den 100 beliebtesten Programmen aus 31 Kategorien auf Google Play waren nur 2 betroffen.

7.3. Benutzerfreundlichkeit

Unsere Bemühungen für größtmögliche Nutzerfreundlichkeit wurden an einigen wenigen Stellen vom Android-System boykottiert. So zeigt der Android-Startbildschirm auch die Icons deaktivierter Anwendungen (in ausgegrauter Farbe) an, weshalb für jede installierte Anwendung zwei Icons aufscheinen (die ursprüngliche, deaktivierte Version und die gepatchte Version). Dieses Problem lässt sich aber zum Beispiel durch Bereitstellung eines angepassten Applikations-Launchers relativ einfach beheben. Außerdem wird in der Standardkonfiguration nach der Installation von gepatchten Anwendungen von Google Play Protect[5] in einem Dialog nachgefragt, ob die unbekannte Software auf Googles Servern auf bekannte Merkmale von Schadsoftware überprüft werden soll. Auch dieses Problem lässt sich glücklicherweise einfach beheben, indem diese Überprüfung in den Google-Einstellungen deaktiviert wird. Für Nutzer, die nur Apps vom offiziellen Play-Store beziehen, ergibt sich daraus keine Einschränkung der Geräte-Sicherheit, da Anwendungen dort schon beim Upload automatisiert analysiert werden.

8. Zusammenfassung

In diesem Projekt wurde eine Lösung erarbeitet, um in kompilierten Android-Anwendungen Sicherheitslücken zu schließen, die durch falsche Verwendung von kryptografischen APIs entstehen. Die Lösung läuft direkt auf dem Android-Gerät und benötigt weder Root-Berechtigungen, noch ein modifiziertes Betriebssystem. Dieser Bericht beschrieb die beiden Kernbestandteile des Systems, den Patch zur Überwachung und Korrektur von Crypto-API-Misuse und die Applikation, die den Patch in installierte Anwendungen injiziert sowie verschiedene Verwaltungsaufgaben erfüllt. In zwei anschaulichen Beispielen haben wir die potentiell verheerenden Auswirkungen von Crypto-API-Misuse demonstriert und gezeigt, wie unsere Lösung den Diebstahl sensibler Daten verhindert.

9. Bibliographie

[1] D. B. Y. F. C. K. Manuel Egele, „An empirical study of cryptographic misuse in android applications," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[2] S. I. S. J. R. A. D. B. V. S. Martin Georgiev, „The most dangerous code in the world: Validating SSL certificates in non-browser software," 2012.

[3] D. L. T. L. R. H. D. Siqi Ma, „CDRep: Automatic Repair of Cryptographic Misuses in Android Applications," in *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, 2016.

[4] M. H. G. M. E. R. W. Damjan Buhov, „Pin it! Improving Android network security at runtime," in *IFIP Networking Conference, Networking and Workshops*, 2016.

[5] N. H. S. A. Y. A. M. B. S. F. Marten Oltrogge, „Why Eve and Mallory Still Love Android: Revisiting TLS (In)Security in Android Applications," in *30th USENIX Security Symposium*, 2021.

[6] B. H. S. F. Charles Weir, „From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security," in *29th USENIX Security Symposium*, 2020.

[5] Google Play Protect: https://support.google.com/googleplay/answer/2812853?hl=en

[7] K. R. F. J. P. M. H. M. L. M. M. H. Daniel Votipka, „Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It,“ in *29th USENIX Security Symposium*, 2020.

[8] C. Q. S. P. C. W. L. Yanick Fratantonio, „Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop,“ in *IEEE Symposium on Security and Privacy (SP)*, 2017.