18. Juni 2019 Studie · Mobile Sicherheit · 23 Min. Lesezeit

Maßnahmen mobiler Betriebssysteme gegen Malware

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

Stand 2019-06-18
Quelle https://technology2.a-sit.at/veroeffentlichungen/massnahmen-mobiler-betriebssysteme-gegen-malware/

Dieses Projekt evaluiert Maßnahmen mobiler Betriebssysteme um Anwenderinnen und Anwender vor Malware zu schützen. Es wird ein Überblick über aktuell verwendete Konzepte auf Android und iOS geboten. Anschließend werden mithilfe von statischer Analyse Beispiele von Schadsoftware analysiert und deren Laufzeitverhalten auf Android-Emulatoren in Hinblick auf die erlangten Berechtigungen, den Zugriff auf sensible Daten und die Möglichkeit weiteren Code nachzuladen, untersucht.

1. Motivation

Schadsoftware, öfter bekannt als „Malware“, umfasst Software deren Zweck es ist, für Benutzerinnen und Benutzer nachteilige Effekte zu bewirken. Dabei können etwa Daten verschlüsselt werden (Ransomware), Funktionen des Systems verändert oder übernommen werden (Clickjacking, © A-SIT, Partner für Sicherheitsfragen Botnetze, Cryptominer), ohne das Wissen der Anwenderinnen und Anwender sensible Informationen ausgelesen werden (Spyware) oder Maßnahmen zur Infektion weiterer Systeme ergriffen werden (Virus). Meist sind die Übergänge zwischen einzelnen Kategorien fließend und können auch kombiniert werden, sodass Beispiele solcher Software nicht eindeutig zugeordnet werden können. [1] Da Malware Ressourcen wie etwa Nutzerdaten oder Systemrechenleistung nutzt, erleiden Anwenderinnen und Anwender nicht nur finanziellen Schaden, sondern unter Umständen auch erhebliche Verletzungen der Privatsphäre. Während auf dem PC die Verwendung von Drittanbieter-Antiviren-Programmen zur Abwehr weit verbreitet ist, sind Smartphones von Haus aus mit Schutzkonzepten versehen. So sind installierte Apps voneinander durch Sandboxing getrennt und können nicht ohne besondere Erlaubnis durch die Nutzerin oder den Nutzer auf Daten einer anderen App zugreifen, während Desktopprogramme meist willkürlicher Zugriff auf das ganze System erlaubt wird. Diese Trennung umfasst auch die Verwendung von Hardware-Schnittstellen wie etwa Mikrofon und Kamera. Um Smartphone-Applikationen überhaupt aus offiziellen Quellen installieren zu können, muss diese Applikation von der Entwicklerin oder dem Entwickler zuvor

signiert werden. Dies soll verhindern, dass Schadsoftware zusammen mit legitimen Programmen ausgeliefert wird. Zunehmend jedoch setzen auch Desktop-Betriebssysteme auf die beschriebenen Techniken.

Solche Schutzmechanismen können allerdings auch umgangen werden. Durch Ausnutzen von Sicherheitslücken, die dem Hersteller noch unbekannt („Zero-Day Exploit“) oder noch nicht vom Selbigen behoben wurden („One-Day Exploit“), kann ein System gezielt mit Malware infiziert werden. Häufiger jedoch wird ein Schadprogramm von der Benutzerin oder dem Benutzer unwissentlich selbst installiert. Auf Android-Systemen können die Nutzerin oder der Nutzer etwa dazu gebracht werden, die Installation von Applikationen „aus unbekannten Quellen“ zu erlauben, was die Installation beliebiger Programme ermöglicht. Oft locken Drittanbieter-Stores mit dem Gratis-Download von Apps, die in offiziellen Quellen zahlungspflichtig sind. In vielen Fällen sind diese Gratis-Versionen modifiziert und beinhalten zusätzlich Schadprogramme.

Applikationen wie Google Play Protect [2] sollen zur Sicherheit der Anwenderinnen und Anwender beitragen, indem Pakete im Play Store mit Hilfe von Machine Learning analysiert werden und anschließend Applikationen am Gerät selbst mit den gelernten Signaturen verglichen werden. Bei einer als potenziell schädlich erkannten App wird der Benutzerin oder dem Benutzer ein Warnhinweis angezeigt, die Ausführung der App verhindert oder die Deinstallation vorgeschlagen.

2. Projektziel

In diesem Projekt wurde ein Überblick über bestehende Maßnahmen in Kapitel 3 erarbeitet und anschließend anhand von aktuell kursierenden Beispielen untersucht, inwiefern die Schutzmechanismen von modernen mobilen Betriebssystemen Schadsoftware eindämmen. Hierbei wurde aufgrund der größeren Verfügbarkeit von Schadsoftware der Fokus der Malwareuntersuchung auf die Android Plattform gelegt. Sämtliche Beispiele werden zuerst einer statischen Analyse unterzogen, um festzustellen welche Berechtigungen die Applikation erfordert und für welche Zwecke diese im Quellcode genutzt werden. Im Rahmen der anschließend erfolgenden dynamischen Untersuchung werden Logdateien sowie ein- und ausgehende Netzwerkpakete ausgewertet. Des Weiteren wird an dieser Stelle auch geprüft, ob und welche Berechtigungen zur Laufzeit angefragt werden und ob zusätzlich weitere Applikation installiert werden. Abschließend soll evaluiert werden, inwiefern Google Play Protect die Applikation während der Ausführung als Schadsoftware erkennt und welche Maßnahmen ergriffen werden.

3. Bestehende Schutzmaßnahmen

Das Betriebssystem von Mobilgeräten, die nicht „rooted“ beziehungsweise „jailbroken“ sind, beinhalten bereits eine Reihe von Maßnahmen zum Schutz der Nutzerdaten. Sowohl auf Android als auch auf iOS bestehen Konzepte, um den Zugriff auf Applikationsfremde Daten einzuschränken und die Integrität der Applikation und des Codes sicherzustellen. Obwohl die Ansätze beider Plattformen sich ähneln, sind diese auf verschiedene Weisen umgesetzt. Diese Unterschiede werden nachfolgend erläutert.

3.1. Signieren von Applikationen

Zum Schutz vor Modifikation des Programmcodes müssen Applikationen auf Android und iOS vor der Verbreitung signiert werden. Android-Applikationen werden mit einem von der Entwicklerin oder dem Entwickler selbst ausgestellten Zertifikat signiert und bei der Installation vom Betriebssystem geprüft. Bei

einem Update wird die neue Version ebenfalls einer Zertifikatsprüfung unterzogen und nur bei Übereinstimmung ersetzt. Es gibt jedoch keinen Weg, bei einer erstmaligen Installation zu überprüfen, ob die Applikation verändert wurde und mit einem neuen Zertifikat signiert wurde. Aufgrund der offenen Philosophie des Android-Betriebssystems können Apps weiteren Code von anderen Quellen herunterladen und ausführen. Dieser Code ist nicht Teil der Signaturprüfung und kann aus schadhaften Instruktionen bestehen. Zertifikate zum Signieren von iOS Apps müssen von der Entwicklerin oder dem Entwickler bei Apple beantragt werden. Die Entwicklerin oder der Entwickler werden dabei verifiziert und die Applikation einem manuellen Review unterzogen. Bei diesem wird die Einhaltung der Developer Guidelines [3] durch Ausführen der Applikation überprüft. Die Signatur der Applikation wird vor jeder Ausführung validiert und erstreckt sich auch auf alle verwendeten Bibliotheken. [4] Der Review-Prozess kann umgangen werden mit dem Kauf eines Enterprise-Zertifikates. Gedacht für die Entwicklung von Firmeninternen Applikationen werden damit signierte Apps nicht im App Store gelistet und können direkt auf Mitarbeitergeräten installiert werden. Einzige Voraussetzung ist das Vorhandensein eines entsprechenden Profils, das die Nutzerin oder der Nutzer am Gerät installieren muss. Enterprise-Zertifikate werden oftmals zum Aufbau von Drittanbieter-Marktplätzen genutzt, in denen Benutzer damit signierte Applikationen beziehen können, die nicht den Regeln des App Stores entsprechen. Nicht selten enthalten diese Apps neben Glücksspielen oder pornografischen Inhalten auch bösartigen Code. Bei Bekanntwerden solcher Fälle wird das entsprechende Enterprise-Zertifikat von Apple zurückgezogen und die Ausführung damit vertriebener Applikationen unterbunden. Ein dynamisches Nachladen von weiteren Instruktionen ist unter iOS aufgrund von Hardwareschutzmaßnahmen nicht möglich. Dies wird in Abschnitt 3.5 genauer ausgeführt.

3.2. Sandboxing

Auf beiden Plattformen werden Applikationen in voneinander isolierten Umgebungen ausgeführt. Diese Umgebung ist als Sandbox bekannt und verhindert den Zugriff auf Hardware-Komponenten und Daten außerhalb der Umgebung. Android ordnet dafür jeder App bei der Installation eine UNIX User ID (UID) zu, die Applikation wird anschließend immer mit dieser UID ausgeführt. Beim Zugriff auf Ressourcen anderer Applikation prüft der Linux Kernel, der die Basis des Android Betriebssystems darstellt, die UID des aufrufenden Prozesses sowie die Zugriffsbits der Ressource und verweigert oder erlaubt diesen abhängig vom Ergebnis. [5] Auf iOS wird jede Applikation in einen dedizierten Ordner mit zufällig erstellten Namen installiert. Zugriffe außerhalb dieses Ordners sind nur über Systemschnittstellen möglich. Die benutzten Schnittstellen müssen in einer eigenen Datei angegeben werden und sind Teil des von Apple durchgeführten Reviews. Nach positivem Ausgang wird diese als Teil der Applikation mitsigniert, die Signaturüberprüfung erfolgt bei jedem Aufruf der Systemschnittstelle. Diese ist mit einem Appleinternen Zertifikat signiert und läuft als unprivilegierter Nutzer, um das Risiko einer Privilegienerweiterung zu minimieren. Applikationserweiterungen wie etwa „Teilen“ werden in einer eigenen Sandbox ausgeführt und können mit der zugehörigen Applikation ebenfalls nur über Systemschnittstellen kommunizieren. Ein spezieller Fall sind Tastaturerweiterungen, die auch in anderen Applikationen verwendet werden müssen. Bei diesen wird sämtlicher direkter Netzwerkzugriff sowie die Netzwerkverbindung der zugehörigen App deaktiviert, um die Weiterleitung von eingegebenen Text an Dritte zu verhindern. Des Weiteren werden Tastaturerweiterungen nicht für sichere Eingabefelder und Passcode-Eingaben verwendet. [4]

3.3. Applikationsberechtigungen

In vielen Szenarien ist es nötig, dass Apps diese Sandbox verlassen und etwa Hardware-Komponenten wie die Kamera verwenden können. Zu diesem Zweck können Anwenderinnen und Anwender auf beiden Plattformen entsprechende Berechtigungen erteilen. Das Android-Berechtigungssystem wird in mehrere Kategorien unterteilt, von diesen sind Installationsberechtigungen und Laufzeitberechtigungen die bekanntesten. Die erste Kategorie beinhaltet Berechtigungen, die „kein großes Privatsphären- oder Sicherheitsrisiko darstellen“ [6] und bei Installation automatisch erteilt werden. Dazu gehören etwa Zugriff auf Bluetooth oder WLAN, Internet oder das Erstellen eines Weckers. Laufzeitberechtigungen wurden mit Android 6.0 eingeführt und umfassen „gefährliche“ Berechtigungen, die ab dieser Version zur Laufzeit explizit erbeten werden müssen. Im Gegensatz zu Installationsberechtigungen können diese auch jederzeit widerrufen werden. Beispiele für diese Art von Berechtigungen sind Kamerazugriff, das Lesen und Schreiben von SMS oder die Verwendung der Anrufhistorie. Vor Android 6.0 wurden diese Berechtigungen, nach einem Hinweis beim Download im Play Store, ebenfalls bei Installation automatisch erteilt. Das beschriebene Berechtigungssystem besitzt jedoch eine Reihe von Einschränkungen. So ist es ohne weiteres möglich, mittels „ungefährlicher“ Installationsberechtigungen Hintergrundservices in Applikationen zu verwenden, die etwa empfangene SMS-Nachrichten oder Push-Notifikationen lesen können und somit auch auf Nachrichten, die für eine 2-Faktor-Authentifizierung genutzt werden, Zugriff erhalten. Anwenderinnen und Anwendern wird dabei nur bei Installation angezeigt, dass die Applikation die entsprechenden Berechtigungen benötigt, jedoch nicht warum diese nötig ist. Durch die offene Philosophie des Android-Betriebssystems ist es möglich, Apps von Drittanbietern zu installieren. Dabei kann die Anzeige von benötigten Berechtigungen komplett entfallen und diese würden automatisch erteilt werden. Bei iOS müssen sämtliche genutzten Berechtigungen beim Code Review kenntlich gemacht und von der App zur Laufzeit erbeten werden. Die Benutzerin oder der Benutzer können erteilte Berechtigungen jederzeit wieder entziehen, Systemschnittstellen verhindern den entsprechenden Zugriff mit sofortiger Wirkung.

3.4. Zugriff auf Benachrichtigungen

Zum Nachweis der Identität werden vermehrt multiple Faktoren genutzt. Dafür ist besonders die Kombination aus Zugangsdaten und einem Besitznachweis, etwa des vorab gekoppelten Smartphones oder der Telefonnummer, aufgrund der großen Verfügbarkeit beliebt. Der Besitz einer Telefonnummer kann nachgewiesen werden, indem an diese Nummer eine SMS mit einem Code geschickt wird. Dieser Code besitzt nur ein kurzes Gültigkeitsfenster und muss von der Besitzerin oder dem Besitzer eingegeben werden. Für den Besitz eines gekoppelten Smartphones wird der Code meist über Push-Nachrichten an eine entsprechende Applikation gesendet. Der Code wird üblicherweise von einer Serverapplikation erstellt und an Server des Push-Anbieters übertragen, der den Inhalt an die Zielgeräte weiterleitet. Die meisten Android-Applikationen nutzen dafür Firebase Cloud Messaging, einen Dienst von Google, dessen Verbindungen transportverschlüsselt sind. Der Inhalt ist abseits des Transports auf den Servern des Anbieters im Klartext verfügbar und sollte aus diesen Gründen End-zu-Endverschlüsselt werden. [7] Wie in Abschnitt 3.3 beschrieben, können Android-Applikationen Zugriff auf die Benachrichtigungszentrale erhalten, damit sämtliche lokalen Benachrichtigungen lesen und auch deren Anzeige unterbinden. Ab Android 9.0 kann allerdings einzelnen Apps zur Laufzeit der Zugriff auf Mitteilungen komplett untersagt werden. Die Android-Philosophie ermöglicht es auch, multiple SMS-Applikationen installiert zu haben. Sollte eine Applikation dafür registriert sein, wird bei Erhalt einer neuen SMS deren Inhalt an die App weitergegeben. Im Falle von Schadsoftware kann damit ein zugesendeter Authentifizierungscode vor der Benutzerin oder dem Benutzer verborgen und an Dritte gesendet werden. Beginnend mit Android 9.0

können Laufzeitberechtigungen, die Anrufe oder SMS betreffen, nur noch von der App beantragt werden, die als standardmäßige SMS- beziehungsweise Telefon-Applikation in den Systemeinstellungen festgelegt wurde. [8] Im Gegensatz zu Android-Applikation können iOS Apps nur Benachrichtigungen lesen, die auch der App zugeordnet sind. Die Verwendung von Push-Notifikationen setzt ein dafür registriertes Zertifikat voraus, das von Apple ausgestellt werden muss und eine global eindeutige Applikations-ID beinhaltet. Dies hat zur Folge, dass Apple-Server die einzige Möglichkeit zur Übermittlung von Push-Mitteilungen darstellen. Der Mitteilungstransport erfolgt verschlüsselt, der Inhalt könnte aber auf den Servern von Apple im Klartext gelesen werden. Seit iOS 10 empfiehlt es sich, den Inhalt bereits am Hintergrundservice zu verschlüsseln und mittels einer Systemschnittstelle vor der Anzeige der Nachricht am Gerät zu entschlüsseln. [9] Unter iOS ist es nicht möglich, den kompletten Inhalt einer SMS auszulesen. Falls ein Authentifizierungscode darin enthalten ist, wird dieser anhand eines Musters vom Betriebssystem erkannt. Sollte auf der aktuell angezeigten App ein Eingabefeld mit passendem Typ ausgewählt sein, wird der Code auf der Tastatur zur automatischen Eingabe vorgeschlagen. Dies muss die Benutzerin oder der Benutzer jedoch explizit bestätigen.

3.5. Weitere Laufzeitschutzmaßnahmen

Als weitere Maßnahme beinhalten beide Betriebssysteme noch Komponenten, welche die Ausführung von Applikation beziehungsweise deren Programmcode zur Laufzeit überwachen. Android setzt dafür auf die Applikation Google Play Protect, die Schadsoftware erkennen und eindämmen soll. Dazu werden sämtliche Apps im Google Play Store gescannt und mithilfe eines Machine-Learning-Modells analysiert. Signaturen von Applikationen, die als bösartig erkannt wurden, werden in regelmäßigen Abständen an Android-Geräte verteilt. Auf den Geräten selbst erfolgt eine ständige Überprüfung der installierten Applikationen auf eine Übereinstimmung mit diesen Signaturen. Sollte dies der Fall sein, wird je nach Schweregrad ein Warnhinweis eingeblendet, die Ausführung verhindert oder die App sofort deinstalliert. [10] Unter iOS wird als erweiterte Schutzmaßnahme auf Hardwarekomponenten gesetzt. Da iOS-Applikationen im Gegensatz zu Android aus ausführbarem Maschinencode bestehen, wird der Speicheradressraum standardmäßig randomisiert, um das Ausnutzen von Speicher-basierten Schwachstellen zu erschweren. Zusätzlich ist sämtlicher Adressraum, der keinen signierten Code enthält, für den Prozessor als nicht ausführbar gekennzeichnet (*ARM never execute*). Dies verhindert, dass weitere Instruktionen zur Laufzeit von einem Server nachgeladen werden können, da die Applikation diesen nur in nicht ausführbaren Adressen speichern kann. Ein Schreibzugriff auf Adressen, die als ausführbar gekennzeichnet sind, wird bereits Hardwareseitig verhindert. [4]

4. Untersuchte Malware

Die Applikationen *Spotify Crack*, *iCoyote* und *Squaremaker* wurden ausgewählt, da diese auf der Malware-Analyse-Plattform *HybridAnalysis* [1] zum Zeitpunkt des Projektbeginns die letztgeprüften Android-Apps waren, die auch als Schadsoftware klassifiziert wurden. Beispiele für *Fakebank* und *Reddrop* wurden von *Contagio Mobile* [2] zur Verfügung gestellt. Dabei handelt es sich um bekannte Schadsoftware, die allerdings noch immer im Umlauf ist. Als letzte Applikation wurde *Exodus* untersucht, da Beispiele davon während des Projektzeitraumes aus Drittanbieter-Marktplätzen wie *ApkPure* [3] leicht verfügbar waren.

4.1. Spotify Crack

Package Name: com.spotify.music SHA256: 5a50189d53db57f5741952b82e8b6cd74775e1285fb054a15209adc30b51ad56 Bei *Spotify Crack* handelt es sich um einen modifizierten Client für den Musikstreaminganbieter *Spotify*, der auf der offiziellen Applikation basiert und die Authentifizierung für den Konsum von Premiumdiensten umgehen soll.

1 https://www.hybrid-analysis.com/ 2 https://contagiominidump.blogspot.com/ 3 https://apkpure.com/

4.2. iCoyote

Package Name: com.coyotesystems.android SHA256: 449043c934a649530c93ea6e5103e1c42813e0b04c0795d45e41c6394c30eb20 *iCoyote* ist eine Verkehrswarnapp, bei der Nutzerinnen und Nutzer Geschwindigkeitslimits oder Behinderungen wie etwa Stau oder Baustellen eintragen können und bei der Fahrt darauf aufmerksam gemacht werden.

4.3. Squaremaker

Package Name: com.baiwang.squaremaker SHA256: 41213cc0f358d4f7bc28ea1cd037e42c739aaadaee7a852c386228da5b280e9e Diese App bietet die Möglichkeit, Filme und Videos aufzunehmen, welche danach mit vorgefertigten Filtern und Effekten versehen und über soziale Netzwerke verbreitet werden können.

4.4. Fakebank

Package Name: com.example.kbtest/com.xinhan.smsmanager SHA256: 4aeccf56981a32461ed3cad5e197a3eedb97a8dfb916affc67ce4b9e75b67d98 *Fakebank* ist eine modifizierte Version der offiziellen Banking-App der Shinhan Bank in Südkorea, mit der Kundinnen und Kunden ihre Bankgeschäfte abwickeln können. Dazu wird die Eingabe eines Passworts und einer Transaktionsbestätigungsnummer, die per SMS erhalten wird, zur Bestätigung jeder Transaktion genutzt.

4.5. Reddrop

Package Name: com.wimkek.gpctplzy SHA256: 03bf40f154460e245fd242c2f297170eafd1417c1de0f9ac8356d29f8855fe54 Diese Version von *Reddrop* ist ein Spiel, entwickelt mit dem Plattform-übergreifendem Spiele Framework Cocos2d-x.

4.6. Exodus

Package Name: assistenza.linea SHA256: db59407f72666526fca23d31e3b4c5df86f25eff178e17221219216c6975c63f Diese App präsentiert sich Nutzerinnen und Nutzern als Support-Programm eines italienischen Mobilfunkanbieters, die Benutzerinnen und Benutzer bei Problemen mit dem Mobilfunknetz verwenden können.

5. Statische Analyse

Mithilfe der statischen Analyse soll sichergestellt werden, dass es sich bei der Applikation um schädliche Software handelt sowie bösartiger Code identifiziert werden. Als Vorbereitung dazu wird zuerst das Android Paket entpackt und anschließend das Manifest, der Quellcode und die zusätzlichen Ressourcen extrahiert.

Das Manifest beinhaltet Informationen wie etwa den Namen der installierten App, unterstützte Versionen und benötigte Berechtigungen und Systemservices. Im Paket ist diese nur in einem proprietären Binär-Format enthalten und wird zuerst in eine lesbare XML-Struktur umgewandelt. Dafür werden das Android Asset Packaging Tool *aapt* [11] sowie das Open-Source Programm *AXMLPrinter* [12] verwendet.

Die enthaltenen Berechtigungen und Services werden anschließend unter Betrachtung des Zwecks der Applikation geprüft. Die Verwendung von „gefährlichen Berechtigungen" [13] wie etwa *RECORD_AUDIO* oder *CAMERA* deutet bei Apps, deren Verwendungszweck diese Funktion nicht rechtfertigt, oft auf bösartigen Code hin. Als nächster Schritt werden die statischen Ressourcen untersucht. Diese bestehen aus Mediendateien, Layout- und User Interface-Definitionen sowie Lokalisierungsdateien, die statische Strings beinhalten. Letztere werden auch zur Konfiguration der App benutzt und können daher URLs oder IP-Adressen enthalten. Des Weiteren ist es möglich, mittels Kodierungsverfahren wie Base64 ausführbaren Code als String darin zu verstecken. Zuletzt wird der Programmcode, der im *Dalvik Executable* [14] Format in der Datei *classes.dex* enthalten ist, mit dem Open-Source Programm Ghidra [15] dekompiliert. In diesem Prozess werden die ausführbaren *Dalvik Virtual Machine* -Instruktionen in ein Java-ähnliches Format rückgeführt. Dieses leichter lesbare Format kann anschließend auf die Verwendung der zuvor ermittelten Berechtigungen und Ressourcen analysiert werden. Verstärkt werden in diesem Schritt Teile untersucht, die Netzwerkverbindungen betreffen oder das Starten von Hintergrundprozessen zum Ziel haben. Die hieraus gewonnenen Erkenntnisse bilden die Grundlage für die anschließende dynamische Analyse.

6. Dynamische Analyse

Im Rahmen der dynamischen Analyse wird jede App separat auf der Testumgebung mittels *Android Debug Bridge (adb)* [16] installiert. Die Testumgebung besteht aus einem Android Emulator mit Android 9.0 und installierten Google Play Services. Da Android 9.0 derzeit (Stand: 14. Mai 2019) nur die *x86* Prozessorarchitektur unterstützt, wird im Falle einer Applikation für die *arm* Architektur als neueste verfügbare Version Android 7.1.1 verwendet. Nach Installation werden typische Anwendungsfälle manuell durchgespielt und dabei die Systemund Applikationslogs sowie der entstehende Netzwerkverkehr aufgezeichnet. Die Logdateien werden daraufhin auf Ereignisse wie etwa geblockte Systemschnittstellenzugriffe oder auftretende Programmfehler untersucht. In der Aufzeichnung des Netzwerkverkehres wird geprüft, ob und welche Daten des Geräts und des Benutzers an Server geschickt werden und ob dem Nutzer dies auch bekannt ist. Zusätzlich ist darin auch ersichtlich, ob weiterer Code nachgeladen wird. Gesammelt werden die Logdateien mit dem Protokollanalyseprogramm *Wireshark* [17] über die *adb* Schnittstelle. Um den Netzwerkverkehr aufzuzeichnen wird *Burp Suite* [18] in der Community Version als Man-In-The-Middle-Proxy verwendet. Dabei werden sämtliche Netzwerkanfragen vom Emulator zuerst an den Proxy-Server der *Burp Suite* geschickt, der diese aufzeichnet und danach an den Zielserver weiterleitet. Die Antwort des Servers erhält wiederum zuerst der Proxy-Server, der diese an den Emulator weiterschickt. Während dem Ausführen der Applikation wird zusätzlich beobachtet, ob Google Play Protect aktiv wird und welche Maßnahmen davon vorgeschlagen werden.

7. Resultate

Die Resultate der Analysen werden für jede untersuchte Applikation einzeln beschrieben und die wichtigsten Punkte hervorgehoben. Abschließend wird auf die vom Betriebssystem ergriffenen Maßnahmen zur Einschränkung der Schadsoftware eingegangen.

7.1. Spotify Crack

Diese App benötigt eine Vielzahl von Berechtigungen, unter anderem Vollzugriff auf die gespeicherten Nutzeraccounts, den Netzwerkstatus und Lese- und Schreibrechte für das Dateisystem. Im Vergleich zum offiziellen Android Spotify Client werden die gleichen Berechtigungen benötigt. Die durchsuchten String-Ressourcen beinhalten Übersetzungen für europäische, amerikanische und asiatische Sprachen sowie URLs des Spotify Supports und der Privacy Policies.

Die Package-Identifier „com.spotify.music" im Quellcode der Applikation weisen ebenfalls auf die Verwendung von Code der originalen Applikation hin. Der Vollzugriff auf Nutzeraccounts wird für Frameworks wie BMW Car Connect, Facebook Login oder Sony Entertainment Account benötigt. Der Netzwerkstatus wird für das Streaming von Musik sowie für diverse Werbeplattformen wie Google Ads und Moat Analytics benötigt. Einzig der eingebundene Sony Entertainment Account Manager führt eine eindeutige Identifizierung des Gerätes mittels *International Mobile Equipment Identity* (IMEI) und SIM-Anbieterdaten erfolgreich durch. Die Dateisystem-Berechtigungen werden für das Importieren eigener Musikdateien sowie den Download weiterer Musikdateien benötigt. Installiert wurde die App auf Android 7.1.1, da diese Version nur für die *arm* Plattform verfügbar ist. In den Systemlogs waren keine auffälligen Ereignisse zu sehen, jedoch dürften die im Code vorhandenen Serveradressen veraltet sein, da diese auf Netzwerkanfragen nicht mehr antworteten. Google Play Protect kennzeichnete dieses Paket nicht als bösartig und schlug auch keine Maßnahmen vor.

7.2. iCoyote

Das Manifest von iCoyote beinhaltet einige gefährliche Berechtigungen, zum Beispiel Schreibzugriff auf das Dateisystem, die Möglichkeit Systemeinstellungen zu verändern, Hintergrunddownloads durchzuführen und den Systemstart festzustellen. Die String-Ressourcen sind in westeuropäischen Sprachen lokalisiert. Diese Region dürfte das Hauptanwendungsgebiet der App sein, da im Code die französisch/belgischen SIM-Anbieter SFR und Mobistar vorkommen und deren Verwendung erweiterte Funktionen freischalten. Dafür werden die Anbieterinformationen und Hardware-Identifikatoren in den Shared Preferences [19] gespeichert. Die Abfrage nach dem genauen Standort des Geräts wird für das Eintragen von Baustellen, Staus und Unfällen genutzt. Der Schreibzugriff auf das Dateisystem und Hintergrunddownloads werden benötigt, um Verkehrskarten offline zu speichern. Die Nutzeraccounts werden zum Teilen von Informationen über Facebook und Twitter verwendet. Teile des Programmcodes weisen auch auf die Möglichkeit hin, Updates der App aus Drittanbieterquellen zu installieren. Beim Start der App wird die IMEI angefordert, jedoch war diese nicht in Netzwerkanfragen enthalten. Google Play Protect wies bei dieser Applikation nicht auf Schadsoftware hin.

7.3. Squaremaker

Squaremaker verlangt unter anderem Berechtigungen zum Auslesen des Netzwerkstatus und der Geräteinformationen, zur Verwendung der Kamera und des Mikrofons, sowie Zugriff auf das Dateisystem und den genauen Standort des Geräts. In den String-Ressourcen finden sich Übersetzungen in mehrere asiatische Sprachen. Die Geräteinformationen werden sämtliche Geräteinformationen ausgelesen und an „DuAd"-Server geschickt. Die Verwendung der Kamera, des Mikrofons und der Zugriff auf das Dateisystem wird benötigt, um die Aufnahme von Bildern und Videos in der App zu ermöglichen, die anschließend bearbeitet werden können. Der Schreibzugriff auf das Dateisystem wird aber auch genutzt, um ein Java-Archiv (1526594665595.jar) herunterzuladen, welches anschließend als Hintergrundservice ausgeführt

wird. Bei der Ausführung wurde festgestellt, dass jede Interaktion mit der App mehrere Anfragen an Werbeanbieter auslöst und einen Klick auf Werbebanner vortäuscht. Bei all diesen Anfragen wurden sämtliche verfügbaren Gerätedaten wie IMEI, Gerätehersteller, Modell und *Media Access Control* (MAC)-Adressen inkludiert. Die MAC-Adressen wurden auf den fiktiven Wert „02:00:00:00:00.00“ geändert, alle anderen Identifikatoren wurden unverändert gesendet. Obwohl sämtliche Gerätedaten weitergegeben wurden und Werbebetrug durchgeführt wird, bewertete Google Play Protect diese App nicht als schädlich.

7.4. Fakebank

Die benötigten Berechtigungen von Fakebank umfassen die Möglichkeit zum Auslesen der Geräteinformationen und des Netzwerkstatus, das Installieren und Löschen von anderen Applikationen sowie das Senden und Empfangen von SMS. Die Lokalisierungsdateien beinhalten nur sehr wenige Strings in koreanischer Sprache.

Im Code findet sich die Implementation eines SMS-Receivers, der eingehende Nachrichten auswertet. Anschließend wird der Inhalt, Sender- und Empfängertelefonnummer sowie sämtliche Hardware-Identifikatoren an die URLs kkk.kakatt.net und banking1.kakatt.net gesendet. Beide URLs sind mittlerweile nicht mehr zu erreichen. Der Programmcode des SMS-Receivers enthält auch eine E-Mailadresse, registriert bei einem chinesischen Anbieter, die jedoch nicht verwendet wird. Nach dem Start der App wird der SMS-Receiver als Hintergrundservice gestartet, während Nutzerinnen und Nutzern ein Loginfenster der Bank angezeigt wird. Sowohl bei Eingabe des Benutzernamens und Passworts als auch beim Empfangen von SMS-Nachrichten wird versucht, diese an die oben genannten Server zu schicken. Die Netzwerkanfragen führen jedoch zum Absturz der Applikation, da diese vom Hauptthread aus gestartet werden und Android 9.0 dies blockiert [20]. Google Play Protect erkannte die App als Schadsoftware und blendete einen Warnhinweis ein.

7.5. Reddrop

Reddrop benötigt den Zugriff auf das Dateisystem, SMS und MMS, den genauen Standort sowie die Möglichkeit, andere Applikation neu zu starten, Systemeinstellungen zu verändern, Nutzeraccounts auszulesen und Bildschirmsperren auszuschalten. In den beigepackten Grafik-Ressourcen finden sich mehrere native Shared Library-Dateien, eine Java-Bibliothek sowie ein weiteres Android-Package. Wie schon bei Fakebank wertet ein SMS-Receiver Nachrichten aus und schickt diese Informationen zusammen mit den Geräteinformationen an eine hinterlegte IP-Adresse. Der Code beinhaltet auch Funktionen zum Senden von SMS-Nachrichten sowie zum Download von weiteren Java-Archiven und Android-Packages. Das in den Grafik-Ressourcen versteckte Android-Paket ermittelt Geräteinformationen mittels Mustererkennung in den Android Konfigurationsdateien, besitzt die Möglichkeit Pay-SMS zu versenden und diese anschließend wieder aus den Nachrichten zu löschen. Beim Ausführen der App auf Android 7.1.1 wurde unmittelbar nach dem Start ein Command-and-Control Server mit den Geräteinformationen kontaktiert, von diesem eine Konfiguration mit weiteren Server-Adressen empfangen und von diesen Servern weitere Java-Archive heruntergeladen und, wie aus den Systemlogs ersichtlich, installiert. Sensible Daten wie Telefonnummer, *International Mobile Subscriber Identity* (IMSI), IMEI, SIM-Kartennummer, Gerätemodell und verbundene Wifi Access Points werden auch mit Payment-Providern geteilt. Jede Interaktion mit der Applikation führte zum versuchten Versand einer Pay-SMS. Google Play Protect meldete diese App nicht als schädlich, keine Maßnahmen wurden vorgeschlagen.

7.6. Exodus

Die Berechtigungen, die von Exodus verlangt werden, umfassen das Feststellen des Standorts, Zugriff auf das Dateisystem, Mikrofon, Kamera und Lesezugriff auf SMS-Nachrichten, Kontakte und Anrufhistorie. Die String-Dateien beinhalten zwei Zertifikate als Base64-codierte Strings. Der Code von Exodus beginnt mit einer Überprüfung, ob dieser auf einem Emulator ausgeführt wird. Falls dies der Fall ist, werden Teile der App deaktiviert, wie etwa der Download eines weiteren Android-Paketes, das anschließend ausführbar gemacht wird und als Service gestartet wird. Des Weiteren wird das Archiv mike.jar gestartet, das im Stande ist, weitere Daten wie etwa installierte Applikationen, Kalendereinträge, Browserhistorien, den Mailverkehr und Nachrichten beliebter Messenger auszulesen. Nach Abschalten der Emulator-Überprüfung kontaktierte die Applikation in regelmäßigen Abständen den Command-And-Control-Server, der jedoch nicht mehr erreichbar ist. In diesen Anfragen befanden sich Hardware-Identifier wie die Seriennummer, Telefonnummer und SIM-Anbieter. Exodus wurde von Play Protect nicht als Schadsoftware klassifiziert.

7.7. Schutzmaßnahmen und deren Wirksamkeit

Jede App wird auf Android in einer isolierten Umgebung (Sandbox) installiert, der Zugriff auf Daten oder Geräte außerhalb ist nur möglich, indem entsprechende Berechtigungen erteilt werden.

Normale Berechtigungen wie etwa Internetzugang werden bei der Installation automatisch gewährt, gefährliche Berechtigungen, zum Beispiel der Zugriff auf das Mikrofon müssen seit Android 6.0 zur Laufzeit von der Benutzerin oder dem Benutzer erteilt und widerrufen werden können. Alle untersuchten Beispiele umgingen diese Einschränkung, indem der Parameter *targetSdkVersion* auf eine ältere Version gesetzt wurde. Dadurch werden auch gefährliche Berechtigungen bei der Installation erteilt und somit die Einschränkungen der Sandbox umgangen. Beim Einreichen einer App oder eines Updates in den Play Store wird dieser Parameter geprüft und muss eine aktuelle Android-Version beinhalten, andernfalls erfolgt die Ablehnung der App oder des Updates. Bei bereits bestehenden Applikation oder dem Bezug aus Drittanbieterquellen erfolgt diese Prüfung nicht. Mithilfe der erlangten Berechtigungen konnten Squaremaker und Reddrop weitere Android-Pakete und Java-Archive laden. Da bei Exodus der Host nicht erreichbar war, wurde kein weiterer Code nachgeladen.

Sämtliche Apps nutzen Hardwaredaten wie Geräteseriennummer, IMEI, IMSI oder MAC-Adressen der Netzwerkgeräte, um Smartphones eindeutig zu identifizieren. Dabei wurden nur die physischen MAC-Adressen von WLAN und Bluetooth-Schnittstellen durch die Adresse „02:00:00:00:00:00“ ersetzt [21], alle anderen Werte waren unverändert verfügbar.

Die auf Android-zertifizierten Smartphones vorinstallierte Schutzsoftware Google Play Protect erkannte die untersuchten Anwendungen mit Ausnahme von Fakebank nicht als bösartig. Obwohl Play Protect „Informationen zu den Netzwerkverbindungen des Geräts, zu potenziell schädlichen URLs, zum Betriebssystem und zu Apps, die über Google Play oder andere Quellen auf Ihrem Gerät installiert wurden“ [10] auswertet, lösten sensible Daten in Netzwerkanfragen keine Warnung aus. Im Falle von Fakebank wurde kurz nach dem Start der App ein Hinweis eingeblendet und die weitere Ausführung verhindert. In Tabelle 1 sind die beschriebenen Ergebnisse zusammengefasst.

App	Platform	Android Version	Zugriff auf sensible Daten möglich	Wurde Code nachgeladen	Play Protect Erkennung
Spotify Crack	arm	7.1.1	Ja (Berechtigung)	Nein	Nein
iCoyote	x86	9.0	Ja (Berechtigung)	Nein	Nein
Squaremaker	arm	7.1.1	Ja (Berechtigung)	Ja	Nein
Fakebank	x86	9.0	Ja (Berechtigung)	Nein	Warnhinweis
Reddrop	arm	7.1.1	Ja (Berechtigung)	Ja	Nein
Exodus	x86	9.0	Ja (Berechtigung und Rooting)	Nein (Host nicht erreichbar)	Nein

Tabelle 1: Ergebnisse der Untersuchung

8. Conclusio

Trotz Fortschritten in der Verwaltung von Berechtigungen und der Verfügbarkeit von zusätzlichen Schutzmechanismen wie Play Protect ist es Malware noch immer mit überschaubarem Aufwand möglich, Zugriff auf sensible Daten zu erhalten. Sämtliche untersuchten Android-Applikationen konnten mithilfe von Installationsberechtigungen Geräteinformationen auslesen und diese zur eindeutigen Identifikation nutzen. In zwei von drei Fällen konnte weiterer Schadcode ohne Wissen der Benutzerin oder des Benutzers nachinstalliert werden, der dritte Fall scheiterte nur am nicht verfügbaren Server. Nur in einem von sechs Fällen erkannte Google Play Protect die Applikation als bösartig, verhinderte die weitere Ausführung der App und benachrichtigte die Benutzerin oder den Benutzer.

Da vorinstallierte technische Maßnahmen nur bedingt wirksam scheinen, bleibt der bestmögliche Schutz vor Malware die aufmerksame Prüfung der erforderlichen Berechtigungen und der Bezug von Applikationen aus offiziellen Quellen. Auch die Maßnahmen des iOS Betriebssystems unterbinden die Ausführung von Schadsoftware nicht vollständig, erschweren die Verbreitung durch verpflichtende Reviews und die zentrale Zertifikationsstelle allerdings erheblich.

Referenzen

[1] Malwarebytes, „Malwarebytes," [Online]. Available: https://www.malwarebytes.com/malware.

[Zugriff am 15 03 2019].

[2] Google Ltd., „Google Play Protect," [Online]. Available:

https://www.android.com/intl/de_de/play-protect/. [Zugriff am 14 Mai 2019].

[3] Apple Inc., „Apple Review Guidelines," [Online]. Available: https://developer.apple.com/app store/review/guidelines/. [Zugriff am 12 06 2019].

[4] Apple Inc., „iOS Security," 2019.

[5] N. Elenkov, Android Security Internals: An In-Depth Guide to Android's Security Architecture, San Francisco: No Starch Press, 2014.

[6] J. V. S. C. B. N. K. René Mayrhofer, „The Android Platform Security Model," arXiv preprint arXiv:1904.05572, 2019.

[7] Google Ltd., „Project Capillary," [Online]. Available: https://github.com/google/capillary.

[Zugriff am 12 06 2019].

[8] Android Developer Resources, „Datenschutz, Sicherheit und Täuschung," [Online]. Available:

https://play.google.com/about/privacy-security-deception/permissions/. [Zugriff am 18 Juni 2019].

[9] Apple Inc., „Apple Developer Documentation," [Online]. Available:

https://developer.apple.com/documentation/usernotifications/unnotificationserviceextension.

[Zugriff am 12 06 2019].

[10] Google Ltd, „Wie funktioniert Google Play Protect?," [Online]. Available:

https://support.google.com/googleplay/answer/2812853?hl=de. [Zugriff am 14 Mai 2019].

[11] Android Open Source Project, „aapt2," [Online]. Available:

https://developer.android.com/studio/command-line/aapt2. [Zugriff am 10 Mai 2018].

[12] T. Strazzere, „AXMLPrinter," [Online]. Available: https://github.com/rednaga/axmlprinter.

[Zugriff am 10 Mai 2019].

[13] Android Developer Resources, „Dangerous Permissions," [Online]. Available:

https://developer.android.com/guide/topics/permissions/overview#dangerous_permissions.

[Zugriff am 13 Mai 2019].

[14] Android Open Source Project, „Dalvik Executable Format," [Online]. Available:

https://source.android.com/devices/tech/dalvik/dex-format. [Zugriff am 14 Mai 2019].

[15] National Security Agency, „Ghidra," [Online]. Available: https://ghidra-sre.org/. [Zugriff am 14

[16] Android Open Source Project, [Online]. Available:

https://developer.android.com/studio/command-line/adb. [Zugriff am 14 Mai 2019].

[17] Wireshark Foundation, „Wireshark," [Online]. Available: https://www.wireshark.org/. [Zugriff am 14 Mai 2019].

[18] Portswigger Ltd., „Burp Suite,“ [Online]. Available: https://portswigger.net/burp. [Zugriff am 14

[19] Android Developer Resources, „Shared Preferences,“ [Online]. Available:

https://developer.android.com/reference/android/content/SharedPreferences. [Zugriff am 14

[20] Android Open Source Project, „Android Developer Resources,“ [Online]. Available:

https://developer.android.com/reference/android/os/NetworkOnMainThreadException. [Zugriff am 14 Mai 2019].

[21] Android Open Source Project, „Identifiers in Android 8.0 and higher,“ [Online]. Available:

https://developer.android.com/training/articles/user-data-ids#identifiers-android-8. [Zugriff am 14 Mai 2019].