A-SIT Technology

E-Government Innovationszentrum

2. April 2024 Analyse · Mobile Sicherheit · 17 Min. Lesezeit

Einfluss von Änderungen an der Laufzeitumgebung auf Android Applikationen

Gerald Palfinger



Autorinnen und Autoren	Gerald Palfinger
Stand	2024-03-18
Quelle	https://technology2.a-sit.at/veroeffentlichungen/einfluss-von-aenderungen-an-der-laufzeitumgebung-auf-android-applikationen/

Kernaussagen

- Von 683 als fingerprintbar erkannten Methoden und Feldern wurden für 617 (90,5 %) Patches erstellt.
- Neun Methoden und Felder mussten auf die Sperrliste, etwa uiMode (Flackern der Oberfläche) und getDisplays() (Absturz mit ArrayIndexOutOfBoundsException).
- Mit dem vollständigen Patch-Paket blieben 97 der 120 getesteten populären Android-Apps lauffähig.
- Schon mit leerem Patch-Paket (nur Neusignierung) stürzten 7 Apps ab; 3 weitere Apps liefen bereits unverändert nicht.

- Patches an UI-Methoden führten zu Darstellungsunterschieden, etwa andere Schaltflächenfarben, die Elemente blieben aber voll funktionsfähig.

Die Programmierschnittstelle von Android bietet eine Vielzahl an Funktionen und Informationsquellen, um Applikationen zu erstellen. Diese Informationsquellen ermöglichen es jedoch auch, einen Fingerabdruck eines Geräts zu erstellen. Doch nicht jede Applikation benötigt Zugriff auf alle bereitgestellten Informationen. Ein Ansatz, um die Erstellung eines Fingerabdrucks zu erschweren besteht deshalb darin, gewisse Informationsquellen durch Änderungen an der Laufzeitumgebung zu deaktivieren beziehungsweise deren Rückgabewerte über möglichst viele Geräte zu vereinheitlichen. In diesem Bericht wird der Einfluss solcher Änderungen an der Laufzeitumgebung auf Applikationen unter Android untersucht. Dazu werden Tests mit Hilfe eines Tools ausgeführt, welches automatisiert Benutzereingaben auslöst, während die zu testenden Applikationen laufen. Mit Hilfe dieses Ansatzes werden die Änderungen an der Laufzeitumgebung mit den 120 populärsten Anwendungen aus dem Google Play Store untersucht. In den Untersuchungen zeigte sich, dass ein Großteil der getesteten Applikationen weiterhin lauffähig ist. Durch Änderungen an Methoden, die für die Benutzeroberfläche zuständig sind, kann es jedoch zu kleinen Darstellungsunterschieden kommen.

1. Einleitung

Mobile Anwendungen stellen eine breite Palette von Diensten und Funktionen zur Verfügung, oft ohne (direkte) Kosten für die Nutzenden. Um mit diesen kostenlosen Anwendungen Einnahmen zu erzielen, sind die Entwickler häufig auf personalisierte Werbung angewiesen [1]. Um zielgerichtete Werbung zu liefern, sammeln mobile Anwendungen persönliche Informationen von den Geräten, auf denen sie laufen [2]. So können sie ihre Nutzerinnen und Nutzer über verschiedene Anwendungen hinweg verfolgen. Diese Praxis ermöglicht es den Anwendungsentwicklern zwar, ihre Dienste kostenlos anzubieten, beeinträchtigt aber die Privatsphäre der Anwenderinnen und Anwender erheblich. Um dem entgegenzuwirken und die Privatsphäre der Nutzerinnen und Nutzer zu schützen, wurden im Laufe der Jahre mehrere Schutzmechanismen in Android integriert. So wurde beispielsweise mit Android 6 das Berechtigungssystem überarbeitet, um den Benutzerinnen und Benutzern die Kontrolle über den Zugriff auf Datenquellen zu geben, die als datenschutzrechtlich sensibel gelten, wie z. B. Kontakte oder Fotos. Diese so genannten gefährlichen Berechtigungen erfordern die ausdrückliche Zustimmung der Benutzerin bzw. des Benutzers, bevor eine Anwendung auf die durch sie geschützten Daten zugreifen kann. Im Laufe der Jahre sind weitere Berechtigungen hinzugekommen, und einige bestehende wurden auf die Stufe “gefährlich” verschoben. Darüber hinaus wurde mit Android 10 der Zugriff auf nicht rücksetzbare Benutzerkennungen für Anwendungen von Drittanbietern eingeschränkt [3]. Umgekehrt werden Anwendungen angehalten, eine rücksetzbare Kennung wie die Werbekennung von Google zu verwenden. Trotz dieser Versuche der Betriebssystemhersteller, den Zugang zu personenbezogenen Daten einzuschränken, sammeln Anwendungen nach wie vor Daten über die Geräte, auf denen sie ausgeführt werden, bzw. manche beginnen sogar, diese zu sammeln [4]. Insbesondere können Anwendungen immer noch benutzer- und gerätespezifische Informationen kombinieren, um einen Fingerabdruck zu erstellen, der als Ersatz für eine eindeutige Kennung verwendet werden kann [5].

Um die Fingerprintbarkeit von Android zu verringern, wurde ein Tool entwickelt, welches automatisiert Patches für die Programmierschnittstelle des Betriebssystems erstellen kann. Diese Patches ändern die Rückgabewerte von Methoden, den Inhalt von Feldern und Content Providern, welche fingerprintbare Information enthalten. Diese Patches werden als Patch-Paket zusammengefasst und können so auf einzelnen Anwendungen angewendet werden. In diesem Bericht wird ein solches Patch-Paket erstellt und danach auf verschiedenen populären Anwendungen angewandt. Dabei wird untersucht, wie sich die Änderungen an der Programmierschnittstelle auf die Anwendungen auswirken. Weiters wird untersucht, wie viele der gefundenen fingerprintbaren Informationsquellen automatisch gepatcht werden konnten und welche Informationsquellen aufgrund von Inkompatibilitäten mit Applikationen vom Patching ausgeschlossen werden mussten.

2. Hintergrund

In diesem Abschnitt werden wichtige Hintergrundinformationen zu den wichtigsten Ansätzen für das Patching von Applikationen gegeben. Anschließend werden verwandte Arbeiten zum Schutz vor dem Fingerprinting von Browsern vorgestellt.

2.1. Applikations-Patching

Um die Fingerprintbarkeit der Programmierschnittstelle von Android durch Änderung der bereitgestellten Informationen zu verringern, muss die Laufzeitumgebung oder das Anwendungspaket abgeändert werden. Insbesondere muss es möglich sein, die Methoden, Felder und Inhaltsanbieter zu ändern, die über die API zugänglich sind. Es gibt verschiedene Ansätze, um dieses Ziel zu erreichen. Einige von ihnen beruhen auf der Durchführung von Änderungen am System, entweder durch Erlangung von Root-Rechten oder durch Bereitstellung eines geänderten Systemabbilds. Es gibt mehrere Lösungen für die Anwendung von Patches auf das System, wie LSPosed [1] oder Magisk-Module [2] . Die Änderung des Systems betrifft jedoch das gesamte Gerät, nicht nur einzelne Anwendungen, und erhöht damit die Anfälligkeit des Geräts für Angriffe. Alternativ können Anwendungen innerhalb einer Container-Anwendung, wie VirtualApp [3], ausgeführt werden. Dieses Konzept ermöglicht es der Container-Anwendung, Änderungen an der Anwendung vorzunehmen, ohne dass Änderungen am System erforderlich sind. Bei diesem Ansatz 1 https://github.com/LSPosed/LSPosed 2 https://github.com/topjohnwu/Magisk 3 https://github.com/asLody/VirtualApp muss die Container-Anwendung jedoch alle Berechtigungen, die eine Anwendung möglicherweise benötigt, im Voraus anfordern. Der Container befindet sich im selben Prozess wie die Anwendung, so dass er im Allgemeinen nicht in der Lage ist, die Anwendung daran zu hindern, eine vorab angeforderte Berechtigung zu verwenden. Außerdem werden bei diesem Ansatz einige Betriebssystemintegrationen nicht mehr funktionieren und die vom System gemeldete Signatur des Anwendungspakets wird verändert.

Schließlich können Änderungen direkt am Anwendungspaket vorgenommen werden. Bei diesem Ansatz können der ausführbare Programmcode und die im Paket enthaltenen Ressourcen geändert werden. Um das geänderte Paket auf einem Gerät zu installieren, muss es neu gepackt und neu signiert werden, wodurch auch alle Signaturüberprüfungen nicht mehr gültig sind, ähnlich wie bei der Ausführung innerhalb einer Container-Anwendung. Allen drei Ansätzen ist gemeinsam, dass sie in der Regel mit Hilfe einer Remote-Attestierung erkannt werden können, indem entweder die Signatur der Anwendung oder der Zustand des Bootloaders überprüft wird [6]. Wir haben die A2P2-Patching-Pipeline [7] gewählt, um das Anwendungspaket direkt zu patchen. Dieser Ansatz ermöglicht es uns, anwendungsunabhängige Patches

zu erstellen, die auf einzelne Anwendungspakete angewendet werden können. Da sie auf das Anwendungspaket angewendet werden, ist keine Änderung des Systems erforderlich, wodurch unerwünschte Nebeneffekte auf das System und andere Anwendungen vermieden werden. Darüber hinaus können die Patches bei Bedarf für einzelne Anwendungspakete abgeändert oder deaktiviert werden.

2.2. Verwandte Arbeiten

In der Literatur gibt es nur wenige Arbeiten zum Schutz mobiler Betriebssysteme vor Fingerprinting. Im Zusammenhang mit Webbrowsern wurden jedoch mehrere Ansätze zum Schutz vor Fingerprinting vorgeschlagen. FPGuard [8] überwacht zum Beispiel die API-Aufrufe von Websites und verwendet Heuristiken, um Fingerprinting-Versuche zu erkennen. Wenn die Browsererweiterung Fingerprinting-Aktivitäten feststellt, ändert sie die an die Website zurückgegebenen Informationen, um den Fingerprinting-Versuch zu vereiteln. PriVaricator [9] verwendet verschiedene Zufallsverfahren, um die von Websites abrufbaren Informationen zu ändern. In ähnlicher Weise verändert auch die FP-Block-Erweiterung die Daten, die von Fingerprinting-fähigen Informationsquellen bereitgestellt werden, nach dem Zufallsprinzip. Der Hauptunterschied besteht darin, dass die Informationen nur über verschiedene Domänen hinweg randomisiert werden, um Probleme zu reduzieren. In [10] wurde eine Browsererweiterung namens JShelter vorgestellt. Die Erweiterung zielt darauf ab, die Fingerprintbarkeit zu verringern, indem die JavaScript-Programmierschnittstelle der Browser geändert wird. Insbesondere schränkt die Erweiterung bestimmte Methoden und Eigenschaften ein oder ändert sie und ist in der Lage, bestimmte Sensoren zu simulieren. Diese Methoden und Eigenschaften ermöglichen es normalerweise, Informationen über das verwendete Gerät, die Browserinstanz oder die Umgebung abzurufen. In ähnlicher Weise haben sowohl der Tor-Browser [11] als auch Firefox [12] Maßnahmen ergriffen, um dem Fingerprinting entgegenzuwirken. Zu diesen Maßnahmen gehören die Änderung der von bestimmten Informationsquellen gemeldeten Werte und die Deaktivierung bestimmter Funktionen.

3. Erstellte Patches

3.1. Methoden und Felder

Eine zusammenfassende Darstellung der gefundenen fingerprintbaren Methoden und Felder sowie der dafür erzeugten Patches ist in Tabelle 1 zu finden. Insgesamt wurden 598 Methoden und 85 Felder gefunden, die fingerprintbare Informationen liefern. Davon konnten 33 Methoden und 9 Felder nicht gepatcht werden, weil die entsprechende Klasse nicht Teil der öffentlichen Android-API war. Außerdem konnten 5 Methoden nicht gepatcht werden, weil der Rückgabetyp verschleiert ist. Für alle Methoden, die in ihrer Signatur Exceptions deklarieren, konnte die entsprechende Klasse aufgelöst werden. Allerdings konnten 15 Methoden nicht aufgerufen und 4 Felder auf dem Referenzgerät nicht abgerufen werden. Daher wurden für insgesamt 544 Methoden und 72 Felder automatisch Patches erstellt. Darüber hinaus konnten für 10 der 16 Methoden manuell Referenzwerte definiert werden, die auf von anderen Geräten erhaltenen Werten basierten. 8 der Methoden und 1 Feld mussten in die Sperrliste aufgenommen werden, weil sie sich negativ auf bestimmte Anwendungen auswirken. Insgesamt wurden also 546 Methoden und 71 Felder gepatcht.

Von den 546 gepatchten Methoden sind 446 Instanzmethoden aus 117 verschiedenen Klassen. Davon sind 379 Methoden Teil der öffentlichen API und konnten daher direkt über die Patchmethoden von A2P2 gepatcht werden. Für die restlichen 67 versteckten Methoden musste ein Patch für die Reflection-API erstellt werden, um Aufrufe auf die jeweilige gepatchte Version der Methode umzuleiten. Zusätzlich wurden Patches für 108 statische Methoden in 61 Klassen erstellt. 77 der statischen Methoden sind öffentlich zugänglich, während die restlichen 31 Methoden versteckt sind. Insgesamt wurden Patches für 448 öffentliche und 98 versteckte Methoden erstellt. Von den insgesamt 71 gepatchten Feldern sind 43 nicht-statische Felder aus 13 Klassen. Die anderen 28 Felder sind statisch und werden in 6 verschiedenen Klassen geändert. Da die Felder entweder beim Start der Anwendung oder bei der Objekterstellung mit Hilfe der Reflection-API festgelegt werden, wird nicht zwischen öffentlichen und verborgenen Feldern unterschieden.

Tabelle 1 Übersicht über die Methoden und Felder, die als fingerprintbar erkannt wurden, sowie die dafür erstellten Patches.

Col1	METHODEN	FELDER	TOTAL
# Erkannte fingerprintbare Methoden und Felder	598	85	683
- Dazugehörige Klasse konnte auf dem Referenzgerät nicht gefunden werden	33	9	42
- (Rückgabe-) Typ existiert nicht	5	0	5
- Ausnahmentyp existiert nicht	0	0	0
- Kein Referenzwert vorhanden	15	4	19
= Automatisch erstellte Patches	544	72	616
+ Manuell definierte Referenzwerte	10	0	10
- Methoden, die zur Sperrliste hinzugefügt werden mussten	8	1	9
= Erstellte Patches	546	71	617
	91,5%	83,5%	90,5%

3.2. Sperrliste

Einige Methoden und Felder mussten auf die Sperrliste gesetzt werden. Konkret wurden acht Methoden und ein Feld auf die Sperrliste gesetzt, da sie in einer Reihe von Anwendungen Probleme verursachten. So führte die Änderung des Wertes des nichtstatischen Feldes uiMode in der Klasse android.content.res.Configuration zu einem starken Flackern der Benutzeroberfläche, wodurch die betroffenen Anwendungen unbenutzbar wurden. Auf Smartphones lässt sich mit diesem Feld feststellen, ob der Nachtmodus eingeschaltet ist. Darüber hinaus kann es verwendet werden, um zu erkennen, ob die Anwendung beispielsweise auf einem Fernseher oder VRR-Headset läuft. Zusätzlich wurden die Methoden getAllNetworks() und getActiveNetwork() des ConnectivityManager in die Sperrliste aufgenommen. Diese Methoden ermöglichen es, alle verfügbaren Netzwerke bzw. das aktive Netzwerk abzufragen. Das Patchen dieser Methoden führte jedoch in einigen Anwendungen zu Konnektivitätsproblemen. Die Methode

getConfiguration() der Klasse android.content.res.Resources wurde auf die Sperrliste gesetzt, da das Entpacken eines alten Konfigurationsobjekts in einigen Anwendungen zu Problemen führte. Dennoch war unser Framework in der Lage, Patches für einige der Methoden und Felder innerhalb der Klasse Configuration zu erstellen. Außerdem wurde die Methode getText() der Klasse android.widget.TextView auf die Sperrliste gesetzt, da sie dazu führte, dass Anwendungen falschen Text aus Text-Views abriefen. In der Klasse android.hardware.display.DisplayManager wurde die Methode getDisplays() in die Sperrliste aufgenommen, da der Rückgabewert nicht korrekt deserialisiert werden konnte, was dazu führte, dass ein leeres Array zurückgegeben wurde. Dieses Problem wurde während unserer Wirksamkeitstests festgestellt, da der Rückgabewert von dem des Referenzgeräts abwich. Außerdem verursachte es ein Problem in einer Anwendung, da diese die Rückgabe von mindestens einer Anzeige erwartete. Dies führte zu einem Absturz der Anwendung mit einer ArrayIndexOutOfBoundsException. Darüber hinaus wurden zwei Methoden in der android.view.WindowManager-Klasse zur Sperrliste hinzugefügt. Konkret wurden die Methoden getCurrentWindowMetrics() und getMaximumWindowMetrics() auf die Sperrliste gesetzt, da der Parameter windowInsets in den gesammelten Referenzwerten null ist. Bei den Kompatibilitätstests führte dies bei fünf Anwendungen zu einem Absturz mit einer NullPointerException. Schließlich haben wir auch getCallingUid() von android.os.Binder ausgeschlossen, da es bei bestimmten Sicherheitsüberprüfungen zu Problemen führen könnte.

3.3. Content Provider

Insgesamt wurden fünf Content Provider gefunden, die fingerprintbare Informationen liefern. Alle von ihnen wurden erfolgreich gepatcht. Daher enthält unsere Referenzdatenbank jeweils fünf Tabellen für jeden der entdeckten Content Provider. Insgesamt enthält die Referenzdatenbank 441 Einträge. Die mit Abstand meisten Fingerprinting-Einträge wurden bei den settings-Content Provider gefunden. Neben einigen Konstanten enthalten diese Provider hauptsächlich Systemeinstellungen, die vom Benutzer bzw. der Benutzerin des Geräts angepasst werden können. Der Settings-Provider ist in drei Unter-Provider unterteilt. Im Einzelnen sind dies settings/global, settings/secure und settings/system. Die Referenztabelle für den Content Provider settings/global enthält 221 Einträge. Sie enthält zum Beispiel den vom Benutzer bzw. der Benutzerin frei wählbaren Gerätenamen, die Bootanzahl und die Mobilfunkeinstellungen. Für den Provider settings/secure enthält die Referenztabelle 130 verschiedene Einträge. Dazu gehören unter anderem die Einstellungen für die Barrierefreiheit, die Eingabemethode und der Autofill-Dienst sowie die Einstellungen für den Sperrbildschirm. Der Anbieter settings/system enthält schließlich 47 Einträge. Dieser Anbieter enthält in erster Linie Informationen über Medien- und Benachrichtigungseinstellungen, darunter z. B. eingestellte Benachrichtigungstöne oder Vibrationseinstellungen. Der Anbieter media/internal/audio/media enthält Informationen über Klingel-, Benachrichtigungs- und Alarmtöne. Die von unserem Testgerät erhaltene Referenztabelle enthält 42 verschiedene Einträge. Neben verschiedenen Daten über die Sounds enthält diese Tabelle auch eine Spalte “date_modified”. Bei nicht gepatchten Geräten entspricht diese Spalte dem Zeitpunkt des ersten Bootvorgangs des Geräts für die vorinstallierten Sounds. Der Anbieter service-state enthält schließlich Informationen über das Mobilfunknetz, wie z. B. die Art des Datennetzes oder die Nummer des Netzbetreibers. Da in unserem Referenzgerät eine einzige SIM-Karte installiert war, enthält dieser Provider einen Eintrag.

4. Experimente

Da unser Ansatz die Laufzeitumgebung von Anwendungen verändert, bewerten wir die Kompatibilität, indem wir sie mit beliebten Anwendungen testen. Die Kompatibilitätstests wurden auf einem Google Pixel 7a mit Android 14 und Sicherheitspatches vom 5. Januar 2024 durchgeführt. Insbesondere wurden die Tests mit den sechs beliebtesten Anwendungen aus 21 verschiedenen Kategorien durchgeführt. Eine der Anwendungen war ausschließlich für Android TV-basierte Geräte verfügbar, wodurch sie mit unserem Testgerät nicht kompatibel war. Weitere fünf Anwendungen wurden in zwei Kategorien aufgeführt, so dass insgesamt 120 verschiedene Anwendungen für die Tests zur Verfügung standen. Um die Testumgebung vorzubereiten, wurde die Android Debug Bridge (adb) verwendet, um die gepatchte Anwendung automatisch zu installieren. Darüber hinaus wurde der UI/Application Exerciser Monkey [1] verwendet, um die Applikation zu starten und automatisch Ereignisse wie Berührungen oder Wischgesten in der laufenden Anwendung auszuführen. Die voreingestellte Distribution der Ereignisse wurde nicht verändert, da eine Abänderung wie in [13] gezeigt wurde zu keiner höheren Abdeckung führen würde. Eine Ausnahme bilden hierbei die Systemtasten-Ereignisse. Diese wurden deaktiviert, da darunter auch Tasten wie die Home- oder Anruftaste fallen, welche die zu testende Applikation entweder in den Hintergrund verschiebt oder die Telefonanwendung startet. Zusätzlich wurde eine Wartezeit von 200 Millisekunden zwischen den Ereignissen ausgewählt. Während diese Wartezeit zwar laut [13] die Abdeckung nicht erhöht, erleichtert diese Wartezeit die manuelle Sichtkontrolle der Applikation durch den Tester. Während der Application Monkey zwar Abstürze der untersuchten Applikation durch Überwachung des Systemprotokolls auf unbehandelte Exceptions erkennt, kann der Application Monkey Änderungen am Programmablauf oder Fehler, die während der Ausführung angezeigt werden, nicht automatisiert erkennen. Diese werden in den folgenden Untersuchungen durch eine manuelle Sichtprüfung der automatisch durchlaufenden Tests erkannt.

Die Anwendungen werden in drei verschiedenen Konfigurationen verglichen. Zunächst wurden die Anwendungen ohne jegliche Änderungen ausgeführt, um etwaige Inkompatibilitäten mit dem Testgerät zu ermitteln. Anschließend wurde die Patching-Pipeline von A2P2 verwendet, um die Anwendung neu zu signieren. Bei diesem Test wurden keine zusätzlichen Patches angewendet, d. h. es wird ein leeres Patch-Paket verwendet. Dies ermöglicht die Erkennung von Problemen, die auftreten können, wenn die Anwendungen mit einem anderen Signaturzertifikat neu gepackt werden. Schließlich wurde das komplette Patch-Paket angewendet. Um die Tests zwischen den drei verschiedenen Testfällen vergleichbar zu machen, wurde ein fixer Ausgangswert (Seed) für den Application Monkey gesetzt. Dadurch wird für jeden Testfall die gleiche Sequenz an Ereignissen an die Applikation gesendet.

Tabelle 2 Übersicht über die Ergebnisse der Applikationstests.

Col1	ANZAHL
Mögliche Gesamtanzahl (6 Applikationen * 21 Kategorien)	126
- Anzahl der Applikationen, die in mehreren Kategorien vorkommen	6
= Anzahl der zu testenden Applikationen	120
- Nicht patchbare Applikationen	2
- Ausführungsprobleme auch im unmodifizierten Zustand	3
= Potenziell lauffähige Applikationen	115
- Absturz mit Ausnahme (Exception)	7
- Andere Ausführungsprobleme (z.B. Fehlermeldung bei Applikationsstart)	8
= Verbleibende patchbare Anwendungen	100
- Absturz mit Ausnahme (Exception)	3
= LAUFFÄHIGE GEPATCHTE APPLIKATIONEN	97

Eine Übersicht über die Ergebnisse der Applikationstests ist in Tabelle 2 ersichtlich. Zwei der 120 Anwendungen konnten nicht gepatcht werden, da das Patching-Framework das gepatchte Anwendungspaket nicht neu signieren konnte. Die übrigen 118 wurden erfolgreich gepatcht und installiert. Allerdings traten bei drei Anwendungen bereits Probleme auf, wenn sie unverändert ausgeführt wurden. Zwei von ihnen schlossen sich auf dem Ladebildschirm, während bei einer Anwendung ein Dialogfenster mit dem Hinweis erschien, dass kritische Anwendungsdaten fehlten und die Anwendung neu installiert werden müsse.

Von den übrigen 115 Anwendungen stürzten insgesamt sieben mit einer Ausnahme ab, wenn sie mit einem leeren Patch-Paket gepatcht wurden. Vier von ihnen stürzten aufgrund eines IncompatibleClassChangeError ab. Obwohl einige Anwendungen verschleiert waren, stellte sich beim Vergleich der Stack Traces heraus, dass alle in der gleichen Zeile in der DrawScope-Klasse des Compose UI Frameworks von Android abstürzten. Die verbleibenden drei Anwendungen wurden aufgrund von fehlgeschlagenen Signaturüberprüfungen abgebrochen. Während der Tests wurden acht weitere Anwendungen vorzeitig abgebrochen. Zwei dieser Anwendungen zeigten eine Fehlermeldung an, die darauf hinwies, dass eine Datei beschädigt sei und die Anwendung neu installiert werden müsse. Die übrigen sechs Anwendungen wurden geschlossen, ohne dass eine Fehlermeldung angezeigt wurde oder eine Exception ausgelöst wurde.

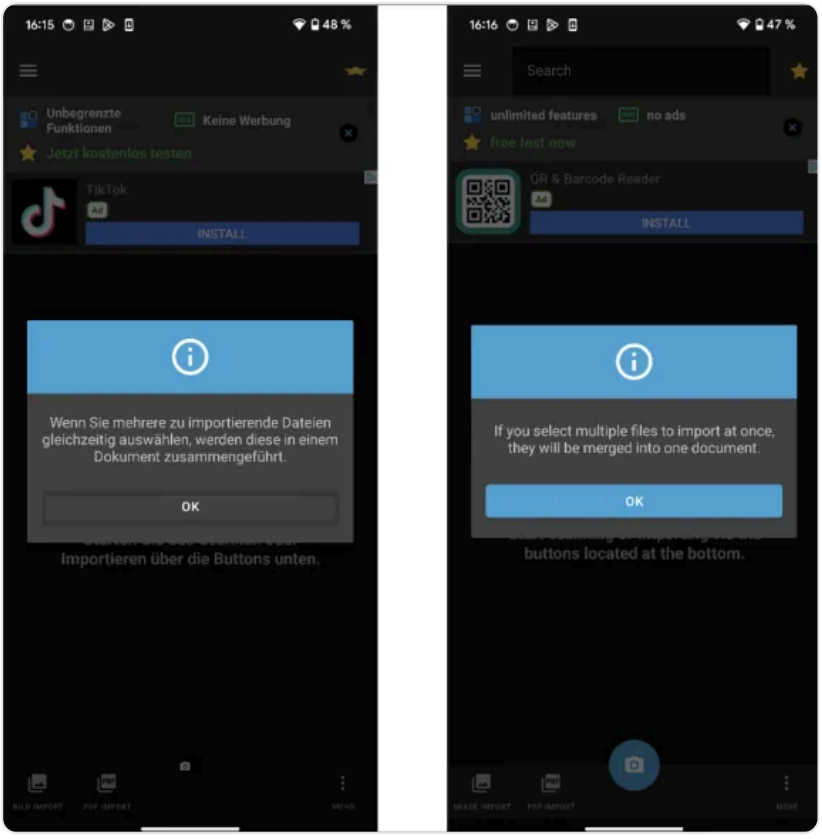


Abbildung 1: Grafische Unterschiede von Bedienelementen zwischen einer ungepatchten und gepatchten Applikation.

Von den verbleibenden 100 Anwendungen, die mit unserem Testgerät und A2P2 kompatibel sind, brachen drei während der Laufzeit ab, wenn sie mit dem vollständigen Patch-Paket gepatcht wurden. Insbesondere stürzte eine dieser Anwendungen im nativen Programmcode ab. Die beiden anderen Anwendungen waren nicht in der Lage, auf die im Anwendungspaket gespeicherten Ressourcen zuzugreifen. Außerdem traten bei zwei Anwendungen während der Tests Probleme mit der Netzwerkkonnektivität auf. Darüber hinaus wurden Patches für einige Benutzeroberflächenklassen in der API generiert, was zu visuellen Unterschieden bei einigen Elementen der Benutzeroberfläche im Vergleich zum leeren Patch-Paket führte. Beispielsweise hatten einige Schaltflächen eine andere Hintergrundfarbe, während Schalter eine andere Größe hatten. Trotz dieser Unterschiede blieben alle getesteten Elemente der Benutzeroberfläche voll funktionsfähig, wenn sie betätigt wurden. Ein beispielhafter Unterschied zwischen ungepatchter und gepatchter Applikation ist in Abbildung 1 ersichtlich.

5. Fazit

In diesem Bericht wurde der Einfluss von Änderungen an der Laufzeitumgebung auf Applikationen unter Android untersucht. Dazu wurden Tests mit Hilfe des UI/Application Exerciser Monkey ausgeführt. Dieser führte automatisiert Benutzereingaben auf den zu testenden Applikationen aus. Mit Hilfe des Monkeys wurden die Auswirkungen der Änderungen an der Laufzeitumgebung zur Reduzierung der Fingerprintbarkeit auf den 120 populärsten Anwendungen aus dem Google Play Store untersucht. In der Untersuchung zeigte sich, dass ein Großteil der getesteten Applikationen weiterhin lauffähig ist. Durch Änderungen an Methoden, die für die Benutzeroberfläche zuständig sind, kam es jedoch zu Darstellungsunterschieden bei verschiedenen Elementen der Benutzeroberfläche. Diese waren jedoch weiterhin benutzbar.

Referenzen

[1] Statista, „Most popular app monetization methods by publishers from the United States as of December 2023,“ Statista, 2023. [Online]. Available: https://www.statista.com/statistics/1119916/app-monetization-methods-united-states-apppublishers/. [Zugriff am 26 02 2024].

[2] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich und P. Gill, „Apps, Trackers, Privacy, and Regulators: A Global Study of the Mobile,“ in 25th Annual Network and Distributed System Security Symposium, NDSS, San Diego, California, USA, 2018.

[3] Android Developers, „Privacy Changes in Android 10 - Restriction on non-resettable device identifiers,“ Google Inc., 2019. [Online]. Available: https://developer.android.com/about/versions/10/privacy/changes#non-resettable-device-ids/. [Zugriff am 28 02 2024].

[4] K. Kollnig, A. Shuba, M. V. Kleek, R. Binns und N. Shadbolt, „Goodbye Tracking? Impact of iOS App Tracking Transparency and Privacy Labels,“ in FAccT ‘22: 2022 {ACM} Conference on Fairness, Accountability, and Transparency, Seoul, Republic of Korea, 2022.

[5] W. Wu, J. Wu, Y. Wang, Z. Ling und M. Yang, „Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers,“ IEEE Access, Bd. 4, pp. 8073-8083, 2016.

[6] B. Prünster, G. Palfinger und C. Kollmann, „Fides: Unleashing the Full Potential of Remote Attestationt,“ in Proceedings of the 16th International Joint Conference on e-Business and Telecommunications - SECRYPT, Prag, 2019.

[7] F. Draschbacher, „A2P2 - An Android Application Patching Pipeline Based On Generic Changesets,“ in International Conference on Availability, Reliability and Security, Benevento, Italy, 2023.

[8] A. FaizKhademi, M. Zulkernine und K. Weldemariam, „FPGuard: Detection and Prevention of Browser Fingerprinting,“ in Data and Applications Security and Privacy XXIX - 29th Annual IFIP, Fairfax, VA, USA, 2015.

[9] N. Nikiforakis, W. Joosen und B. Livshits, „PriVaricator: Deceiving Fingerprinters with Little White Lies,“ in Proceedings of the 24th International Conference on World Wide Web>, Florence, Italy, 2015.

[10] L. Polcák, M. Salon, G. Maone, R. Hranický und M. McMahon, „JShelter: Give Me My Browser Back,“ in Proceedings of the 20th International Conference on Security and Cryptography, SECRYPT, Rome, Italy, 2023.

[11] M. Perry, E. Clark, S. Murdoch und G. Koppen, „The Design and Implementation of the Tor Browser - Cross-Origin Fingerprinting Unlinkability,“ The Tor Project, Inc., 2019. [Online]. Available: https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkabilityp. [Zugriff am 28 02 2024].

[12] Mozilla Corporation, „Enhanced Tracking Protection in Firefox for desktop,“ Mozilla Corporation, 2023. [Online]. Available: https://support.mozilla.org/en-US/kb/enhanced-tracking-protectionfirefox-desktop/. [Zugriff am 27 02 2024].

[13] P. Patel, G. Srinivasan, S. Rahaman und I. Neamtiu, „On the effectiveness of random testing for Android: or how i learned to stop worrying and love the monkey,“ in Proceedings of the 13th International Workshop on Automation of Software, Gothenburg, Sweden, 2018.

---

1. https://developer.android.com/studio/test/other-testing-tools/monkey ↩