A-SIT

Zentrum für sichere Informationstechnologie - Austria

Implementierung und Evaluierung neuartiger Juice-Jacking-Angriffe

Implementierung und Evaluierung neuartiger Juice-Jacking-Angriffe

Autor:
Florian Draschbacher:
florian.draschbacher@a-sit.at

Datum: 22.04.2025

Abstract/Zusammenfassung:
Juice-Jacking bezeichnet eine Familie von Angriffen auf Mobilgeräte, in denen ein manipuliertes Ladekabel dazu genutzt wird, Daten zu extrahieren oder Malware zu installieren. Anfällig für diese Attacke sind Nutzer insbesondere überall dort, wo anstelle eines eigenen Ladegeräts bestehende Infrastruktur genutzt wird, etwa in Form von Ladestationen an Flughäfen. Nach der Integration von Nutzerbestätigungsdialogen bzw. vergleichbarer Interaktionsmodalitäten für USB-Verbindungen zur Datenübertragung wurden Juice-Jacking-Angriffe weitgehend als obsolet betrachtet und auch von der Wissenschaft nicht mehr weiterverfolgt.

Im Rahmen dieses Projekts wurde die Effektivität von Nutzer-Interaktion als Mittel gegen Juice-Jacking-Angriffe auf mobilen Geräten der populärsten Hersteller evaluiert. Dazu wurden Möglichkeiten gesucht und implementiert, die es einem Angreifer erlauben, die Interaktion zu umgehen und so auch auf aktuellen Geräten Daten zu extrahieren.

Inhalt

1. Einleitung 2
2. Hintergrund 3
2.1. USB 3
2.2. USB-Anschlüsse und -Versionen 3
2.3. USB Type-C 4
2.4. USB-Konnektivität von mobilen Geräten 5
3. Implementierungen von Juice-Jacking-Absicherungen 5
3.1. Geräte 5
3.2. Implementierungsvarianten 6
4. Prinzip und Sicherheit der Juice-Jacking-Mitigations 7
5. Angriffe 7
6. Evaluierung der Angriffe und Fehlerbehebung 9
6.1. Fehlerbehebung 9
7. Zusammenfassung 10
Referenzen 11

1. Einleitung

Das Konzept von Juice-Jacking-Angriffen sorgte rund um das Jahr 2011 für Furore. Sicherheitsforscher berichteten damals erstmals über die Möglichkeit, mittels bösartiger Ladegeräte Dateien von mobilen Endgeräten zu stehlen, oder mittels dieses Mechanismus unbemerkt Apps (z.B. Spyware) zu installieren. Diese Angriffe nutzten dabei die Tatsache aus, dass auf Mobilgeräten wie Smartphones oder Tablets dieselbe Schnittstelle sowohl zum Laden als auch zur Datenübertragung genutzt wird. Dennoch setzten die Betriebssysteme zunächst in Bezug auf die USB-Schnittstelle auf das Prinzip Trust-By-Default. Es wurde angenommen, dass allein durch die Notwendigkeit der durch den Nutzer durchgeführten physischen Kopplung implizit sichergestellt werden kann, dass auch der softwareseitige Verbindungsaufbau im Sinne des Nutzers ist. Es handelte sich bei dieser Annahme allerdings um einen Fehler im Trust Model: Der Nutzer kann nur aufgrund der physischen Erscheinung des Kabels oder angeschlossenen Zubehörs nicht wissen, welche Funktion es erfüllt. So war es möglich, einen Computer äußerlich als harmloses Ladegerät erscheinen zu lassen, das bei erfolgter Verbindung allerdings dann eine Datenverbindung herstellen konnte. Über die Datenverbindung konnte das Ladegerät dann etwa Dateien stehlen oder unbemerkt Apps installieren.

Schon kurz nach dem Auftauchen dieser Juice-Jacking-Angriffe besserten sowohl Google als auch Apple ihre mobilen Betriebssysteme Android und iOS nach. Konkret wurde die Notwendigkeit für Nutzerinteraktion eingeführt, bevor ein Computer eine USB-Datenverbindung zum Mobilgerät herstellen kann. Wie genau diese Nutzerinteraktion ausfallen muss, wurde je nach Hersteller unterschiedlich umgesetzt. Überall allerdings soll die Maßnahme sicherstellen, dass der Nutzer über den folgenden Datenaustausch informiert ist bzw. ihn auch selbst angestoßen hat.

Im Unterschied zu Verbindungen zwischen einem Computer (als USB-Host) und Mobilgerät (als USB-Device) wurde für Verbindungen zwischen einem Mobilgerät (als USB-Host) und einem Peripheriegerät (als USB-Device) allerdings keine Änderung vorgenommen. So ist es auch bei aktuellen Smartphones und Tablets noch immer möglich, USB-Eingabegeräte wie Tastaturen oder Mäuse, bzw. USB-Ausgabegeräte wie Kopfhörer oder Monitore ohne zusätzliche Nutzerinteraktion direkt nach der physischen Kopplung in Betrieb zu nehmen. In den letzten Jahren konzentrierten sich Angriffe auf Basis von bösartigen Ladegeräten daher darauf, als USB-Peripheriegerät Daten zu stehlen. So simulierten diese Angriffe etwa einen USB-HDMI-Adapter, um mittels Optical Character Recognition (OCR) Text aus dem Bildschirminhalt zu extrahieren [1]. Weitere Angriffe agierten verdeckt als USB-Audio-Interface, um mittels Audio-Eingabe Befehle an den Voice Assistant am Gerät zu erteilen und mittels Audio-Ausgabe und Spracherkennung Daten zu extrahieren [2]. Allen diesen Ansätzen ist jedoch gemein, dass ihre Fähigkeiten bei weitem nicht an jene der früheren Juice-Jacking-Angriffe heranreichen. Es hatte sich also die Meinung durchgesetzt, dass die eingeführten Gegenmaßnahmen unbemerkte Datenverbindungen zwischen einem im Ladegerät versteckten Computer und einem Mobilgerät effektiv verhindern.

Im Rahmen dieses Projekts wurde diese Einschätzung systematisch einer Prüfung unterzogen. Dazu analysieren wir zunächst, wie mobile Betriebssystem und Betriebssystem-Varianten verschiedener Hersteller die Nutzerinteraktion zur Aktivierung von USB-Dateifreigabe umsetzen. Danach gehen wir den theoretischen Grundlagen nach, auf deren Basis die Sicherheit dieser Maßnahme fußt. Schließlich zeigen wir, dass diese theoretischen Grundlagen nicht mit der praktischen Realität von modernen mobilen Betriebssystemen übereinstimmen. Wir präsentieren mehrere Angriffe, die es erlauben, die Nutzerinteraktion zur Aktivierung von USB-Dateifreigabe zu umgehen.

2. Hintergrund

In diesem Abschnitt sollen jene Technologien erklärt werden, die für das weitere Verständnis der späteren Ausführungen notwendig sind.

2.1. USB

Bei USB, dem Universal Serial Bus, handelt es sich um eine Schnittstelle, mit der Peripheriegeräte an einen Computer angeschlossen werden können. Bei dem Standard handelt es sich um den ersten erfolgreichen Versuch, die vielen proprietären Konnektoren, die bis dahin von verschiedenen Herstellern für die unterschiedlichsten Ein- und Ausgabegeräte genutzt wurden, zu vereinheitlichen. Seit der Einführung Mitte der 1990er-Jahre hat sich USB im Endnutzer-Bereich für nahezu jede Art von Peripheriegerät durchgesetzt. Es gibt USB-Mäuse und -Tastaturen, Festplatten, DVD-Laufwerke und andere Speichermedien, die den Anschluss ebenso nutzen wie Drucker, Kameras, und viele mehr. Die weite Verbreitung des USB-Standards und entsprechender Buchsen führte nach wenigen Jahren schon soweit, dass viele Geräte USB-Anschlüsse zur Stromversorgung verwenden, selbst wenn sie gar keine Ein- oder Ausgabegeräte sind. So gibt es mittlerweile Fahrradlichter oder Ventilatoren, die am USB-Anschluss geladen bzw. betrieben werden können.

Jede Kommunikation findet bei USB zwischen einem USB-Host und einem USB-Device statt. Der USB-Host ist hier das Gerät, das die Kontrolle über die Verbindung hat und im Wesentlichen Funktionalität oder Daten vom USB-Device anfordern kann. In umgekehrter Richtung fließen Daten nur dann, wenn dies vom USB-Host veranlasst wurde. Es kann also von einer gerichteten Kommunikation gesprochen werden.

Konzeptionell werden Daten bei der Nutzung von USB immer nur zwischen *einem* USB-Host und *einem* USB-Device übertragen (nicht also zwischen mehreren USB-Hosts oder mehreren USB-Devices). Es besteht allerdings die Möglichkeit, an einem USB-Host mehrere USB-Devices zu betreiben. Realisiert wird diese Bus-Topografie mittels USB-Hubs, die Kommunikations-Pakete vom Host an mehrere verschiedene Devices weiterleiten können. Man kann bei USB also von einer baumförmigen Bus-Architektur sprechen.

Wird ein USB-Device an einen Host angesteckt, so beginnt der Host die Kommunikation, indem er das Device nach dessen Deskriptoren befragt. Es handelt sich dabei um Datenstrukturen, die Aufschluss über die Eigenschaften des USB-Geräts geben. Übertragen werden hier zum Beispiel der Hersteller- und Produktname des Geräts, ebenso wie die zur Verfügung gestellte Funktionalität des Geräts in Form von Interface-Deskriptoren. Mittels dieser Deskriptoren ist es dem Host möglich, den Typ des angeschlossenen Geräts zu ermitteln und einen entsprechenden Gerätetreiber zu laden.

2.2. USB-Anschlüsse und -Versionen

Seit der ersten Veröffentlichung des USB-Standards Mitte der 1990er-Jahre [3] wurden mehrere Überarbeitungen publiziert. Im Wesentlichen unterschieden sich die Überarbeitungen von ihren jeweiligen Vorgängern durch die Möglichkeit zu höherer Datenübertragungsrate. Während weitgehende Rückwärtskompatibilität zu älteren USB-Standards besteht, wurden auch neue Funktionalitäten eingeführt, die Hardware-Änderungen benötigten.

In den ersten beiden Version 1.0 und 1.1 [4] sieht der USB-Standard Anschlüsse ausschließlich mit 4 Verbindungen vor. Dabei sind 2 Verbindungen (Pins) für die Stromversorgung des USB-Devices in

Verwendung (5V bzw. Masse/Ground), sowie ein differenzielles Leitungspaar der Datenübertragung gewidmet.

Es wurden verschiedene USB-Buchsen bzw. -Stecker spezifiziert, die die Rolle des jeweiligen Anschlusses in der USB-Verbindung festlegen. Hier wird zwischen den folgenden Anschlüssen unterschieden:

- USB-A: Es handelt sich hier um den gängigen USB-Anschluss, wie er etwa von USB-Sticks bekannt ist. USB-Host ist immer jenes Gerät, das über die USB-A-Buchse verfügt.
- USB-B: Bei USB-Devices, die kein fix verbautes Kabel (oder keinen fix verbauten USB-A-Stecker) haben, kommt eine USB-B-Buchse zum Einsatz. Geräte, die über eine USB-B-Buchse (nach USB 1.0 bzw. 1.1) verfügen, arbeiten stets in der Rolle als USB-Device (sind also niemals Host).

Eine wesentliche Änderung in Version 2.0 des USB-Standards [5] in Bezug auf Juice-Jacking-Angriffe war die Einführung von USB On-The-Go [6]. Es handelt sich dabei um die Möglichkeit, ein USB-fähiges Gerät sowohl als Device als auch als Host nutzen zu können. Umgesetzt wurde diese Möglichkeit, indem die neu eingeführten verkleinerten Varianten der USB-A- und USB-B-Verbindungen (Mini USB sowie Micro USB) mit einem zusätzlichen Pin ausgestattet wurden. Verschiedene Belegungen dieses Pins signalisieren die Rolle des verbundenen Geräts in der USB-Verbindung. OTG-fähige Geräte prüfen nach Herstellung der physischen Verbindung den Status dieses sogenannten ID- oder Sense-Pins, um ihren USB-Stack in den jeweils passenden Modus zu schalten. Die Regelung, welcher Kommunikationspartner die Rolle als Host bzw. Device übernimmt, wird hier also immer noch in Hardware fixiert. Es stehen verschiedene Kabel zur Verfügung, in denen die Beschaltung des ID-Pins am Micro-USB-B-Konnektor entweder den Anschluss von USB-Peripheriegeräten (wie Mäuse, Tastaturen, etc.) an eine USB-OTG-Buchse erlaubt, oder eine Verbindung des OTG-fähigen Gerätes an einen Computer.

Micro-USB-Anschlüsse (häufig auch OTG-fähig) fanden vor der Einführung von USB Type-C weite Verbreitung bei Smartphones und Tablets, insbesondere solchen, die mit Android betrieben wurden.

Version 3.0 des USB-Standards [7] führte zusätzliche Pins im USB-A-Konnektor sowie neue abwärtskompatible Micro- bzw. Mini-Anschlüsse ein. Die zusätzlichen Pins erlauben durch zwei parallele Datenpaare schnellere Übertragungsgeschwindigkeiten.

2.3. USB Type-C

Seit etwas mehr als fünf Jahren setzt sich unter Mobilgeräten immer mehr der USB Type-C Standard [8] durch. Es handelt sich um einen neuen USB-Konnektor, der einige Vorteile gegenüber Micro-USB (Typ B) hat:

- Umdrehbar
  Der Stecker kann in beiden Orientierungen in die Buchse gesteckt werden.
- Höhere Datenraten
  Durch höhere Anzahl an Pins können Datenraten von mehreren Gbit/s erreicht werden.
- Höherer Ladestrom
  Der Konnektor wurde für mindestens 15 Watt Leistung ausgelegt.

An dieser Stelle muss festgehalten werden, dass der USB Type-C-Anschluss weitgehend unabhängig von der USB-Spezifikation genutzt werden kann, die die Datenübertragung regelt. Konkret bedeutet das, dass Geräte mit USB Type-C teils nur USB 2.0-Geschwindigkeiten zur Datenübertragung unterstützen. Für den neuesten USB 4.0-Standard wird allerdings verpflichtend die Nutzung des Type-C-Anschlusses

vorgeschrieben. Damit wird die Unterscheidung zwischen Host und Device auf Anschluss-Ebene, die bis USB 3.0 noch immer in Teilen vorhanden war, abgeschafft.

Während frühe Anwender von Type-C diesen einfach als physische Steckverbindung für einen USB 2.0-Anschluss nutzten, bietet heutzutage nahezu jedes mobile Endgerät auch Unterstützung für Power Delivery [9]. Es handelt sich dabei um ein zusätzliches Kommunikationsprotokoll, mit dem zwei per USB Type-C verbundene Geräte Details über ihren Daten- und Strom-Austausch vereinbaren können. Diese Kommunikation findet über ein separates Pin-Paar im USB-Type-C-Konnektor statt. Neben einigen standardisierten Nachrichten erlaubt Power Delivery es Geräteherstellern auch, das Protokoll durch proprietäre Nachrichten zu erweitern.

2.4. USB-Konnektivität von mobilen Geräten

Moderne mobile Geräte können in der Regel sowohl als USB-Host als auch als USB-Device agieren. Als USB-Device unterstützt iOS etwa das Picture Transfer Protocol (PTP), Android zusätzlich auch das auf PTP aufbauende, aber umfangreichere Protokoll Media Transfer Protocol (MTP). Beide Protokolle erlauben es dem USB-Host zunächst eine Liste der verfügbaren Dateien bzw. Ordnern vom Mobilgerät abzufragen, und dann Inhalte sowohl zu lesen als auch zu schreiben. Während unter Android mit MTP ein Zugriff auf Dateien beliebigen Datentyps möglich ist, beschränkt sich der Zugriff unter iOS auf Bild- und Videodateien. Wie oben bereits erwähnt, muss der Dateizugriff über USB sowohl unter iOS als auch unter Android erst explizit durch den Nutzer freigegeben werden.

Als USB-Host unterstützen beide Betriebssystemfamilien den Anschluss von Peripheriegeräten. So können etwa USB-Mäuse oder -Tastaturen verwendet werden, um die Benutzeroberfläche zu bedienen. Auch USB-basierte Audio-Interfaces und Speichergeräte wie USB-Sticks, Speicherkartenleser oder Festplatten werden großflächig unterstützt. Besonders höherpreisige Mobilgeräte unterstützen auch den Betrieb von externen Bildschirmen am USB-Anschluss.

Zusätzlich bieten iOS und Android auch plattform-spezifische Zubehörprotokolle, die über USB Funktionalitäten zur Verfügung stellen, die über die Basis-USB-Spezifikation hinaus gehen.

3. Implementierungen von Juice-Jacking-Absicherungen

In diesem Abschnitt beschreiben und analysieren wir die Implementierungen von Juice-Jacking-Absicherungen der verschiedenen Hersteller. Alle Ansätze eint, dass Nutzerinteraktion benötigt wird, um den USB-Dateizugriff freizuschalten.

3.1. Geräte

Für die Analyse ziehen wir die in Tabelle 1 aufgelisteten Geräte und Betriebssysteme heran. Die Auswahl umfasst ein aktuelles Gerät von jedem der 5 Android-Hersteller mit dem höchsten weltweiten Marktanteil im Mai 2024. Diese sind Samsung, Xiaomi, Oppo, Vivo und Huawei. Zusätzlich erweitern wir unsere Auswahl um ein aktuelles Gerät von Apple, dem Marktführer im Bereich der mobilen Geräte. Für Samsung, den führenden Android-Hersteller, inkludieren wir 4 Geräte, um eine Bandbreite von Low-End- bis High-End-Geräten eines Herstellers abzudecken. Wir wählen auch ein aktuelles Pixel-Handy von Google aus, da diese mit einem weitgehend unveränderten AOSP-Build laufen. Schließlich inkludieren wir ein Gerät von Honor, um die vielen kleineren Gerätehersteller zu repräsentieren.

Hersteller	Modell	Betriebssystem	USB Power Delivery
Samsung	Galaxy A14	Android 13 (One UI 5)	Ja
Samsung	Galaxy S20 FE	Android 13 (One UI 5)	Ja

Samsung	Galaxy A33	Android 14 (One UI 6)	Ja
Samsung	Galaxy A23	Android 14 (One UI 6)	Ja
Xiaomi	12	Android 13 (MIUI 14)	Ja
Vivo	Y36	Android 13 (Funtouch OS 13)	Teilweise (kein Data Role Swap)
Oppo	A58	Android 13 (Color OS 13)	Ja
Huawei	Nova 12i	Android 12 (EMUI 14)	Ja
Honor	90 Lite	Android 13 (Magic OS 7.1)	Ja
Google	Pixel 7a	Android 14	Ja
Apple	iPad Pro 2022	iOS 17.4.1 (iPadOS 17.4.1)	Ja

Tabelle 1: Die analysierten Mobilgeräte

Alle untersuchten Geräte sind mit einem USB-C-Anschluss ausgestattet. Alle unterstützen auch prinzipiell das USB Power Delivery Protokoll. Erwähnenswert ist hier jedoch, dass das Vivo Y36 nur jenen Teil des Protokolls unterstützt, mit dem der Ladestrom zwischen Ladegerät und Mobilgerät vereinbart wird. Protokoll-Nachrichten, die der Regelung der Datenrolle in der Verbindung dienen, werden nicht unterstützt. Dies betrifft insbesondere den USB Power Delivery Data Role Swap, mit dem die Kommunikationspartner dynamisch die Rolle als USB-Host und USB-Device tauschen können.

3.2. Implementierungsvarianten

Insgesamt können wir in unserer Sammlung an Geräten 3 verschiedene Implementierungen von Juice-Jacking-Mitigations feststellen.

Variante 1: Bestätigungsdialog

Sobald das Mobilgerät über USB mit einem USB-Host verbunden wird, wird ein Dialog angezeigt, der über zwei Bedienelemente verfügt. Damit kann die Datenverbindung zum USB-Host entweder bestätigt oder abgelehnt werden. Diese Implementierung findet sich bei allen Geräten der Hersteller Samsung und Apple.

Variante 2: Auswahldialog

Ähnlich wie oben wird auch hier ein Dialog angezeigt, sobald das Mobilgerät mit einem USB-Host verbunden wird. Jedoch werden dem Nutzer hier alle Möglichkeiten an Funktionen angezeigt, mit denen die USB-Verbindung belegt werden kann. Neben dem Dateiaustausch stehen je nach Hersteller etwa Möglichkeiten zur Freigabe der mobilen Internetverbindung an den USB-Host zur Verfügung. Möchte der Nutzer keinen Datenaustausch erlauben, so muss er aus der Liste den Eintrag „Nur Laden“ (oder ähnlich) auswählen. Diese Implementierung findet sich bei Geräten der Hersteller Xiaomi, Oppo, Huawei und Honor.

Variante 3: Benachrichtigung und Einstellungen

Bei dieser Variante zeigt das System lediglich eine Benachrichtigung an, die den Benutzer darüber informiert, dass das Gerät an einen USB-Host angesteckt wurde. Zunächst ist aber keine Datenübertragung erlaubt. Um das zu ändern, muss der Benutzer über ein Tippen auf die Benachrichtigung die Systemeinstellungen öffnen, um dort eine andere Funktion für die USB-Verbindung auszuwählen. Die verfügbaren Möglichkeiten gleichen in etwa jenen, die auch im oben beschriebenen Auswahldialog angeboten werden. Diese Implementierung findet sich bei Geräten der Hersteller Google und Vivo.

Für jede dieser 3 Varianten wird also Benutzerinteraktion benötigt. In keiner Variante wird allerdings der Nutzer authentifiziert. Stattdessen wird vorausgesetzt, dass das Gerät entsperrt wird, bevor die Benutzerinteraktion zur Aktivierung einer USB-Datenverbindung ausgeführt werden kann. Da zum

Entsperren des Geräts eine Nutzerauthentifizierung nötig ist, wird diese für die Aktivierung der USB-Datenverbindung nicht mehr verlangt.

4. Prinzip und Sicherheit der Juice-Jacking-Mitigations

Die im vorigen Abschnitt beschriebenen Lösungen sollen Juice-Jacking-Angriffe verhindern, bei denen ein bösartiges Ladegerät Daten vom Gerät stiehlt. Damit diese Absicherung gewährleistet ist, darf es einem solchen bösartigen Ladegerät nicht möglich sein, die Nutzerinteraktion selbstständig auszuführen. Es darf also nicht möglich sein, gleichzeitig eine Datenverbindung zum Mobilgerät aufzubauen und Input-Events zu senden.

Im Kern verlässt sich diese Maßnahme gegen Juice-Jacking auf die USB-Spezifikation. Gemäß dieser kann sich ein USB-Anschluss zu jedem Zeitpunkt nur entweder als USB-Host oder als USB-Device verhalten. Ein bösartiges Ladegerät könnte also nur entweder als USB-Device (etwa als Tastatur) Eingabe-Events senden, um so die Nutzerinteraktion zur Aktivierung von USB-Datenfreigabe zu fälschen, oder als USB-Host eine USB-Datenverbindung aufbauen, für die die Freigabe nötig ist. Die Datenfreigabe kann nur erteilt werden, solange eine Verbindung zu einem USB-Host besteht.

Soweit die Theorie. Wie wir herausgefunden haben, bestehen allerdings bei allen getesteten mobilen Betriebssystemen Möglichkeiten, mit denen ein bösartiges Ladegerät sehr wohl Input-Events während eine USB-Datenverbindung senden kann, und so selbstständig die Datenfreigabe erteilen kann.

Alle Betriebssysteme erfordern das Entsperren des Bildschirms, bevor die USB-Datenfreigabe erteilt werden kann. Smartphone-Nutzer verwenden jedoch in der Regel ihr Gerät auch während des Ladevorgangs immer wieder. Dazu müssen sie es natürlich entsperren. Für einen Angreifer stellt in diesem Szenario also der Sperrbildschirm kein wirkliches Hindernis dar, sondern lediglich die Notwendigkeit von Nutzerinteraktion für die USB-Dateifreigabe.

5. Angriffe

Wir haben im Rahmen dieses Projekts drei verschiedene Möglichkeiten gefunden, mit denen ein bösartiges Ladegerät die Interaktion zur USB-Datenfreigabe autonom ausführen kann. So kann das Ladegerät vom Nutzer unbemerkt Daten extrahieren. Während alle Möglichkeiten unter Android funktionieren, funktioniert nur eine auch unter iOS. Eine Gegenüberstellung der drei Angriffsmöglichkeiten findet sich in Tabelle 2.

Angriffsmethode	Dauer	Vorbedingungen	Systeme	Auffälligkeit
AOAP	0,1 - 1,5 Sekunden	Keine	Android <15*	Kaum wahrnehmbar
Race Condition	Wenige Sekunden	Keine	Android <15*	Potentiell wahrnehmbar
Bluetooth HID	Mehrere Sekunden	Je nach Gerät: BT bereits aktiviert	Android <15* & iOS <18.4	Sichtbar, aber harmlos wirkend

* Gerätehersteller muss die Änderungen von Google in seine Android-Version integriert haben

Tabelle 2: Übersicht über die gefundenen Angriffsmethoden

Obwohl diese Möglichkeiten alle oben genannten Implementierungen von Nutzerinteraktionen umgehen können, sind einzelne Varianten besonders unauffällig zu umgehen. Konkret müssen bei jenen

Implementierungen, die von sich aus schon einen Dialog zeigen, weniger Schritte ausgeführt werden. Es ist also lediglich für einige wenige Augenblick der Dialog sichtbar, bis er vom bösartigen Ladegerät bestätigt wird. Auch die verschiedenen Möglichkeiten zur autonomen Erteilung der USB-Datenfreigabe unterscheiden sich in ihrer Auffälligkeit für den Benutzer.

Angriff 1: Android Open Accessory Protocol

Frühe Android-Geräte unterstützten nicht flächendeckend die Rolle als USB-Host. Es war also zum Beispiel aufgrund mangelnder Hardware-Unterstützung nicht bei jedem Gerät möglich, eine USB-Tastatur anzuschließen. Jedes Gerät verfügte aber zum Datenaustausch mit einem Computer über die Möglichkeit, als USB-Device zu agieren. Auch war bei USB-Konnektoren vor USB-C die Power-Rolle mit der Daten-Rolle fix verknüpft. Der USB-Host war stets dafür verantwortlich, ein angeschlossenes USB-Device mit Strom zu versorgen. Um diese Einschränkungen zu umgehen schuf Google das Android Open Accessory Protocol (AOAP). Es erlaubt die Implementierung von Funktionalität als USB-Host, die normalerweise USB-Devices vorbehalten ist. So ist es beispielsweise möglich, mit speziellen USB Control Requests als USB-Host Eingabe-Events an ein USB-Device zu schicken. Gemäß der Spezifikation des Android Open Accessory Protocol muss dazu das Android-Gerät zunächst mit einem anderen USB Control Request in einen speziellen Accessory-Modus versetzt werden. In diesem Modus ist jede andere USB-Funktionalität unterbunden. Es sollte also aus technischer Sicht nicht möglich sein, gleichzeitig eine MTP-Datenverbindung aufzubauen und mittels Android Open Accessory Protocol Eingabe-Events zu senden. Tatsächlich konnten wir hier entscheidende Divergenzen zwischen der Spezifikation und der Implementierung von AOAP feststellen: Bei Android-Geräten sämtlicher Hersteller konnten wir mit AOAP Eingabe-Events schicken, ohne vorher den Accessory-Modus zu aktivieren. Es kann also parallel eine MTP-Datenverbindung aufgebaut werden und mit den gesendeten Eingabe-Events die Interaktion zur Datenfreigabe absolviert werden.

Angriff 2: Race Condition in Androids Input-Dispatcher

Android baut auf einen Linux-Kernel auf. Ein Kernel-Treiber ist verantwortlich dafür, Input-Events von Eingabegeräten zu lesen und an höhere Ebenen weiterzureichen. Unter Android nimmt die InputReader-Komponente im InputManagerService-Systemservice die Events entgegen und hinterlegt sie in eine Warteschlange, von wo sie vom InputDispatcher konsumiert und verteilt werden. Diese Warteschlange fungiert als Puffer, der sicherstellt, dass Eingabe-Ereignisse in der richtigen Reihenfolge verarbeitet werden und nicht verloren gehen, auch wenn das System beschäftigt ist. Wir stellen fest, dass das Android-Eingabe-Subsystem die Input-Events in dieser Warteschlange behält, auch wenn das Eingabegerät, das sie erzeugt hat, nicht mehr angeschlossen ist. Zusätzlich serialisiert Androids Input-Dispatcher absichtlich Tastatur-Events. Er wartet bis alle vorherigen Input-Events vollständig verarbeitet wurden, bevor ein Tastatur-Event verteilt wird. Das bedeutet, dass ein einzelner Prozess, der übermäßig komplexe Logik in seinem Tastatur-Eingabe-Handler ausführt, die Input-Event-Verteilung für alle anderen Prozesse aufhält. Wir beobachten, dass solche Tastatur-Eingabe-Handler in Systemprozessen oder vorinstallierten Anwendungen auf allen untersuchten Android-Geräten zu finden sind. Ein bösartiges Ladegerät kann dies ausnutzen, indem es als USB-Peripheriegerät startet und die Input-Warteschlange mit einer speziell gestalteten Sequenz von Input-Events überflutet. Das Ladegerät tauscht dann mittels USB Power Delivery Data Role Swap die Daten-Rollen. Das Ladegerät agiert nun als USB-Host, während das Mobilgerät noch immer damit beschäftigt ist, die Input-Events des Angreifers zu verteilen. Diese Input-Events können daher dazu benutzt werden, die Interaktion zur Bestätigung der Datenverbindung mit dem bösartigen Ladegerät auszuführen.

Angriff 3: Bluetooth-Eingabegerät

Schließlich haben wir auch eine dritte Angriffsmöglichkeit gefunden, die nicht nur Android, sondern auch iOS betrifft. Hier versteckt der Angreifer im bösartigen Ladegerät zusätzlich ein Bluetooth-Modul. Der

Angriff startet, indem das Ladegerät als USB-Eingabegerät agiert. Es sendet an das Mobilgerät Input-Event, die die Systemeinstellungen für Bluetooth öffnen. Dies macht das Bluetooth-Modul im Mobilgerät sichtbar für alle anderen Bluetooth-Geräte in der Umgebung. Das bösartige Ladegerät kann dies nutzen, um die Bluetooth-Adresse des mobilen Geräts zu erfassen. Dann kann es eine Verbindungs-Anfrage an diese Adresse versenden. Als Reaktion auf diese Anfrage zeigt das Mobilgerät dem Nutzer einen Bestätigungsdialog für die Verbindung an. Als USB-Eingabegerät kann das bösartige Ladegerät diesen Dialog autonom bestätigen. Das Ladegerät kann nun über die Bluetooth-Verbindung ebenfalls Input-Events an das Mobilgerät schicken. Es schaltet sodann das USB-Interface in den USB-Host-Modus um, und zwingt das Mobilgerät mittels USB Power Delivery Data Role Swap in die Rolle als USB-Device. Nun kann das Ladegerät über die Bluetooth-Verbindung die Interaktion zur Freigabe der USB-Datenverbindung absolvieren.

Im Rahmen dieses Projekts wurden alle drei Angriffe auf Basis eines eigens angefertigten Ladegerät-Prototyps umgesetzt. Der Prototyp besteht aus einer Platine mit Raspberry Pi RP2040 Mikrokontroller und USB-PD-Controllern. Gesteuert wird diese Platine von einem Raspberry Pi 4/5 Single Board Computer, auf dem die Angriffslogik in Python implementiert wurde.

6. Evaluierung der Angriffe und Fehlerbehebung

Auf allen getesteten Geräten funktioniert zumindest eine der oben beschriebenen Angriffsvarianten. Die meisten Android-Geräte zeigten sich sogar auf alle drei Methoden anfällig.

Der mächtigste der gefundenen Angriffe ist jener, der das Android Open Accessory Protocol ausnutzt. Besonders auf Geräten, die als Nutzerinteraktion einen Dialog nutzen, ist der Angriff kaum zu bemerken. So ist der Bestätigungsdialog auf einem Samsung Galaxy S20 FE bei diesem Angriff nur etwa 133 Millisekunden lang zu sehen. Es handelt sich dabei etwa um die halbe Dauer eines menschlichen Blinzelns. Danach hat das bösartige Ladegerät unbemerkt Zugriff auf die Nutzerdaten am Gerät. Ein unbedarfter Endanwender hat keine Chance, einen Angriff zu bemerken.

6.1. Auswirkungen der Angriffe

Alle Angriffe erlauben es einem bösartigen Ladegerät, Zugriff auf am Mobilgerät gespeicherte Nutzerdaten zu erhalten. Auf allen getesteten Geräten ist zumindest ein Zugriff auf alle Screenshots, Fotos und Videos möglich. Unter Android erstreckt sich der Zugriff auch auf weitere Daten, wie etwa Downloads, Dokumente, die Liste installierter Apps oder bestimmte App-Daten, zum Beispiel über Messenger empfangene Medien.

6.2. Fehlerbehebung

Im Rahmen dieses Projekts wurden alle Angriffe an die betroffenen Hersteller gemeldet. Die Hersteller haben die Angriffe anerkannt und in einigen Fällen auch bereits Fehlerbehebungen ausgeliefert. So muss sich der Nutzer in iOS 18.4 nun mittels Pin, Fingerabdruck oder Gesichts-Scan authentifizieren, wenn er den Dialog zur USB-Datenfreigabe bestätigt. In Android 15 wurde die Angriffe verunmöglicht, indem ebenfalls Authentifizierung erzwungen wird, wenn in den Systemeinstellungen der USB-Modus geändert wird. Dennoch bleiben viele Geräte nach wie vor anfällig auf die hier gezeigten Angriffe. Dies ist darauf zurückzuführen, dass Android 15 noch nicht von vielen Android-Herstellern an ihre Geräte ausgeliefert wird. Außerdem ist angesichts der bisherigen Unterschiede zwischen verschiedenen Android-Herstellern unklar, ob alle Hersteller die von Google in Android 15 eingebauten Gegenmaßnahmen in ihre Android-Varianten übernehmen.

6.3. Gefahr für Endnutzer

Endanwender sind besonders dort der Gefahr für Juice-Jacking-Angriffe ausgesetzt, wo gleichzeitig häufig Bedarf für eine Lademöglichkeit besteht, jedoch lediglich öffentliche Infrastruktur genutzt werden kann. Beispielhaft sollen hier Flughäfen erwähnt werden, wo Nutzer gerade im Transit nach einigen Stunden Reise trotz leerem Handy-Akku erreichbar bleiben möchten. Flughäfen bieten daher meist USB-Buchsen oder sogar -Kabel an, über die Reisende ihre Mobilgeräte mit Strom versorgen können.

Hier bietet sich eine ideale Möglichkeit für einen Angreifer. Durch Austausch der Elektronik oder Firmware im Ladegerät kann diese so präpariert werden, dass der Angreifer von allen zukünftig angeschlossenen Mobilgeräten private Daten stehlen kann. Denkbar ist auch ein für den Endnutzer unauffälliger Aufsatz auf bestehende Infrastruktur, ähnlich wie in der Vergangenheit etwa Bankomaten für Skimming-Angriffe präpariert wurden.

6.4. Empfehlungen für Endnutzer

Endnutzer haben mehrere Möglichkeiten, um sich vor Juice-Jacking-Angriffen zu schützen:

- Nutzung von eigenen Ladegeräten bzw. Powerbanks
  Vermeiden Sie es, Ihr Mobilgerät an einer öffentlichen (oder mit Unbekannten geteilten) USB- oder USB-C-Buchse aufzuladen. Nehmen Sie stattdessen Ihre eigene Powerbank mit. Häufig stehen zusätzlich zu USB-Buchsen auch normale Netzdosen bereit. Hier hilft es, wenn Sie ihr eigenes Ladegerät mitbringen.

- USB-Datenblocker
  Für besonders vorsichtige bzw. gefährdete Anwender gibt es Produkte, die zwischen die öffentliche Ladeinfrastruktur und das eigene Mobilgerät gesteckt werden und auf physischer Ebene die Datenkommunikation unterbinden. Es besteht hier allerdings die Möglichkeit, dass auch Schnellladeprotokolle unterbunden werden, worunter die Ladegeschwindigkeit leidet.

- Laden bei ausgeschaltetem Gerät
  Sollte es unvermeidlich sein, das eigene Mobilgerät an eine fremde USB-Buchse zu stecken, schalten Sie Ihr Gerät vorher vollständig aus. Im ausgeschalteten Zustand ist in der Regel keine USB-Kommunikation möglich und der Speicher des Geräts zusätzlich verschlüsselt.

- Installieren Sie Betriebssystem-Updates sobald verfügbar
  Immer wieder werden selbst in den modernsten mobilen Betriebssystemen Sicherheitslücken im USB-Stack gefunden. Wenn Sie Updates direkt nach Verfügbarkeit installieren, sind Sie vor Ausnutzung dieser bekannten Sicherheitslücken bestmöglich geschützt.

7. Zusammenfassung

In diesem Projekt wurde die Sicherheit der gängigen mobilen Betriebssysteme iOS und Android im Hinblick auf Angriffe durch bösartige Ladegeräte untersucht. Dazu wurde zunächst ein Überblick über die verschiedenen Gegenmaßnahmen geboten. Diese wurden dann aus technischer Sicht analysiert, um das zugrundeliegende USB-Trust-Modell zu beleuchten. Schließlich konnten wir anhand von drei verschiedenen Angriffs-Methoden zeigen, dass die bestehenden Maßnahmen nicht in der Lage sind, USB-basierten Datendiebstahl durch bösartige Ladegeräte zu unterbinden. Einige unserer neuartigen Angriffe benötigen nur wenige Millisekunden, sind für Endanwender kaum zu erkennen und funktionieren auf vielen gängigen Mobiltelefonen nach wie vor.

Referenzen

[1] W. H. L. S. M. S. K. Weizhi Meng, „JuiceCaster: Towards automatic juice filming attacks on smartphones," *Journal of Network and Computer Applications ,* Bd. 68, 2016.

[2] H. G. Q. Y. Yuanda Wang, „GhostTalk: Interactive Attack on Smartphone Voice System Through Power Line," in *Network and Distributed Systems Security (NDSS) Symposium*, 2022.

[3] Compaq, Digital Equipment Corporation, IBM PC Company, Intel, Microsoft, NEC, Northern Telecom, „Universal Serial Bus Specification," 01 1996. [Online]. Available: https://web.archive.org/web/20180130144424/https://fl.hw.cz/docs/usb/usb10doc.pdf. [Zugriff am 01 2024].

[4] Compaq, Digital Equipment Corporation, IBM PC Company, Intel, Microsoft, NEC, Northern Telecom, „Universal Serial Bus Specification Revision 1.1," 09 1998. [Online]. Available: https://fabiensanglard.net/usbcheat/usb1.1.pdf. [Zugriff am 01 2024].

[5] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, „Universal Serial Bus Specification Revision 2.0," 04 2000. [Online]. Available: http://sdpha2.ucsd.edu/Lab_Equip_Manuals/usb_20.pdf. [Zugriff am 01 2024].

[6] USB Implementers Forum, Inc, „On-The-Go Supplement to USB 2.0 Specification," 12 2001. [Online]. Available: https://www.rockbox.org/wiki/pub/Main/DataSheets/OTG1_0a.pdf. [Zugriff am 01 2024].

[7] Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, NEC Corporation, ST-NXP Wireless, Texas Instruments, „Universal Serial Bus 3.0 Specification," 11 2008. [Online]. Available: http://www.softelectro.ru/usb30.pdf. [Zugriff am 01 2024].

[8] USB 3.0 Promoter Group, „Universal Serial Bus Type-C Cable and Connector Specification," 08 2014. [Online]. Available: https://www.usb.org/sites/default/files/USB%20Type-C%20Spec%20R2.0%20-%20August%202019.pdf. [Zugriff am 01 2024].

[9] USB 3.0 Promoter Group, „Universal Serial Bus Power Delivery Specification," 07 2012. [Online]. Available: https://www.usb.org/document-library/usb-power-delivery. [Zugriff am 01 2024].