ChoiceJacking: Compromising Mobile Devices through Malicious Chargers like a Decade ago

Florian Draschbacher
Graz University of Technology
A-SIT Austria

Lukas Maar
Graz University of Technology

Mathias Oberhuber
Graz University of Technology

Stefan Mangard
Graz University of Technology

Abstract

JuiceJacking is an attack in which malicious chargers compromise connected mobile devices. Shortly after the attack was discovered about a decade ago, mobile OSs introduced user prompts for confirming data connections from a USB host to a mobile device. Since the introduction of this countermeasure, no new USB-based attacks with comparable impact have been found.

In this paper, we present a novel family of USB-based attacks on mobile devices, ChoiceJacking, which is the first to bypass existing JuiceJacking mitigations. We observe that these mitigations assume that an attacker cannot inject input events while establishing a data connection. However, we show that this assumption does not hold in practice. We present a platform-agnostic attack principle and three concrete attack techniques for Android and iOS that allow a malicious charger to autonomously spoof user input to enable its own data connection. Our evaluation using a custom cheap malicious charger design reveals an alarming state of USB security on mobile platforms. Despite vendor customizations in USB stacks, ChoiceJacking attacks gain access to sensitive user files (pictures, documents, app data) on all tested devices from 8 vendors including the top 6 by market share. For two vendors, our attacks allow file extraction from locked devices. For stealthily performing attacks that require an unlocked device, we use a power line side-channel to detect suitable moments, i.e., when the user does not notice visual artifacts.

We responsibly disclosed all findings to affected vendors. All but one (including Google, Samsung, Xiaomi, and Apple) acknowledged our attacks and are in the process of integrating mitigations.

1 Introduction

Mobile devices such as smartphones and tablets have grown into indispensable everyday companions, storing sensitive user data such as private pictures or documents. Until about a decade ago, a significant threat to these devices were JuiceJacking attacks [16, 19]. These attacks exploit the fact that mobile phones use a single physical connector for charging and for data exchange. A malicious party could hide a computer in a charger to stealthily establish a data connection during charging. Since all USB connections were trusted by default, the attacker could access arbitrary files stored on the device or gain code execution without the user's knowledge.

Mobile platform developers including Google, its Android partners and Apple quickly recognized the threat posed by JuiceJacking attacks. As a mitigation, modern versions of Android and iOS require user consent[1] before a *data connection from a computer* can be established. However, to date, no such requirement has been added for USB connections from a mobile device to *peripherals and accessories* such as keyboards or mice. Most USB-based attacks presented after the introduction of JuiceJacking countermeasures, therefore, act as malicious peripherals. Nohl et al. [29] proposed BadUSB, which reprograms the firmware of USB peripherals, such as thumb drives, to inject input events. However, as an input device, the approach requires an additional channel for extracting data. Meng et al. [23, 24] suggested a malicious charger that contains a USB-to-HDMI interface for accessing the screen content of mobile devices. Although this setup allows data extraction, it can only access the content visible on screen. Wang et al. [50] hide an audio interface in a malicious charger, but are only able to extract data accessible through the voice assistant. Some researchers exploit physical side-channels of cable connections to extract data from charging devices. However, passive side-channels can only extract information from events triggered by the user [7, 52], and active side-channels require detailed prior knowledge about the victim [12, 50]. In summary, none of the attacks published since the introduction of JuiceJacking mitigations matches the impact of JuiceJacking attacks. Additionally, none so far explored the potential of combining characteristics of malicious USB hosts and malicious USB devices (e.g., peripherals).

In this paper, we present a novel attack family, ChoiceJacking, which has an impact comparable to JuiceJacking

[1] In this paper, we use the term "consent" to refer exclusively to technical means that enforce a user's intent

attacks, but works on modern mobile devices. It introduces the concept of hybrid USB attacks that combine aspects of both host-based and device-based attacks on mobile devices. ChoiceJacking attacks operate under the same threat model and attacker goals as JuiceJacking attacks. They leverage malicious chargers to gain *file access or code execution* on mobile devices. While we confirm the attack principle on iOS as well, this paper focuses on ChoiceJacking attacks for variants of Android, the most popular mobile platform.

We base our work on a fundamental flaw in the JuiceJacking mitigations implemented in major mobile platforms. The core idea of these mitigations is to require explicit user confirmation before a USB host can establish a data connection. This user confirmation is implemented by requiring certain user interface interactions, e.g., confirming a user prompt that explains what access is exposed. This countermeasure works under the assumption that a USB host cannot inject input events to autonomously confirm the user prompt. When considering the USB protocol in isolation, this is a valid assumption. The protocol dictates that a USB port can only operate as either a *USB host* (e.g., a computer) or a *USB device* (e.g., a mouse or keyboard) at a given time. However, from a system perspective, this assumption does not hold on mobile platforms. We identify three concrete attack techniques for Android and iOS that allow breaking this assumption. All attack techniques combine aspects of both a malicious USB host and USB device. They enable the attacker to take control of the user interface while a USB user confirmation prompt is shown and to autonomously complete the confirmation.

The most effective attack technique exploits flaws in the USB-based Android Open Accessory Protocol (AOAP). This protocol enables an accessory to register as an input device despite operating as a USB host. It does not require any user consent. According to the AOAP specification, Android devices are only supposed to accept AOAP messages while in a special accessory mode that does not allow data connections. However, we find that modern real-world Android devices accept certain AOAP messages at all times, allowing injection of input events while establishing USB data connections. This allows a malicious charger to trigger the user consent prompt intended to mitigate JuiceJacking attacks and autonomously confirm it. In our evaluation, this technique allows gaining file access on all Android devices.

For our second attack technique, we identify a race condition in Android's input subsystem. Android still trusts *USB peripherals* by default, so that a malicious charger can act as an input device without requiring user consent. This allows a malicious charger to flood the input event queue with key events as a *USB peripheral*. When the charger then (through USB Power Delivery) switches its USB interface to operate as a *USB host* and trigger the user consent prompt, the Android OS is still processing the injected input events. If the attacker specifically crafts the input events used for flooding the queue, this technique allows bypassing the user confirmation prompt. It succeeds on 9 of 10 evaluated Android devices.

In the third attack technique, a malicious charger can act as a USB peripheral to make the mobile device establish a connection to a Bluetooth input device integrated in the charger. The charger can then switch its USB interface to operate as a USB host, which triggers the user consent prompt intended to mitigate JuiceJacking attacks. However, the malicious charger can use the connected Bluetooth input device to autonomously accept the user consent prompt. 10 of 11 evaluated devices across both Android and iOS are susceptible to this technique.

On most evaluated devices, gaining data access through ChoiceJacking attacks requires the victim device to be unlocked at some point during charging. Once this condition is met, the attack only takes in the range of milliseconds on many devices, which is merely a short flickering on the screen that may remain unnoticed. After this brief flash of visual artifacts, the attacker has stealthy file access or code execution on the victim device until the USB connection is removed. For two vendors (Honor and Oppo), ChoiceJacking attacks gain MTP file access on *locked devices*, by exploiting further implementation flaws in USB handling logic. On devices from Xiaomi, ChoiceJacking attacks gain ADB development access (code execution) even if they are not development-enabled before the attack. For enabling entirely stealthy ChoiceJacking attacks, we present a power line side-channel for detecting suitable attack times, e.g., when the device screen is unobserved during a phone call.

Contributions. The main contributions of our work are

(1) We uncover fundamental flaws in the JuiceJacking mitigations of mobile platforms that render them largely ineffective on state-of-the-art devices.

(2) We introduce the novel concept of USB attacks on mobile platforms that combine traits of host-based and device-based attacks. We provide three concrete attack techniques that instantiate this concept to exploit JuiceJacking mitigation flaws and establish PTP, MTP or ADB connections on Android or iOS without user consent.

(3) We demonstrate the effectiveness of our attacks on 11 current-generation mobile devices from 8 vendors. Additionally, we present a power line side-channel that allows attackers to identify suitable moments for ChoiceJacking attacks.

Disclosure. We reported 16 vendor-specific security issues to 6 Android vendors, 4 upstream Android vulnerabilities to Google, and one vulnerability to Apple. All but one vendors confirmed our findings and are in the process of developing or rolling out patches. Google and Samsung already assigned CVEs[2]. Google plans to fix the upstream design flaws in an upcoming Android release.

Outline. Section 2 provides background. Section 3 introduces the threat model and principle of ChoiceJacking attacks. We present our attack techniques in Section 4. In

[2]CVE-2024-43085 and CVE-2024-20900, respectively

Section 5, we detail the prototype used for the evaluations in Section 6. Section 7 presents our power line side-channel for detecting suitable attack moments. Section 8 discusses susceptible platforms, existing mitigations and related work, before Section 9 concludes this paper.

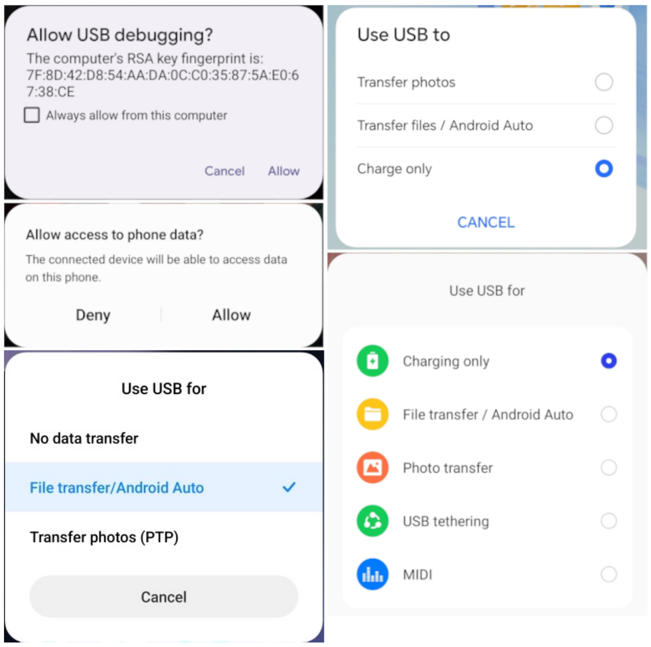
2 Background

This section briefly introduces USB and the Android OS, which this paper focuses on.

2.1 USB, USB Type-C and USB PD

The Universal Serial Bus (USB) allows connecting a large variety of different hardware peripherals to a computer through a single connector and low-level communication protocol. Since its first publication in 1996 [6], the standard has been widely adopted in all sorts of electronic devices. Fundamentally, USB connections on a logical level are always formed between a single USB host and a USB device. All communication is controlled by the USB host. The USB device may only transmit data to the host after the latter issues a corresponding request. Usually, the USB host is a computer of some form, while the USB device is a peripheral such as a mouse or keyboard. Once a physical connection is established, the USB host queries the USB device for its USB descriptors. These are data structures that describe the supported functionality of the device, as well as metadata. This information allows the host computer to identify the attached peripheral and load suitable drivers. Originally, the role a communication partner assumed in a USB connection was hardcoded in the connector type it featured. Host devices used a USB Type-A connector that differed in shape from the Type-B connector used on the device side.

USB Type-C and USB Power Delivery. The Type-C connector [47] was added to the USB ecosystem to solve multiple issues with the previous Type-A and Type-B connector families. It features a dedicated Configuration Channel (CC) line that communication partners may use to advertise their port as either Upward Facing Port (UFP; what used to be a USB device port) or Downward Facing Port (DFP; USB host) through specific resistor configurations. Additionally, more complex role arrangements can be negotiated by exchanging USB Power Delivery (USB PD) [3] messages over the CC line. USB PD introduces a distinction between a port's power role (source or sink) and its data role (UFP or DFP). The communication partners advertise their capabilities as part of an initial PD handshake. On connection establishment, the DFP (as identified through resistors) always starts out as the power source and USB host. Later on, both partners can dynamically request power or data role swaps. For example, this enables the implementation of USB hubs that either act as power source or sink depending on whether their power supply is connected.


Allow USB debugging?
The computer's RSA key fingerprint is:
7F:8D:42:D8:54:AA:DA:0C:C0:35:87:5A:E0:6
7:38:CE
☐ Always allow from this computer
Cancel Allow

Use USB to
Transfer photos
Transfer files / Android Auto
[✓ Charge only]
CANCEL

Allow access to phone data?
The connected device will be able to access data on this phone.
Deny Allow

Use USB for
No data transfer
[✓ File transfer/Android Auto]
Transfer photos (PTP)
Cancel

Use USB for
[✓ Charging only]
File transfer / Android Auto
Photo transfer
USB tethering
MIDI

Figure 1: USB dialogs displayed by different Android variants

2.2 Android

With a market share of over 71 %[3], Android is the dominating mobile operating system. It is developed in an open-source fashion in the Android Open Source Project (AOSP) led by Google. The OS consists of a Linux kernel and a purpose-built userspace under the Apache 2.0 license. As a result of this licensing arrangement, device vendors may adapt large parts of the operating system. Almost all major Android phone manufacturers try to differentiate their products by shipping customized Android variants. These customizations have led to numerous security vulnerabilities in the past [21, 35, 43].

Modern Android devices feature a PD-enabled USB Type-C port that supports acting as DFP and UFP. When acting as a DFP, Android supports USB Human Interface Device (HID) class devices such as USB mice and keyboards. When acting as a UFP, Android exposes interfaces for exchanging files and for development access.

USB File Access. File exchange works through the Picture Transfer Protocol (PTP) for transferring photos and videos and the Media Transfer Protocol (MTP), an extension of PTP that enables transferring files of arbitrary formats. For every single USB connection that wishes to use any of these protocols, Android requires explicit user consent. Depending on the protocol and the vendor-specific Android variant, this works either through confirming a dialog or through changing a system setting.

USB Development Access. On Android, development access is implemented through the Android Debug Bridge

[3]As of Q1 2024: https://gs.statcounter.com/os-market-share/mobile/worldwide

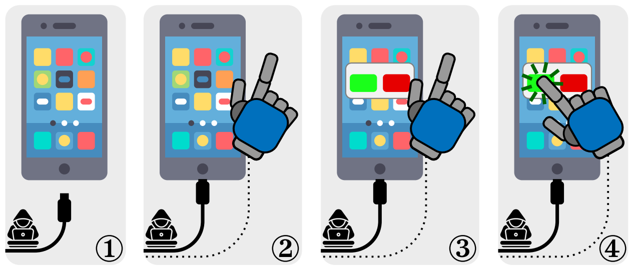


Figure 2: Principle of ChoiceJacking attacks. ① Victim device is attached to the malicious charger. ② The charger establishes an extra input channel. ③ The charger initiates a data connection. User consent is needed to confirm it. ④ The charger uses the input channel to spoof user consent.

(ADB) protocol and daemon. It grants the USB host shell access to the device, which can be used for installing apps, accessing the file system or executing arbitrary binaries. Using the ADB development access interface involves a three-staged setup procedure. As a first step, a hidden development system settings menu needs to be unlocked. This step is permanent and only needs to be carried out once, but requires user authentication, e.g., by presenting a valid unlock pin. Through the now unlocked development settings menu, it is possible to change configurations that lower the system's security. For example, it allows changing the default USB mode, so that the user consent described above is not needed for every MTP or PTP connection. As the second step for enabling ADB access, the user needs to enable USB debugging in the unlocked development settings menu. As the third and final step, Android requires user confirmation through a UI dialog to establish trust in a specific computer before it can interact with the ADB interface. When acting as a UFP, Android also supports platform-specific accessories that use the Android Open Accessory Protocol (AOAP) [1]. Like standard USB peripherals, AOAP accessories are trusted by default and may implement HID functionality. Still, as a major difference to standard USB peripherals, AOAP accessories act as USB hosts.

3 Threat Model and Attack Principle

In this section, we describe our threat model, which largely aligns with prior work [19, 44, 50], including the one from JuiceJacking. Subsequently, we discuss the flawed design of JuiceJacking mitigations and the principle of our ChoiceJacking attacks that bypass them.

3.1 Threat Model

Our threat model largely aligns with that of JuiceJacking attacks and various previous USB-based attacks on mobile devices [19,44,50]. We assume a scenario in which the victim uses phone charging infrastructure that has been tampered with by an attacker. Many public spaces offer mobile device charging possibilities that are anonymously shared between a large number of people. Examples are phone charging stations at airports or cafes, rentable power banks at museums, or USB wall sockets in hotel rooms. In our threat model, manipulations entail replacing the electronics in the charger or reprogramming the firmware of existing electronics. The goal of the attacker is to extract sensitive information from a mobile device that is connected to the malicious charger. Since the malicious charger integrates a wireless module (e.g., Wifi), the attacker does not need to be physically present for carrying out ChoiceJacking attacks. While some ChoiceJacking attacks require an unlocked screen, we alleviate this constraint through the power line side-channel in Section 7.

ChoiceJacking attacks are untargeted. An attacker can place malicious chargers in public places to harm as many victims as possible. As part of this untargeted setup, all information needed to carry out the attack is collected from the device once it is connected to the malicious charger. The attack will succeed on all susceptible charging devices (i.e., running Android or iOS and, for some vendors, unlocked while charging), and will thereby cause considerable damage to a large number of people over time. ChoiceJacking represents a primitive for obtaining access to users' private files stored on the mobile device. These files commonly include sensitive information such as private pictures, documents or app data. Through leaking this data, ChoiceJacking attacks can have severe consequences for affected invidiuals. Depending on the device configuration, ChoiceJacking attacks can also gain permanent code execution on the mobile device.

3.2 Design Flaws in JuiceJacking Mitigations

We observe that integrating user interaction as a countermeasure for JuiceJacking fundamentally conflicts with mobile platforms' default trust in *USB peripherals*. Mobile devices by default trust accessories and input devices connected to their USB port. However, at the same time, mobile OS developers introduced the requirement for explicit user consent for data connections to *USB hosts* as a countermeasure against JuiceJacking. On some Android variants, and also on iOS, a dialog informs users about the requested data access and offers the possibility to deny the connection. On other Android variants such as AOSP, the user has to manually change the USB mode in the system settings to enable data communication over the connection to the USB host. In all cases, when working as designed, this mitigation entails that (1) *the screen is unlocked*, and (2) *some UI interaction is performed* that explicitly confirms the data connection.

Screen Unlock. Before user consent can be expressed through the user interface, mobile platforms require the screen to be unlocked. On properly configured systems, unlocking the screen requires user authentication, for example through entering an unlock pin or presenting a fingerprint. The requirement to unlock the screen is therefore intended to ensure that all subsequent actions on the device are carried out by its legitimate owner. However, in the context of charger-based attacks, this guarantee does not necessarily hold. Users frequently unlock their device during charging, e.g., to make phone calls or look up information online. Prior work exploits this fact in malicious chargers to extract the screen unlock pin [7], observe web browsing activity [52] or read user input [23] of charging mobile devices. A malicious charger can also use this opportunity to stealthily act as an input device and interact with the phone's UI. This works because mobile platforms still trust USB *peripherals and accessories* by default, as described in Section 2. We conclude that not all UI interaction carried out while the device is unlocked and connected to a charger is necessarily caused by or known to the user. This voids the first part of the JuiceJacking mitigations already on a design level. Further flaws in vendor-specific Android variants additionally void it on the implementation level. These allow attacks that entirely bypass the requirement for the screen to be unlocked prior to USB data access.

UI Interaction. The protection therefore hinges on the fact that UI interaction is required for activating a data connection to a USB host. This measure is intended to ensure that the user is aware of potential consequences and consciously making the choice to enable the data connection. It assumes that a malicious charger cannot operate as a USB host and as an input device at the same time. Otherwise, the attacker could autonomously carry out the user interaction needed to confirm its own data connection. This assumption is based on limitations of the USB protocol, which only allows a USB port to either act as a USB host or a USB peripheral at one point in time. When considering USB communication in isolation, an attacker would therefore have to decide on one of these two roles. They could either establish the USB data connection as a USB host or carry out user interaction as a USB peripheral. However, we show that when considering a holistic view on mobile platforms, there are multiple options for a malicious charger to *conceptually* hold both these roles at the same time. This is the basis for ChoiceJacking attacks.

3.3 Attack Principle

ChoiceJacking attacks are the first USB attacks to combine aspects of host-based and device-based attacks. In all ChoiceJacking attacks, the attacker compromises mobile devices through a malicious charger. To hide its malicious intent, the charger initially behaves like a normal phone charger. Once it detects a suitable moment (e.g., through the power line side-channel described in Section 7), it mounts an attack. As a common pattern for all ChoiceJacking attacks, the malicious charger issues input events while initiating a data connection to the victim device as a USB host. Conceptually, this works by exploiting initial USB access to establish a second channel to the victim device. This second channel can subsequently be used for injecting input events while the primary USB channel operates as a USB host initiating a data connection. Figure 2 illustrates this principle. It works on all major mobile platforms.

We present three concrete attack techniques that instantiate this novel attack principle. While the presented techniques focus on the Android platform, one of them works on iOS as well. The first technique uses implementation flaws in the Android Open Accessory Protocol (AOAP) as its (second) input channel. The second technique exploits a race condition in the Android input subsystem as its conceptual input channel. The third technique uses initial USB peripheral access to establish a BT input device connection as a second input channel. Additionally, we present a power line side-channel that allows a malicious charger to detect suitable attack moments.

Two of our attack techniques require switching USB roles, which a malicious charger can accomplish through USB Power Delivery. For determining the most effective ChoiceJacking attack for a specific device, an attacker can use the device information exposed through USB descriptors and USB PD Discover Identity messages. As described in Section 2.1, a USB device sends its USB device descriptor to the USB host as part of the initial handshake. Among other data, the descriptor contains information on the device's vendor, product, and version, as well as its unique serial number. This information can also be obtained before the USB handshake, by using the USB PD Discover Identity mechanism. Alternatively, timing or power side-channel based fingerprinting approaches as presented by Bates et al. [5] and Spolaor et al. [37] can be used for accurately identifying devices.

3.4 Attack Impact

Through a ChoiceJacking attack, the malicious charger either gains access to the victim device's PTP/MTP file transfer interface or its ADB development interface. On Android, both interfaces permit read/write control over the entirety of the victim device's public storage. Gaining MTP file transfer access does not pose any special requirements on the victim device. In other words, virtually all Android devices are affected. Most intuitively, an attacker can exploit this access to extract personal data such as pictures, videos, voice recordings, downloads or documents. Prior research has also shown that applications commonly store sensitive app-specific data in public storage. As a result, access to public storage could in the past be exploited for e.g., tampering over-the-air updates [20], gaining code execution in privileged processes [20, 22] or mounting DoS attacks against apps [22]. On devices that are development-enabled (see Section 2.2), ChoiceJacking additionally grants a malicious charger shell access to the victim device. Among others, this allows an attacker to gain persistent code execution. On iOS, ChoiceJacking attacks gain access to the victim device's pictures and videos.

4 Attack Techniques

We identify three concrete attack techniques, T1 through T3, that exploit the fundamental flaws in JuiceJacking mitigations described in Section 3. In a novel hybrid form, they combine aspects of malicious USB hosts and malicious USB devices. They allow a malicious charger to bypass JuiceJacking mitigations for access to either PTP/MTP or ADB. While the described attack techniques focus on Android, T3 is platform-agnostic and also works on iOS. As described in Section 3, the malicious charger can use information obtained from initial communication to identify the most suitable attack for a victim device.

4.1 T1: Input Accessory

By sending a particular USB control request, the AOAP protocol described in Section 2.2 allows a USB host to put an attached Android device into a special accessory mode. In this accessory mode, the USB port is no longer available to other USB interfaces such as MTP. The USB host can then send subsequent USB control requests to register as a HID device and inject HID input events. However, we find that all investigated Android devices violate the AOAP specification by accepting AOAP HID messages while not in accessory mode. This means that a USB host can inject input events while initiating an ADB or MTP connection. A malicious charger can exploit this implementation flaw to autonomously complete all user confirmations for these data connections. Usually, AOAP accessories can only charge the device with 500 mA, which slows down charging noticeably [1]. For comparison, modern phone chargers supply at least 2400 mA. To work around this limitation, the attacker can carry out USB Power Delivery power negotiation before sending AOAP messages. The full steps for this attack technique are:

1. The unlocked victim device is connected to the charger.
2. The charger registers a HID input device through AOAP.
3. The malicious charger initiates an MTP or ADB data connection to trigger the user consent prompt.
4. The user consent prompt is shown on the device.
5. The charger autonomously accepts the user consent prompt by injecting the suitable HID events.

On some vendor-specific Android variants, further implementation flaws in AOAP can be exploited for gaining file access on locked devices. By triggering a fault in AOAP input event handling, it is possible to stealthily force re-initialization of the USB interface in MTP mode. Details about these attacks can be found in Section 6.

4.2 T2: Flooding Input Dispatcher

The second attack technique, T2, exploits a race condition in Android's input dispatcher for injecting input events while initiating a USB data connection. Android's input subsystem builds on its Linux counterpart. At the lowest level, Linux kernel drivers are responsible for interacting with connected input peripherals (e.g., USB HID devices) to obtain input events. The `InputReader` in the `InputManagerService` Android system service parses these Linux input protocol events and puts them into a queue for consumption by the input dispatcher. This queue acts as a buffer that ensures that input events are processed in order and are not lost even if the system is busy. We note that the Android input subsystem keeps input events in this queue even if the input device that generated them is no longer connected. Additionally, Android's input dispatcher intentionally serializes key events. It waits for all previous input events to have been fully processed before a key event is dispatched[4]. This means that a single process that performs overly complex logic in its key event handler will delay event dispatching for all other processes or global event handlers. We observe that such key event handlers are found in system processes or preinstalled applications on all evaluated Android devices. A malicious charger can exploit this by starting as a USB peripheral and flooding the event queue with a specially crafted sequence of key events. It then switches its USB interface to act as a USB host while the victim device is still busy dispatching the attacker's events. These events therefore accept user prompts for confirming the data connection to the malicious charger. The technique is illustrated in Figure 3. The exact steps are:

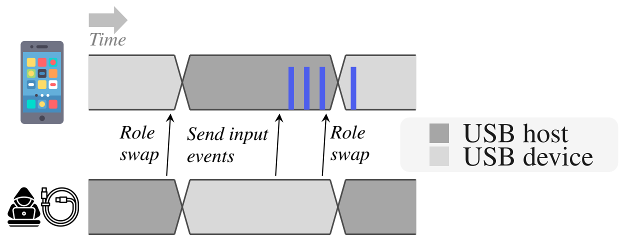


Figure 3: Race condition in Android's input dispatcher. A malicious charger can inject input events that are dispatched after its role swap to a USB host.

1. The unlocked device is connected to the charger.
2. At a suitable moment, the charger performs a USB PD Data Role Swap. The Android device now acts as a USB host, while the charger is a USB HID device.
3. The charger generates a large number of HID input events that are known to trigger complex handling logic and confirm the user prompt. Two options are possible for arranging events that delay processing (delayers) and those that confirm the user prompt (confirmers):

   (a) *Alternating arrangement*. By alternating delayers and confirmers, the user prompt is accepted as soon as it is shown.

[4] `InputDispatcher`: https://android.googlesource.com/platform/frameworks/native/+/refs/heads/main/services/inputflinger/dispatcher/InputDispatcher.cpp#2217

(b) *Sequential arrangement*. Filling the queue purely with delayers and only adding confirmers at the end ensures that confirmers do not interfere with the user interface before the user prompt is shown.

4. The charger performs a USB PD Data Role Swap. It is now the USB host, the mobile device is the USB device.
5. As the USB host, the charger triggers the user prompt.
6. The input generated previously confirms the user prompt.

4.3 T3: Bluetooth Input Device

In the third attack technique, T3, the malicious charger acts as a USB peripheral to autonomously pair a Bluetooth (BT) HID device to the victim device. Modern BT chips have very small footprints and can be easily fitted into maliciously modified chargers. The malicious charger can use this BT input device to confirm the data connection initiated by switching its USB interface into USB host mode. For pairing the BT input device to the victim device, the charger injects events for (1) making the device discoverable, and (2) confirming the pairing dialog. On Android, making a device discoverable simply requires opening the BT pairing screen in the system settings. The full steps for this attack technique through a PD-capable and BT-capable malicious charger are therefore:

1. The victim device is connected to the malicious charger. The device has its screen unlocked.
2. At a suitable moment, the charger performs a USB PD Data Role (DR) Swap. The mobile device now acts as a USB host, the charger acts as a USB input device.
3. The charger generates input to ensure that BT is enabled.
4. The charger navigates to the BT pairing screen in the system settings to make the mobile device discoverable.
5. The charger starts advertising as a BT input device.
6. By constantly scanning for newly discoverable Bluetooth devices, the charger identifies the BT device address of the mobile device and initiates pairing.
7. Through the USB input device, the charger accepts the Yes/No pairing dialog appearing on the mobile device. The Bluetooth input device is now connected.
8. The charger sends another USB PD DR Swap. It is now the USB host, and the mobile device is the USB device.
9. As the USB host, the charger initiates a data connection.
10. Through the Bluetooth input device, the charger confirms its own data connection on the mobile device.

5 Proof-Of-Concept Implementation

ChoiceJacking attacks are mounted from a malicious phone charger. Since we consider reverse-engineering and modifying existing commercial charger products out of scope for this work, we simulate such an environment using a custom printed circuit board (PCB) and a Raspberry Pi single-board computer (SBC). The PCB operates the hardware-level components needed for the attacks, while the Raspberry Pi SBC runs the attack logic and controls the PCB. The interaction between the components is shown in Figure 4.

Custom PCB. Attack techniques T2 and T3 described in Section 4 require a malicious charger capable of (1) operating as both USB device and USB host, (2) providing power to the attached mobile device in both modes, and (3) switching the USB data role of both the charger and mobile device. Technique T1 only requires the charger to send AOAP messages as a USB host. It is worth noting that T1, therefore, can be carried out from any commodity hardware that supports USB host mode (e.g., off-the-shelf computers or MCUs). To fulfill the requirement for all attack techniques, we designed a custom PCB around the Raspberry Pi RP2040 microcontroller unit (MCU). During operation, the PCB is connected to the USB PD power supply, the mobile device, and the Raspberry Pi SBC. The MCU controls all components on the board. The chip's native USB interface implements a USB HID keyboard and mouse, while a second bit-banged USB port allows serial communication to the external Raspberry Pi SBC. The PCB also contains a USB Power Delivery controller that acts as a DFP to the mobile device. A USB multiplexer connects the mobile device's USB data lines to either the MCU or the Raspberry Pi SBC.

Raspberry Pi. The Raspberry Pi 4 SBC in our implementation serves three purposes. First, it communicates with the MCU on the PCB to control its hardware-level functionality. Second, it acts as the USB host that establishes the data connection to the mobile device. Third, it operates as a Bluetooth HID device required for attack technique T3.

Attack Costs. Overall, the cost of our prototype is less than 100 USD. This means ChoiceJacking attacks can be mounted even by unsophisticated attackers. Although larger in physical dimensions than an off-the-shelf charger, the prototype can be hidden behind a wall, e.g., behind the enclosure of a charging station. Alternatively, the electronics on our PCB can further be miniaturized so that they even fit into the USB-C connector. This allows to form a malicious charger simply by attaching a malicious cable to an otherwise benign USB-C socket. A similar miniaturization has been accomplished in commercial products for BadUSB attacks[5].

6 Evaluation

Using the implementation described in Section 5, we evaluate the susceptibility of mobile devices from the most popular vendors to ChoiceJacking attacks. While we focus our evaluation on different Android devices, we also include a device running iOS, the other major mobile platform. Since every Android manufacturer ships a customized OS variant, different Android devices are affected in different ways by our attacks. This section demonstrates that despite the diversity of

[5]O.MG Cable: https://shop.hak5.org/products/omg-cable

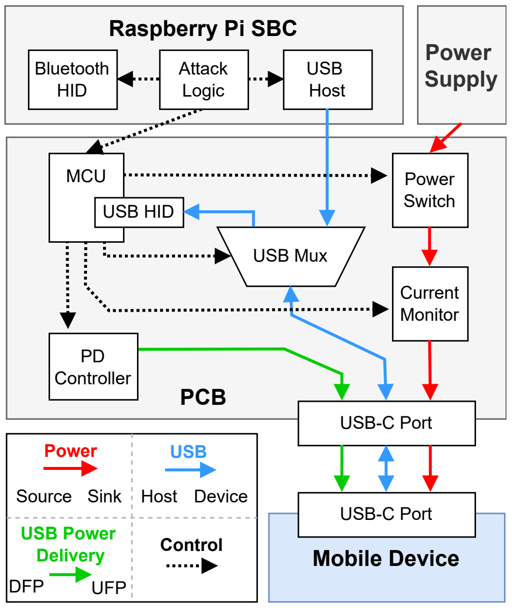


Figure 4: Our simulated malicious charger. Bidirectional USB lines represent the possibility for data role swaps.

tested devices, all are vulnerable to CHOICEJACKING attacks.

Our selection of test devices is shown in Table 1. It includes a recent device from each of the 5 Android vendors with the highest worldwide market share as of May 2024 as reported by Statcounter [38]. These are Samsung, Xiaomi, Oppo, Vivo and Huawei. We additionally extend our evaluation to a recent device from Apple, the leader in mobile device market share. For Samsung, the leading Android manufacturer, we select 4 devices to include a range from low-end to high-end devices of one manufacturer. We also select a recent Pixel phone from Google, as these run a largely unmodified AOSP build. Finally, we also select a device from Honor to represent the long tail of smaller device manufacturers. For all of the covered devices, we evaluate on a recent available OS release (either Android 12, 13, or 14, and iOS 17).

For every device, we evaluate all attack combinations, i.e., using any of attack techniques T1 to T3 from Section 4 to gain MTP or ADB access. Since PTP is a strict subset of MTP, it was not evaluated separately for devices that support both.

We provide timings for the fastest attack technique per Android device and target protocol (ADB, MTP) in Figure 5. For all attacks, we measure the duration in which the malicious charger visibly interacts with the user interface of the victim device. This entails triggering UI changes e.g., through injected input events or through USB state changes. We measure timings by counting frames in video recordings taken at 30 frames per second and full HD resolution. In the following, we evaluate concrete CHOICEJACKING attacks for each individual mobile device vendor.

6.1 Samsung

Samsung's OneUI Android variant by default acts as an MTP device when connected to a USB host. However, actual device storage contents only become visible once the user has confirmed a prompt displayed on the device following the MTP handshake from the host. The prompt can be confirmed by jointly pressing both hardware volume keys.

T1. Attack technique T1 succeeds on all Samsung devices to autonomously confirm Samsung's proprietary MTP prompt while the screen in unlocked. The attack takes 167 ms on the Galaxy A14, 133 ms on the Galaxy S20 FE, 267 ms on the Galaxy A33 and 300 ms on the Galaxy S23. For development-enabled devices, the ADB trust prompt can also be overcome via technique T1 within 233 ms, 200 ms, 233 ms and 267 ms on the Galaxy A14, S20 FE, A33 and S23, respectively.

T2. Attack technique T2 allows bypassing the MTP prompt. On Android 13 and earlier, volume key events delay the processing of subsequent input events and accept the user prompt. As a result, CHOICEJACKING attacks can accept the MTP user prompt on the Galaxy A14 and S20 FE even before it becomes visible. Using T2 for gaining MTP access takes 17.5 s on average over the 4 Samsung devices. On development-enabled Galaxy A14, S20 FE and A33 devices, T2 also allows confirming the ADB trust prompt in less than 11 s. We could not find a suitable delayer sequence for the ADB trust prompt on the Galaxy S23.

T3. Technique T3 gains MTP access on an unlocked or ADB access on an unlocked, development-enabled Samsung device within 24.3 s and 25.3 s on average, respectively.

Takeaway: All Samsung devices in our test set are susceptible to CHOICEJACKING attacks for file access and code execution in *under* 0.3 s.

6.2 Xiaomi

Xiaomi's MIUI Android variant displays a proprietary dialog for selecting the USB mode for all connections to a USB host longer than a second. Xiaomi also heavily modified the developer settings on their OS variant. Enabling USB debugging effectively lowers an Android device's security, as explained in Section 2.2. To ensure that the user is consciously making the choice to enable USB debugging, its activation normally requires user authentication. However, Xiaomi entirely removed the authentication step from this procedure in their Android variant. Enabling the developer settings menu simply requires repeatedly tapping on the build version field in the system settings. Once enabled, the developer settings menu can already be used e.g., to change the default USB mode. For further activating USB debugging (ADB), the user needs to wait 10 s and accept a warning prompt.

T1. On the Xiaomi 12, attack technique T1 allows bypassing the USB mode selection dialog while the screen is un-

Vendor	Device	OS	USB PD	Exploitable Attack Techniques		
				T1 Targets	T2 Targets	T3 Targets
Samsung	Galaxy A14	Android 13 (OneUI 5)	Yes	MTP, ADB	MTP, ADB	MTP, ADB
Samsung	Galaxy S20 FE	Android 13 (OneUI 5)	Yes	MTP, ADB	MTP, ADB	MTP, ADB
Samsung	Galaxy A33	Android 14 (OneUI 6)	Yes	MTP, ADB	MTP, ADB	MTP, ADB
Samsung	Galaxy S23	Android 14 (OneUI 6)	Yes	MTP, ADB	MTP	MTP, ADB
Xiaomi	12	Android 13 (MIUI 14)	Yes	MTP, ADB[a]	MTP, ADB[a]	MTP, ADB[a]
Vivo	Y36	Android 13 (Funtouch OS 13)	Yes (No DR swap)	MTP, ADB	-	-
Oppo	A58	Android 13 (ColorOS 13)	Yes	MTP[b], ADB	MTP	MTP
Huawei	nova 12i	Android 12 (EMUI 14)	Yes	MTP, ADB	MTP, ADB	MTP, ADB
Honor	90 Lite	Android 13 (MagicOS 7.1)	Yes	MTP[b], ADB	MTP	MTP
Google	Pixel 7a	Android 14	Yes	MTP, ADB	ADB	MTP, ADB
Apple	iPad Pro 2022	iOS 17.4.1 (iPadOS 17.4.1)	Yes	-	-	PTP

[a] ADB access can be gained even if the device is not development-enabled before the attack [b] Attack works on locked devices

Table 1: Tested devices and their susceptibility to CHOICEJACKING attacks based on our 3 different attack techniques.

locked to access MTP functionality. The attack takes 967 ms. If the device is development-enabled (USB debugging is activated), the same technique can also be used for confirming the ADB trust prompt in 933 ms. It is worth noting that attack technique T1 can also be used for gaining ADB access on an unlocked Xiaomi device that is not development-enabled. This works by using the AOAP input device to carry out the UI interaction for enabling development settings and USB debugging, before confirming the ADB trust dialog. Overall, this attack takes 36 s on the Xiaomi 12. The attack on a non-development-enabled device is comparatively slow, because it involves additional menu navigation and idle time caused by the warning prompt delay described above.

T2. Attack technique T2 is successful at gaining MTP or ADB access on an unlocked device. For MTP on an unlocked device, the attack takes 13 s and uses volume key events as delayers in a sequential arrangement. For ADB on a development-enabled and non-development-enabled device, the attack takes 27 s and 67 s, respectively. The difference in these timings can again be attributed to the additional steps needed on non-development-enabled devices.

T3. Attack technique T3 allows obtaining MTP or ADB access on an unlocked device. Reaching MTP access on an unlocked device takes 29 s. ADB access can also be reached on a development-enabled or non-development-enabled device within 23 s and 55 s, respectively. The attack on the non-development-enabled device is slowed down by the additional steps described above.

Takeaway: On the evaluated Xiaomi device, CHOICEJACKING gains file access or code execution in under 1 s. Due to Xiaomi's OS customizations, CHOICEJACKING attacks can gain code execution on unlocked *non-development-enabled* devices.

6.3 Vivo

Vivo's Funtouch OS Android variant does not display a user prompt for USB mode selection. Instead, the user can change USB mode by tapping on a particular system notification. Since this notification is always shown above all other notifications, it can be reliably selected and opened through keyboard inputs. It is worth noting that the entry-level Vivo Y36 device in our test set does not support data role switches through USB PD.

T1. Attack technique T1 on the Vivo Y36 allows to open the USB mode settings and enable MTP mode. This attack takes 1267 ms. T1 can also gain ADB access on a development-enabled device within 367 ms.

T2 & T3. As the tested Vivo device only partially supports USB PD, it is not susceptible to attack techniques T2 and T3.

Takeaway: CHOICEJACKING attacks on the evaluated Vivo device gain file access or code execution in under 1.3 s.

6.4 Oppo

Oppo's ColorOS displays a user prompt for selecting the USB mode. The prompt is triggered by the USB host querying the device's USB descriptors immediately following USB cable attachment. It is worth noting that ColorOS disables USB debugging on every USB disconnect or PD data role swap. The attack cannot re-enable USB debugging, because the system settings UI does not accept keyboard input for the corresponding UI switch. Mouse input cannot be used because the screen position of the switch depends on the device configuration and screen orientation and is, therefore, impossible to predict.

T1. Attack technique T1 succeeds in gaining MTP access on the Oppo A58 both while the screen is locked and unlocked. On an unlocked screen, AOAP HID events are injected to autonomously select MTP in the USB mode user prompt within 667 ms. Gaining MTP access while the screen is locked can be accomplished by injecting an input event immediately after registering the HID device through AOAP. Because the OS takes some time to set up the registered HID device for use, an error in the AOAP implementation is raised. As part of error handling, the Android Auto service on the device (part of the AOAP implementation) interferes with Oppo's USB stack, resulting in MTP mode being enabled after about 35 s. Technique T1 also gains ADB access on unlocked development-enabled devices within 2233 ms.

T2. Attack technique T2 can be exploited to bypass the USB mode user prompt for gaining MTP access on an unlocked Oppo device. The attack navigates to the launcher by injecting a home key press and uses the key event handler in the launcher application as a delayer in an alternating arrangement. Overall, the attack takes 14 s.

T3. Attack technique T3 allows access to MTP on the Oppo A58 within 27 s. Due to USB debugging being disabled automatically, we failed to exploit T2 and T3 for ADB access.

Takeaway: CHOICEJACKING attacks gain file access and code execution on the evaluated Oppo device in under 2.3 s. A bug in the implementation of accessory mode allows extracting files *while the screen is locked*.

6.5 Huawei

Huawei's EMUI Android variant displays a proprietary USB mode dialog for connections to a USB host that last longer than 2 seconds. The dialog is triggered by querying the device's USB descriptors. Among others, it offers the possibility to enable MTP access. It is worth noting that EMUI is not a licensed Android variant, so it does not undergo Google's rigourous compatibility testing. As a result, the OS's accessory mode is not fully compliant with Google's specification.

T1. Despite Huawei's AOAP implementation not being fully specification-compliant, EMUI accepts AOAP HID messages when not in accessory mode. As a result, the Huawei nova 12i is susceptible to CHOICEJACKING attacks for gaining MTP access while the screen is unlocked. This attack takes 200 ms to complete. Attack technique T1 can also be used for autonomously accepting the ADB trust dialog on a development-enabled device in 1933 ms.

T2. This technique gains MTP access on an unlocked device and ADB access on an unlocked development-enabled device. These attacks take 20 s and 23 s for MTP and ADB, respectively. Volume key events are used as delayers in a sequential arrangement for MTP and a hybrid one for ADB.

T3. Via a Bluetooth HID connection, MTP and ADB access can be gained in 25 s and 27 s, respectively.

Takeaway: The Huawei device is susceptible to CHOICEJACKING attacks that gain file access and code execution in under 1.9 s.

6.6 Honor

In its MagicOS Android variant, Honor implemented a proprietary user prompt for choosing the USB mode. The prompt is triggered by querying the USB descriptors immediately following connection establishment as a USB device. Although the Android device presents as an MTP device to the host immediately, the actual storage contents are only exposed once the user selects MTP in this USB mode prompt. Like Oppo's ColorOS, we observe that MagicOS automatically disables USB debugging after USB disconnects or PD data role swaps. Due to the same limitations in the system settings UI, it also cannot be re-enabled as part of the attack.

T1. The Android Open Accessory Protocol (AOAP) can be exploited to reach MTP access on the Honor 90 Lite independent of the screen lock state. While the screen is unlocked, the injected AOAP HID events select MTP in the displayed USB mode prompt. The attack completes in 367 ms. If the screen is locked, the same implementation flaw as found on the Oppo device can be exploited. It takes about 30 s for MTP access to be granted. Technique T1 also gains ADB access on an unlocked development-enabled device within 1400 ms.

T2. Attack technique T2 can be exploited to bypass the USB mode user prompt to gain MTP access. Our attack uses the Google assistant shortcut as a delayer in an alternating arrangement. The attack briefly interrupts the device's power supply to trigger the USB mode dialog after the USB PD data role swap. Overall, the attack takes 28 s.

T3. Technique T3 allows gaining MTP file access. The Bluetooth pairing screen can be reached through the search bar in Honor's app launcher. Due to the USB debugging option being reset automatically, T2 and T3 cannot be exploited for ADB access.

Takeaway: The evaluated Honor device is susceptible to CHOICEJACKING attacks for file access and code execution within 1.4 s. A bug in the implementation of accessory mode allows extracting files *while the screen is locked*.

6.7 Google

Devices by manufacturer Google run unmodified AOSP Android. As a result, these devices do not display a user prompt when a connection to a USB host is established. To activate MTP mode, the user needs to open the USB mode selection screen either through the system settings or through a system notification and select the corresponding option. In contrast to Vivo's customized OS, AOSP does not permit keyboard

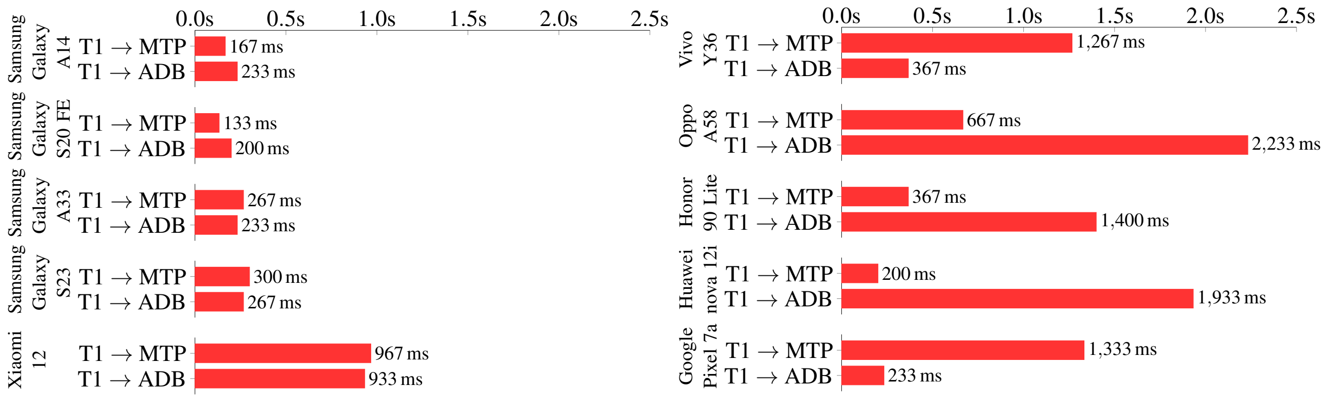


Figure 5: Timings for CHOICEJACKING attacks on Android devices. In the best case, MTP file access is gained in 133 ms.

input to reliably select the notification if other notifications are shown as well, because their order cannot be predicted.

T1. The Pixel 7a is susceptible to input events injected through AOAP for autonomously activating MTP within 1333 ms. Since the exact position of the USB mode system notification among other notifications (e.g., from emails) is unpredictable, the attack launches the USB mode selection screen through the system settings. Technique T1 can also be used for confirming the ADB trust dialog within 233 ms.

T2. The race condition in Android's `InputDispatcher` can be exploited for gaining ADB access on a development-enabled device within 10 s. Our attack opens the preinstalled calculator app using a global key shortcut, then uses number key presses as delayers in an alternating arrangement.

T3. Attack technique T3 gains MTP access on an unlocked device and ADB access on an unlocked development-enabled device. Both attacks take 19 s.

Takeaway: On the Google Pixel 7a, CHOICEJACKING attacks can extract files or gain code execution in 1.3 s.

6.8 Apple

When attached to a USB host, iOS exposes a PTP interface, which allows exchanging photos and videos. Initiating a PTP connection displays a user prompt on the mobile device, which must be confirmed before the host can access any photos or videos. The BT pairing screen can be reached through system-wide Spotlight search. In addition to supporting PTP data connections to a computer, iOS devices can also expose a development interface through USB. However, establishing trust in a computer requires user authentication even if the device is already development-enabled. As a result of requiring user authentication for establishing trust in a computer, CHOICEJACKING attacks cannot access the development interface on iOS.

T3. As attack techniques T1 and T2 are specific to Android, we restrict our evaluation to T3. It successfully gains PTP access on an unlocked device within 23 s.

Takeaway: On the iPad Pro 2022, CHOICEJACKING attacks can extract photos and videos within 23 s.

6.9 Summary of Results

Our evaluation highlights the scope of CHOICEJACKING attacks. All tested devices across platforms and manufacturers are susceptible. On devices from Oppo and Honor (18 % of all tested devices), it is possible to exploit a flaw in the accessory mode implementation to gain file access on locked devices. On all Android devices, our attacks can gain MTP file access in under 1.4 s, for a median of 333 ms over all evaluated Android devices. Code execution through ADB on all unlocked development-enabled Android devices can be reached in less than 1.3 s, for a median of 316 ms. On devices from Xiaomi, it is possible to gain code execution through ADB on an unlocked but non-development-enabled device.

Attack technique T1 proved best in terms of susceptible devices and attack speed. It succeeds on all evaluated Android devices. In the best case (Samsung Galaxy S20 FE), it only takes 133 ms, which is less than half the duration of a human blink [17]. At this speed, the attack only leaves a short flickering on the screen that may remain unnoticed. The median attack duration for T1 was 334 ms. Technique T3 exhibits the same success rate across platforms (91 %), but performs slower at a median of 24.5 s. This increase in duration can be attributed to the BT pairing process. At a median duration of 13 s, technique T2 only succeeded on 8 devices for MTP and 6 devices for ADB access. 9 devices (81 %) from 6 manufacturers (75 %) are susceptible to all 3 attack techniques for at least some protocol.

The results for the 4 Samsung devices indicate that attacks based on T1 work almost unchanged for different devices from the same vendor. Although the devices cover a range from low-end to high-end and span two major OS versions, the identical attack sequence works on all of them. This finding indicates that an attacker can target many different device models through a small number of different attack sequences.

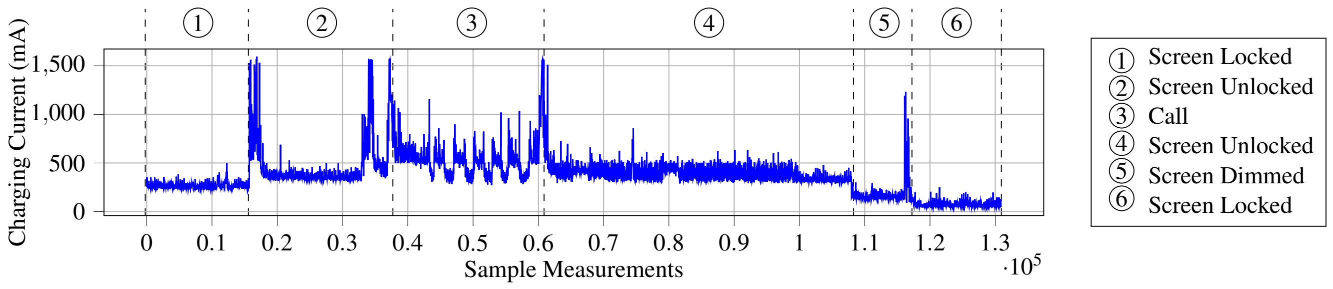


Figure 6: Power trace for phone call cycle on Xiaomi 12

7 Power Line Side-Channel

This section presents a power line side-channel (PLSC) for detecting a suitable moment to perform ChoiceJacking attacks, i.e., when the victim device is left unobserved.

The PLSC exploits the fact that any operation carried out on a mobile device creates unique fluctuations in its power consumption. A malicious charger that measures the charging device's power consumption can, therefore, learn details about its activity (e.g., user interactions). Prior work has demonstrated that PLSCs reveal various information, e.g., button-accurate screen tap locations [7] or web browsing activity [52]. The work by Cronin et al. [7] is particularly relevant for our PLSC since it infers touch locations from current fluctuations caused by screen content changes as part of visual feedback. Their findings already show that such PLSCs exist on a wide range of mobile devices and are invariant to the battery charging level and various device configurations such as screen brightness. Screen content changes not only indicate user interaction but also allow learning about the user's focus of attention. Therefore, such measurements are a basis for detecting suitable moments for ChoiceJacking attacks.

7.1 Desired Detection Scenario

For operating stealthily, an attacker is interested in moments when a user does not notice the UI state changes incurred by ChoiceJacking attacks on an unlocked device. We observe that there are scenarios in which mobile devices are unlocked, yet the user is unable to observe the screen (see Section 8.1). Phone calls are particularly illustrative examples of such scenarios, so we use them for our proof-of-concept PLSC. During phone conversations, users typically hold their mobile device close to their face, so they can communicate through the integrated microphone and speaker. To conserve energy and prevent unintended touch events, most devices include proximity sensors that detect these events and turn off the screen. Crucially, while the screen is turned off, we notice that external input devices can still fully interact with the UI in this scenario. Furthermore, mobile phone calls usually take about two orders of magnitude longer than the 1.4 seconds ChoiceJacking attacks need at worst for gaining *file access* [31]. We therefore aim for a PLSC that detects when the screen is turned off during a phone call. We extend the findings of previous works [7, 52] and show that a PLSC can be used to detect this scenario.

7.2 Power Trace Analysis

As an initial indication for the existence of a suitable PLSC, we collect power traces using the malicious charger prototype described in Section 5. We base our experiments on the Xiaomi 12 from the test set described in Section 6, expecting similar results for other phones (as per prior work [7,52]). The Xiaomi 12 was chosen because of the possibility for ChoiceJacking attacks to gain code execution *on non-development-enabled devices*. This represents the maximum attack gain among all evaluated ChoiceJacking attacks, yet it requires the device to be unattended for the longest time (36 s for technique T1). In line with prior work [7, 52], we deduce power from measuring current. The power trace collected at 1 kHz from the Xiaomi 12 is shown in Figure 6, illustrating the device's power consumption before, during and after a call. ① and ② show the consumption in the locked and unlocked states respectively before the call, while ⑤ and ④ show the consumption after the call. In the transition from ② and ③, the power traces indicate the possibility to distinguish the start of the call from other events, such as screen lock and unlock. During the call in ③, we repeatedly turned the screen off and on by triggering the device's proximity sensor. These events can be observed as visually distinguishable fluctuations in current draw between 600 mA and 400 mA in ③.

7.3 PLSC Evaluation

We implement and evaluate a neural network for automatically detecting the desired scenario from a malicious charger. This entails (1) detecting phone calls, i.e., call start and end events, as well as (2) screen on and off events. To demonstrate that the approach by Cronin et al. [7] can be adapted for detecting calls, we construct a similar one-dimensional convolutional neural network (CNN) classifier. The model distinguishes call start and end events from other events that commonly occur during mobile device usage. We consider

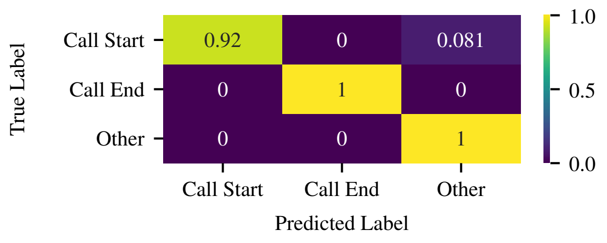


Figure 7: Confusion matrix of our PLSC CNN model.

16 other common events (e.g., web browsing, doing calculations, messaging, ...). For training the model, we collected 20 power traces for each of the 18 events of 5 seconds each (360 traces total). Evaluating the resulting model on a test set of 10 fresh traces per event (180 traces total) yields an accuracy of 92.78 %. The resulting confusion matrix can be found in Figure 7. While our model does not report any false negatives for call start or end events, some other events are reported as call ends. The main goal for the PLSC is to avoid launching CHOICEJACKING attacks while the user might notice them. False negatives (not starting the attack although it would be possible) at the identified prevalence are therefore negligible for our application. We conclude that the CNN allows determining when the user is in a phone call. For further determining when the screen is turned off during the phone call, a simple threshold comparator can be used, as observable from the trace in Figure 6.

8 Discussion and Related Work

In this section, we elaborate attack scenarios and techniques, discuss mitigations to CHOICEJACKING attacks and introduce related literature.

8.1 Suitable Attack Scenarios

While some CHOICEJACKING attacks work on locked devices or succeed within a matter of milliseconds, others require an unlocked device and cause on-screen artifacts for a duration of multiple seconds. These attacks are effective in any charging scenario where the unlocked phone screen is outside the user's focus of attention. A very illustrative example of such a scenario can be found in phone calls, as discussed in Section 7. Similarly suitable are scenarios where the user listens to a music video while the device charges from a rented power source (e.g., a hotel room's USB wall socket). It is worth noting that Youtube, which accounts for almost half of all music streaming [11], does not allow locked-screen playback for non-paying customers, and users are typically immersed in other tasks while listening to music [11]. Mobile devices are also commonly attached to rented chargers (power bank, car, etc.) while using navigation apps, where the user's attention is focused on the surroundings rather than the screen. These scenarios or further variations thereof are experienced by most mobile device users on a daily basis.

8.2 Rationale for Multiple Attack Techniques

Our paper presents three attack techniques (T1 through T3), which might appear redundant at first glance. However, presenting several distinct attack techniques serves two important missions. First, it expands the scope of CHOICEJACKING attacks, both in terms of affected devices and platforms. Attack technique T3 is relatively slow due to the BT pairing procedure, but it succeeds on both iOS and Android. Attack technique T1 performs fastest, but it requires support for AOAP, which can be missing in unlicensed AOSP-derived platforms. Attack technique T2 exploits a race condition in a core OS component, so it is likely to succeed even on heavily customized Android variants that lack support for AOAP. Second, presenting multiple techniques illustrates that the attacks are rooted in a systemic lack of considering the security repercussions of dual-role USB connectivity. It therefore intends to provoke a more wholistic way of thinking about dual-role USB and connectivity adversaries in mobile platforms.

8.3 Existing Mitigations

A number of mitigations have been suggested for JuiceJacking attacks that may also be effective against CHOICEJACKING. Additionally, mobile OSs already integrate some means to limit the damage an attacker with control over the unlocked device's UI may do. All of these existing mitigations suffer from limitations to their effectiveness in practical scenarios.

USB Data Blockers. A frequent suggestion for mitigating malicious chargers are USB data blockers, which break USB data lines. However, this mitigation assumes that the user is aware of the potential for attacks, rendering it ineffective for the average end user. Data blockers also interfere with modern power negotiation schemes, thereby degrading charge speed.

User Authentication for Security-critical Functions. Android and iOS require user authentication for security-critical functionality such as enabling the USB debugging interface. While this is an effective mitigation against malicious chargers carrying out these operations, there are flaws in Android's implementation. When asking for user authentication, the UI does not provide any clue about the operation it is needed for. To the average user, the authentication screen might look like an innocuous screen unlock, which they routinely authenticate to. A malicious charger can exploit this weakness by triggering the authentication prompt and simply waiting for the user to confirm it.

Lockdown Mode. Both Android (since Android 15) and iOS (since iOS 16) include a Lockdown mode that can be activated to thwart immediate danger in insecure environments. On both OSs, enabling Lockdown mode shuts down USB lines entirely while the lock screen is employed, so that any kind of USB communication is disabled. However, USB protections are restricted to the lock screen. As soon as the device is unlocked, USB connectivity is re-enabled. Most CHOICEJACKING attacks operate while the device is unlocked, so that Lockdown mode is ineffective against them. Additionally, this mitigation needs to be enabled manually, and, therefore, requires the user's awareness of potential attacks.

8.4 Suggested Mitigation

We propose user prompts for all kinds of USB access (host, device, accessory mode) as a mitigation to both our novel CHOICEJACKING and existing JuiceJacking attacks. CHOICEJACKING attacks exploit the fact that mobile OSs by default trust USB and accessory input devices, although these can be used for elevating the attacker's privileges by interacting with the UI to enable data connections. Since there may be many ways to elevate privileges beyond the flaws we highlighted, we argue that the default trust in USB input devices and accessories needs to be cut. This can be realized by introducing prompts that ask for explicit user consent before an input device or accessory is allowed to interact with the system. Once the USB Type-C Authentication Specification [46] has been widely adopted, the user consent may be displayed for each device upon first attachment and saved for subsequent connections. A similar solution was proposed by Tian et al. [42] and has been adopted in desktop OSs, e.g., in macOS [2].

8.5 Related Work

This section briefly surveys existing work on other USB-based and general connectivity threats.

USB Threats. Prior works have discovered numerous other threats related to USB connectivity on mobile devices. After JuiceJacking and related attacks [16, 19] had been mitigated, several works exploited USB peripherals to extract screen contents [23, 24] or inject voice commands [50]. Tian et al. [43], Imtiaz et al. [13] and Pereira et al. [34] shed light on the consequences of vendor customizations to mobile devices' USB interfaces. They uncover possibilities for flashing firmware files or extracting user data through AT commands. Recent works by Kim et al. [14, 15], Wu et al. [51] and Peng et al. [33] present fuzzers for uncovering memory safety issues in USB and USB PD stacks. In contrast to the design flaws our work focuses on, all these works identify implementation issues such as memory safety issues and logic bugs. Su et al. [41] and Dumitru et al. [8] exploit physical properties of USB to observe or inject off-path communication. Finally, Tischer et al. [45] and Meng et al. [24] present user studies showing end users are unaware of USB-related threats.

Exploiting Mobile Device Connectivity. Design and implementation weaknesses have recently been found in various other communication interfaces of mobile devices. In 2023, Marc Newlin [26] showed a family of attacks that allows bypassing Bluetooth pairing authentication to inject input events on Android phones. Vanhoef et al. [48, 49] and Schepers et al. [36] have shown multiple attacks on WPA protocols that allow intercepting and hijacking Wifi communication. Cellular standards GSM [4] and UMTS [25] have long been known to allow eavesdropping. Stute et al. [39, 40] proposed multiple eavesdropping attacks on Apple's proprietary wireless protocols. Fahl et al. [9], Oltrogge et al. [32] and others reveal flaws in mobile apps' use of TLS, allowing Man-In-The-Middle attacks on server communication. None of these interfaces allow arbitrary file access as possible over USB.

Power Side-Channels On Mobile Devices. Wired, wireless and standalone power side-channels (PSC) for mobile devices are known in literature. Cronin et al. [7] exploit a PSC attack that allows a malicious charger to infer unlock pins. They detect button-accurate screen presses through the effects of visual button tap feedback on the screen's power consumption. Yang et al. [52] and Wang et al. [50] show that PSCs allow inferring a user's browsing activity and loudspeaker output. Genkin et al. [10] use the USB PSC to extract ECDSA keys from mobile devices. Ni et al. [27] and Oberhuber et al. [30] demonstrate that many of these PSCs can also be exploited indirectly, from neighbouring devices in multiport chargers or from software. La Cour et al. [18] prove that website fingerprinting is also possible through a PSC from a wireless charger. Ni et al. [28] further improve on this, identifying the running app and entered keystrokes of a mobile device charging from a wireless power bank.

9 Conclusion

Since user confirmations for USB data connections were introduced on mobile platforms, their effectiveness against JuiceJacking attacks remained unchallenged. In this paper, we observed that the design of this countermeasure is based on the flawed assumption that a USB host cannot inject input events. We presented a novel attack family, CHOICEJACKING, and three concrete attack techniques that break this assumption on Android and iOS. They allow bypassing JuiceJacking mitigations and gaining file access or code execution through a malicious charger. We constructed a prototype of a malicious charger to evaluate CHOICEJACKING attacks on 11 recent mobile devices from 8 manufacturers. Our evaluation showed that CHOICEJACKING attacks succeed across platforms and vendors and can gain USB-based file access in as little as 133 milliseconds in the best case. While most attacks require an unlocked device, we demonstrated attacks for two manufacturers that work while the screen is locked. Finally, we presented a power line side-channel that can be used for detecting suitable attack moments. Most affected manufacturers have already acknowledged the threat posed by CHOICEJACKING attacks. We are hopeful they will all integrate the countermeasures we suggest, which will lead to a lasting improvement to USB security in the mobile landscape.

Acknowledgements

We are grateful to Jakob Heher and Stefan More for their valuable ideas, feedback and encouragement, in particular during the early stages of this project. We also extend our thanks to the anonymous reviewers and our shepherd for their time, effort, and insightful contributions, which helped bring this work to publication. This project has received funding from the Austrian Research Promotion Agency (FFG) via the SEIZE project (FFG grant number 888087). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

Ethics Considerations

We responsibly disclosed all identified vulnerabilities to affected vendors. In addition to the 16 Android issues mentioned in Section 1, we also disclosed our findings regarding susceptibility of iOS devices to CHOICEJACKING attacks to Apple. We filed our reports between June and July of 2024, and have since been assisting vendors in reproducing the issues on their side and integrating suitable mitigations. Samsung assigned us moderate severity CVE-2024-20900 for the attack principle and rolled out first improvements to their USB mode selection implementation. Apple added user authentication prompts for USB connections in iOS 17.5, but is yet to integrate this mitigation into iOS 18. Xiaomi, Huawei, Vivo and Honor confirmed our findings and are still working on corresponding patches to their Android variants. Google assigned us high severity CVE-2024-43085 for the underlying flaws that caused MTP to be exposed on locked Oppo and Honor devices. A corresponding fix has been rolled out in the November 2024 Android Security Bulletin (ASB). They are also working on fixes for the reported design flaws in upstream AOSP. All evaluations were carried out in a lab environment, on dedicated research devices. We anticipate that the publication of our work will positively impact USB security of mobile platforms and help raise user awareness for USB-based threats.

Open Science

We provide all artifacts for our paper to the public[6]. The artifact package includes our prototype charger, the corresponding firmware, the Python framework for controlling it through USB, the Python code for all attacks and the PLSC. Additionally, we publish video recordings demonstrating our attacks on 11 different devices (10 Android devices and 1 iOS device), which we used for measuring attack timings.

[6] https://zenodo.org/records/14726532

References

[1] Android Open Source Project, "Android Open Accessory 2.0," 2012. [Online]. Available: https://android.googlesource.com/platform/docs/source.android.com/+/03fbc41/src/tech/accessories/aoap/aoa2.md

[2] Apple, Inc, "macOS Ventura 13 Release Notes: Accessory Security," 2022. [Online]. Available: https://developer.apple.com/documentation/macos-release-notes/macos-13-release-notes#Accessory-Security

[3] Apple Inc, Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, Renesas, STMicroelectronics, and Texas Instruments, "USB Power Delivery Specification Revision 2.0," 2017. [Online]. Available: https://www.usb.org/document-library/usb-power-delivery

[4] E. Barkan, E. Biham, and N. Keller, "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication," in *CRYPTO*, 2003.

[5] A. Bates, R. Leonard, H. Pruse, D. Lowd, and K. R. B. Butler, "Leveraging USB to establish host identity using commodity devices," in *NDSS*, 2014.

[6] Compaq, Digital Equipment Corporation, IBM PC Company, Intel, Microsoft, NEC, Northern Telecom, "Universal Serial Bus Specification 1.0," 1996. [Online]. Available: https://web.archive.org/web/20180130144424/https://fl.hw.cz/docs/usb/usb10doc.pdf

[7] P. Cronin, X. Gao, C. Yang, and H. Wang, "Charger-Surfing: Exploiting a power line Side-Channel for smartphone information leakage," in *USENIX Security*, 2021.

[8] R. Dumitru, D. Genkin, A. Wabnitz, and Y. Yarom, "The impostor among US(B): Off-Path injection attacks on USB communications," in *USENIX Security*, 2023.

[9] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why eve and mallory love android: an analysis of android ssl (in)security," in *CCS*, 2012.

[10] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "Ecdsa key extraction from mobile devices via nonintrusive physical side channels," in *CCS*, 2016.

[11] International Federation of the Phonographic Industry, "Music Consumer Insight Report," 2018. [Online]. Available: https://www.ifpi.org/wp-content/uploads/2020/07/091018_Music-Consumer-Insight-Report-2018.pdf

[12] Y. Jiang, X. Ji, K. Wang, C. Yan, R. Mitev, A. Sadeghi, and W. Xu, "WIGHT: wired ghost touch attack on capacitive touchscreens," in *S&P*, 2022.

[13] I. Karim, F. Cicala, S. R. Hussain, O. Chowdhury, and E. Bertino, "Opening pandora's box through atfuzzer: dynamic analysis of AT interface for android smartphones," in *ACSAC*, 2019.

[14] K. Kim, S. Kim, K. R. B. Butler, A. Bianchi, R. Kennell, and D. J. Tian, "Fuzz the power: Dual-role state guided black-box fuzzing for USB power delivery," in *USENIX Security*, 2023.

[15] K. Kim, T. Kim, E. Warraich, B. Lee, K. R. B. Butler, A. Bianchi, and D. Jing Tian, "Fuzzusb: Hybrid stateful fuzzing of usb gadget stacks," in *S&P*, 2022.

[16] B. Krebs, "Beware of Juice-Jacking," 2011. [Online]. Available: https://krebsonsecurity.com/2011/08/beware-of-juice-jacking/

[17] K.-A. Kwon, R. J. Shipley, M. Edirisinghe, D. G. Ezra, G. Rose, S. M. Best, and R. E. Cameron, "High-speed camera characterization of voluntary eye blinking kinematics," *J R Soc Interface*, vol. 10, no. 85, 2013.

[18] A. S. La Cour, K. K. Afridi, and G. E. Suh, "Wireless Charging Power Side-Channel Attacks," ser. CCS, 2021.

[19] B. Lau, Y. Jang, C. Song, T. Wang, P. H. Chung, and P. Royal, "Mactans: Injecting Malware into iOS Devices via Malicious Chargers," in *Black Hat USA*, 2013.

[20] Y.-T. Lee, W. Enck, H. Chen, H. Vijayakumar, N. Li, Z. Qian, D. Wang, G. Petracca, and T. Jaeger, "PolyScope: Multi-Policy access control analysis to compute authorized attack operations in android systems," in *USENIX Security*, 2021.

[21] L. Maar, F. Draschbacher, L. Lamster, and S. Mangard, "Defects-in-Depth: Analyzing the Integration of Effective Defenses against One-Day Exploits in Android Kernels," in *USENIX Security*, 2024.

[22] S. Makkaveev, "Man-in-the-Disk:Android Apps Exposed via External Storage," 2019. [Online]. Available: https://research.checkpoint.com/2018/androids-man-in-the-disk/

[23] W. Meng, F. Fei, W. Li, and M. H. Au, "Harvesting smartphone privacy through enhanced juice filming charging attacks," in *ISC*, ser. Lecture Notes in Computer Science, vol. 10599, 2017.

[24] W. Meng, L. W. Hao, M. S. Ramanujam, and S. P. T. Krishnan, "Juicecaster: Towards automatic juice filming attacks on smartphones," *J. Netw. Comput. Appl.*, vol. 68, pp. 201–212, 2016.

[25] U. Meyer and S. Wetzel, "A man-in-the-middle attack on umts," in *Proceedings of the 3rd ACM Workshop on Wireless Security*, 2004.

[26] M. Newlin, "Hi, My Name Is Keyboard," 2023. [Online]. Available: https://github.com/skysafe/reblog/tree/main/cve-2023-45866#hi-my-name-is-keyboard

[27] T. Ni, Y. Chen, W. Xu, L. Xue, and Q. Zhao, "Xporter: A study of the multi-port charger security on privacy leakage and voice injection," in *MobiCom*, 2023.

[28] T. Ni, J. Li, X. Zhang, C. Zuo, W. Wang, W. Xu, X. Luo, and Q. Zhao, *Exploiting Contactless Side Channels in Wireless Charging Power Banks for User Privacy Inference via Few-shot Learning*, 2023.

[29] K. Nohl and J. Lell, "BadUSB - On Accessories That Turn Evil," in *Black Hat USA*, 2014.

[30] M. Oberhuber, M. Unterguggenberger, L. Maar, A. Kogler, and S. Mangard, "Power-Related Side-Channel Attacks using the Android Sensor Framework," in *NDSS*, 2025.

[31] ofcom, "Mobile Matters: Using crowdsourced data to assess people's experience of using mobile networks," 2021. [Online]. Available: https://www.ofcom.org.uk/siteassets/resources/documents/research-and-data/telecoms-research/mobile-matters/2021/mobile-matters-2021-report.pdf

[32] M. Oltrogge, N. Huaman, S. Amft, Y. Acar, M. Backes, and S. Fahl, "Why eve and mallory still love android: Revisiting TLS (In)Security in android applications," in *USENIX Security*, 2021.

[33] H. Peng and M. Payer, "USBFuzz: A framework for fuzzing USB drivers by device emulation," in *USENIX Security*, 2020.

[34] A. Pereira, M. E. Correia, and P. Brandão, "USB connection vulnerabilities on android smartphones: Default and vendors' customizations," in *Communications and Multimedia Security*, ser. Lecture Notes in Computer Science, vol. 8735, 2014, pp. 19–32.

[35] A. Possemato, S. Aonzo, D. Balzarotti, and Y. Fratantonio, "Trust, but verify: A longitudinal analysis of android oem compliance and customization," in *S&P*, 2021.

[36] D. Schepers, A. Ranganathan, and M. Vanhoef, "Framing frames: Bypassing Wi-Fi encryption by manipulating transmit queues," in *USENIX Security*, 2023.

[37] R. Spolaor, H. Liu, F. Turrin, M. Conti, and X. Cheng, "Plug and Power: Fingerprinting USB Powered Peripherals via Power Side-channel," in *IEEE INFOCOM*, 2023.

[38] statcounter GlobalStats, "Mobile Vendor Market Share Worldwide - May 2024," 2024. [Online]. Available: https://gs.statcounter.com/vendor-market-share/mobile

[39] M. Stute, A. Heinrich, J. Lorenz, and M. Hollick, "Disrupting continuity of apple?s wireless ecosystem security: New tracking, dos, and mitm attacks on ios and macos through bluetooth low energy, awdl, and wi-fi," in *USENIX Security*, 2022.

[40] M. Stute, S. Narain, A. Mariotto, A. Heinrich, D. Kreitschmann, G. Noubir, and M. Hollick, "A billion open interfaces for eve and mallory: MitM, DoS, and tracking attacks on iOS and macOS through apple wireless direct link," in *USENIX Security*, 2019.

[41] Y. Su, D. Genkin, D. Ranasinghe, and Y. Yarom, "USB snooping made easy: Crosstalk leakage attacks on USB hubs," in *USENIX Security*, 2017.

[42] D. J. Tian, A. Bates, and K. Butler, "Defending against malicious usb firmware with goodusb," in *ACSAC*, 2015.

[43] D. J. Tian, G. Hernandez, J. I. Choi, V. Frost, C. Raules, P. Traynor, H. Vijayakumar, L. Harrison, A. Rahmati, M. Grace, and K. R. B. Butler, "ATtention spanned: Comprehensive vulnerability analysis of AT commands within the android ecosystem," in *USENIX Security*, 2018.

[44] J. D. Tian, N. Scaife, D. Kumar, M. D. Bailey, A. Bates, and K. R. B. Butler, "Sok: "plug & pray" today - understanding USB insecurity in versions 1 through C," in *S&P*, 2018.

[45] M. Tischer, Z. Durumeric, S. Foster, S. Duan, A. Mori, E. Bursztein, and M. Bailey, "Users Really Do Plug in USB Drives They Find," in *S&P*, 2016.

[46] USB 3.0 Promoter Group, "Universal Serial Bus Type-C Authentication Specification," 2019. [Online]. Available: https://www.usb.org/document-library/usb-authentication-specification-rev-10-ecn-and-errata-through-january-7-2019

[47] USB3.0 Promoter Group, "Universal Serial Bus Type-C Cable and Connector Specification Release 2.3," 2023. [Online]. Available: https://www.usb.org/document-library/usb-type-cr-cable-and-connector-specification-release-23

[48] M. Vanhoef, "Fragment and forge: Breaking Wi-Fi through frame aggregation and fragmentation," in *USENIX Security*, 2021.

[49] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the dragonfly handshake of wpa3 and eap-pwd," in *S&P*, 2020.

[50] Y. Wang, H. Guo, and Q. Yan, "Ghosttalk: Interactive attack on smartphone voice system through power line," in *NDSS*, 2022.

[51] Y. Wu, T. Zhang, C. Jung, and D. Lee, "Devfuzz: Automatic device model-guided device driver fuzzing," in *S&P*, 2023.

[52] Q. Yang, P. Gasti, G. Zhou, A. Farajidavar, and K. S. Balagani, "On inferring browsing activity on smartphones via usb power analysis side-channel," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 5, pp. 1056–1066, 2017.