WATCH YOUR PHONE

Novel USB-Based File Access Attacks Against Mobile Devices

Florian Draschbacher & Lukas Maar

About Us

We are PhD students at Graz University of Technology's Institute of Information Security

Florian Draschbacher

florian.draschbacher@tugraz.at

Research Areas:

- Mobile Security
- Application Analysis
- Hardware Aspects

isec

Research Areas:

- System Security
- Kernel Security
- Side-Channel Security

Lukas Maar

lukas.maar@tugraz.at

Introduction

- Mobile devices store sensitive user data
  - Pictures, Messages, Credentials, …
- USB connectivity is a known attack vector
  - Extract data, compromise device

- **We present novel USB data extraction attacks for two scenarios:**
  - Manipulated Chargers: Attacker needs to bypass user prompts
  - Physical Access: Attacker needs to bypass lock screen

History of USB Attacks

Manipulated USB Devices

Exploit high-level trust model

- **Malicious Hosts**
  JuiceJacking (2011)
  Mitigated with user prompts
- **Malicious Peripherals**
  BadUSB (2014-)
  JuiceFilming (2016-)
  Ghosttalk (2022-)

Physical Access

Exploit individual low-level flaws

- **Example: Checkm8 (2019)**
  Code execution through
  Use-After-Free in USB stack
- **Commercial forensics tools**
  Cellebrite UFED (2008-)
  MSAB XRY (2009-)
  Magnet GrayKey (2017-)

Background: USB on Mobile

- Mobile devices use multi-function USB-C ports

- **Power**
  Charge phone or supply peripherals
- **USB**
  Data exchange with PC or peripherals
- **USB Power Delivery**
  Negotiation of power and data roles

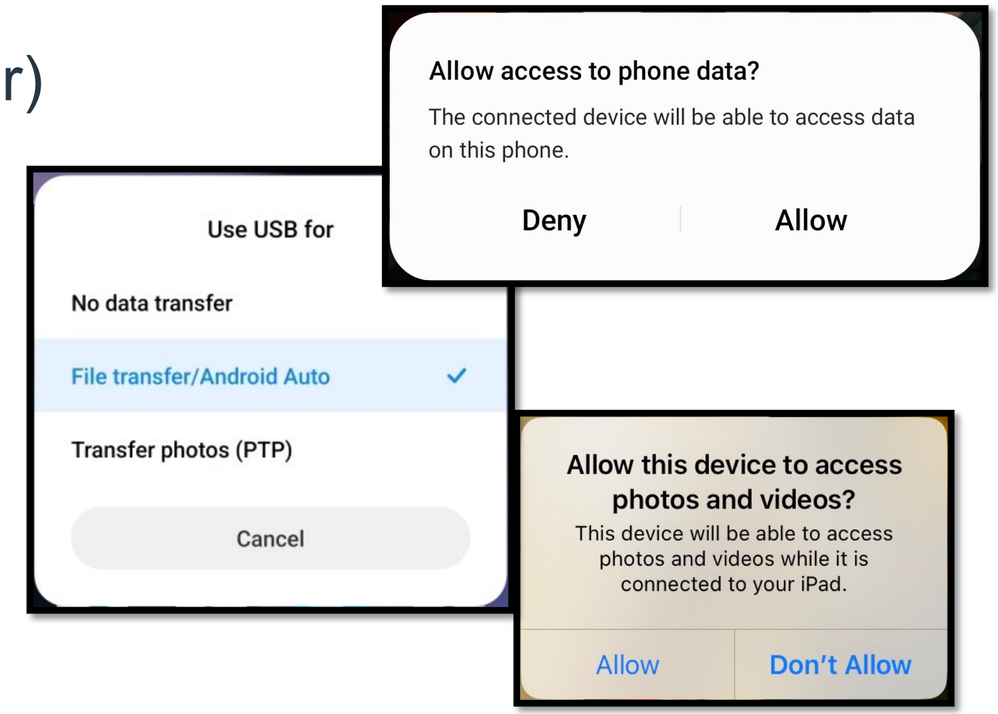
Image: Kyu3 / CC BY-SA 4.0 / wikimedia

Still: A USB port **either** acts as USB host or USB device at a given time

Background: USB File Access on Mobile

User prompts mitigate malicious chargers that extract files (“JuiceJacking”)

1. User connects device to USB Host (computer)
2. MTP interface doesn’t yet show any files
3. User unlocks screen
4. User accepts prompt / changes USB mode
5. Files show through MTP



Manipulated Charger Attacks

Bypassing JuiceJacking Mitigations On State-of-the-Art Mobile Devices

Attack Setup

- **Attacker plants malicious phone charger**
  Reflashed firmware or exchanged electronics

- **Charger attacks any charging mobile device**
  Data extraction, …

- **Easy to mount in public places**
  Chargers at airports, museum, hotel rooms, …

Pictures: wikimedia.org, ChatGPT, own work

Key Observations on JuiceJacking Mitigations

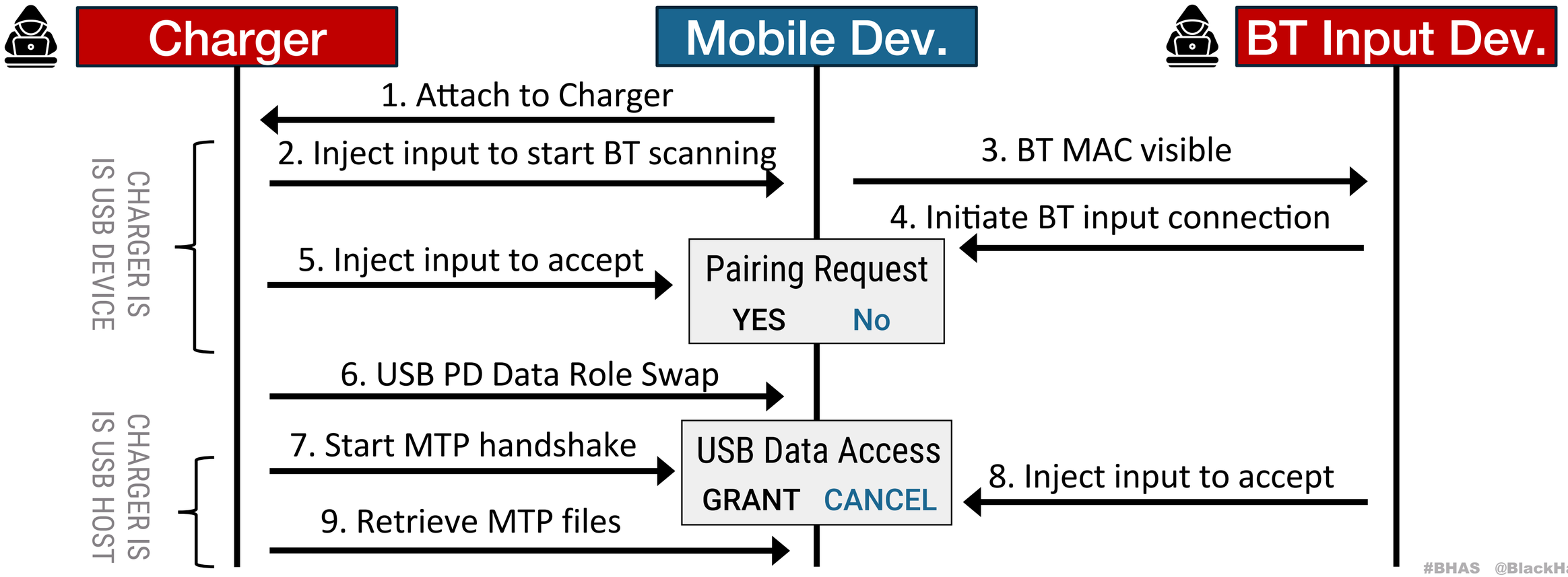
Goal: Ensure user consciously enables USB file access

1. **Require Screen Unlock**
   Idea: Subsequent actions executed by legitimate owner
   Observation: Users routinely unlock screen while charging

2. **Require User Consent**
   Idea: USB partner cannot establish MTP and inject input at same time
   Flaw: Impossible by USB specification, but possible in practice!

Attack: ChoiceJacking via PD & BT HID

Use USB PD to switch USB data roles, hide BT device in charger



iOS: CVE-2025-24193, Patched
Android: Partly Patched

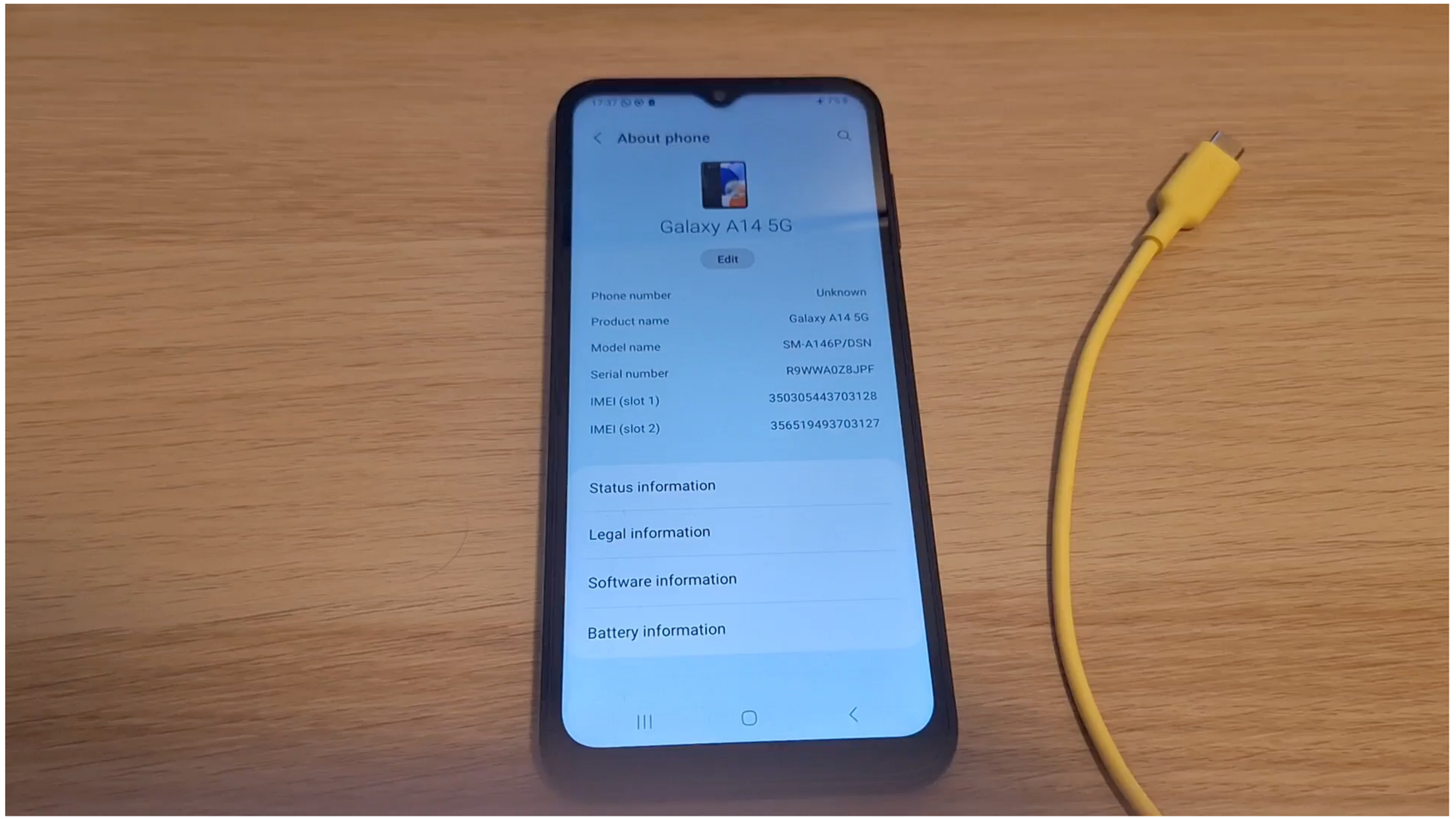
Attack: ChoiceJacking via AOAP

Android Open Accessory Protocol (AOAP)
Allows USB host to inject input events, even if not in accessory mode

1. Connect Android device to malicious charger
2. Initiate MTP connection to trigger user prompt
3. Inject input events through AOAP to confirm prompt
4. Stealthily access files from device

Android Patch Pending

Demo: Access Files on Samsung Galaxy A14



Physical Access Attacks on Android

Entirely bypassing lock screen and user consent prompts

Attack Setup

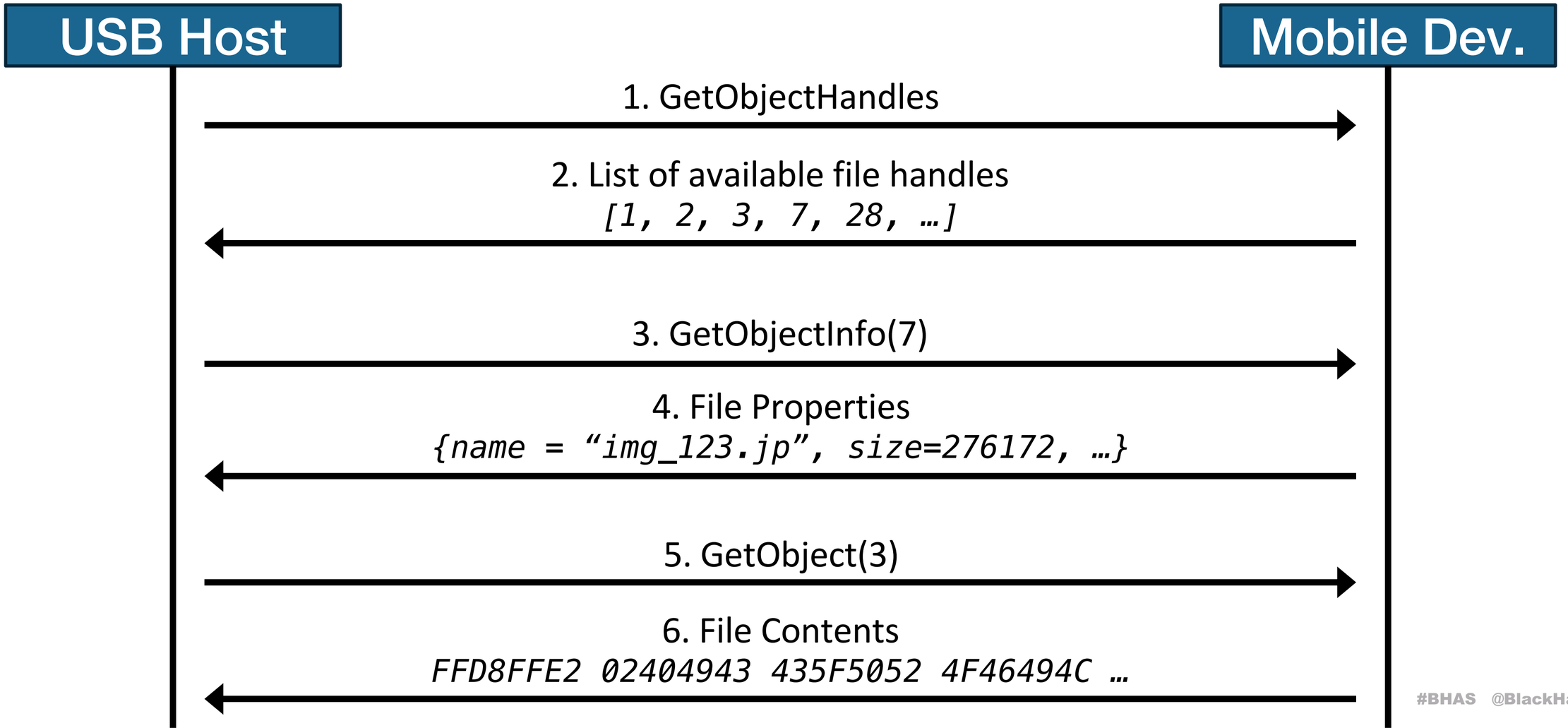
- **Assumption**: Physical device access
- **Assumption**: Unlocked once since last reboot

- **Note**: Android Disk Encryption only effective until first unlock after boot

> For data that must be encrypted with a key associated with user credentials, such as a PIN or password, use credential encrypted storage. Credential encrypted storage is available after the user has successfully unlocked the device and until the user restarts the device. If the user enables the lock screen after unlocking the device, credential encrypted storage remains available.

Source: developer.android.com

Media Transfer Protocol (MTP)

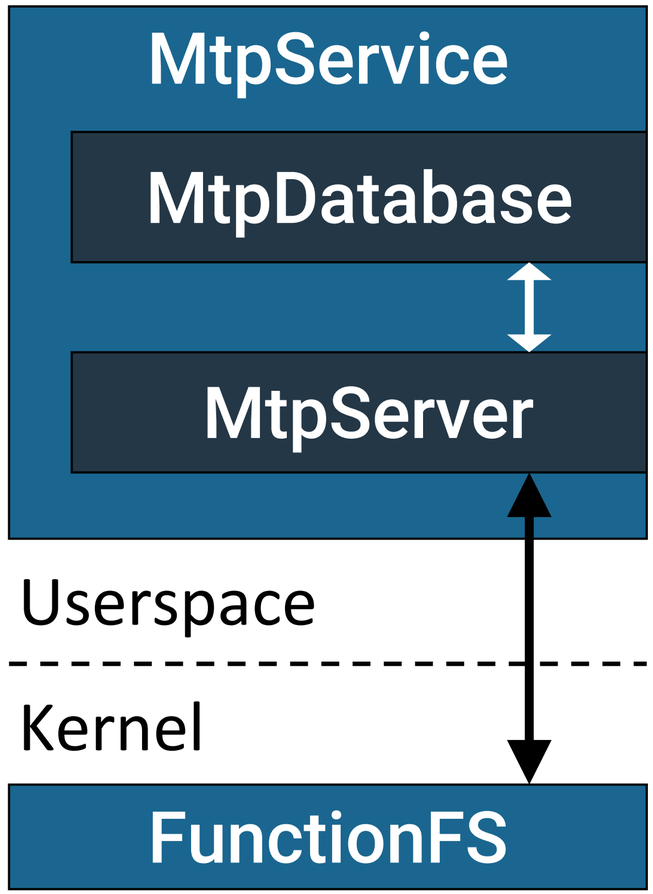


Android MTP Stack

- **USBManager starts MtpService**
  When connected to host
- **USBManager maintains enabled USB functions**
  `mCurrentFunctions` field
- **USB prompts set current (= enabled) functions**
  Eg. By calling

```
usbMgr.setCurrentFunctions(FUNCTION_MTP)
```

- **MtpDatabase is only populated if MTP enabled**



Populating MtpDatabase

MtpService

```
if ((UsbManager.getCurrentFunctions() & UsbManager.FUNCTION_MTP) != 0)
  for (StorageVolume v : volumes.values())
    database.addStorage(v);
```

MtpDatabase

```
MtpStorageManager mManager;

public void addStorage(StorageVolume storage) {
  MtpStorage mtpStorage = mManager.addMtpStorage(storage);
  mServer.addStorage(mtpStorage);
  …
```

MtpStorageManager keeps track of file handles

MtpDatabase Vendor Customizations

MtpDatabase (customized)

```
private int getObjectFilePath(int handle, char[] outFilePath, …) {
  if (handle <= 10000000) {
    MtpStorageManager.MtpObject obj = mManager.getObject(handle);
    …  }
  Uri objectsUri = MediaStore.Files.getContentUri("external_primary");
  String[] arg = new String[]{Integer.toString(handle - 10000000)};
  Cursor c = resolver.query(objectsUri, PROJECTION, ID_WHERE, arg, …);
  String path = c.getString(1);
  path.getChars(0, path.length(), outFilePath, 0);
```

MtpDatabase Vendor Customizations

MtpDatabase (customized)

```
private int getObjectFilePath(int handle, char[] outFilePath, …) {
  if (handle <= 10000000) {
    MtpStorageManager.MtpObject obj = mManager.getObject(handle);
    …  }
  Uri objectsUri = MediaStore.Files.getContentUri("external_primary");
  String[] arg = new String[]{Integer.toString(handle - 10000000)};
  Cursor c = resolver.query(objectsUri, PROJECTION, ID_WHERE, arg, …);
  String path = c.getString(1);
  path.getChars(0, path.length(), outFilePath, 0);
```

MtpDatabase Vendor Customizations

- **MtpServer includes sanity checks for most requests**

MtpServer

```
MtpResponseCode MtpServer::doGetObject() {
  if (!hasStorage())
    return MTP_RESPONSE_INVALID_OBJECT_HANDLE;
  ...
```

`hasStorage()` only returns true if database populated

- **However: No sanity checks in doTruncateObject!**

Attack: Erase All Files From Huawei nova 12i

For all ***file handles f*** starting from `10000000`:

1. **Start edit through MTP BeginEditObject(f)**

   Opens file descriptor for `MtpDatabase.getObjectFilePath(f)`

2. **Invoke MTP TruncateObject(f, 0)**

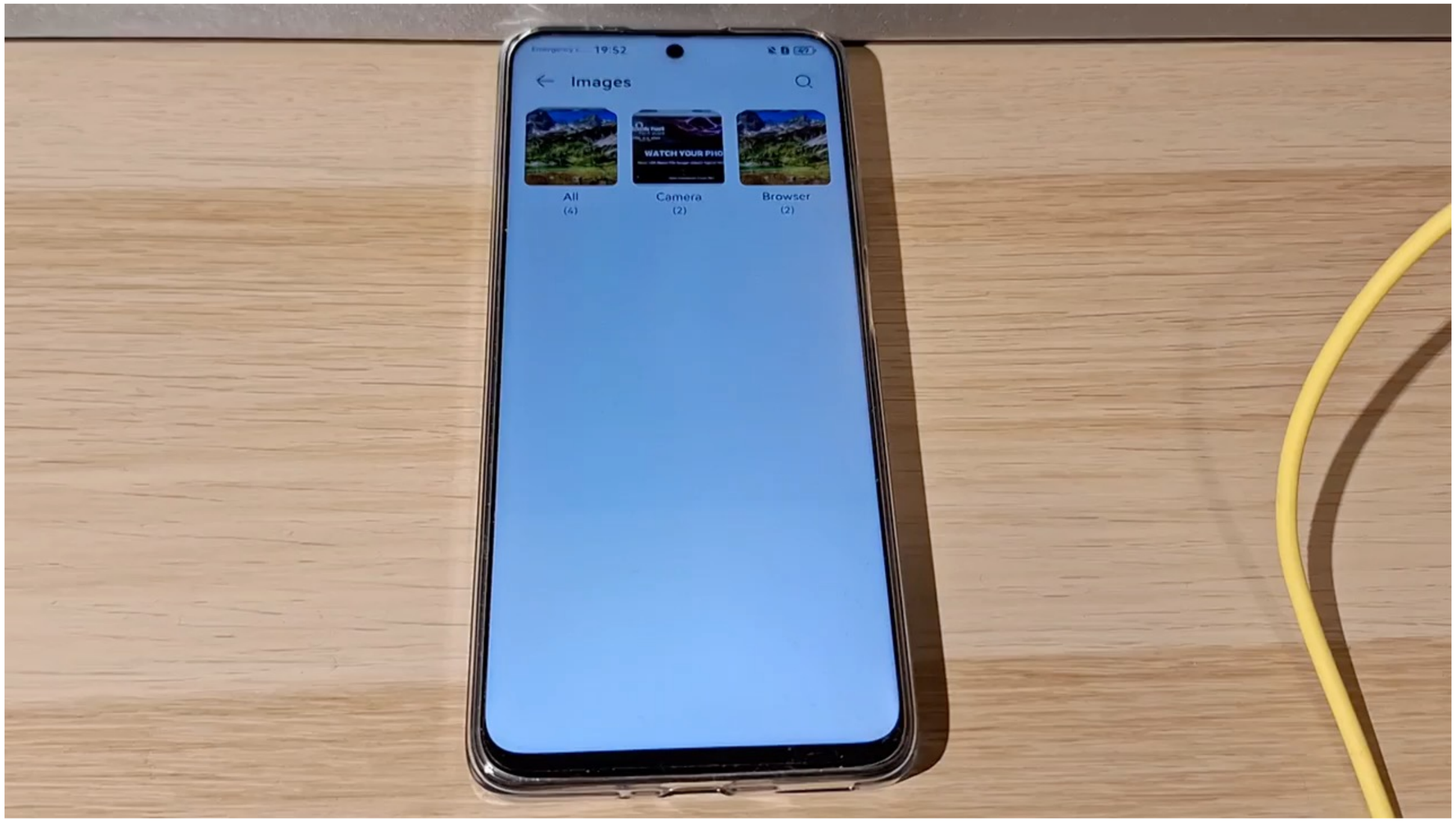
   Calls `ftruncate(0)` on file descriptor

3. **Invoke MTP EndEditObject(f)**

Result: Effectively erase all user files from device

CVE-2024-54096, Patched

Demo: Erase All Files From Huawei nova 12i



Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}
```

```
private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}
```

TIMEOUT

```
public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}

private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}

public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

TIMEOUT

Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}
```

```
private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}
```

TIMEOUT

```
public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}
```

```
private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}
```

```
public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

TIMEOUT

Invoking setEnabledFunctions via USB

`setEnabledFunctions()` **needs calling with** `functions` **other than** `NONE`

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}
```

```
private void startAccessoryMode() {
    ...
    setEnabledFunctions(UsbManager.FUNCTION_ACCESSORY);
    mHandler.sendMessageDelayed(MSG_ACC_MODE_ENTER_TIMEOUT, 10000);
}
```

Idea: Start accessory mode!

Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}

private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}

public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

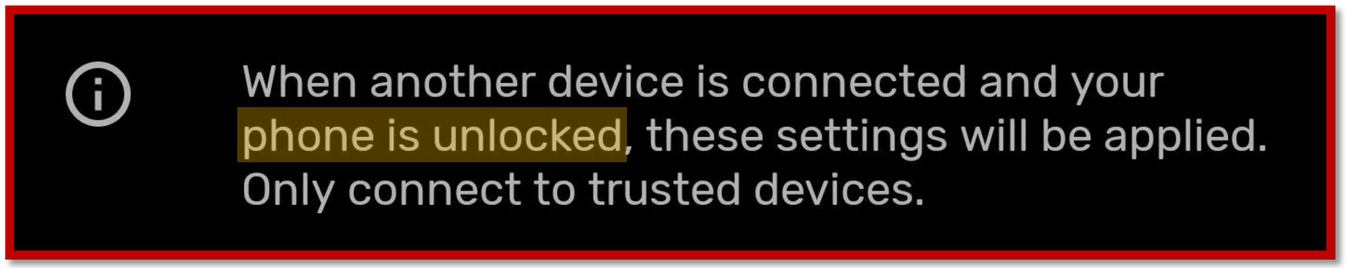
TIMEOUT

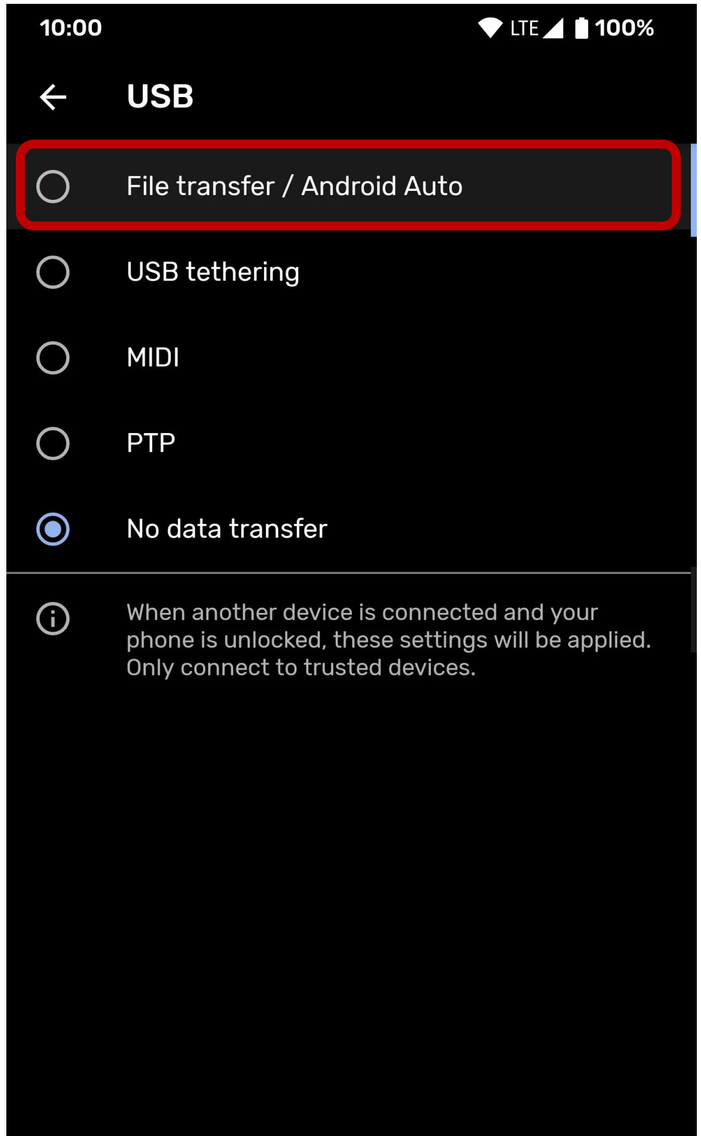
mScreenUnlockedFunctions

- **Can it be set to MTP mode?**
  This would default to MTP upon timeout

- **Yes:** Developer settings → Default USB config
  Requires developer settings to be unlocked first

- **Supposed to enable MTP while device unlocked**





Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}
```

```
private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}
```

TIMEOUT

```
public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

Android USB Stack

Can we enable MTP USB function through UsbManager State Machine?

```
protected void setEnabledFunctions(long functions) {
    setUsbConfig(functions, functions == UsbManager.FUNCTION_NONE);
}

private void setUsbConfig(long config, boolean chargeFuncs) {
    mUsbGadgetHal.setCurrentUsbFunctions(config, chargeFuncs);
    sendMessageDelayed({.what=MSG_TIMEOUT, .arg1=chargeFuncs}, 3000);
}

public void handleMessage(Message msg) {
    if (msg.what == MSG_TIMEOUT && msg.arg1 != 1)
        setEnabledFunctions(mScreenUnlockedFunctions);
}
```

TIMEOUT

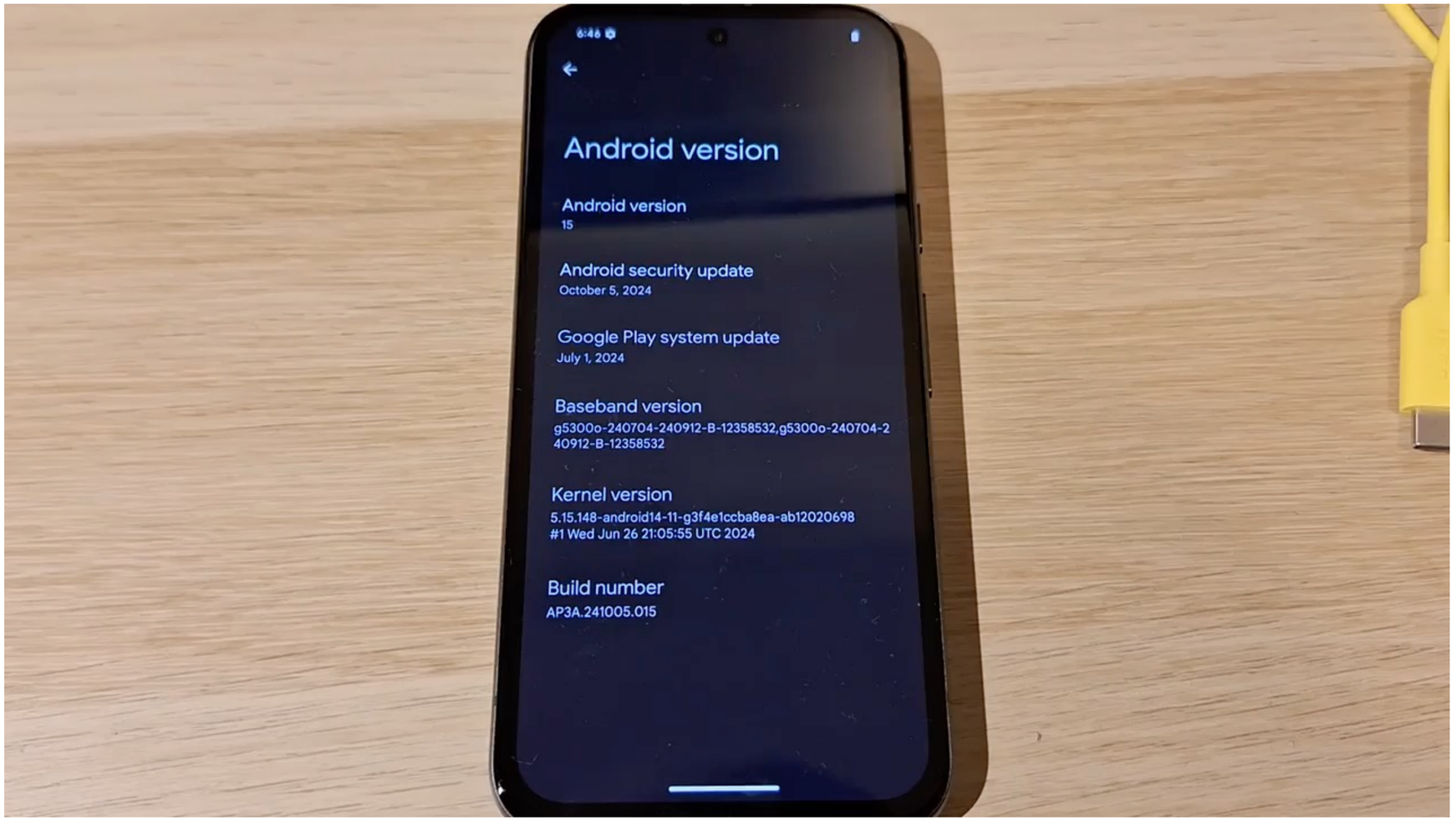
Attack: Retrieve Files from Locked Device

Assumption: Default USB Configuration is set to MTP / File Transfer

1. **Send USB control message to enable accessory mode**
   Switches USB descriptor & awaits re-enumeration
2. **Wait 3 seconds until** `MSG_TIMEOUT`
   Enables populated MTP interface
3. **MTP access for 7 seconds until** `MSG_ACC_MODE_ENTER_TIMEOUT`
4. **Repeat**

CVE-2024-43085, Patched in November 2024 ASB

Demo: Read Files from Locked Pixel 8a



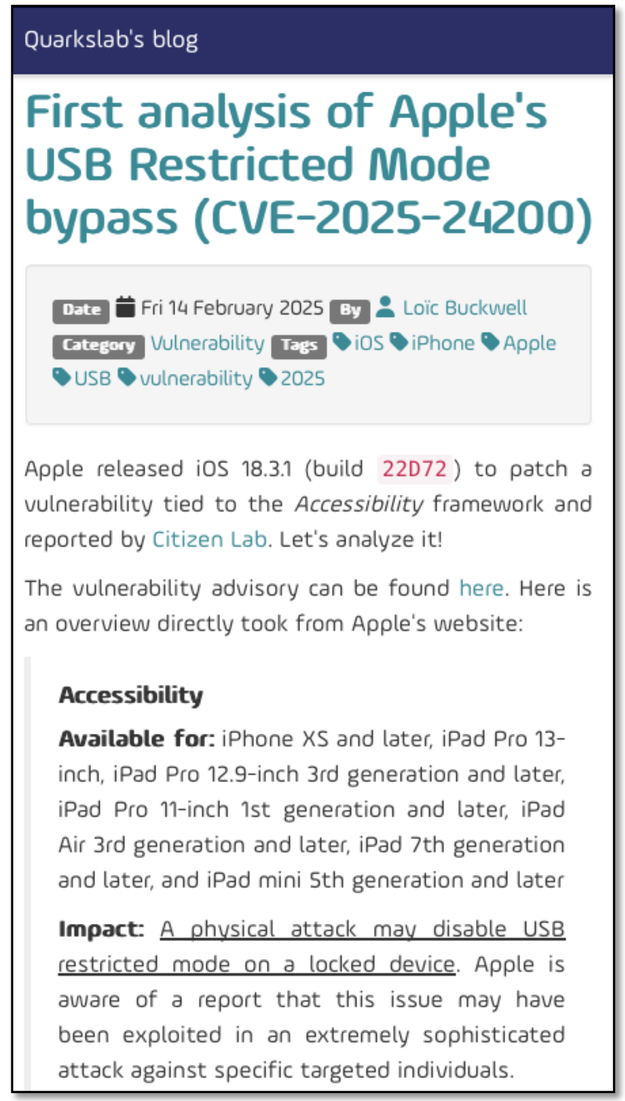
Mitigations

- **User authentication for USB file access prompts**
  Slow vendor adoption

- **Lockdown / Restricted Modes (improved lock screen)**
  Slow vendor adoption, flaws exist

Recommendations:

- Install updates!
- Bring your own power bank
- Otherwise: Shut down device while charging



Quarkslab's blog

First analysis of Apple's USB Restricted Mode bypass (CVE-2025-24200)

Date Fri 14 February 2025 By Loïc Buckwell
Category Vulnerability Tags iOS iPhone Apple USB vulnerability 2025

Apple released iOS 18.3.1 (build 22D72) to patch a vulnerability tied to the *Accessibility* framework and reported by Citizen Lab. Let's analyze it!

The vulnerability advisory can be found here. Here is an overview directly took from Apple's website:

> **Accessibility**
>
> **Available for:** iPhone XS and later, iPad Pro 13-inch, iPad Pro 12.9-inch 3rd generation and later, iPad Pro 11-inch 1st generation and later, iPad Air 3rd generation and later, iPad 7th generation and later, and iPad mini 5th generation and later
>
> **Impact:** A physical attack may disable USB restricted mode on a locked device. Apple is aware of a report that this issue may have been exploited in an extremely sophisticated attack against specific targeted individuals.

Source: quarkslab.com

BlackHat SoundBytes

- **ChoiceJacking: JuiceJacking-style attacks are still possible**
  Malicious chargers can bypass user prompts to extract files

- **File extraction still possible on locked devices**
  Flaws exist even in state-of-the-art devices

- **Watch your phone**
  Don’t hand it to strangers, only use trusted chargers

BlackHat SoundBytes

- **ChoiceJacking: JuiceJacking-style attacks are still possible**
  Malicious chargers can bypass user prompts to extract files
- **File extraction still possible on locked devices**
  Flaws exist even in state-of-the-art devices
- **Watch your phone**
  Don’t hand it to strangers, only use trusted chargers

Questions?