This page intentionally left blank.

Here is my lens. You know my methods. —Sherlock Holmes

AUTHOR CONTACT

hmahalik@gmail.com

200

Twitter: @heathermahalik

20852

domenica.crognale@gmail.com

SANS INSTITUTE

11200 Rockville Pike, Suite

North Bethesda, MD

301.654.SANS(7267) Twitte

DFIR RESOURCES

info@sans.org

digital-forensics.sans.org

registration@sans.org

Twitter: @sansforensics

SANS EMAIL

GENERAL INQUIRIES:

REGISTRATION:

TUITION: tuition@sans.org

PRESS/PR: press@sans.org

FOR585 | Smartphone Forensic Analysis In-Depth

This page intentionally left blank.

585.2

Android Forensics

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With this CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, USER AGREES THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If User does not agree, User may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express

FOR585-2-H01-01

This page intentionally left blank.

This page intentionally left blank.

This page intentionally left blank.

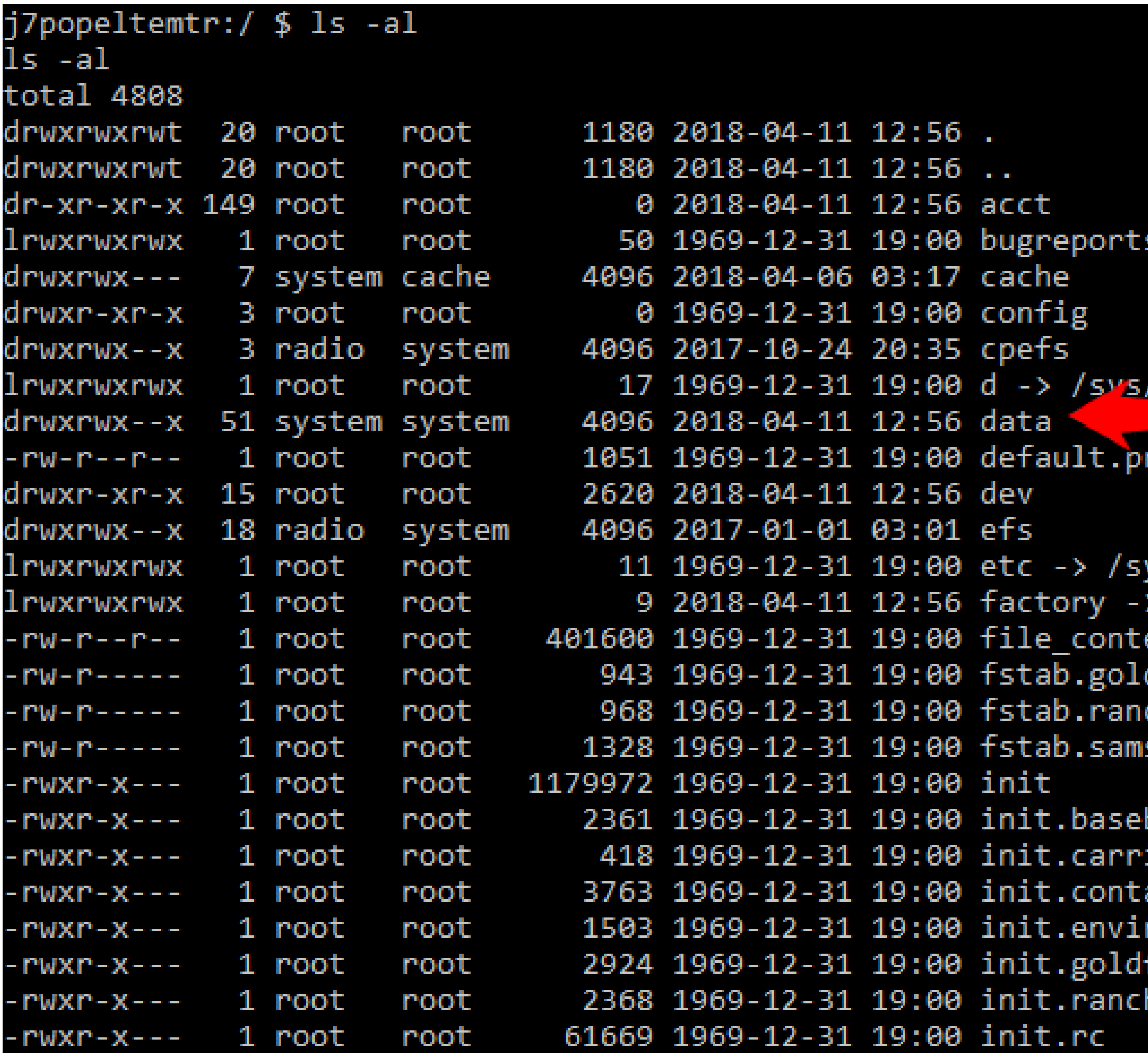
- Command Line Utility

The Android Debug Bridge (ADB) can be used to communicate with the device. ADB requires that USB Debugging be enabled to pull data from the device. For rooted devices, the /Data Partition can be pulled through ADB.[1] Commercial tools even rely on ADB when their traditional acquisition methods fail. For example, Cellebrite UFED Touch and UFED4PC have two options to acquire the devices physically; one is ADB Pull.[2] ADB can be installed on your forensic workstation for free. We have already set everything up in VM so that you can run ADB commands from any location within your course SIFT workstation. Don't forget that these options exist the next time your tools cannot acquire an Android. We will discuss more ADB options for accessing data from Androids later in this section.

In order to pull the /USERDATA partition or /data/data, you'll need to have root access to the device. This will make more sense as you dive into the file systems if the concept isn't something you are familiar with.

References:

- Mahalik, Tamma, and Bommisetty, *Practical Mobile Forensics*, Second Edition (Birmingham, UK: Packt, 2016).
- Tamma and Tindall, *Learning Android Forensics* (Birmingham, UK: Packt, 2015).



```
j7popeltemtr:/ $ ls -al
ls -al
total 4808
drwxrwxrwt  20 root   root      1180 2018-04-11 12:56 .
drwxrwxrwt  20 root   root      1180 2018-04-11 12:56 ..
dr-xr-xr-x 149 root   root         0 2018-04-11 12:56 acct
lrwxrwxrwx   1 root   root        50 1969-12-31 19:00 bugreports
drwxrwx---   7 system cache     4096 2018-04-06 03:17 cache
drwxr-xr-x   3 root   root         0 1969-12-31 19:00 config
drwxrwx--x   3 radio  system    4096 2017-10-24 20:35 cpefs
lrwxrwxrwx   1 root   root        17 1969-12-31 19:00 d -> /sys/
drwxrwx--x  51 system system    4096 2018-04-11 12:56 data
-rw-r--r--   1 root   root      1051 1969-12-31 19:00 default.pr
drwxr-xr-x  15 root   root      2620 2018-04-11 12:56 dev
drwxrwx--x  18 radio  system    4096 2017-01-01 03:01 efs
lrwxrwxrwx   1 root   root        11 1969-12-31 19:00 etc -> /sy
lrwxrwxrwx   1 root   root         9 2018-04-11 12:56 factory -
-rw-r--r--   1 root   root    401600 1969-12-31 19:00 file_conte
-rw-r-----   1 root   root       943 1969-12-31 19:00 fstab.gold
-rw-r-----   1 root   root       968 1969-12-31 19:00 fstab.ranc
-rw-r-----   1 root   root      1328 1969-12-31 19:00 fstab.sams
-rwxr-x---   1 root   root   1179972 1969-12-31 19:00 init
-rwxr-x---   1 root   root      2361 1969-12-31 19:00 init.basel
-rwxr-x---   1 root   root       418 1969-12-31 19:00 init.carri
-rwxr-x---   1 root   root      3763 1969-12-31 19:00 init.conta
-rwxr-x---   1 root   root      1503 1969-12-31 19:00 init.envi
-rwxr-x---   1 root   root      2924 1969-12-31 19:00 init.gold
-rwxr-x---   1 root   root      2368 1969-12-31 19:00 init.ranch
-rwxr-x---   1 root   root     61669 1969-12-31 19:00 init.rc
```

- Nothing is recognized and ready to

- Device is connected

  interact with

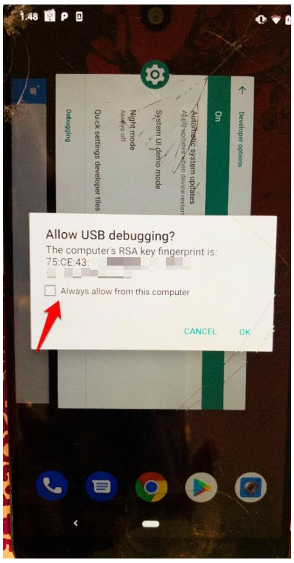
The first step to connecting via ADB is to make sure your workstation can see the device. If you are attempting this within the VM, make sure you have passed the device through to the VM via USB devices. If it’s still not working, unplug the phone and plug it back in. You may have to enter the password if the device has not been trusted with that workstation in the past or if USB authorizations have been cleared on the phone.

The screenshot on the left shows this issue. Nothing appears when you type **adb devices**. The screenshot on the right is what we expect to see when we successfully connect an Android device and have authorized the USB to trust our workstation.

- Device is unauthorized interaction to

- Requires device

  accept the trust



Allow USB debugging?

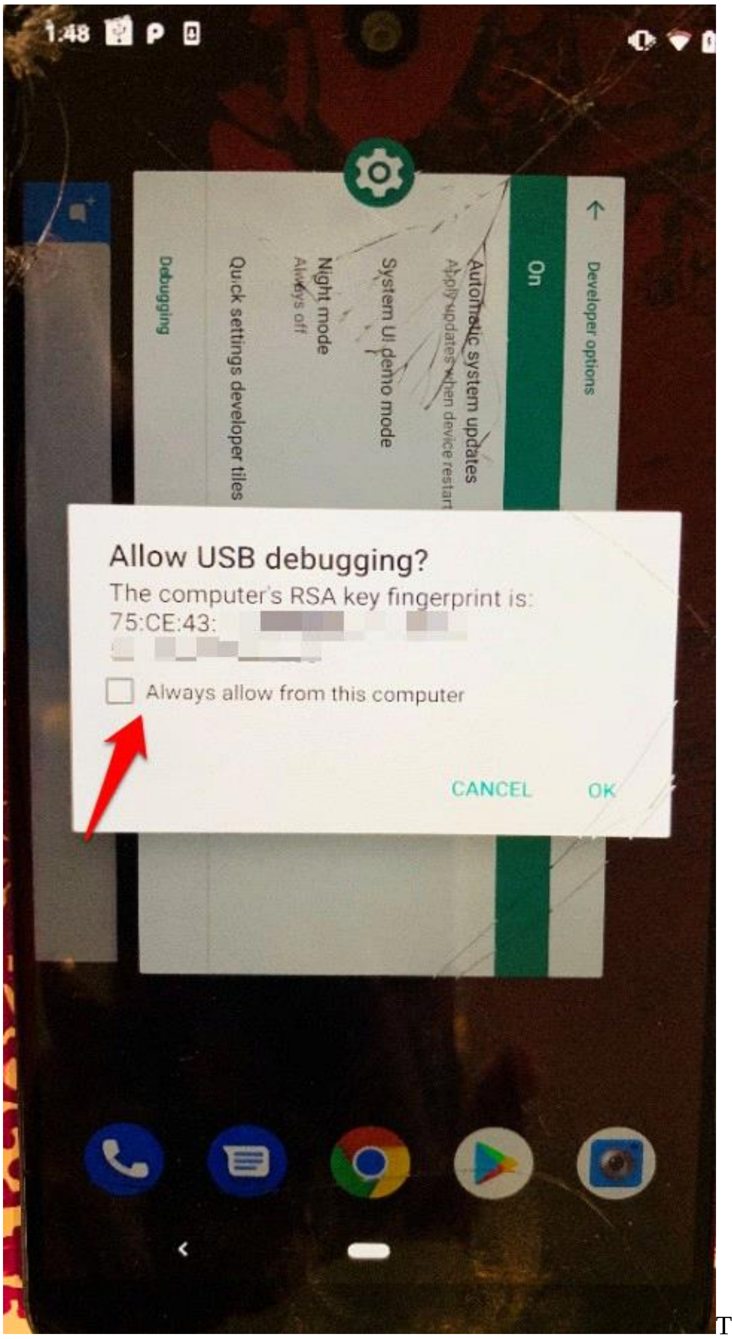
The computer’s RSA key fingerprint is:

75:CE:43: [redacted]

□ Always allow from this computer

Should you see “unauthorized” when attempting to connect through ADB, you need to interact with the phone to ensure that USB debugging has been authorized to connect to that specific computer. Once this step is completed, you may not have to do it again in the future, depending on the device. After checking the box, you should be able to type adb devices a second time and get the result as shown below.

An unauthorized device will look like the screenshots below. The host workstation will show:

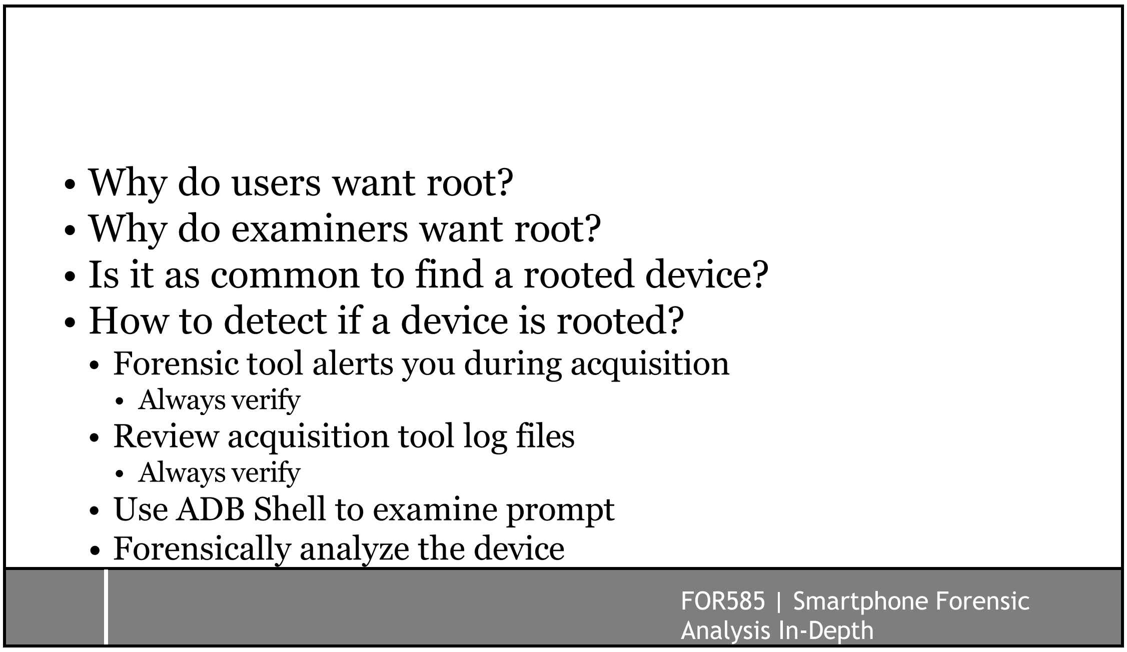


Allow USB debugging?

The computer's RSA key fingerprint is:
75:CE:43: [redacted]

□ Always allow from this computer

The phone will be prompting the user for permission. Should you miss this step, run **adb devices** again and then check the box and enable access.



- Why do users want root?
- Why do examiners want root?
- Is it as common to find a rooted device?
- How to detect if a device is rooted?
  - Forensic tool alerts you during acquisition
    - Always verify
  - Review acquisition tool log files
    - Always verify
  - Use ADB Shell to examine prompt
  - Forensically analyze the device

Users root their Android devices to gain full read/write access to the smartphone. Rooting the device allows the user to have superuser access, which means they have all rights to the smartphone.[1] It is rare to find a device already rooted in the wild today. If you are lucky enough to have a rooted Android device as part of your investigation, you should consider yourself extremely fortunate. You are going to be able to acquire that device with ease! You could pull the entire USERDATA partition for free via ADB.

If you aren’t sure if the device is rooted, a forensic examination should provide you clues. Applications are available for verifying rooted Android devices; however, this is not a forensically sound method. This method would require you to install an app on the user’s device, thus changing the USERDATA Partition. Examples of these apps include Root Checker or Root Checker Basic.[2] The best way to determine root access to the Android is to perform forensic analysis on the device. This is covered in the analytical section. Root access can also be determined using ADB Shell to examine the Android.

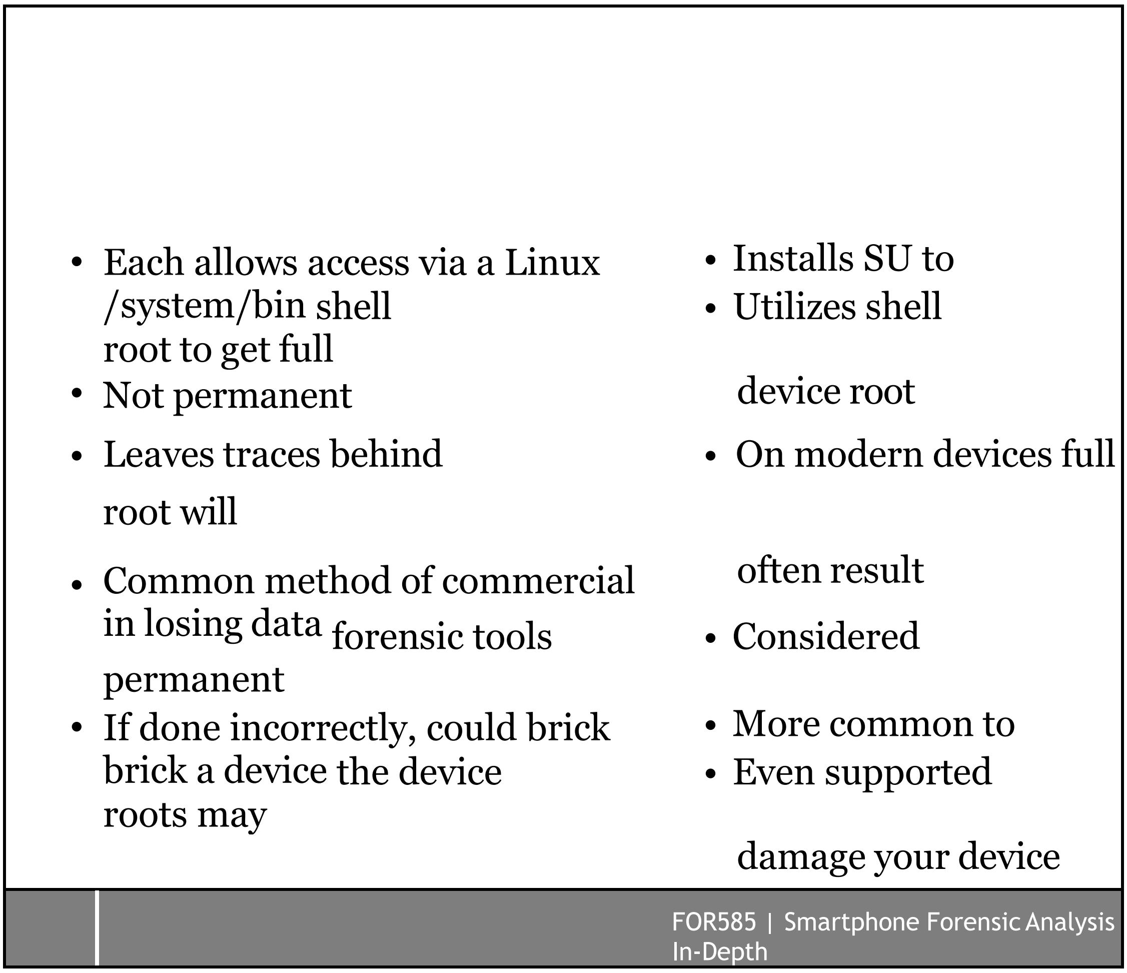
To use ADB Shell, use the command line to determine if the root symbol is present (#). If it isn't, run the "su" command and see if it's there, as explained in the following steps.

Using an ADB Shell, examine the following:

- Does the prompt show $ - means not a root user
- Type “su” or “sudo” into the prompt – what happens?
- Does the prompt show # - means you have root!

References:

- https://for585.com/rooted (How to tell if your phone is rooted)
- https://for585.com/rootcheck (Play Store Root Checker)



- Each allows access via a Linux /system/bin shell
  root to get full
- Not permanent
- Leaves traces behind
  root will
- Common method of commercial in losing data forensic tools
  permanent
- If done incorrectly, could brick brick a device the device
  roots may

- Installs SU to
- Utilizes shell
  device root
- On modern devices full
  often result
- Considered
- More common to
- Even supported
  damage your device

There are different types of root access available for Android devices. Most commercial forensic tools offer a temporary root, which allows temporary superuser access to the device. These temporary roots are supposed to "go away" upon reboot. However, traces are left behind for both temporary and full roots, and we look at this further later in this section. Temporary roots are also called shell and soft roots. NOTE, if you conduct covert operations, you should not root the device if you are worried about leaving a footprint behind. You will leave permanent traces on the phone.

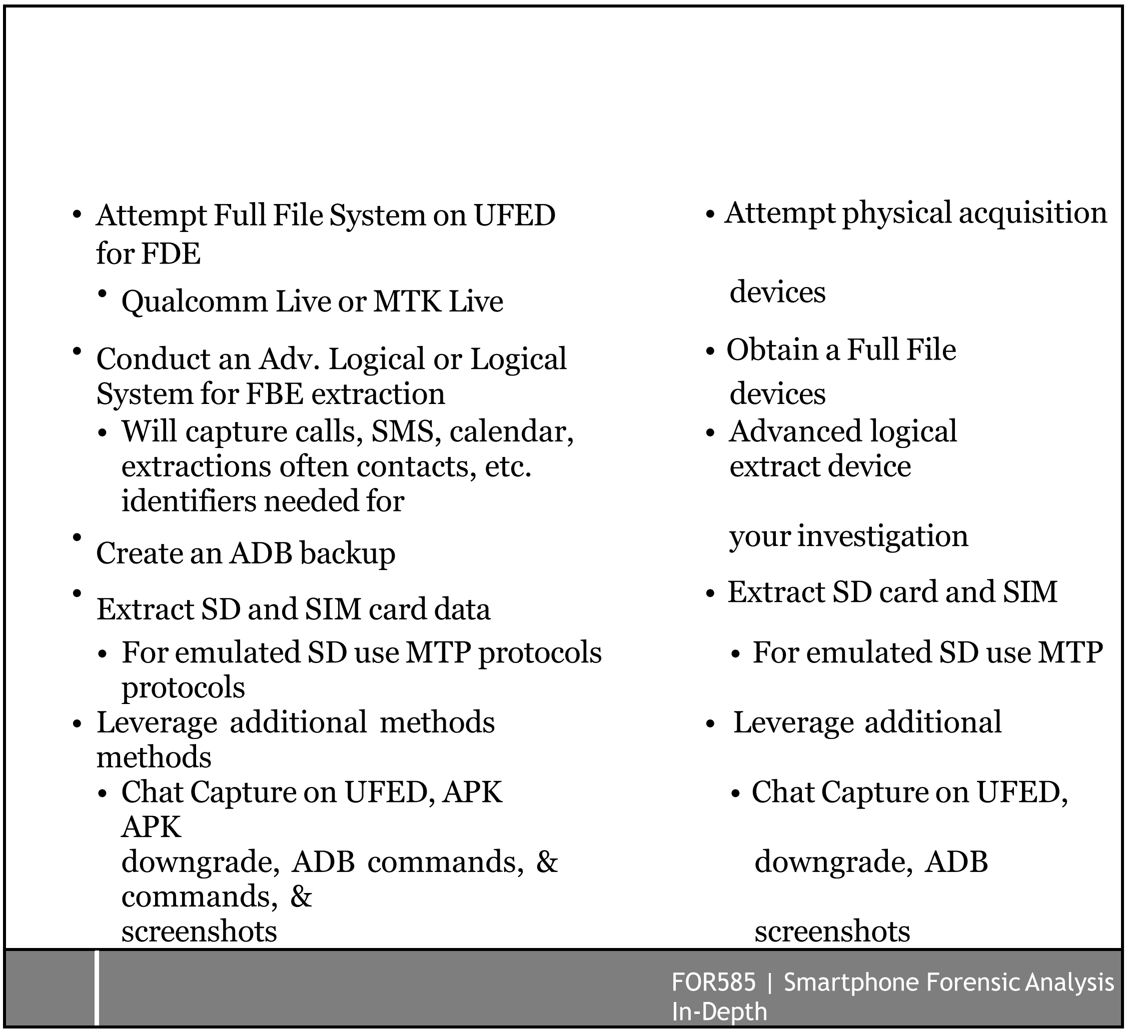
A full root provides persistent root access to the device. This means that, even when the device is rebooted, the device remains rooted. However, how persistent are these roots? Even if a root is removed, traces will be left behind. As expected, traces of a full root are commonly the easiest to detect. For modern devices, it is almost impossible to fully root a device without losing some form of data.

Use caution when downloading and installing roots on the device, as you may lose all user data, be forced to restore the original ROM on the device and be left with nothing. Before rooting a device for physical access, make sure you have acquired the data in all other ways possible. This way, should the device become damaged, or the user data get wiped, you know you acquired all data possible for that device and Android version. If you are trying to find a root in the wild, we have had the best success with https://desktop.firmware.mobi/. Co-author Domenica Crognale did a webcast on root access, using Android emulators and creating test data for application testing. It can be found here:

https://www.sans.org/webcasts/building-android-application-testing-toolbox-106515.

FOR585 | Smartphone Forensic Analysis In-Depth

Rooting a device for forensic access can be risky business. Data can be lost during the process. This is not good for forensics because you lose user data while attempting to access it! Make sure that all roots for forensic use are tested on devices prior to trying it on evidence, where possible. Pixel may erase the user partition before the bootloader is unlocked to install the root. This is a huge risk. Heather Mahalik's website smarterforensics.com has a white paper written by Julie Desautels, a Champlain student, on Google Glass Forensics available in the Reading Room. Even with a root that matches your exact model and revision, you may find that you destroy the device and that it is inaccessible for acquisition. Rooting should always be your last step, and you need to ensure you have permission to proceed. Make sure your company understands that evidence could be destroyed during the rooting process. Keep in mind, it is possible to root devices without wiping the device, but data is commonly lost in the process.



- Attempt Full File System on UFED for FDE
  - Qualcomm Live or MTK Live
- Conduct an Adv. Logical or Logical System for FBE extraction
  - Will capture calls, SMS, calendar, extractions often contacts, etc. identifiers needed for
- Create an ADB backup
- Extract SD and SIM card data
  - For emulated SD use MTP protocols protocols
- Leverage additional methods methods
  - Chat Capture on UFED, APK APK downgrade, ADB commands, & commands, & screenshots

- Attempt physical acquisition

  devices
- Obtain a Full File devices
- Advanced logical extract device

  your investigation
- Extract SD card and SIM
  - For emulated SD use MTP
- Leverage additional
  - Chat Capture on UFED,

    downgrade, ADB

    screenshots

The method for acquiring an Android device often depends on four variables:

- Is the device locked?
- Is the encryption Full Disk or File Based?
- Is USB Debugging enabled?
- To which tools do you have access?

If the device is unlocked, you can essentially access the device using the Logical, File System/Backup, or Full File System/Physical acquisition methods with your preferred smartphone forensic tool or simply using ADB. Make sure that you verify that all data is obtained, and that more than one method is used to acquire the data. If the device is locked and USB Debugging is not enabled, the Cellebrite UFED should be used to attempt to capture a physical acquisition. All Android devices are not supported with this feature. JTAG, ISP, and chip- off may work on locked Android devices where USB Debugging is not enabled, but the data may be encrypted.

Logical acquisition copies "some" of the data that the user can see. This includes active files that are accessible through the operating system. Logical acquisition does not capture deleted data in unallocated areas. What may be captured is data marked as deleted, yet residing in database files waiting to be recovered, as covered in Section 1.

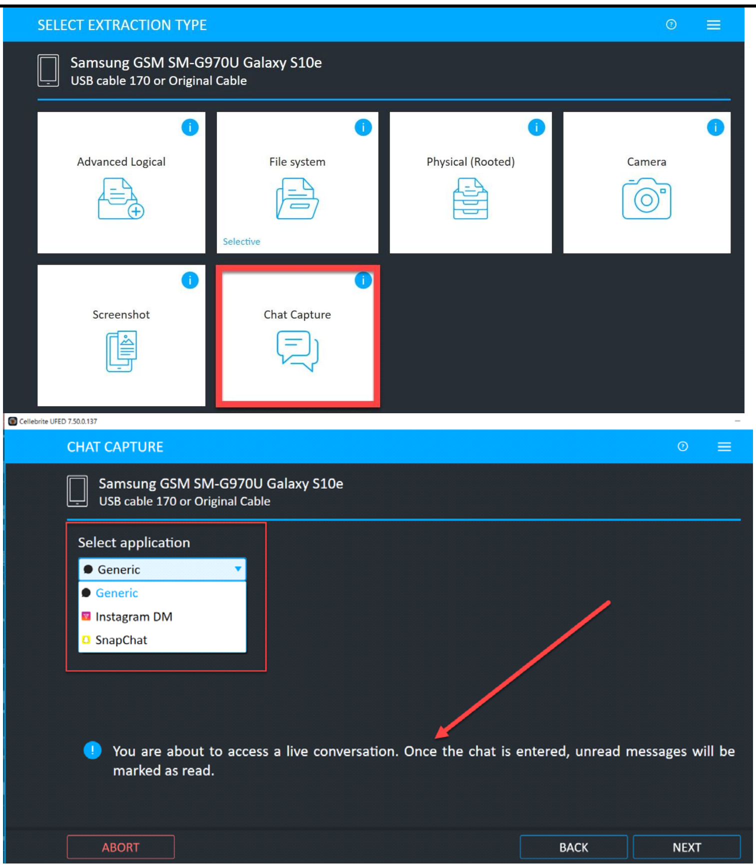
File System acquisitions, by normal standards, provide a logical representation of the files on the device. Forensic tools, such as Cellebrite UFED or Premium, may provide access to the full file system of the device. The logical data is still obtained during a file system acquisition, but the examiner is also presented access to raw files and/or a backup stored within the file system. For devices that aren't supported by Cellebrite, the option to use Generic Android Methods works well. For Android knock-off devices, the Generic Android Method is a great solution! Additionally, I dump most of my Android devices using this method because Physical Analyzer runs all possible plug-ins against the device, which renders more parsed artifacts. It works really well when you can physically dump a device that was otherwise inaccessible!

To access this method, follow these steps:

- Launch UFED.
- Select Manually Browse for Device.
- Select Smartphone.
- Select Android.
- Start with the recommended method and then try the other if it fails.
- Attempt Physical, File System/Backup, and Logical if all are offered.

Several tools are available commercially that support Logical, File System/Backup, and Physical acquisition of Android devices. Each tool generally attempts to root the device for physical access and then reverts to creating a backup via ADB when all else fails. This means that you could really do all of the above acquisition methods for free if you are familiar with ADB and are comfortable interacting with roots and Android devices.

Regardless of how you create an image of an Android, always mount the image or parse the dump to ensure you obtained data in an unencrypted state. There is no worse feeling than obtaining a physical dump and not realizing it was encrypted until it is too late, and the device has been returned. In the next group of slides, we are going to set your expectations for each acquisition image and what data will exist for each method of extraction.

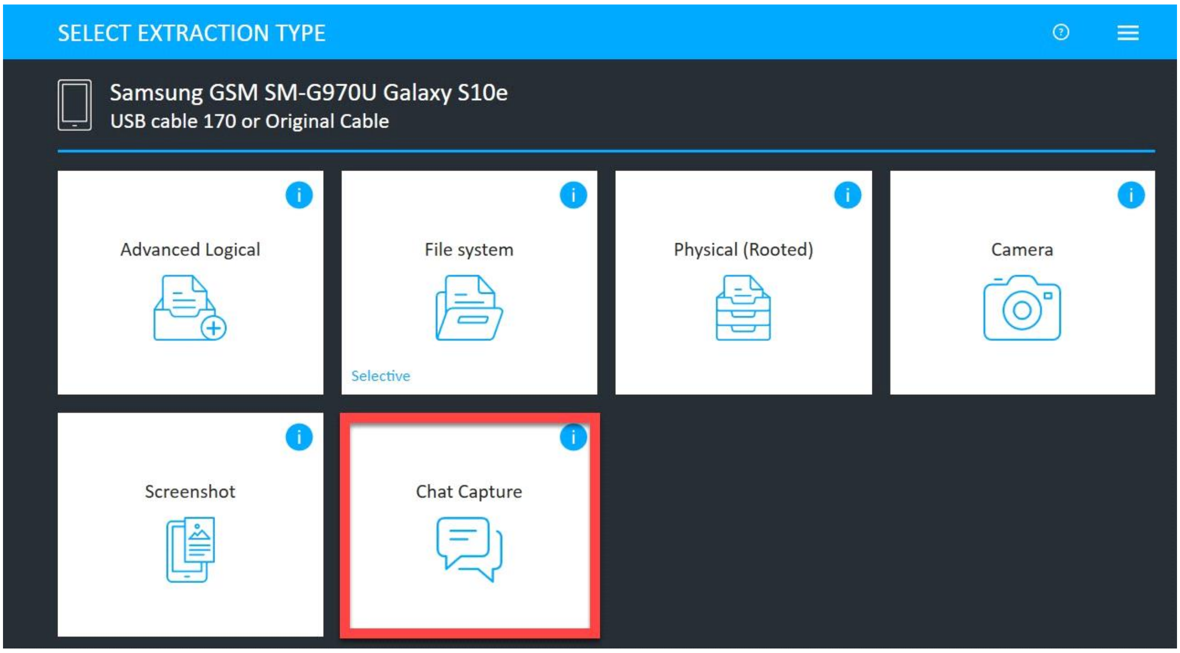


Samsung GSM SM-G970U Galaxy S10e
USB cable 170 or Original Cable

You are about to access a live conversation. Once the chat is entered, unread messages will be marked as read.

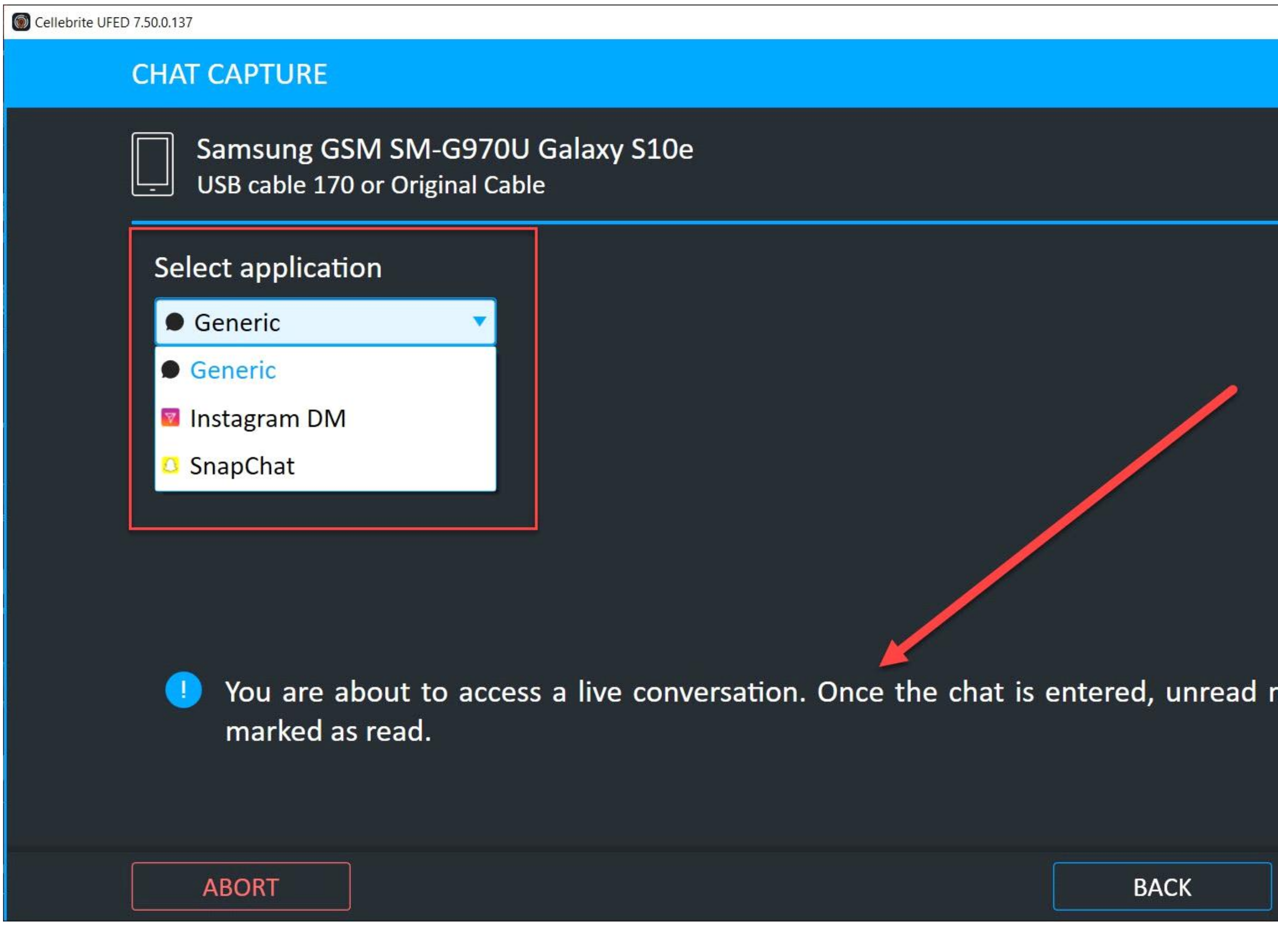
Chat Capture is a feature built into UFED that enables examiners to extract application data of interest when Physical and File System Acquisition are not possible, or do not extract the data. This feature is extremely helpful for encrypted third-party applications. Once extracted, the data can be parsed by your tools of choice. ADB is required for Chat Capture to function, and it’s important that you not touch the phone during the scroll/capture process.

The first step is to find your device in UFED. Next select Chat Capture. Take note that the screens on the device may temporarily change. We know that every contact leaves a trace, so a footprint will be left behind. The main Chat Capture screen enables you to select an application of choice (list of supported apps will be shown) or simply a Generic option which enables the examiner to select the screen and then the scroll up or scroll down. Options on how much data you want are also provided. You can select a timeframe of interest. When you select Next, you can keyword search for conversations or names of interest or names of parties in the chats.



SELECT EXTRACTION TYPE

Samsung GSM SM-G970U Galaxy S10e
USB cable 170 or Original Cable



Cellebrite UFED 7.50.0.137

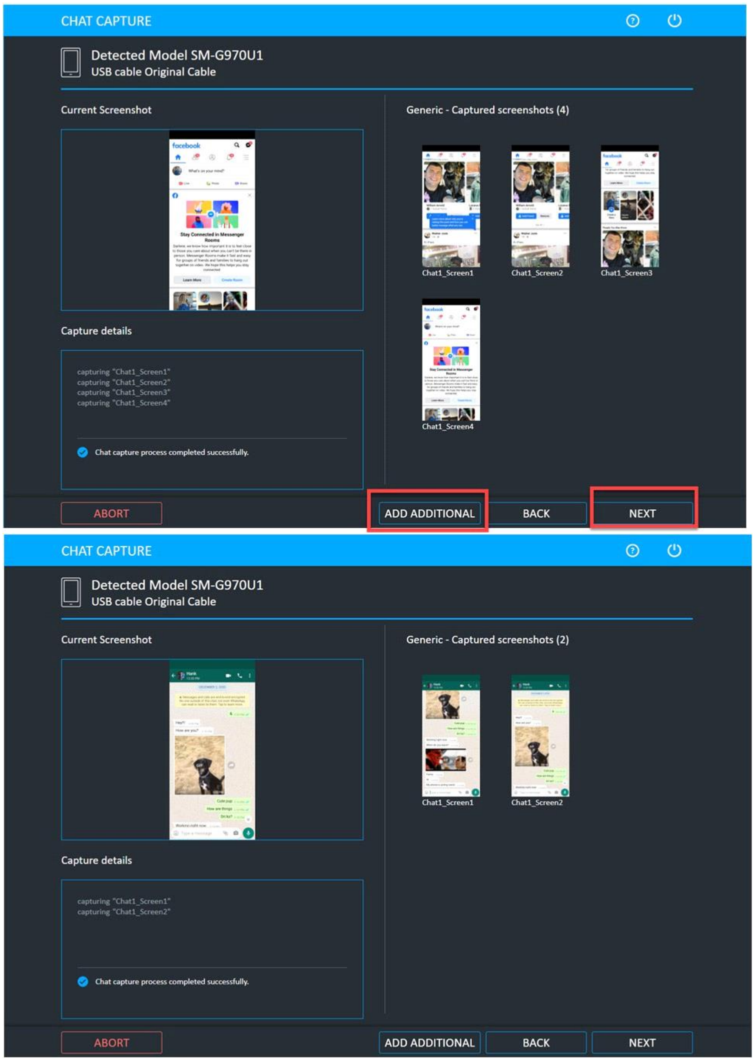
CHAT CAPTURE

Samsung GSM SM-G970U Galaxy S10e
USB cable 170 or Original Cable

Select application

You are about to access a live conversation. Once the chat is entered, unread r
marked as read.

NOTE that once you enter a chat, it will be marked as read. All unread messages will be changed to read messages even after the acquisition completes. This means if the user left messages in a new or unread state, they will be marked as viewed or read messages.



CHAT CAPTURE

Detected Model SM-G970U1
USB cable Original Cable

Current Screenshot

Generic - Captured screenshots (4)

Chat1_Screen1 Chat1_Screen2 Chat1_Screen3 Chat1_Screen4

Capture details

```
capturing "Chat1_Screen1"
capturing "Chat1_Screen2"
capturing "Chat1_Screen3"
capturing "Chat1_Screen4"
```

Chat capture process completed successfully.

CHAT CAPTURE

Detected Model SM-G970U1
USB cable Original Cable

Current Screenshot

Generic - Captured screenshots (2)

Chat1_Screen1 Chat1_Screen2

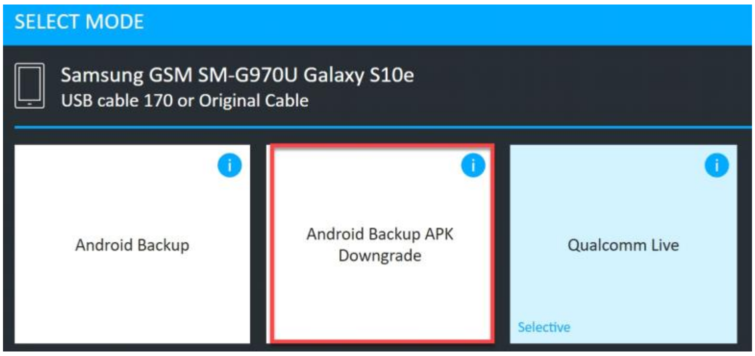
Capture details

```
capturing "Chat1_Screen1"
capturing "Chat1_Screen2"
```

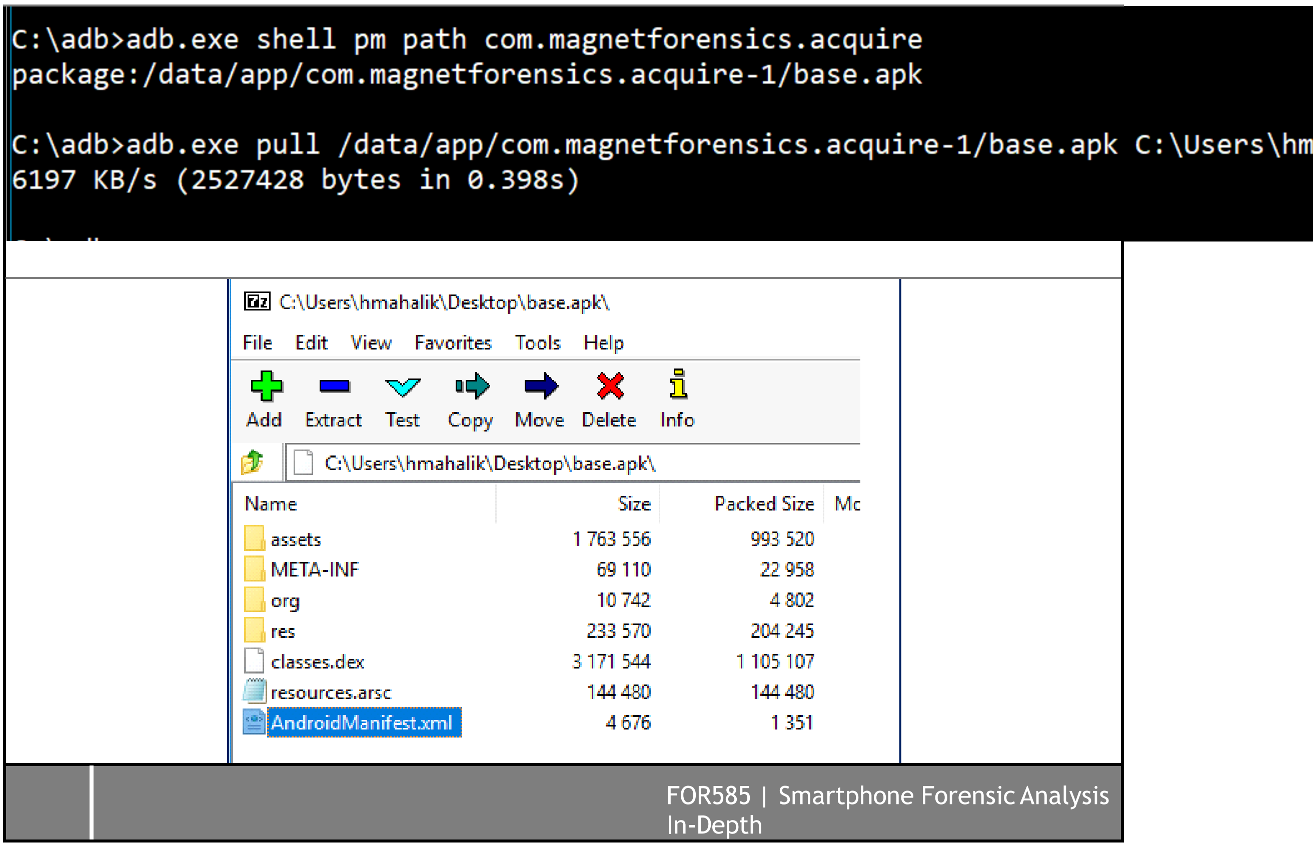
Chat capture process completed successfully.

Chat Capture then begins taking snapshots of the screen from the timeframe selected. Make sure you properly choose the up or down arrows so that you don’t miss a screen. For example, if the chat opens and the user was at the bottom of the conversation, you want to select the scroll up button. The examples above show captures from Facebook and WhatsApp. You can always add additional extractions as well.

If parsed in Physical Analyzer, the data will be placed based upon your selection of generic or a specific application. If you selected WhatsApp or Signal, the results will be placed in Chats. If you selected Generic, the results will be in Analyzed Data>Media>Images. The results are searchable too, as an OCR is created of the screen capture.


SELECT MODE
Samsung GSM SM-G970U Galaxy S10e
USB cable 170 or Original Cable
Android Backup
Android Backup APK Downgrade
Qualcomm Live
Selective

APK downgrade is a solution offered in UFED that is often advertised to be a last effort to preserve evidence on the device. During this method, applications on the device are downgraded in order to extract user data from the application data. For Android 12 devices, you may find that is can be riskier as settings have changed. Stay tuned for a blog as more research occurs for this type of extraction.



```
C:\adb>adb.exe shell pm path com.magnetforensics.acquire
package:/data/app/com.magnetforensics.acquire-1/base.apk

C:\adb>adb.exe pull /data/app/com.magnetforensics.acquire-1/base.apk C:\Users\hm
6197 KB/s (2527428 bytes in 0.398s)
```

C:\Users\hmahalik\Desktop\base.apk\
File
Edit
View
Favorites
Tools
Help
Add
Extract
Test
Copy
Move
Delete
Info
C:\Users\hmahalik\Desktop\base.apk\

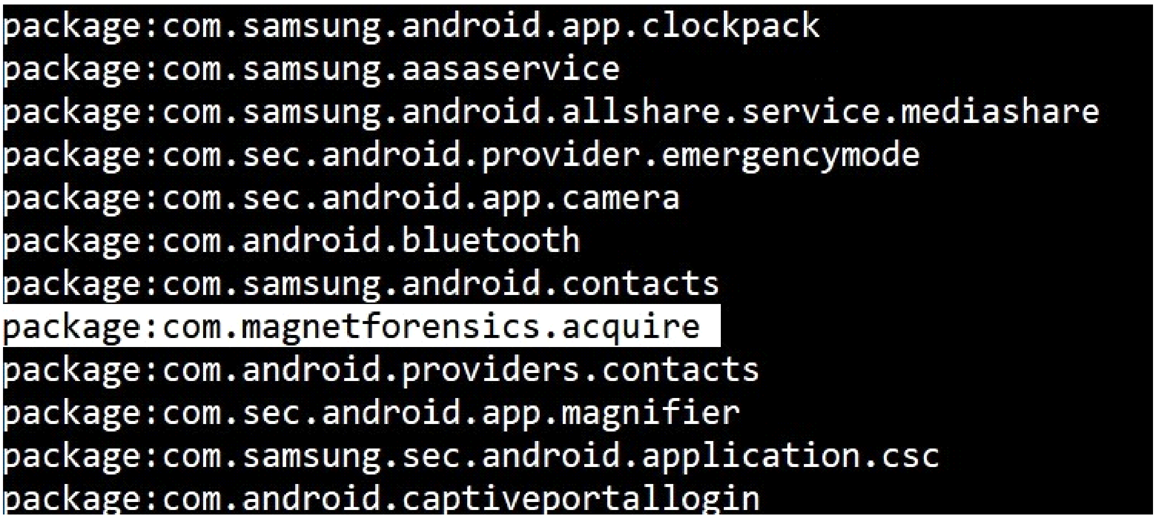
Name	Size	Packed Size	Mc
assets	1 763 556	993 520	
META-INF	69 110	22 958	
org	10 742	4 802	
res	233 570	204 245	
classes.dex	3 171 544	1 105 107	
resources.arsc	144 480	144 480	
AndroidManifest.xml	4 676	1 351	

FOR585 | Smartphone Forensic Analysis In-Depth

In this example, I went back to ADB and ran **adb.exe shell pm path com.magnetforensics.acquire** and received a response with the package information. From here, I was able to do an adb pull to extract that apk. Make sure you state where you want the apk to be saved. From here, I can dive into this application. If you haven't done this in the past, do not worry. This will be covered more in Section 4 where we discuss mobile malware. At this point, we are simply looking for the AndroidManifest.xml file. This file stores the permissions associated to the application. If we examine this file for applications that are installed on the device in order to acquire data, it will make our investigations easier. Why? Because we can testify (if required) to why the app was installed and what it was doing on the device. While this method of extraction doesn't contain the databases full of user artifacts, it provides a glimpse of capabilities, permissions, and more.

In order to know what the package is named, I followed the steps below:

- **adb devices** (makes sure your Android is connected and that you see the device and the serial number)
- **adb.exe shell pm list packages**



```
package:com.samsung.android.app.clockpack
package:com.samsung.aasaservice
package:com.samsung.android.allshare.service.mediashare
package:com.sec.android.provider.emergencymode
package:com.sec.android.app.camera
package:com.android.bluetooth
package:com.samsung.android.contacts
package:com.magnetforensics.acquire
package:com.android.providers.contacts
package:com.sec.android.app.magnifier
package:com.samsung.sec.android.application.csc
package:com.android.captiveportallogin
```

- **adb.exe shell pm path com.magnetforensics.acquire**
- **adb.exe pull /data/app/com.magnetforensics.acquire-1/base.apk <output path>**

Taking a deeper look at logical remnants left behind, we have pulled the base.apk file. Here we are examining the AndroidManifest.xml for the logical acquisition agent installed while using Magnet Acquire. The APK, called base.apk, was extracted using the ADB pull methods discussed in the previous slide. Taking a closer look, we can see that Acquire installs itself in order to collect call logs, contact, dictionary, SMS, Wi-Fi-
/Bluetooth, Browser information, calendar, and background processes. Essentially, it installs itself, grants the permissions required to access content commonly pulled during acquisition, and then collects it. So, you get a logical report, but no actual data. In reality, this may be the best we get other than ADB acquisition pulls.

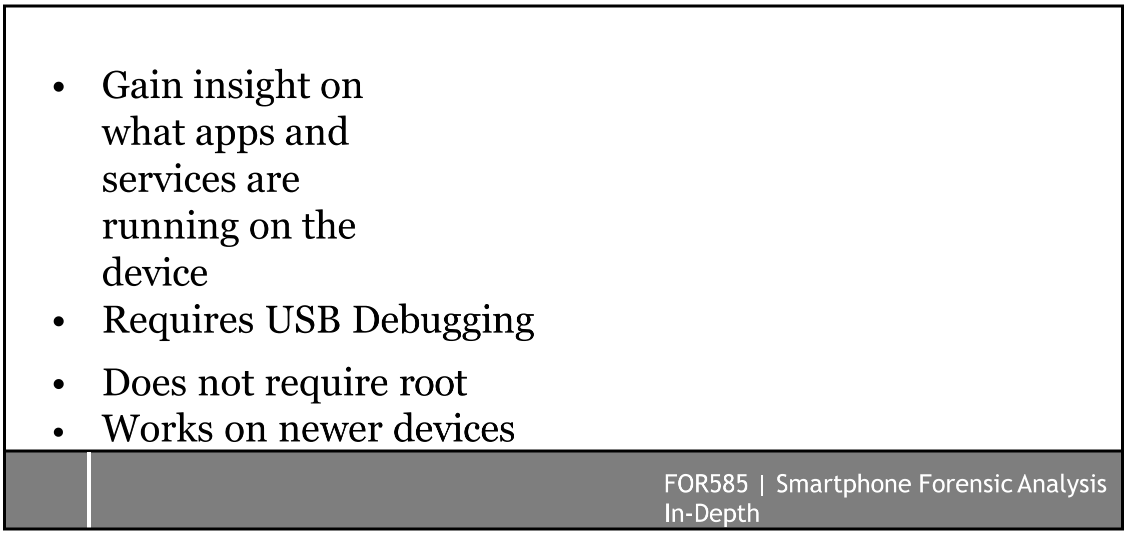
- The tools aren’t acquiring the

data or aren't reporting what you expect

- You cannot afford to add another tool to your toolbox
- There is hope, with a few considerations:
  - USB Debugging is enabled, or you can enable it
  - You have ADB tools (FREE)
  - You must authorize communication between the device and computer

I have so many times been stuck in what I like to call the "loop of failure" when acquiring Android devices. It is so frustrating when you try tool after tool and get the same results, which are just a failure! Make sure you have exhausted all options and remember the Android Generic approaches shared with you so far in this course. When you have hit the wall and nothing is working, what next? When this happens, take matters into your own hands. Roll up your sleeves and go old school on the device. Go back to the original ways of acquiring Android devices and use ADB to interact with the device.

The options when using ADB to interact with a device will be sprinkled throughout this section. If you have an Android of your own, try the bonus lab handout to see how you, too, can have success when running commands to the device.



- Gain insight on what apps and services are running on the device
- Requires USB Debugging
- Does not require root
- Works on newer devices

The first step to connecting with an Android is to validate your connection by typing **adb devices** that will show all devices detected and the current status for each. These statuses normally include:

- offline: The instance is not connected to adb or is not responding.
- device: The instance is connected to the adb server.
- no device: There is no device connected.
- unauthorized: USB debugging isn't authorized.

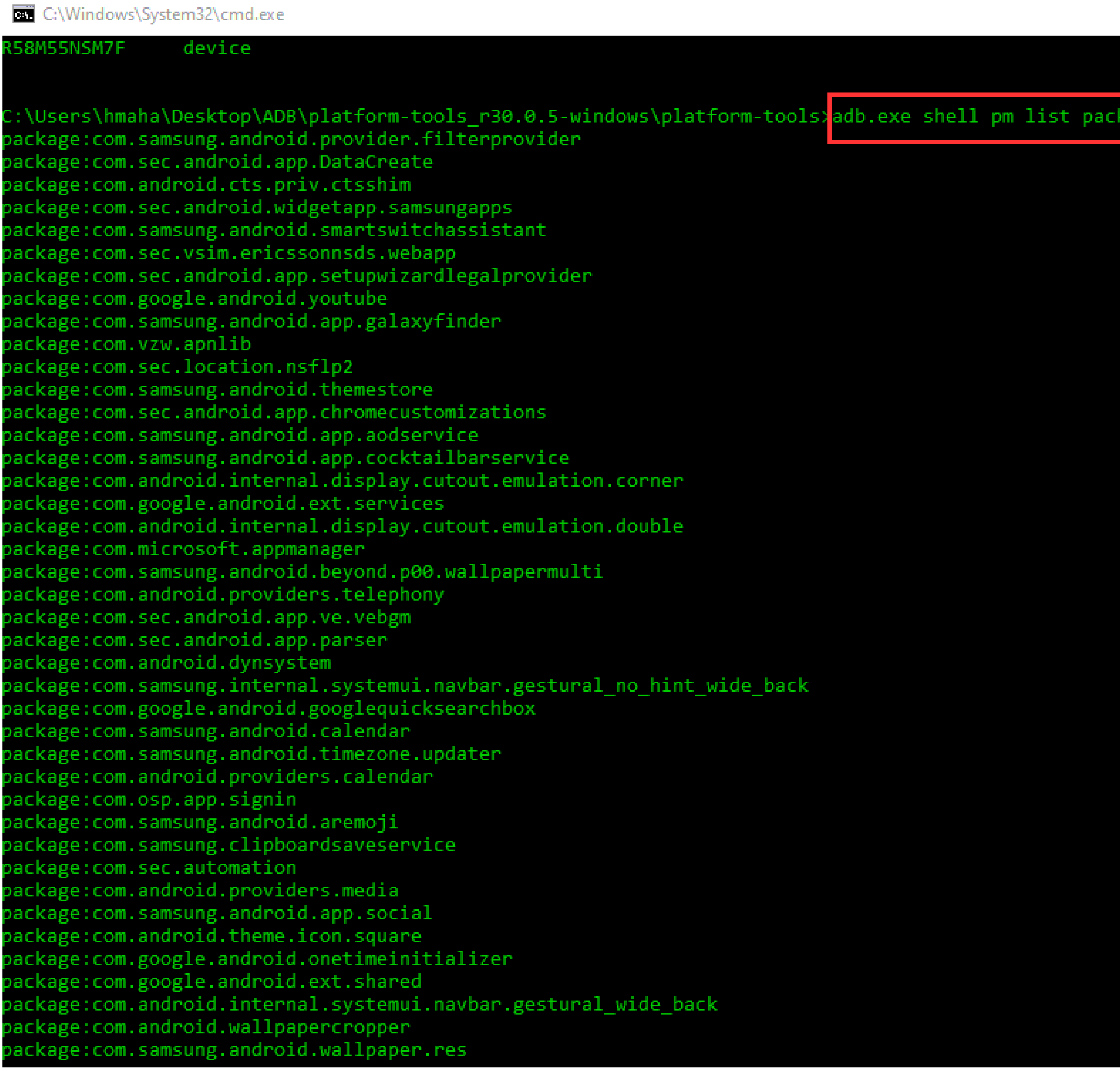
dumpsys is a fantastic option when using ADB to obtain a glimpse of all services running on the Android device. Conducting a dumpsys extraction provides us with the opportunity to collect information about apps and services that exist and are running on the device. A good place to start is to obtain a list of all services running so we get a better understanding of what we may want to extract using dumpsys.[1]

Examiners can interact with devices when the tools just simply cannot provide the access required. This is also a place that may allow you to quickly triage the device. We recommend using ADB to interact with the Android only if you are experienced or as a last-ditch resort after your other acquisition methods have been exhausted. Practice makes you stronger, so get a test device and give it a shot!

Here, we are running **adb.exe shell pm list packages** to view a list of applications and services present on the Android device. Notice that you will see both system and third-party applications listed here. This is being run right from my command line, as ADB tools have been installed.

Reference:

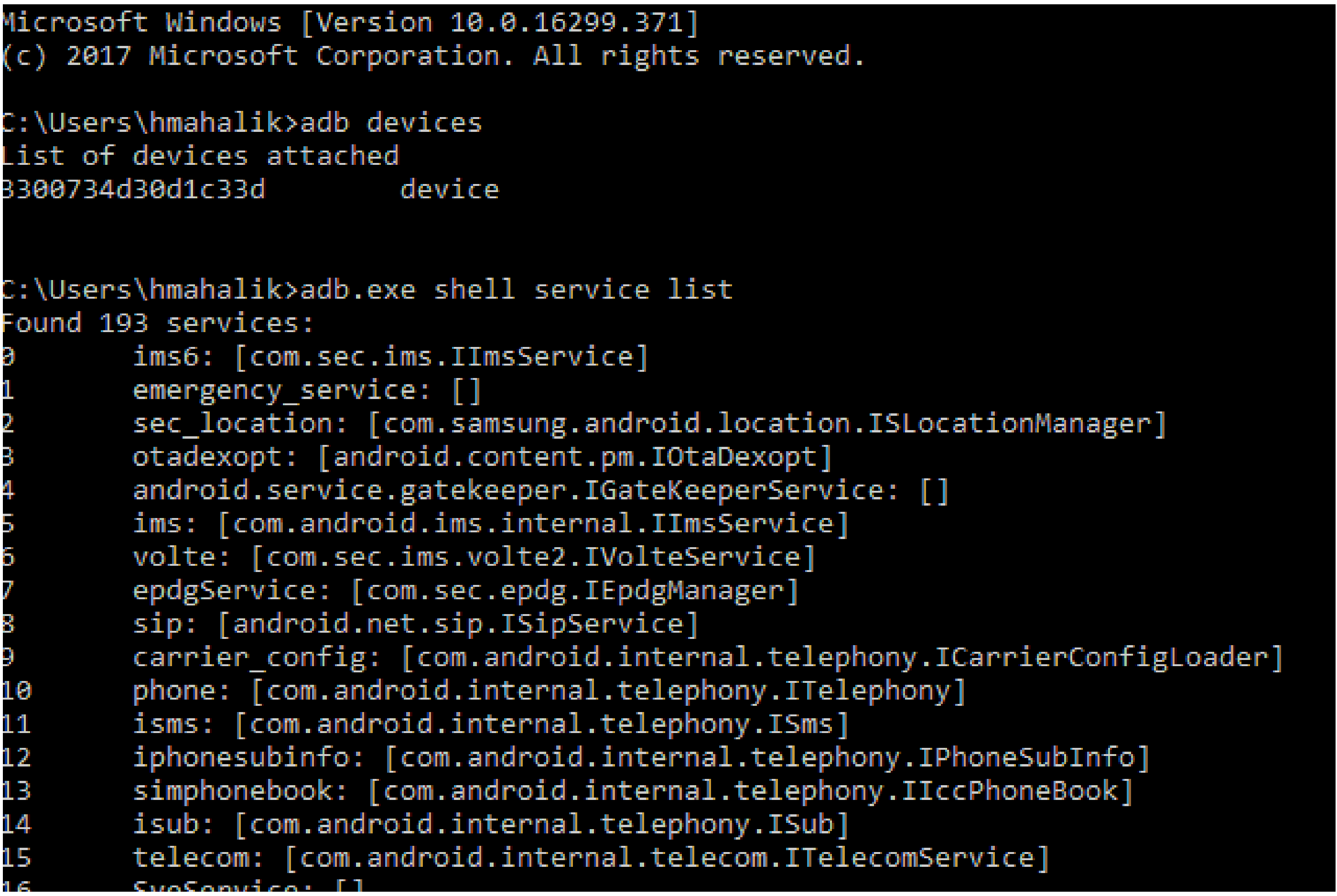
[1] Mahalik, Tamma, and Bommisetty, *Practical Mobile Forensics*, Second Edition (Birmingham, UK: Packt, 2016).



C:\Windows\System32\cmd.exe

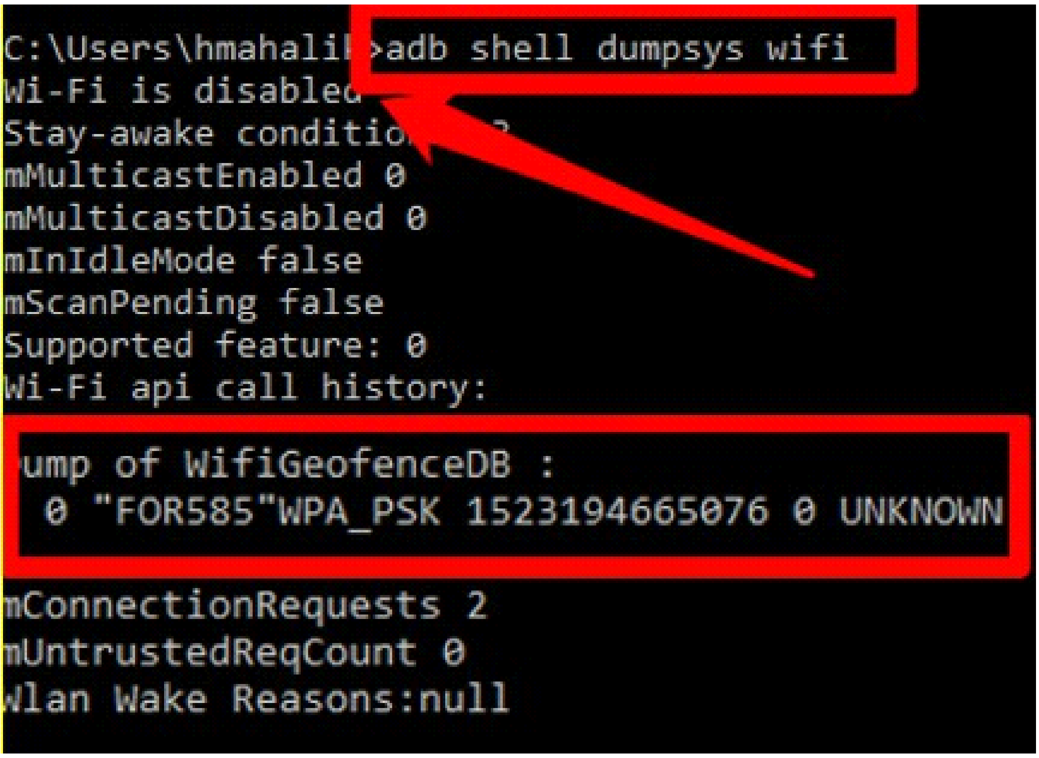
```
R58M55NSM7F     device


C:\Users\hmaha\Desktop\ADB\platform-tools_r30.0.5-windows\platform-tools>adb.exe shell pm list pac
package:com.samsung.android.provider.filterprovider
package:com.sec.android.app.DataCreate
package:com.android.cts.priv.ctsshim
package:com.sec.android.widgetapp.samsungapps
package:com.samsung.android.smartswitchassistant
package:com.sec.vsim.ericssonnsds.webapp
package:com.sec.android.app.setupwizardlegalprovider
package:com.google.android.youtube
package:com.samsung.android.app.galaxyfinder
package:com.vzw.apnlib
package:com.sec.location.nsflp2
package:com.samsung.android.themestore
package:com.sec.android.app.chromecustomizations
package:com.samsung.android.app.aodservice
package:com.samsung.android.app.cocktailbarservice
package:com.android.internal.display.cutout.emulation.corner
package:com.google.android.ext.services
package:com.android.internal.display.cutout.emulation.double
package:com.microsoft.appmanager
package:com.samsung.android.beyond.p00.wallpapermulti
package:com.android.providers.telephony
package:com.sec.android.app.ve.vebgm
package:com.sec.android.app.parser
package:com.android.dynsystem
package:com.samsung.internal.systemui.navbar.gestural_no_hint_wide_back
package:com.google.android.googlequicksearchbox
package:com.samsung.android.calendar
package:com.samsung.android.timezone.updater
package:com.android.providers.calendar
package:com.osp.app.signin
package:com.samsung.android.aremoji
package:com.samsung.clipboardsaveservice
package:com.sec.automation
package:com.android.providers.media
package:com.samsung.android.app.social
package:com.android.theme.icon.square
package:com.google.android.onetimeinitializer
package:com.google.android.ext.shared
package:com.android.internal.systemui.navbar.gestural_wide_back
package:com.android.wallpapercropper
package:com.samsung.android.wallpaper.res
```

Here, I ran **adb devices** to ensure my device was connected and seen by my workstation. Next, I ran **adb.exe shell service list**, and the output was provided. We can see that 193 services are running on this Android device. Once we know the services running, we can elect to extract items of interest for further investigation. This process can be quite addicting, as the excavating never seems to end.



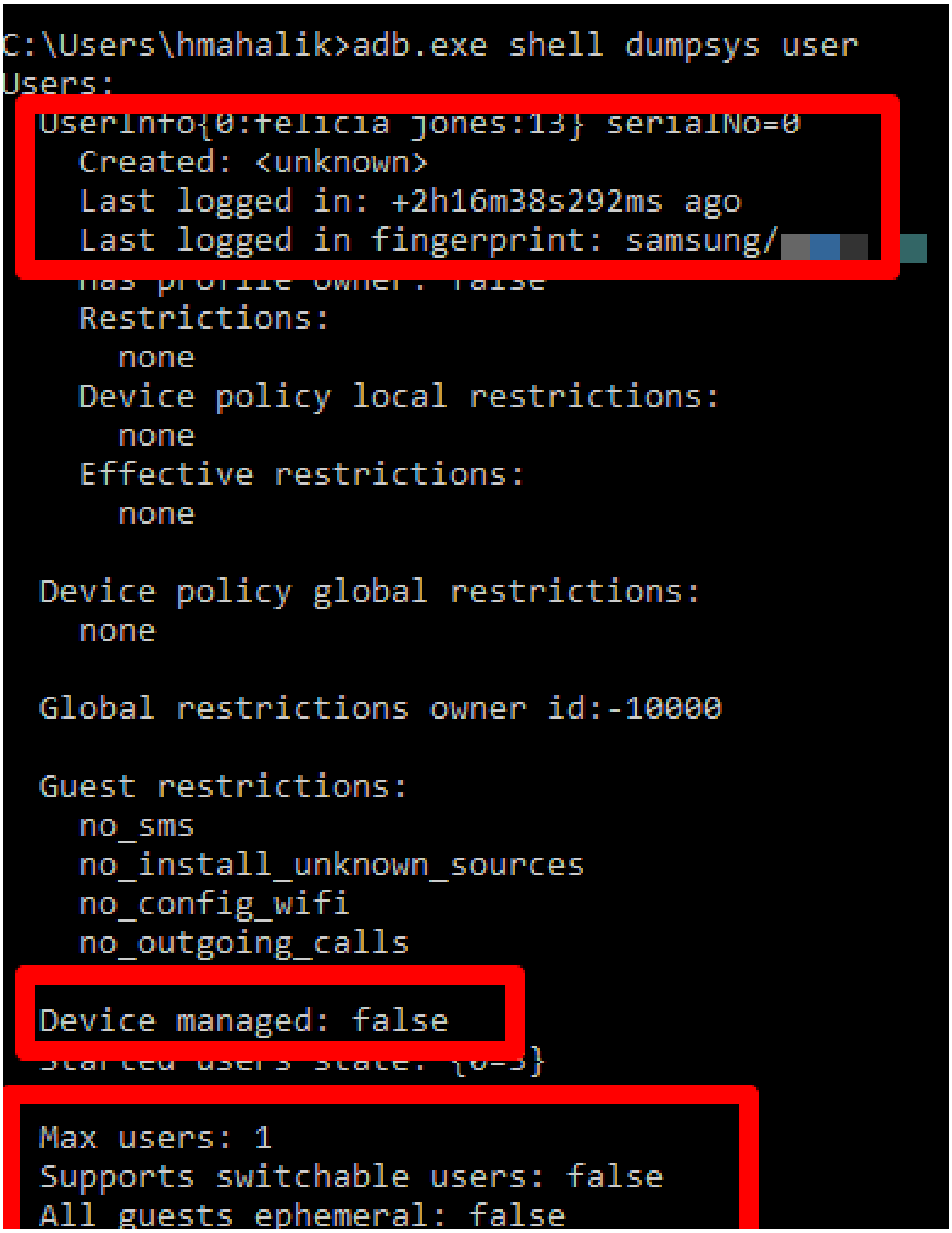
```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\hmahalik>adb devices
List of devices attached
3300734d30d1c33d        device


C:\Users\hmahalik>adb.exe shell service list
Found 193 services:
0       ims6: [com.sec.ims.IImsService]
1       emergency_service: []
2       sec_location: [com.samsung.android.location.ISLocationManager]
3       otadexopt: [android.content.pm.IOtaDexopt]
4       android.service.gatekeeper.IGateKeeperService: []
5       ims: [com.android.ims.internal.IImsService]
6       volte: [com.sec.ims.volte2.IVolteService]
7       epdgService: [com.sec.epdg.IEpdgManager]
8       sip: [android.net.sip.ISipService]
9       carrier_config: [com.android.internal.telephony.ICarrierConfigLoader]
10      phone: [com.android.internal.telephony.ITelephony]
11      isms: [com.android.internal.telephony.ISms]
12      iphonesubinfo: [com.android.internal.telephony.IPhoneSubInfo]
13      simphonebook: [com.android.internal.telephony.IIccPhoneBook]
14      isub: [com.android.internal.telephony.ISub]
15      telecom: [com.android.internal.telecom.ITelecomService]
16      SvcService: []
```

I selected to run **adb shell dumpsys wifi** and examined the results. You should notice that it says Wi-Fi is disabled; this is because my device is currently in Airplane mode. Also, you can see some Wi-Fi connections that I previously connected to. This screenshot was redacted to give you a glimpse of what is there.



```
C:\Users\hmahalil>adb shell dumpsys wifi
Wi-Fi is disabled
Stay-awake conditio
mMulticastEnabled 0
mMulticastDisabled 0
mInIdleMode false
mScanPending false
Supported feature: 0
Wi-Fi api call history:

ump of WifiGeofenceDB :
 0 "FOR585"WPA_PSK 1523194665076 0 UNKNOWN

mConnectionRequests 2
mUntrustedReqCount 0
Wlan Wake Reasons:null
```

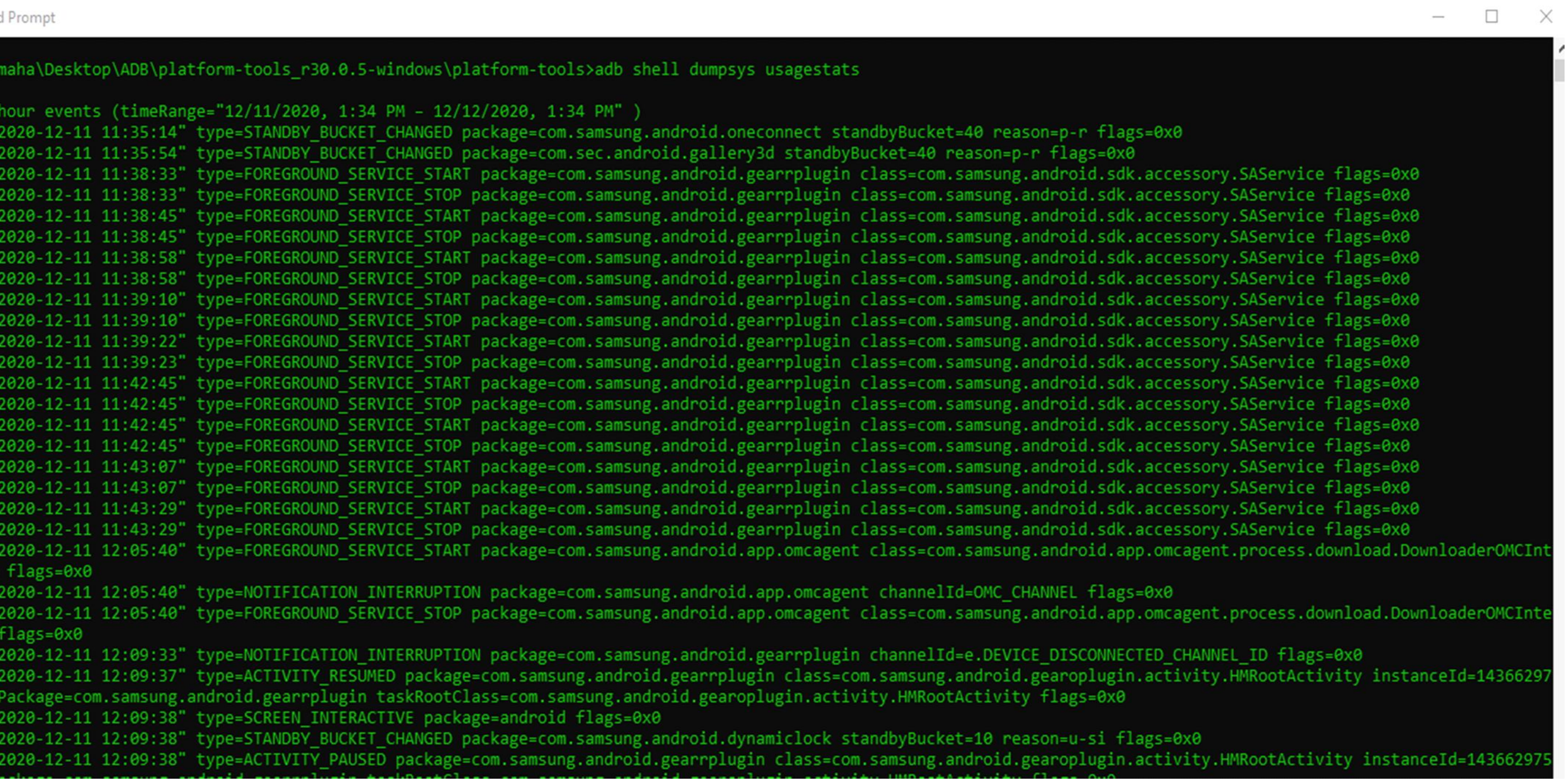
To export your results to a text file, simply refer to the command below.



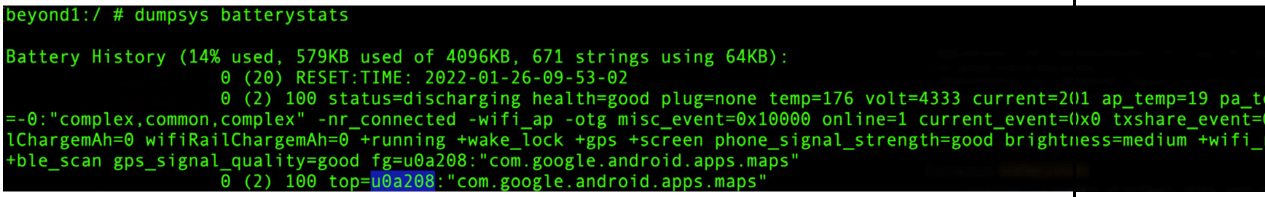
```
C:\Users\hmahalik>adb.exe shell dumpsys user
Users:
  UserInfo{0:felicia jones:13} serialNo=0
    Created: <unknown>
    Last logged in: +2h16m38s292ms ago
    Last logged in fingerprint: samsung/[redacted]
    Has profile owner: false
    Restrictions:
      none
    Device policy local restrictions:
      none
    Effective restrictions:
      none

  Device policy global restrictions:
    none

  Global restrictions owner id:-10000

  Guest restrictions:
    no_sms
    no_install_unknown_sources
    no_config_wifi
    no_outgoing_calls

  Device managed: false
  Started users state: {0=3}

  Max users: 1
  Supports switchable users: false
  All guests ephemeral: false
```

Next, I ran **adb.exe shell dumpsys user** to get a glimpse of my device and the user. We can see the name “Felicia Jones” and login dates and times and the fact that a fingerprint is being used to lock her devices. MDM is not being used and there is only one user on this device.

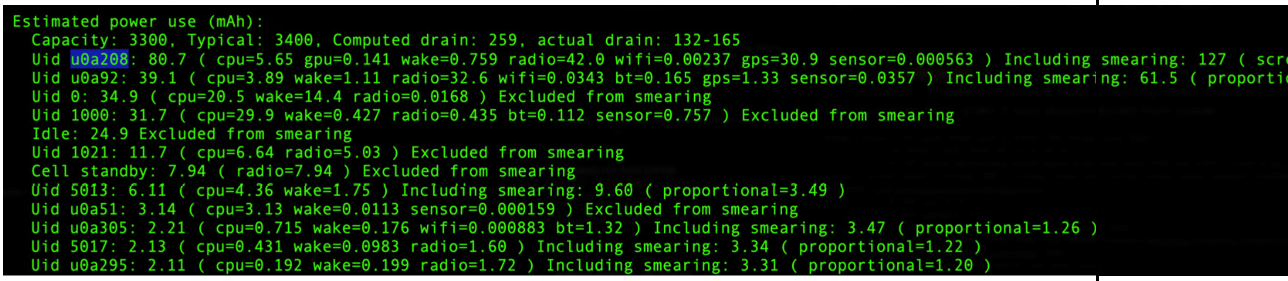
In this example here, we can see that **adb shell dumpsys usagestats** provided us with a listing of usage statistics. We can see some generic date times; let’s ignore those for now. In section 4, you are going to leverage scripts to parse this data thoroughly and to better understand how the data is stored and parsed. The Usagestats may provide insight on file paths and applications that are being used on the device. This includes apps that are running in the background.



```
maha\Desktop\ADB\platform-tools_r30.0.5-windows\platform-tools>adb shell dumpsys usagestats

hour events (timeRange="12/11/2020, 1:34 PM - 12/12/2020, 1:34 PM" )
2020-12-11 11:35:14" type=STANDBY_BUCKET_CHANGED package=com.samsung.android.oneconnect standbyBucket=40 reason=p-r flags=0x0
2020-12-11 11:35:54" type=STANDBY_BUCKET_CHANGED package=com.sec.android.gallery3d standbyBucket=40 reason=p-r flags=0x0
2020-12-11 11:38:33" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:38:33" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:38:45" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:38:45" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:38:58" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:38:58" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:39:10" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:39:10" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:39:22" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:39:23" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:42:45" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:42:45" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:42:45" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:42:45" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:43:07" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:43:07" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:43:29" type=FOREGROUND_SERVICE_START package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 11:43:29" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.gearrplugin class=com.samsung.android.sdk.accessory.SAService flags=0x0
2020-12-11 12:05:40" type=FOREGROUND_SERVICE_START package=com.samsung.android.app.omcagent class=com.samsung.android.app.omcagent.process.download.DownloaderOMCInt
 flags=0x0
2020-12-11 12:05:40" type=NOTIFICATION_INTERRUPTION package=com.samsung.android.app.omcagent channelId=OMC_CHANNEL flags=0x0
2020-12-11 12:05:40" type=FOREGROUND_SERVICE_STOP package=com.samsung.android.app.omcagent class=com.samsung.android.app.omcagent.process.download.DownloaderOMCInte
flags=0x0
2020-12-11 12:09:33" type=NOTIFICATION_INTERRUPTION package=com.samsung.android.gearrplugin channelId=e.DEVICE_DISCONNECTED_CHANNEL_ID flags=0x0
2020-12-11 12:09:37" type=ACTIVITY_RESUMED package=com.samsung.android.gearrplugin class=com.samsung.android.gearoplugin.activity.HMRootActivity instanceId=14366297
Package=com.samsung.android.gearrplugin taskRootClass=com.samsung.android.gearoplugin.activity.HMRootActivity flags=0x0
2020-12-11 12:09:38" type=SCREEN_INTERACTIVE package=android flags=0x0
2020-12-11 12:09:38" type=STANDBY_BUCKET_CHANGED package=com.samsung.android.dynamiclock standbyBucket=10 reason=u-si flags=0x0
2020-12-11 12:09:38" type=ACTIVITY_PAUSED package=com.samsung.android.gearrplugin class=com.samsung.android.gearoplugin.activity.HMRootActivity instanceId=143662975
```

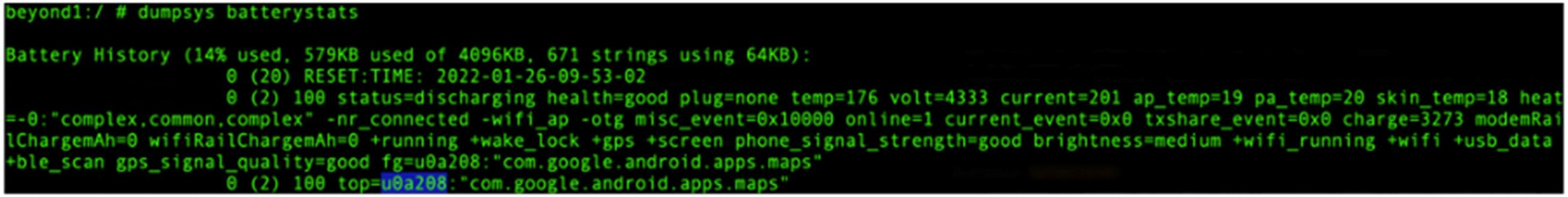


```
beyond1:/ # dumpsys batterystats

Battery History (14% used, 579KB used of 4096KB, 671 strings using 64KB):
                   0 (20) RESET:TIME: 2022-01-26-09-53-02
                   0 (2) 100 status=discharging health=good plug=none temp=176 volt=4333 current=2(1 ap_temp=19 pa_t
=-0:"complex,common,complex" -nr_connected -wifi_ap -otg misc_event=0x10000 online=1 current_event=(x0 txshare_event=
lChargemAh=0 wifiRailChargemAh=0 +running +wake_lock +gps +screen phone_signal_strength=good brightness=medium +wifi_
+ble_scan gps_signal_quality=good fg=u0a208:"com.google.android.apps.maps"
                   0 (2) 100 top=u0a208:"com.google.android.apps.maps"
```

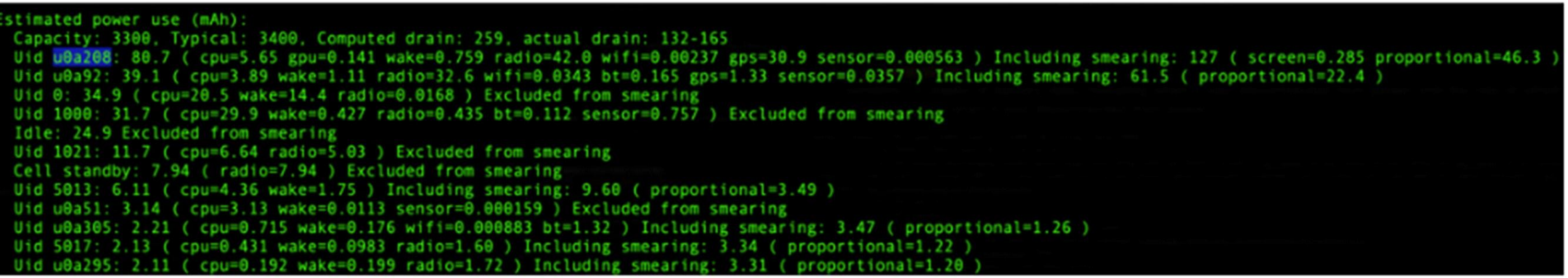


```
Estimated power use (mAh):
  Capacity: 3300, Typical: 3400, Computed drain: 259, actual drain: 132-165
  Uid u0a208: 80.7 ( cpu=5.65 gpu=0.141 wake=0.759 radio=42.0 wifi=0.00237 gps=30.9 sensor=0.000563 ) Including smearing: 127 ( scr
  Uid u0a92: 39.1 ( cpu=3.89 wake=1.11 radio=32.6 wifi=0.0343 bt=0.165 gps=1.33 sensor=0.0357 ) Including smearing: 61.5 ( proporti
  Uid 0: 34.9 ( cpu=20.5 wake=14.4 radio=0.0168 ) Excluded from smearing
  Uid 1000: 31.7 ( cpu=29.9 wake=0.427 radio=0.435 bt=0.112 sensor=0.757 ) Excluded from smearing
  Idle: 24.9 Excluded from smearing
  Uid 1021: 11.7 ( cpu=6.64 radio=5.03 ) Excluded from smearing
  Cell standby: 7.94 ( radio=7.94 ) Excluded from smearing
  Uid 5013: 6.11 ( cpu=4.36 wake=1.75 ) Including smearing: 9.60 ( proportional=3.49 )
  Uid u0a51: 3.14 ( cpu=3.13 wake=0.0113 sensor=0.000159 ) Excluded from smearing
  Uid u0a305: 2.21 ( cpu=0.715 wake=0.176 wifi=0.000883 bt=1.32 ) Including smearing: 3.47 ( proportional=1.26 )
  Uid 5017: 2.13 ( cpu=0.431 wake=0.0983 radio=1.60 ) Including smearing: 3.34 ( proportional=1.22 )
  Uid u0a295: 2.11 ( cpu=0.192 wake=0.199 radio=1.72 ) Including smearing: 3.31 ( proportional=1.20 )
```

The command **adb shell dumpsys battery stats** will show stats per package to help you determine package usage per application. The results may contain up to 2 weeks of information starting with the most recent. In this example we see the UID for Google Maps. This UID will be consistent throughout this file as discussed in section 1 of this course. More information on battery usage and battery stats will be provided toward the end of this section. A good reference is https://github.com/google/battery-historian.



```
beyond1:/ # dumpsys batterystats

Battery History (14% used, 579KB used of 4096KB, 671 strings using 64KB):
                    0 (20) RESET:TIME: 2022-01-26-09-53-02
                    0 (2) 100 status=discharging health=good plug=none temp=176 volt=4333 current=201 ap_temp=19 pa_temp=20 skin_temp=18 heat
=-0:"complex,common,complex" -nr_connected -wifi_ap -otg misc_event=0x10000 online=1 current_event=0x0 txshare_event=0x0 charge=3273 modemRai
lChargemAh=0 wifiRailChargemAh=0 +running +wake_lock +gps +screen phone_signal_strength=good brightness=medium +wifi_running +wifi +usb_data
+ble_scan gps_signal_quality=good fg=u0a208:"com.google.android.apps.maps"
                    0 (2) 100 top=u0a208:"com.google.android.apps.maps"
```



```
stimated power use (mAh):
  Capacity: 3300, Typical: 3400, Computed drain: 259, actual drain: 132-165
  Uid u0a208: 80.7 ( cpu=5.65 gpu=0.141 wake=0.759 radio=42.0 wifi=0.00237 gps=30.9 sensor=0.000563 ) Including smearing: 127 ( screen=0.285 proportional=46.3 )
  Uid u0a92: 39.1 ( cpu=3.89 wake=1.11 radio=32.6 wifi=0.0343 bt=0.165 gps=1.33 sensor=0.0357 ) Including smearing: 61.5 ( proportional=22.4 )
  Uid 0: 34.9 ( cpu=20.5 wake=14.4 radio=0.0168 ) Excluded from smearing
  Uid 1000: 31.7 ( cpu=29.9 wake=0.427 radio=0.435 bt=0.112 sensor=0.757 ) Excluded from smearing
  Idle: 24.9 Excluded from smearing
  Uid 1021: 11.7 ( cpu=6.64 radio=5.03 ) Excluded from smearing
  Cell standby: 7.94 ( radio=7.94 ) Excluded from smearing
  Uid 5013: 6.11 ( cpu=4.36 wake=1.75 ) Including smearing: 9.60 ( proportional=3.49 )
  Uid u0a51: 3.14 ( cpu=3.13 wake=0.0113 sensor=0.000159 ) Excluded from smearing
  Uid u0a305: 2.21 ( cpu=0.715 wake=0.176 wifi=0.000883 bt=1.32 ) Including smearing: 3.47 ( proportional=1.26 )
  Uid 5017: 2.13 ( cpu=0.431 wake=0.0983 radio=1.60 ) Including smearing: 3.34 ( proportional=1.22 )
  Uid u0a295: 2.11 ( cpu=0.192 wake=0.199 radio=1.72 ) Including smearing: 3.31 ( proportional=1.20 )
```

- Developed by Mattia Epifani
- Works on Linux and Mac (and Windows with a Linux subsystem)
- Supports:
  - Linux commands
  - Dumpsys and Bugreport extractions
  - ADB Backup
  - MTP Extraction (emulated SD Card)
  - System partition extraction
  - Content Providers extraction

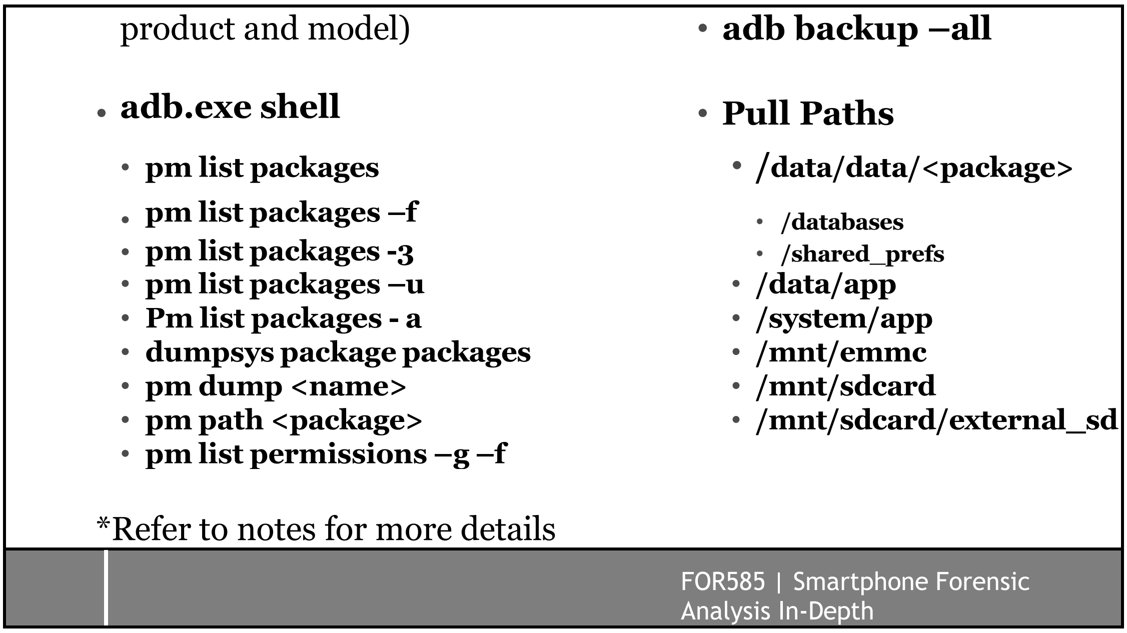
Android Triage, the script developed by Mattia Epifani, will run on Linux and Mac (and Windows with Linux Subsystem). His blog can be found at: https://blog.digital-forensics.it/2021/03/triaging-modern-android-devices-aka.html

And the GitHub link: https://github.com/RealityNet/android_triage

Mattia presented this information and script at the 2021 DFIR Summit: https://www.youtube.com/watch?v=gXN4rRs77Ts

- **adb devices**
- **adb devices –l** (device listed with

- **adb pull <file> <local>**
- **adb backup –shared**



product and model)

- **adb.exe shell**
  - **pm list packages**
  - **pm list packages –f**
  - **pm list packages -3**
  - **pm list packages –u**
  - **Pm list packages - a**
  - **dumpsys package packages**
  - **pm dump <name>**
  - **pm path <package>**
  - **pm list permissions –g –f**

*Refer to notes for more details

- **adb backup –all**
- **Pull Paths**
  - **/data/data/<package>**
    - **/databases**
    - **/shared_prefs**
  - **/data/app**
  - **/system/app**
  - **/mnt/emmc**
  - **/mnt/sdcard**
  - **/mnt/sdcard/external_sd**

FOR585 | Smartphone Forensic Analysis In-Depth

The following adb commands may be of use to you.

adb devices (list of attached devices with serial number
adb devices –l (attached devices listed with product and model)
adb.exe shell pm list packages (list package name)
adb.exe shell pm list packages –f (list package name and include file path)
adb.exe shell pm list packages – a (show hidden files)
adb.exe shell pm list packages -3 (list third-party package names)

adb.exe shell pm list packages –u (list uninstalled package names)
dumpsys package packages (list information on all applications)
adb.exe shell pm dump <name> (list information on one specific package) **adb.exe shell pm path <package>** (list path to application apk file) **adb.exe shell pm list permissions –g –f** (list permission details)

adb pull <file>

<local> adb backup –shared adb backup –all

Pull Paths

- **/data/data/<package>**
  - **/databases**
  - **/shared_prefs**
- **/data/app**
- **/system/app**
- **/mnt/emmc**
- **/mnt/sdcard**
- **/mnt/sdcard/external_sd**

If adb keeps crashing or will not recognize, try the following:

adb

```
kill-server

adb

start-server
```

For more information, refer to Heather Mahalik’s co-authored Practical Mobile Forensics, 2nd edition.

This page intentionally left blank.

- Alphanumeric passcode
- Swipe/pattern lock
- PIN lock
- Face Unlock
- Fingerprint scan

There are several lock options for Android devices. The first is the pattern lock (swipe code), which was introduced by Android. The others include a simple PIN lock, alphanumeric passcode, and biometric locks. The alphanumeric passcode is the hardest to crack. Biometric locks are backed with a pattern lock or a PIN/password. So, they are as difficult to crack as the backup method for protecting the biometrics. Make sure you never take a device that someone unlocks for you without obtaining the passcode! Android 8 required you to enter the passcode to enable Developer Mode, even on unlocked devices.

Currently, tools like Hashcat, Andriller, Oxygen, XRY, and Cellebrite can crack both PIN locks and pattern locks for most devices. The USERDATA/system/

directory holds the gesture.key, password.key, gatekeeper.password.key, and *.key files.1 If these keys are not present, it is because the device was never locked by the user, or your tool extraction wasn't capable of extracting the required files. If the user locks their device and then removes the lock, the files still exist. The gesture.key contains the hash of the pattern. The forensic tools simply decode the hash to obtain the pattern lock. The password.key is also hashed, which is comprised of the password and the salt from the USERDATA/data/com.android.providers.settings/databases/settings.db.2 This file no longer stores the password salt on modern Android devices. The salt can be recovered from /USERDATA/system/locksettings.db. When extracting these two databases, always grab the WAL just in case the salt isn't committed to the database when extraction occurs. For these situations, the WAL will store the salt. When gatekeeper is involved, these rules may not apply.

For devices you cannot bypass, services are offered by CAS (Cellebrite Advanced Services) to unlock most Android phones, even those that are encrypted.

Best practices are to grab any .key-related file that the device may be using to store the password or backup password/PIN hash. These are commonly password.key, fmmpassword.key, fingerprintpassword.key, and sparepassword.key. Beware that some keys may not be accessible on encrypted devices or those being protected by Gatekeeper. The Gatekeeper (introduced with Android 6 Marshmallow) prevents the password from being salted and then cracked by the tools. The password/PIN/pattern is authenticated in a Trusted Execution Environment (TEE), which is essentially a hardware-encoded secret key required to crack the lock.[4] Refer to the blog by Magnet for more information on Gatekeeper: https://www.magnetforensics.com/blog/gatekeeper-password-storage-android-secures-devices/. We will also continue to do research on Gatekeeper, so keep your eye out for blog posts on SmarterForensics.com/blog.

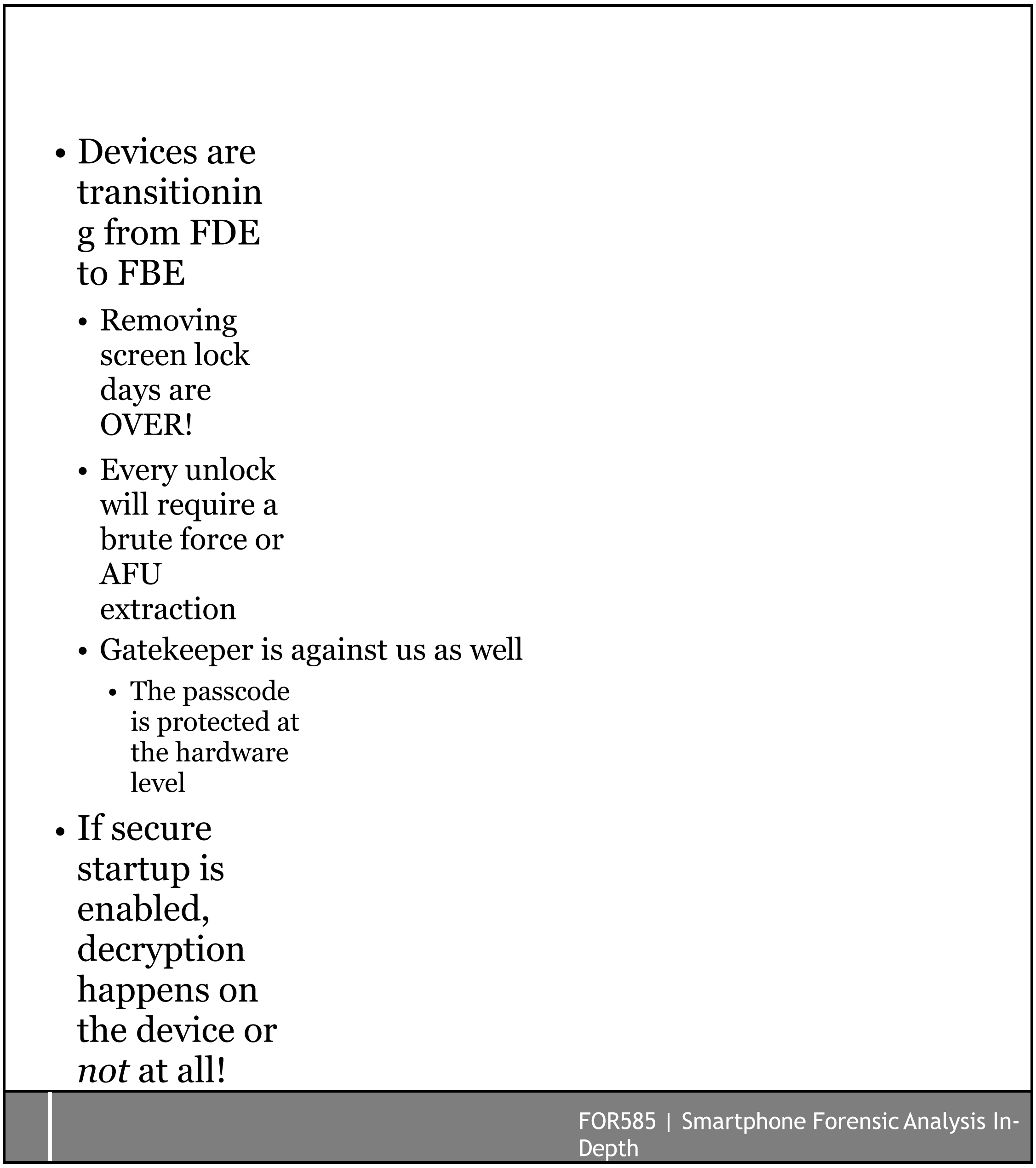
For all .key files you can pull, do it. Tools like Andriller and Hashcat will also accept these files for password cracking. To know which one is in use, start with examining the file size or look at the device if it's readily available. A file size of 0 bytes indicates that file is not in use. Any Hex editor can be used to examine these files.

If the device is rooted, these lock files can be removed via ADB. Make sure you practice on a test device before trying this on live evidence. Again, if you cannot acquire these files from the device, you cannot magically unlock the device for acquisition.

Fingerprint, complex passcodes, and Face Unlock are other options. For Face Unlock settings, the user's face is detected, and the device unlocks. To protect a user from someone using a picture to access her device, the Face Liveness Check was introduced. The liveness check requires the user to blink when using the Face Unlock method.[3]

References:

- https://for585.com/hashcat
- https://for585.com/erge (Face Unlock article)
- https://for585.com/gatekeeper



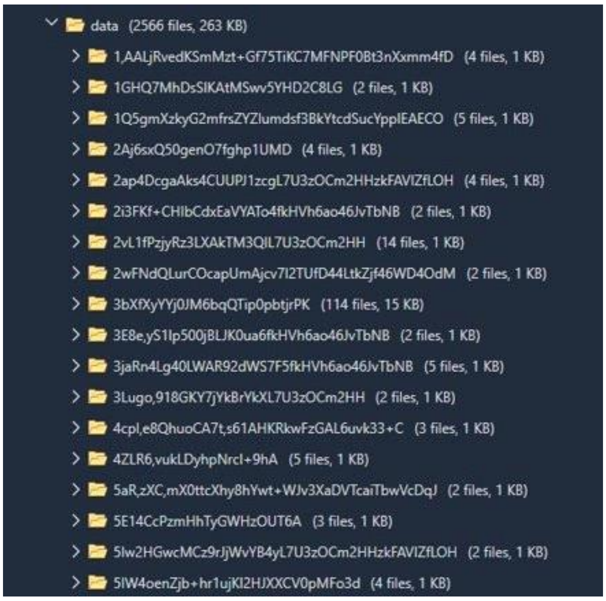
- Devices are transitioning from FDE to FBE
  - Removing screen lock days are OVER!
  - Every unlock will require a brute force or AFU extraction
  - Gatekeeper is against us as well
    - The passcode is protected at the hardware level
- If secure startup is enabled, decryption happens on the device or *not* at all!

FOR585 | Smartphone Forensic Analysis In-Depth

The days of simply leveraging the locksettings.db to crack the passcode or removing the gesture.key files are over. Sadly, for us forensicators, file-based encryption has killed the days of us successfully extracting and cracking the passcode. Samsung is in the process of transitioning all devices to FBE; Motorola, OnePlus and others have already made the move. LG has not because they stopped manufacturing devices as of June 2021. Which means, if you come across an LG, you could possibly leverage older methods to extract the passcode.

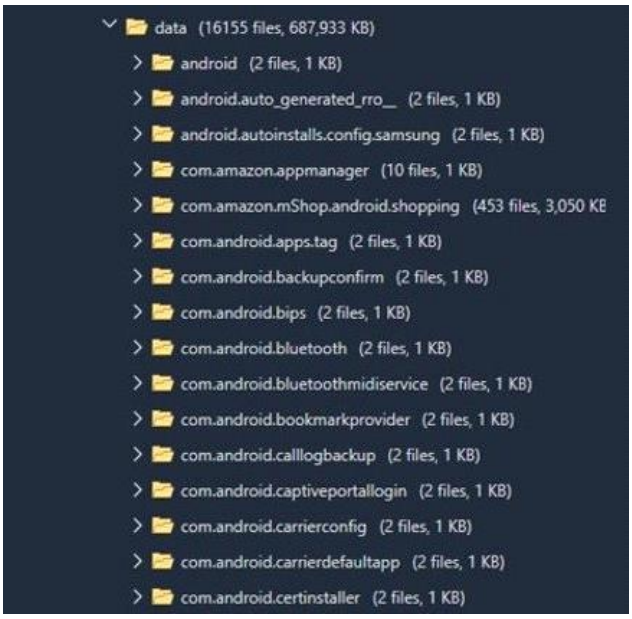
These steps are included below (just in case you get lucky) and there is a bonus lab within your media files for you to try it out!

- BFU



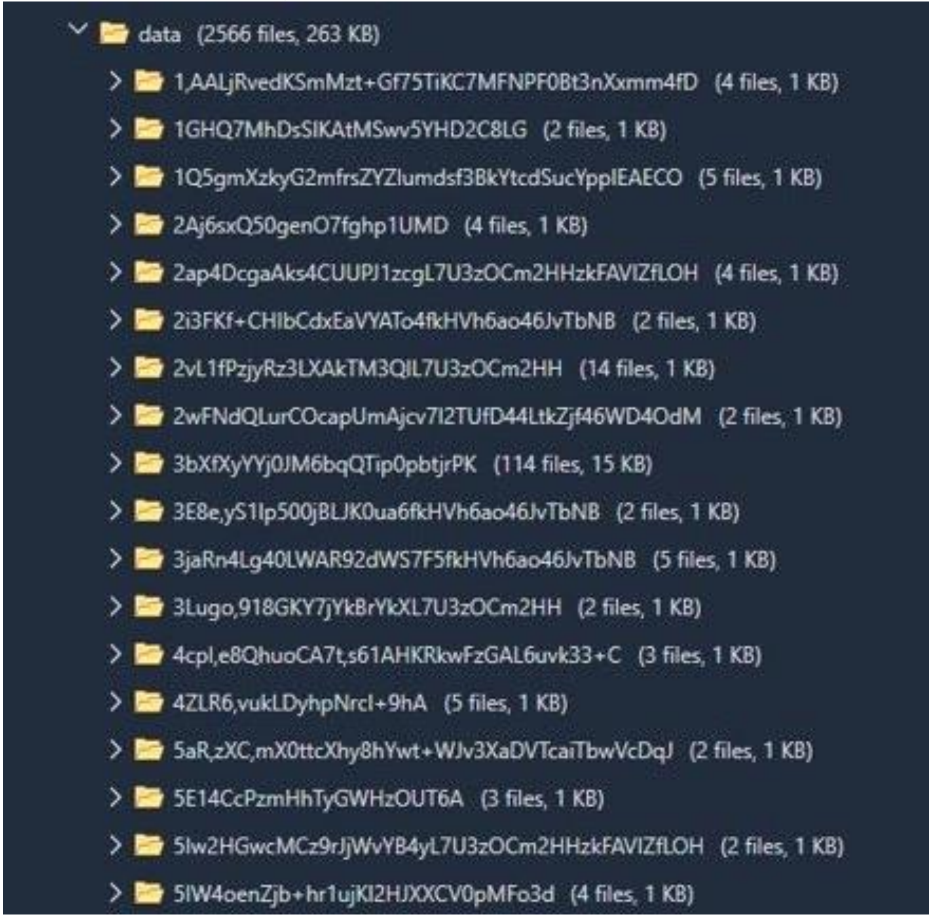
```
v data  (2566 files, 263 KB)
    > 1,AALjRvedKSmMzt+Gf75TiKC7MFNPF0Bt3nXxmm4fD  (4 files, 1 KB)
    > 1GHQ7MhDsSIKAtMSwv5YHD2C8LG  (2 files, 1 KB)
    > 1Q5gmXzkyG2mfrsZYZlumdsf3BkYtcdSucYpplEAECO  (5 files, 1 KB)
    > 2Aj6sxQ50genO7fghp1UMD  (4 files, 1 KB)
    > 2ap4DcgaAks4CUUPJ1zcgL7U3zOCm2HHzkFAVIZfLOH  (4 files, 1 KB)
    > 2i3FKf+CHIbCdxEaVYATo4fkHVh6ao46JvTbNB  (2 files, 1 KB)
    > 2vL1fPzjyRz3LXAkTM3QIL7U3zOCm2HH  (14 files, 1 KB)
    > 2wFNdQLurCOcapUmAjcv7I2TUfD44LtkZjf46WD4OdM  (2 files, 1 KB)
    > 3bXfXyYYj0JM6bqQTip0pbtjrPK  (114 files, 15 KB)
    > 3E8e,yS1lp500jBLJK0ua6fkHVh6ao46JvTbNB  (2 files, 1 KB)
    > 3jaRn4Lg40LWAR92dWS7F5fkHVh6ao46JvTbNB  (5 files, 1 KB)
    > 3Lugo,918GKY7jYkBrYkXL7U3zOCm2HH  (2 files, 1 KB)
    > 4cpl,e8QhuoCA7t,s61AHKRkwFzGAL6uvk33+C  (3 files, 1 KB)
    > 4ZLR6,vukLDyhpNrcl+9hA  (5 files, 1 KB)
    > 5aR,zXC,mX0ttcXhy8hYwt+WJv3XaDVTcaiTbwVcDqJ  (2 files, 1 KB)
    > 5E14CcPzmHhTyGWHzOUT6A  (3 files, 1 KB)
    > 5lw2HGwcMCz9rJjWvYB4yL7U3zOCm2HHzkFAVIZfLOH  (2 files, 1 KB)
    > 5lW4oenZjb+hr1ujKI2HJXXCV0pMFo3d  (4 files, 1 KB)
```

- AFU



```
v data  (16155 files, 687,933 KB)
    > android  (2 files, 1 KB)
    > android.auto_generated_rro__  (2 files, 1 KB)
    > android.autoinstalls.config.samsung  (2 files, 1 KB)
    > com.amazon.appmanager  (10 files, 1 KB)
    > com.amazon.mShop.android.shopping  (453 files, 3,050 KE
    > com.android.apps.tag  (2 files, 1 KB)
    > com.android.backupconfirm  (2 files, 1 KB)
    > com.android.bips  (2 files, 1 KB)
    > com.android.bluetooth  (2 files, 1 KB)
    > com.android.bluetoothmidiservice  (2 files, 1 KB)
    > com.android.bookmarkprovider  (2 files, 1 KB)
    > com.android.calllogbackup  (2 files, 1 KB)
    > com.android.captiveportallogin  (2 files, 1 KB)
    > com.android.carrierconfig  (2 files, 1 KB)
    > com.android.carrierdefaultapp  (2 files, 1 KB)
    > com.android.certinstaller  (2 files, 1 KB)
```

The reality is that most of the data extracted from a BFU is encrypted, which is shown above. However, you may get access to some data to include accounts_ce.db, some configuration files, possible usernames and if you are lucky – passwords! It is worth a shot if you are locked out and stuck. Bottom line, an AFU is what everyone wants and may not be accessible. For AFU, we leverage the keys stored in RAM to access the data. Remember what you learned in Section 1 about hot and cold devices? How you handle the device could determine your level of access.



```
data (2566 files, 263 KB)
  1,AALjRvedKSmMzt+Gf75TiKC7MFNPF0Bt3nXxmm4fD (4 files, 1 KB)
  1GHQ7MhDsSIKAtMSwv5YHD2C8LG (2 files, 1 KB)
  1Q5gmXzkyG2mfrsZYZlumdsf3BkYtcdSucYpplEAECO (5 files, 1 KB)
  2Aj6sxQ50genO7fghp1UMD (4 files, 1 KB)
  2ap4DcgaAks4CUUPJ1zcgL7U3zOCm2HHzkFAVIZfLOH (4 files, 1 KB)
  2i3FKf+CHIbCdxEaVYATo4fkHVh6ao46JvTbNB (2 files, 1 KB)
  2vL1fPzjyRz3LXAkTM3QIL7U3zOCm2HH (14 files, 1 KB)
  2wFNdQLurCOcapUmAjcv7I2TUfD44LtkZjf46WD4OdM (2 files, 1 KB)
  3bXfXyYYj0JM6bqQTip0pbtjrPK (114 files, 15 KB)
  3E8e,yS1Ip500jBLJK0ua6fkHVh6ao46JvTbNB (2 files, 1 KB)
  3jaRn4Lg40LWAR92dWS7F5fkHVh6ao46JvTbNB (5 files, 1 KB)
  3Lugo,918GKY7jYkBrYkXL7U3zOCm2HH (2 files, 1 KB)
  4cpl,e8QhuoCA7t,s61AHKRkwFzGAL6uvk33+C (3 files, 1 KB)
  4ZLR6,vukLDyhpNrcl+9hA (5 files, 1 KB)
  5aR,zXC,mX0ttcXhy8hYwt+WJv3XaDVTcaiTbwVcDqJ (2 files, 1 KB)
  5E14CcPzmHhTyGWHzOUT6A (3 files, 1 KB)
  5lw2HGwcMCz9rJjWvYB4yL7U3zOCm2HHzkFAVIZfLOH (2 files, 1 KB)
  5lW4oenZjb+hr1ujKI2HJXXCV0pMFo3d (4 files, 1 KB)
```

- Is USB Debugging enabled?

System

- Is the device unlocked?
- Can you manually inspect the

smallest footprint device?

Logical/Backup

- Will the acquisition method root the

- Physical or Full File

Acquisition

- Backup –
- Advanced
- Acquire SD card through

the device
device?

separately (if

- Are you willing to risk rooting it yourself?
- What forensic footprint will be left generic Android on behind?
- Will encryption prevent data access?

cables?

- FBE vs FDE

device?

and SD card and SIM

time allows)

- Did you try UFED?
- Did you try multiple
- Are you willing to root the

The first step in acquiring an Android device is to determine if USB Debugging is enabled. If the device is locked, this may not be possible to manually determine. If the device is locked, simply try to acquire the Android. If USB Debugging is enabled, you can bypass any lock on the device. All communications must be disabled on the device. Make sure that you are aware of the forensic acquisition tool you choose to acquire the device and understand the actions taken against the device to obtain the acquisition. (For example, will the tool root the device? Will the root be permanent?).

The preferred acquisition order is normally physical, followed by a file system should the physical fail. However, if you conduct covert operations, a physical may leave the largest footprint and you may want to avoid that. A logical acquisition should be performed every time if you are worried about encryption! The logical acquisition provides examiners with pointers for parsing and decoding raw images that aren't supported by the tool. Finally, the SD card and SIM card(s) should be removed and acquired using a standard forensic tool. FTK Imager is free and works well. Acquiring the SD card through the device creates longer acquisition times and may update the Last Accessed timestamps for the files resident on the SD card. The larger the SD card, the longer the acquisition. For more information on recommended steps, check out the blog posts by Heather Mahalik on smarterforensics.com/blogs.

This page intentionally left blank.

- Most commonly used Linux file system
- Well supported in all Android devices
- Updated features (encryption)
- No size limitations (used to be limited to 4 GB)
- Extremely stable
- Journaling can actually harm the lifespan of the device if in use

- Supported for mounting in most tools

EXT4 is the most common file system found on Android devices. This file system is not only popular among modern smartphones, but it is the most commonly used Linux file system. Updated features in EXT4 were added, and the most well-known included encryption. Unlike older EXT versions, there is no 4 GB size limitation for file storage. While this file system is extremely stable, the use of journaling, if enabled, can wear down the device faster than if journaling is disabled.[1]

Reference:
[1] https://for585.com/extfilesystem

- Directories may differ in tools
- Partitions remain the same

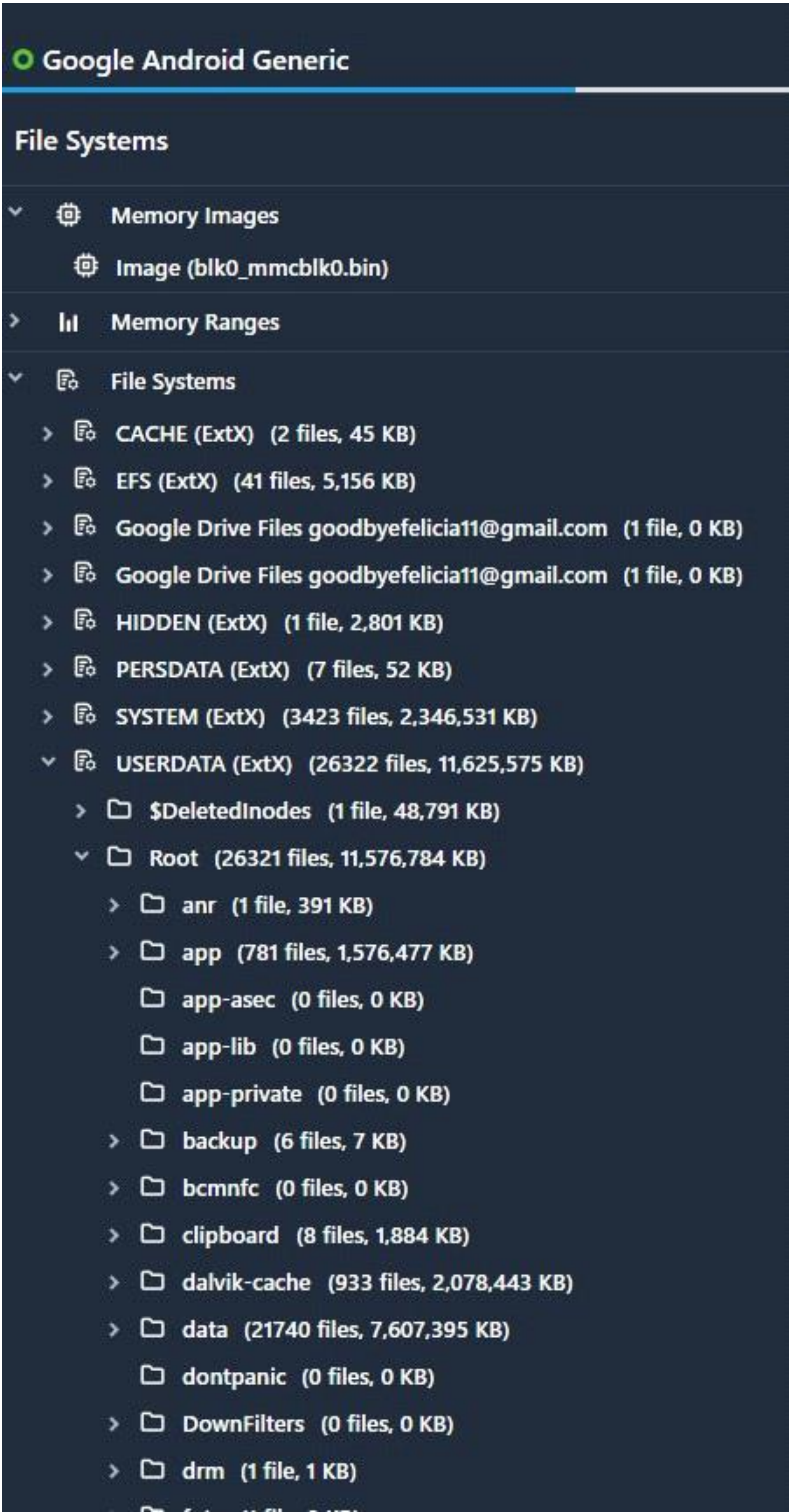
- Data, System, Cache, SD, and so on

- May require manual carving and examination

The best and most comprehensive search should be conducted within the raw image file. The Memory Ranges or partitions are comprised of the USERDATA, EFS, system, SD card, CACHE, and other partitions in their raw form, while the File System shows a normalized version of the same partitions, including a folder structure for each. Autopsy places the name of the partition next to the volume label. Each tool may differ with their methods, so it’s important that you know where to look to ensure you don’t miss any data of forensic value. The File

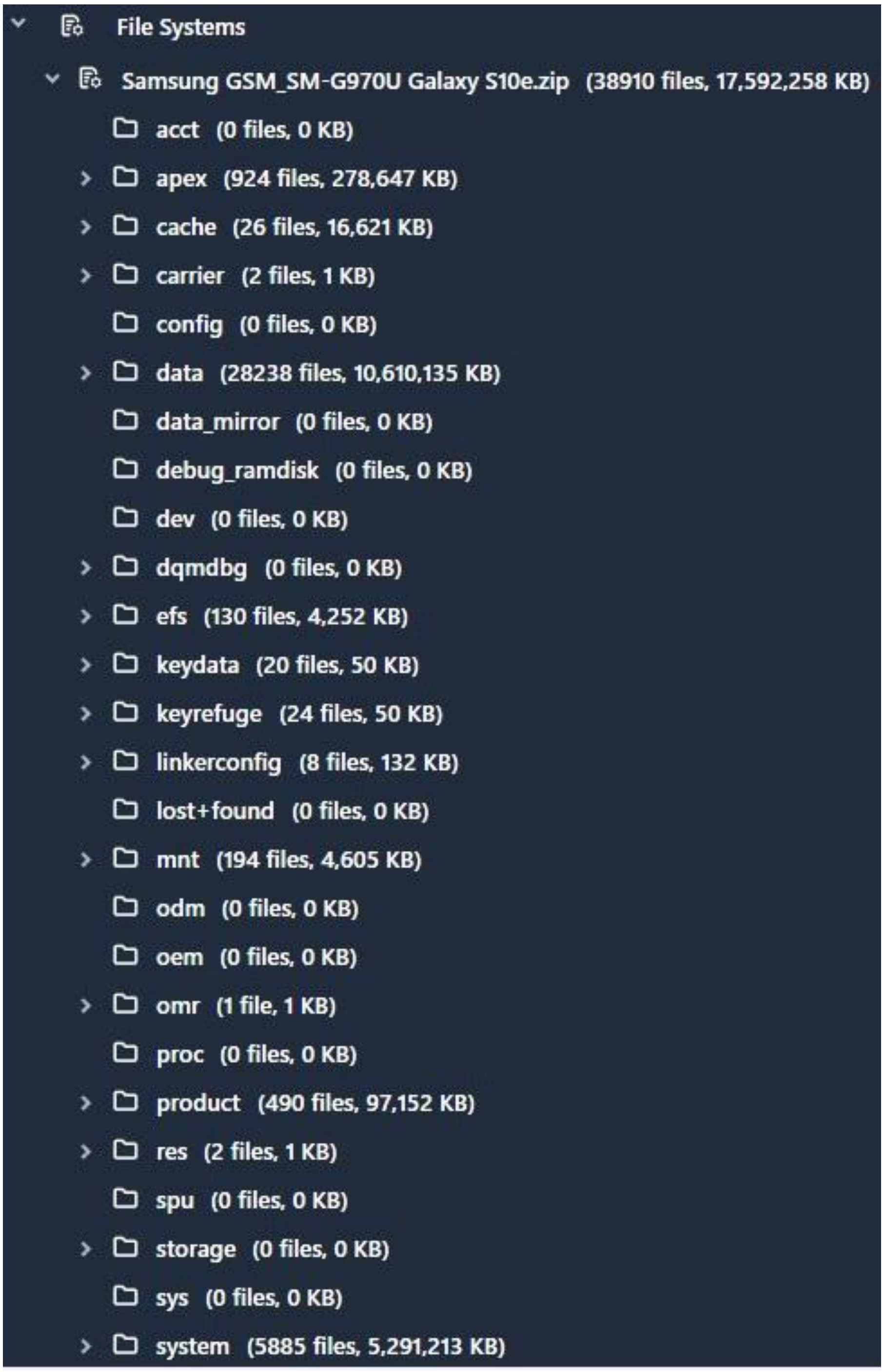
System area of the data shown in Physical Analyzer is the reconstructed file system, as interpreted by the tool. Data is stored natively within the partitions and can be manually examined and decoded for relevance, which is discussed later in this section.

The Analyzed Data section in Physical Analyzer shows all the data that Physical Analyzer knows how to parse. Again, one must use caution here and verify their findings rather than just trusting the tool. More details on how to verify your results and manually decode data will be covered in the analytical portion of this section. Commercial tools will not parse all forms of data available on the Android device. In this section, we will cover different tools and how the data will look in each. You will gain experience with this in your lab work. The examiner must dig deeper to recover the overlooked data. For your lab later in this section, you will have access to a newer device that was physically acquired.

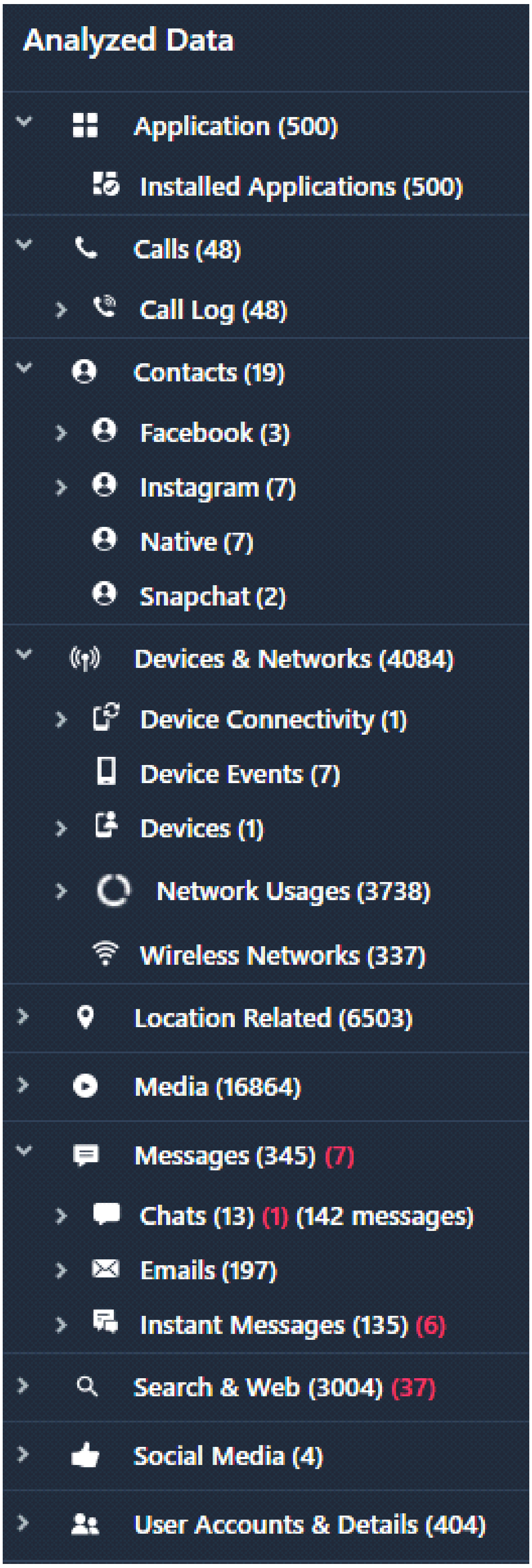


```
Google Android Generic

File Systems

Memory Images
    Image (blk0_mmcblk0.bin)
Memory Ranges
File Systems
    CACHE (ExtX)  (2 files, 45 KB)
    EFS (ExtX)  (41 files, 5,156 KB)
    Google Drive Files goodbyefelicia11@gmail.com  (1 file, 0 KB)
    Google Drive Files goodbyefelicia11@gmail.com  (1 file, 0 KB)
    HIDDEN (ExtX)  (1 file, 2,801 KB)
    PERSDATA (ExtX)  (7 files, 52 KB)
    SYSTEM (ExtX)  (3423 files, 2,346,531 KB)
    USERDATA (ExtX)  (26322 files, 11,625,575 KB)
        $DeletedInodes  (1 file, 48,791 KB)
        Root  (26321 files, 11,576,784 KB)
            anr  (1 file, 391 KB)
            app  (781 files, 1,576,477 KB)
            app-asec  (0 files, 0 KB)
            app-lib  (0 files, 0 KB)
            app-private  (0 files, 0 KB)
            backup  (6 files, 7 KB)
            bcmnfc  (0 files, 0 KB)
            clipboard  (8 files, 1,884 KB)
            dalvik-cache  (933 files, 2,078,443 KB)
            data  (21740 files, 7,607,395 KB)
            dontpanic  (0 files, 0 KB)
            DownFilters  (0 files, 0 KB)
            drm  (1 file, 1 KB)
```

Full file system extractions are the next best acquisition when compared to physical acquisition. However, this extraction method may be the best way to obtain the most information from FBE (File Based Encryption) Android devices. When you get Full file system access, you should expect to pull application and data files from the device. Nothing is guaranteed, unfortunately. This is where it depends on the application developer and whether or not they elect to have their data pulled via an adb backup or that level of extraction. If you are not able to see application data here, you may have to resort to chat capture or obtaining cloud data. Bottom line, you never know what you are going to get! Get as many acquisitions as you can to ensure you get data to examine.

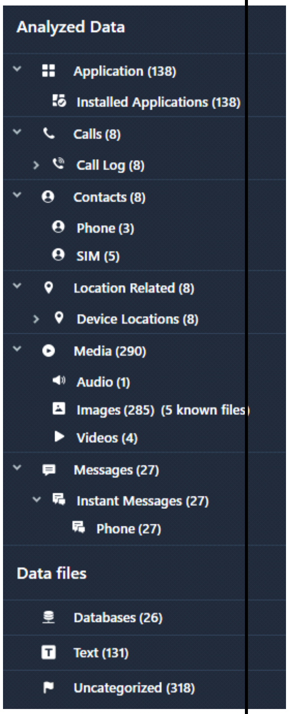


```
File Systems
  Samsung GSM_SM-G970U Galaxy S10e.zip  (38910 files, 17,592,258 KB)
      acct  (0 files, 0 KB)
      apex  (924 files, 278,647 KB)
      cache  (26 files, 16,621 KB)
      carrier  (2 files, 1 KB)
      config  (0 files, 0 KB)
      data  (28238 files, 10,610,135 KB)
      data_mirror  (0 files, 0 KB)
      debug_ramdisk  (0 files, 0 KB)
      dev  (0 files, 0 KB)
      dqmdbg  (0 files, 0 KB)
      efs  (130 files, 4,252 KB)
      keydata  (20 files, 50 KB)
      keyrefuge  (24 files, 50 KB)
      linkerconfig  (8 files, 132 KB)
      lost+found  (0 files, 0 KB)
      mnt  (194 files, 4,605 KB)
      odm  (0 files, 0 KB)
      oem  (0 files, 0 KB)
      omr  (1 file, 1 KB)
      proc  (0 files, 0 KB)
      product  (490 files, 97,152 KB)
      res  (2 files, 1 KB)
      spu  (0 files, 0 KB)
      storage  (0 files, 0 KB)
      sys  (0 files, 0 KB)
      system  (5885 files, 5,291,213 KB)
```



Analyzed Data

- Application (500)
  - Installed Applications (500)
- Calls (48)
  - Call Log (48)
- Contacts (19)
  - Facebook (3)
  - Instagram (7)
  - Native (7)
  - Snapchat (2)
- Devices & Networks (4084)
  - Device Connectivity (1)
  - Device Events (7)
  - Devices (1)
  - Network Usages (3738)
  - Wireless Networks (337)
- Location Related (6503)
- Media (16864)
- Messages (345) (7)
  - Chats (13) (1) (142 messages)
  - Emails (197)
  - Instant Messages (135) (6)
- Search & Web (3004) (37)
- Social Media (4)
- User Accounts & Details (404)



Analyzed Data

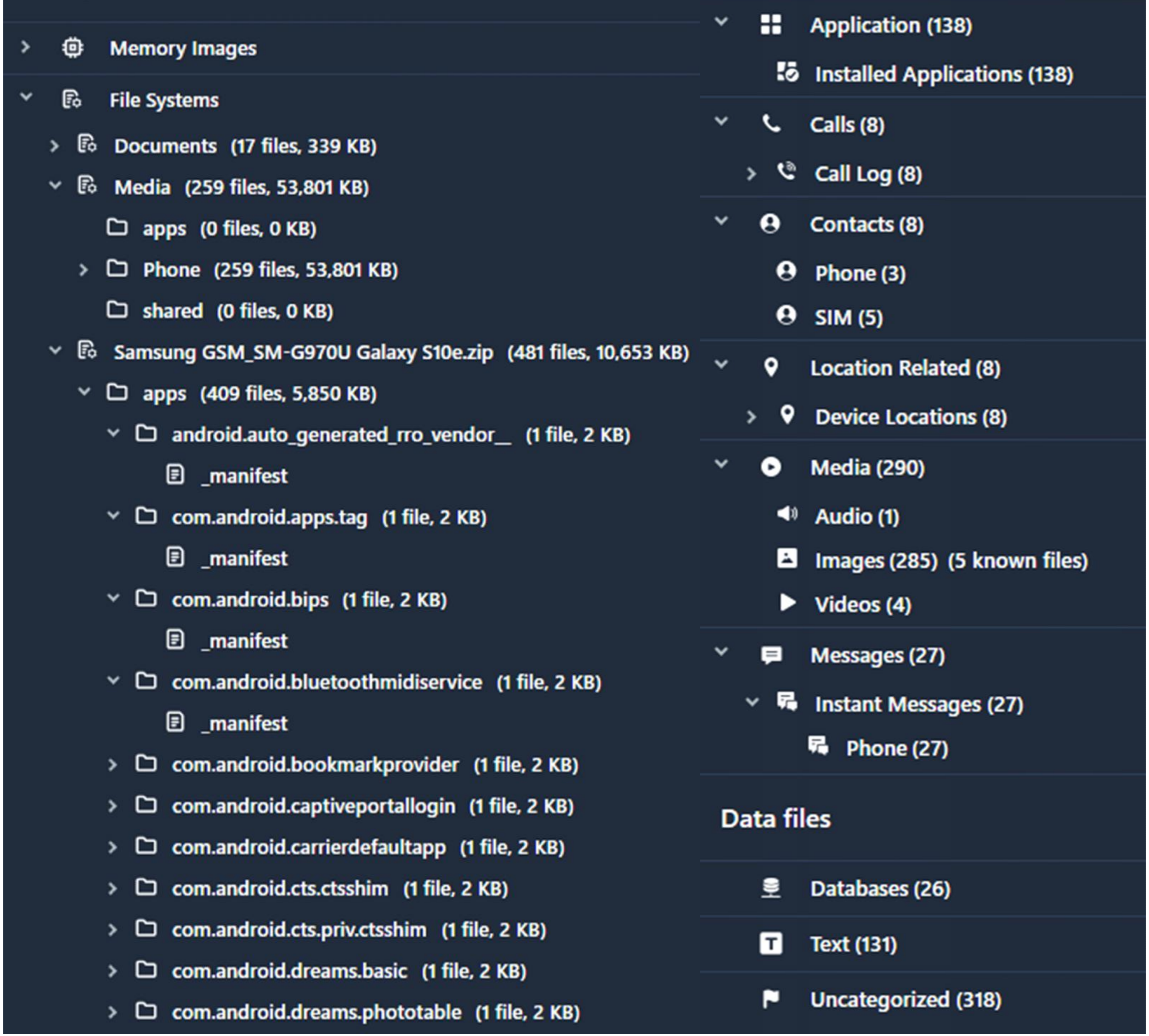
- Application (138)
  - Installed Applications (138)
- Calls (8)
  - Call Log (8)
- Contacts (8)
  - Phone (3)
  - SIM (5)
- Location Related (8)
  - Device Locations (8)
- Media (290)
  - Audio (1)
  - Images (285) (5 known files
  - Videos (4)
- Messages (27)
  - Instant Messages (27)
    - Phone (27)

Data files

- Databases (26)
- Text (131)
- Uncategorized (318)

Advanced logical extractions will leave you with the least amount of data to examine. As discussed in the previous slide, application data created by the user is often not extracted with logical or adb backup extractions. This pertains to Advanced logical extractions as well. When this type of acquisition is your only option, refer back to this section where we discuss methods to obtain more.

As seen in this slide, you will obtain a file system structure and Analyzed data that can be examined in keyword searched. Just keep in mind, you may be searching for something that was not extracted.



```
> Memory Images
v File Systems
  > Documents  (17 files, 339 KB)
  v Media  (259 files, 53,801 KB)
      apps  (0 files, 0 KB)
    > Phone  (259 files, 53,801 KB)
      shared  (0 files, 0 KB)
  v Samsung GSM_SM-G970U Galaxy S10e.zip  (481 files, 10,653 KB)
    v apps  (409 files, 5,850 KB)
      v android.auto_generated_rro_vendor__  (1 file, 2 KB)
          _manifest
      v com.android.apps.tag  (1 file, 2 KB)
          _manifest
      v com.android.bips  (1 file, 2 KB)
          _manifest
      v com.android.bluetoothmidiservice  (1 file, 2 KB)
          _manifest
      > com.android.bookmarkprovider  (1 file, 2 KB)
      > com.android.captiveportallogin  (1 file, 2 KB)
      > com.android.carrierdefaultapp  (1 file, 2 KB)
      > com.android.cts.ctsshim  (1 file, 2 KB)
      > com.android.cts.priv.ctsshim  (1 file, 2 KB)
      > com.android.dreams.basic  (1 file, 2 KB)
      > com.android.dreams.phototable  (1 file, 2 KB)
```

```
v Application (138)
    Installed Applications (138)
v Calls (8)
  > Call Log (8)
v Contacts (8)
    Phone (3)
    SIM (5)
v Location Related (8)
  > Device Locations (8)
v Media (290)
    Audio (1)
    Images (285)  (5 known files)
    Videos (4)
v Messages (27)
  v Instant Messages (27)
      Phone (27)

Data files

  Databases (26)
  Text (131)
  Uncategorized (318)
```

This page intentionally left blank.

The most common evidentiary locations found on Android devices are listed in this slide. Again, depending on the device and version of Android, variations may occur, and you may see information that isn't listed in this slide. The list below contains more directories than the screenshot above. The most important areas to examine are highlighted here (**bold**).[1] The circled items represent locations that may be used by malware. Examine these locations extensively if you are working a malware investigation. While these locations are common, malware may exist in other locations on the device.

NOTE: Depending on your forensic tool or method for examining the file system, the paths may vary. A simple keyword search will get you to the correct location if you find you are getting lost in the file system. Below, the true path is reflected as if the data were pulled directly from the device and not normalized by a forensic tool. Keep in mind that all partitions may not be available for examination.

/(Root)

- **/CACHE** -> contains Gmail attachments, downloads, browser data, and OTA updates (may not exist on devices)
- /EFS -> (Encrypted File System) contains files needed for the device to function in case of failure (Bluetooth address, IMEI and KNOX (Workspace))
- **/USERDATA**
  - **/data** -> (MOST IMPORTANT AREA TO EXAMINE) all application and smartphone data stored here
  - /anr -> contains debugging information
  - /app -> contains Android Marketplace .apk files. *Malware may be found here
  - /backup -> stores backup API for developers; user backup data is not stored here
  - /dalvik-cache -> stores code required for apps to function
  - **/media** -> the internal storage locations equivalent to an SD card. *Malware may be found here
  - /misc -> files related to Bluetooth, dhcp, vpn, Wi-Fi, and more are stored here
  - /property -> contains system properties, including time zone, language settings, and more
  - **/system** -> contains key files such as gesture.key and passwords.key; the accounts.db file contains usernames and passwords for file authentication purposes and more

- /MNT or /NONAME (SD Card) *Malware may be found here
  - /asec -> stores unencrypted app data
  - /DCIM -> stores album thumbnails
  - /Pictures -> stores application and camera images
  - /Multimedia -> videos, audio, music files
  - /downloads -> downloaded files
  - /secure/asec -> stores encrypted app data
  - /<App directories> -> application data
- /SYSTEM
  - /app -> contains .apk files. *Malware may be found here

- /priv-app -> contains /apk files with system level permissions. *Malware may be found here

Reference:

- Andrew Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*

(Waltham, MA: Elsevier, 2011).

FOR585 | Smartphone Forensic Analysis In-Depth

SD cards can contain a vast amount of data. Sometimes the only instance of a file or application may be found on the SD card. Application information found on the SD cards may be encrypted as discussed in section 1. Most of the data stored on the SD and emulated storage are stored in the external.db file in the /USERDATA/data/com.android.providers.media file. This file remains persistent even if the SD card is removed. This means that files and filenames associated with SD cards previously in the device can be recovered and examined. This is a great way to tie an SD card back to a device! Keep in mind that the external.db is found on the device, not the SD card, which means traces are left behind, even if the card is removed.

Location of the external.db changed in Android 11 and 12 due to it being included in Project Mainline[1]. The path for these versions are:

- Pixel (vanilla Android): /USERDATA/data/com.google.android.providers.media.module/
- Samsung: /USERDATA/data/com.android.providers.media.module/

A great reference is a blog written by Josh Hickman https://thebinaryhick.blog/2020/10/19/androids-external- db-everything-old-is-new-again/.

Reference:

[1] https://for585.com/mainline - Project Mainline Reference

- contacts2.db

- Tables of interest: contacts, data, deleted_contacts and raw_contacts
- Table worth considering: accounts (third-party access to contacts tracked here)

- com.google.android.gms/databases/icing_contacts.db – Need root or Physical access!
- com.google.android.gms/databases/pluscontacts.db – Google+

Contacts are stored in the same database file as the call logs for most Android devices. The contacts and/or people tables should be examined for contact information. It appears that in Android 10 and 11 that data was an additional table of interest. Another table is the raw_contacts table. This table stores contact information to include names, email addresses, and phone numbers and associates them with an application, the device, or to a Google account. Should a contact have a picture associated with it, a .jpg file is included and is named to match the contact. The table deleted_contacts stores a contact_id and deleted timestamp for the removed contact. Keep in mind as Android versions update, additional tables may be added that are not parsed by mobile forensic tools.

Additional contacts may be located in icing_contacts.db and plustcontacts.db for Google+ as well as third- party applications.

- contacts2.db > Table of interest: calls
- calllog.db (introduced in OS 7) > Table of interest: calls

Communication data can be recovered from the Android device through multiple forms of acquisition. Call logs, contacts, and messages are commonly acquired communication methods on smartphones. As with other smartphones, Android devices have specific locations for storing communication data.

Call logs are stored in the USERDATA/data/com.android.providers.contacts/databases directory on many

Android devices. The primary database to be examined is the contacts2.db, which contains the call log information. Some fields of interest in the calls table include:[1]

Number: Phone number
Date: UNIX Epoch date-timestamp
Duration: Length of call, in seconds
Type: 1 = Incoming
2 = Outgoing
3 = Missed
4 = Voicemail

For Androids running Nougat (v7) or later, the USERDATA/data/com.android.providers.contacts/databases also contains the calllog.db database. The table of interest for this database is calls. The screenshot shows the calllog.db database. Some devices may populate this database, while others seem to have it, but it's empty. Should you find this database, refer to your class notebook (access will be provided in Section 5, so be patient) for the query to parse it. Some highlights here are shown in the case statements. Notice that there are more fields we would want to include in the contacts2.db. Also keep in mind that your tools may not parse this file correctly!

SQLite Query for Calllog.db

```
select _id,
nu
```

```
mber, name, datetime(date/1000,'unixepoch') as "Timestamp", duration as “Duration_in_Seconds”, CASE when type = 2 then "outgoing" when type = 1 then "incoming" when type = 4 then "voicemail" end as "Call Type", subscription_id, phone_account_address, geo
```

```
coded_location, formatted_number, CASE when deleted = 1 then "deleted" else "NA" end as "Deleted", CASE when dirty = 1
```

```
then "Dirty" else "NA" end as "Valid Entry", transcription
```

from calls

The calllog.db also stores SMS responses to incoming calls. This will be covered in lab 2.1A of this section. Take NOTE of “Type 300”. It matters because the tools don’t handle this information the same way. calls.logtype=100 THEN
"Call"
calls.logtype=20
0 THEN "MMS"
calls.logtype=30
0 THEN
"Message"

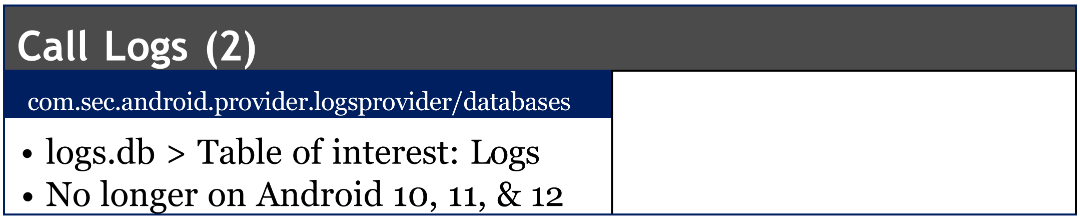
Reference:

- Andrew Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*

(Waltham, MA: Elsevier, 2011).

NOTE: The USERDATA/data/com.android.providers.contacts/contacts2.db contains a plethora of information in various tables. For example, email accounts, synced applications, application status updates (Facebook), dates and times of contact communication, starred contacts, and more can be found in this database.

Google Fi accounts will also contain voicemail transcripts in the table “calls”. The transcript may not be 100% accurate, but still a good indicator of the message. Samsung devices (at least on Android 11) will also show SMS traffic (ingoing/outgoing) and a snippet of the text message. If the message is short enough, the message will be stored in its entirety. This behavior seems to be similar to what was observed in USERDATA/data/com.sec.android.providers.logsprovider on older devices.



Call Logs (2)

com.sec.android.provider.logsprovider/databases

- logs.db > Table of interest: Logs
- No longer on Android 10, 11, & 12

Samsung devices may store the call logs in the USERDATA/data/com.sec.android.provider.logsprovider/databases/logs.db. This database may not be parsed by your tool, which omits call logs from your report. The examiner must look for the call logs and may have to manually examine the data to recover all potential evidence. This database contains other information that may be relevant to the investigation and is not strictly reserved for call logs, so develop a query that works for you. Make sure you look for contacts2.db, calllog.db, and logs.db on all Samsung devices running Android v7 or later. NOTE that this file has not been located on Android 10, Android 11 or Android 12 devices.

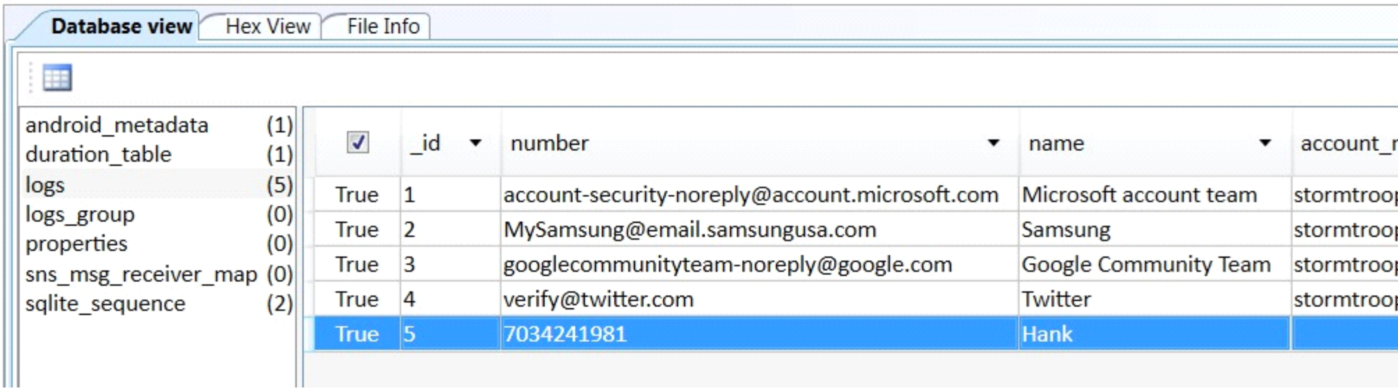
The USERDATA/data/com.sec.android.provider.logsprovider/databases/logs.db file should be manually examined on all Samsung devices where it exists. This file contains more than just call logs. It contains accounts used to verify third-party applications. It can track email accounts, phone numbers from calls, and more. Drafting a query for this database is at the examiner’s discretion, as it contains many tables, columns, and rows of interest. For calls, make sure to examine the logs table and ensure that all necessary columns are parsed. An example of a query for this table would include:

To pull call logs from logs.db, the following columns are of interest:

- The Number
- The Name
- Timestamp
- Duration of Call
- Type of Call

A simple SQLite Query for Logs.db would be:

```
SELECT
name,
DateTime(date/1000,
'UNIXEPOCH') AS
"Timestamp", Duration AS
“Call Duration in Seconds”,
type AS
"1=Incoming
Call, 2=Outgoing
Call" From logs
```



android_metadata	(1)
duration_table	(1)
logs	(5)
logs_group	(0)
properties	(0)
sns_msg_receiver_map	(0)
sqlite_sequence	(2)

☑	_id	number	name	account_r
True	1	account-security-noreply@account.microsoft.com	Microsoft account team	stormtroop
True	2	MySamsung@email.samsungusa.com	Samsung	stormtroop
True	3	googlecommunityteam-noreply@google.com	Google Community Team	stormtroop
True	4	verify@twitter.com	Twitter	stormtroop
True	5	7034241981	Hank	

This query is simply an example and will exclude all other activity outside of call logs. Simply start here and build onto your query to make sure you get all the data relevant to your investigation.

Reference:
Andrew Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android* (Waltham, MA: Elsevier, 2011).

- Group names
- Multiple members
- Group calling
- File transfers

Google Chat is formerly Hangouts and is similar in nature to slack. This chat capability can be embedded within the Gmail application or downloaded from the Play Store as a stand-alone application. The location will either be:

/data/com.google.android.apps.dynamite/dynamite.db – stand-alone app version
/data/com.google.android.gm/databases/user_accounts/%USERACCOUNT% /dynamite.db There are multiple tables of interest.

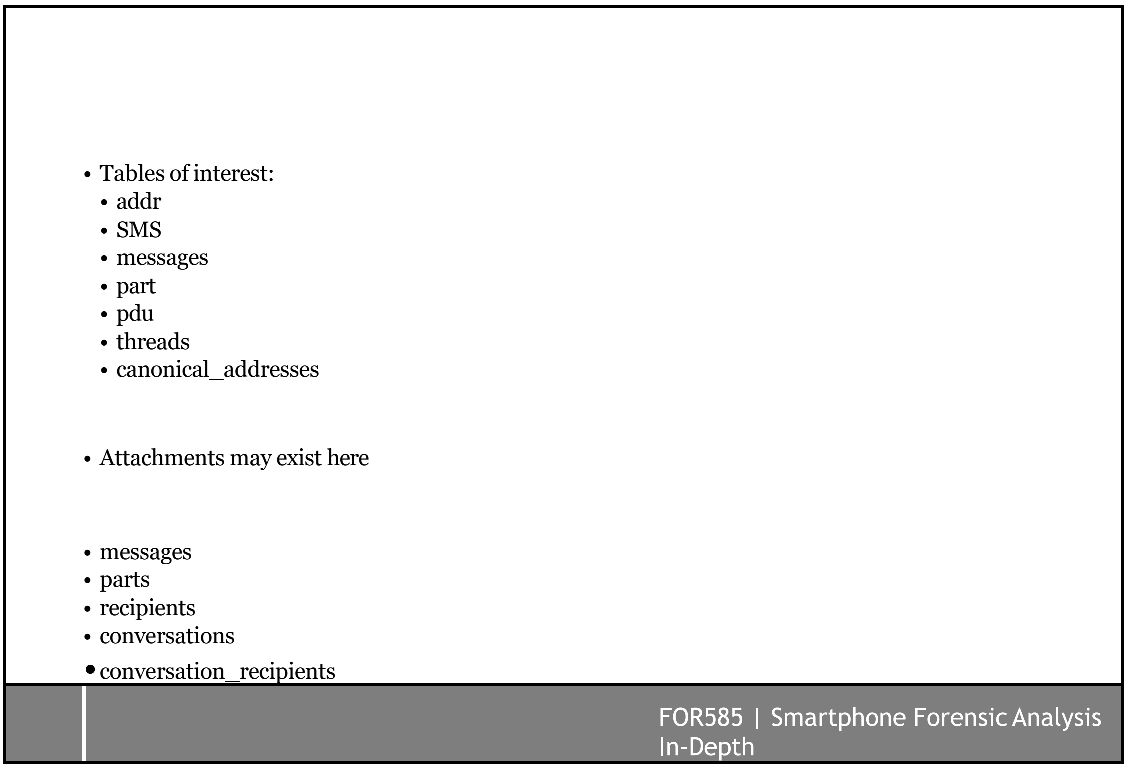
Message attachments and video chat information (URL) is stored in BLOBs that are protobuf. The queries below will return messages BLOBs where they exist. For file attachments, the protobuf data does provide the file type and original file name of the file that was sent/received. However, there is no pointer for the file on the local device (if the file was received). Attached media files can be found in the
~/cache/image_manager_disk_cache directory path. Files in this directory path have a .0 file extension, but they are image files. The Modified Time can be aligned with the BLOB entries in the database to determine what files were transferred when. There may be a shelf life for these files, but I haven't been able to identify it.

Query (For Group Chat info):

```
SELECT
datetime(Groups.create_time/1000000,'unixepo
ch') AS "Created", Groups.name AS "Group
Name",
users.name AS "Group Creator",
datetime(Groups.last_view_time/1000000,'unix
epoch') AS "Last Viewed" FROM
Groups
JOIN users ON
users.user_id=Groups.cre
ator_id ORDER BY
"Created" ASC
```

Query for either dynamite.db

```
SELECT
datetime(topic_messages.create_time/1000000,'unixepoch'
) as "Timestamp", groups.name as "Group name",
users.name as
"Sender",
topic_messages.t
ext_body as
"Message",
topic_messages.annotatio
n as "Message
attachments" FROM
topic_messages
left join groups on
groups.group_id=topic_messages
.group_id left join users on
users.user_id=topic_messages.cre
ator_id order by "timestamp"
ASC
```



- Tables of interest:
  - addr
  - SMS
  - messages
  - part
  - pdu
  - threads
  - canonical_addresses

- Attachments may exist here

- messages
- parts
- recipients
- conversations
- conversation_recipients

FOR585 | Smartphone Forensic Analysis In-Depth

SMS/MMS messages are stored in the USERDATA/data/com.android.providers.telephony/databases/mmssms.db. There are multiple tables of interest, and they seem to keep changing as the versions of Android are released. For Android 11 and 12 mmssms.db exists in two locations:

USERDATA/data/com.android.providers.telephony/ and USERDATA/user_de/%USER_NUMBER%/com.android.providers.telephony/

- The latter contains the ”app_parts” folder

We recommend searching for “com.android.providers.telephony” or “mmsms.db” to ensure all messages are located for examination and verification. The primary table of interest in this database for SMS is the SMS table. The SMS table contains the following fields, which should be examined for relevance to the investigation:

Address: Phone number or email address
Person: Associated entry in address
book (common to be blank) Date:
UNIX Epoch date-time stamp
Read: 1 = message read
0 = message unread
Type: *Note the "Type" flags may vary on different Android devices and must be verified prior to reporting
1 = Inbox
2 = Sent
3 = Drafts
5 = Outbox
6 = Unknown
Body: Message content
service_center: SMSC (service center number) for received messages

For MMS messages, the part table should be examined for the following information:
_data: Location
of attachment in
file system Text:

Message content if available
Ct: Content type (for example, jpeg or text) Cl: Content location, often filename
CID: Content-ID, often filename and the same as the CI

The query below will help:

```
SELECT datetime(sms.date/1000,’unixepoch’) AS “Timestamp (UTC), sms.address AS “Other Party”,
CASE
when sms.type=1 THEN “Incoming”
when
```

```
sms.type=2 THEN “Outgoing” END AS “Message Direction”, CASE when sms.read=0 THEN “No” when sms.read=1 THEN “Yes” END AS “Was Read”
```

```
,
s
m
s.
b
o
d
y
A
S
“
M
es
sa
ge
”
F
R
O
M
s
m
s

M
M
S
:
SELECT
datetime(pdu.date/1000,’unixepoch’) AS
“Timestamp (UTC)”, address.address AS
“Other Party”,
part.ct as “Attachment Type”,
part._data AS
“Attachment Name &
Location”, part.text
AS “Text”
FROM
part
INNER JOIN pdu on pdu._id=part.mid
INNER JOIN address ON address.msg=part.mid
```

- Google Messages and RCS Messages
- Primary tables of interest: conversations, participants, parts, self_participants

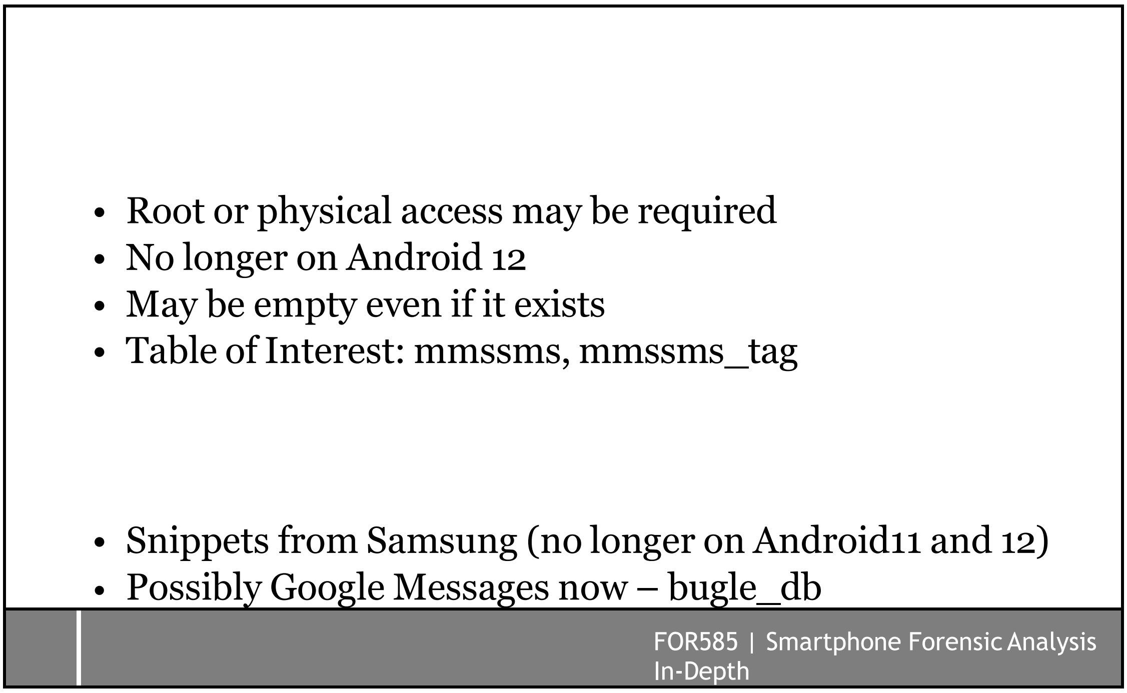
USERDATA/data/com.google.android.apps.messaging/databases/bugle_db: RCS and Google messaging right now, only the S21 is known[1]. For Google Messages, com.google.android.apps.messaging/cache/image_manager_disk_cache/ folder may contain copies of pictures sent/received and/or available to the app in Photos that were merely “read” by the app (not sent or received). Files are stored as .bin files.

Query (Google Messages - bugle_db):

```
SELECT
datetime(parts.timestamp/1000,’unixepoch’) AS
“Timestamp”, parts.content_type AS “Message
Type”,
conversations.name AS “Other
Participant/Conversation Name”,
participants.display_destination AS
“Message Sender”,  parts.text AS
“Message”,
CASE
WHEN
parts.file_size_b
ytes=-1 THEN
‘NA’ ELSE
parts.file_size_b
ytes
END AS “File Size
(bytes)”,
parts.local_cache_pat
h AS “Path To
Attachment” FROM
parts
JOIN messages ON messages._id=parts.message_id
JOIN participants ON
participants._id=messages.sender_i
d   JOIN conversations ON
conversations._id=parts.conversatio
n_id ORDER BY “Timestamp”
ASC
```

Reference:

- https://for585.com/s21 - RCS Messages



- Root or physical access may be required
- No longer on Android 12
- May be empty even if it exists
- Table of Interest: mmssms, mmssms_tag

- Snippets from Samsung (no longer on Android11 and 12)
- Possibly Google Messages now – bugle_db

It is expected that messages will be found within third-party application files, and that will be covered heavily in Section 5 of this course. Examiners must be aware that other locations exist on Android devices that store messages. The additional paths listed may contain message of interest:

- USERDATA/data/com.google.android.gms/databases/icing_mmssms.db: Additional SMS/MMS
- USERDATA/data/com.sec.android.provider.logsprovider/databases/logs.db: Snippets from Samsung devices

RCS, or Rich Communication Services, is a newer standard for messaging that is planned to eventually replace SMS on Android devices. It has yet to be adopted by all devices, thus it's something that will require more research as devices surface that are leveraging this functionality of messaging. The example shown in this slide is the icing_mmssms.db located at USERDATA/data/com.google.android.gms/databases/icing_mmssms.db. You must have either root access to obtain this file or full access via a physical or file system dump. This file contains both SMS and MMS sent and received from the device. A free script was developed to parse these messages, which can be found on your FOR585 VM on the Desktop in the Scripts for class directory. Magnet Forensics has a great white paper on Android messaging, which may be of interest for you to read: https://www.magnetforensics.com/blog/android-messaging-forensics-sms-mms-and-beyond/.

A query to parse this icing_mmssms.db is available in your course

notebook as well as below. select

mmssms._id,
m
m
s
s
m
s

```
.
m
s
g
_
t
y
p
e
,

c
a
s
e
when
mmssms.typ
e = 2 then
"incoming"
when
mmssms.typ
e = 1 then
"outgoing"
end AS
"message
status",
mmssms.add
ress,



datetime(mmssms.date/1000,'UNIXEPOCH','lo
caltime') AS "date", mmssms.body AS
"message",
mm
ssm
s_ta
g.ta
g
AS
"unr
ead
"
fro
m
mm
ssm
s
left join mmssms_tag on mmssms_tag._id=mmssms._id
```

- What happens?
- How is the data stored?
- How will the tools interpret it?

- calls.logtype=100 "Call"
- calls.logtype=200 "MMS"
- calls.logtype=300 "Message"

Spoiler alert – this is coming up in a lab, so we won’t tell you all you need to know about it just yet. Sometimes the best way to learn is to get your hands dirty by practicing.

Graphic from: https://www.digitalcitizen.life/how-use-and-change-quick-responses-available-android- smartphones/

Accounts and passwords can be recovered by most commercial tools. In this slide, the User Accounts and Passwords, parsed by Physical Analyzer and listed in the Analyzed Data section of the tool, contain both usernames and passwords. Sometimes, the data may be nonsensical because of the encryption settings of the application or device. The USERDATA/system/accounts.db file may contain user account information for Google, third-party applications, and other online resources. This file was last leveraged in Nougat (OS 7). For newer Android devices you may see an accounts_de.db or accounts_ce.db, which will be discussed further in these notes.

If you have multiple users, you may also see user/10, user/11 or user/150. Always note that the primary user will be user/0. Use your forensic tools to identify what is being parsed and then do a logical search to ensure nothing is being overlooked. Some tools will parse the accounts files. The passwords may be encrypted and are not easy to obtain, but some smartphone forensic tools can decrypt them. Some passwords may be stored at tokens and are not readable.

Another location that stores account information is com.google.android.gms/shared_prefs/BackupAccount.xml. If the user sets a backup account for recovery, the email listed here will be different from the primary email in the accounts files.

You will find many locations during the labs in this section that store account information. It honestly can be overwhelming. Finally, two fantastic places to go hunting for accounts and passwords is com.android.providers.settings/* and com.google.android.gms/databases/*. Bottom line – usernames, passwords and account information are stored in many places on Android devices. A keyword search may help you hunt these artifacts down.

USERDATA/system_de/0/accounts_de.db and USERDATA/syste_ce/0/accounts_ce.db appear to track user accounts added to the device, even those used by some third-party apps. NOTE that not all user accounts will be placed here, which we found in our testing. You may have to examine the application preferences to uncover all. Some third-party apps will be seen here, even though no user account was associated with the application. Some third-party apps that use a distinct user accounts may not list the account here but there will include an entry for the app. This requires further investigation by the examiner.

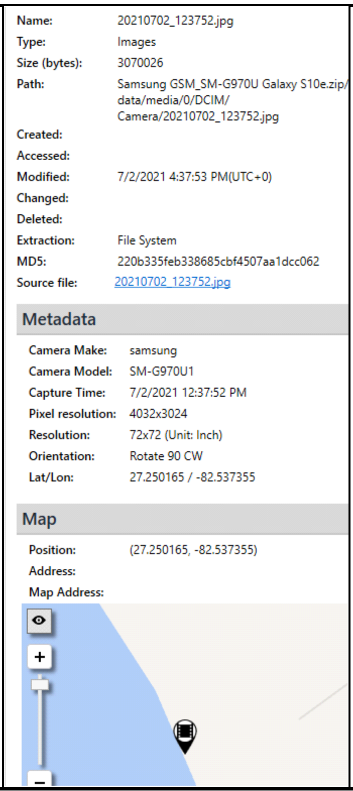
Some tables and columns worthy of mentioning include multiple “action types” in the table “debug_table”

- action_add_account **-** can represent either the time of login (for apps that require it), or the first time an app is used (when a user did not log in or app doesn’t require it)
- action_called_account_remove
- action_account_remove

NOTE: Only “action_add_account” is consistent. The other two are not consistent enough to rely upon.

```
SELECT
accounts.type AS “App”, accounts.name AS “Account”, debug_table.action_type AS “Action”, debug_table.time AS “Action Time” FROM accounts
JOIN debug_table ON debug_table._id=accounts._id
```

NOTE: The times will vary. If the Google account is added during the initial setup of the phone, the time added will be in UTC. The remaining times in the database will be in local time.



Name:	20210702_123752.jpg
Type:	Images
Size (bytes):	3070026
Path:	Samsung GSM_SM-G970U Galaxy S10e.zip/data/media/0/DCIM/Camera/20210702_123752.jpg
Created:	
Accessed:	
Modified:	7/2/2021 4:37:53 PM(UTC+0)
Changed:	
Deleted:	
Extraction:	File System
MD5:	220b335feb338685cbf4507aa1dcc062
Source file:	20210702_123752.jpg

Metadata

Camera Make:	samsung
Camera Model:	SM-G970U1
Capture Time:	7/2/2021 12:37:52 PM
Pixel resolution:	4032x3024
Resolution:	72x72 (Unit: Inch)
Orientation:	Rotate 90 CW
Lat/Lon:	27.250165 / -82.537355

Map

Position:	(27.250165, -82.537355)
Address:	
Map Address:	

As with most graphics, location artifacts, camera information and more may exist. The user can easily disable settings to prevent photos taken with the device from storing this type of metadata, but often we find that location information is prevalent in multimedia files.

The photos top feature can be found in /data/com.google.android.apps.maps/databases/ugc_photos_location_data.db. This database will document the date/time of pictures/videos taken with camera on the device.

Table of Interest:
photos_top_feature

The timestamps are stored as Unix Epoch timestamps, and you will uncover latitude and longitudes for the multimedia files. Values in column “photo_uri” refer to column “_id” in “files” table of external.db. This database was not present on Samsung running Android 11 and may not exist on Android 12 devices.

The file media.db may only be found on Samsung devices running Android 11. The file is located at /data/data/com.samsung.android.providers.med
ia/databases/media.db. Tables of Interest:
files

location
location_datetime_idx
tag_view – contains location data pertaining to the EXIF

The column “File Path” will have artifacts also being stored in Samsung Secure Folder. You may find additional tables of interest, so it’s worth examining all of them that contain information.

SQLite Query:
sele

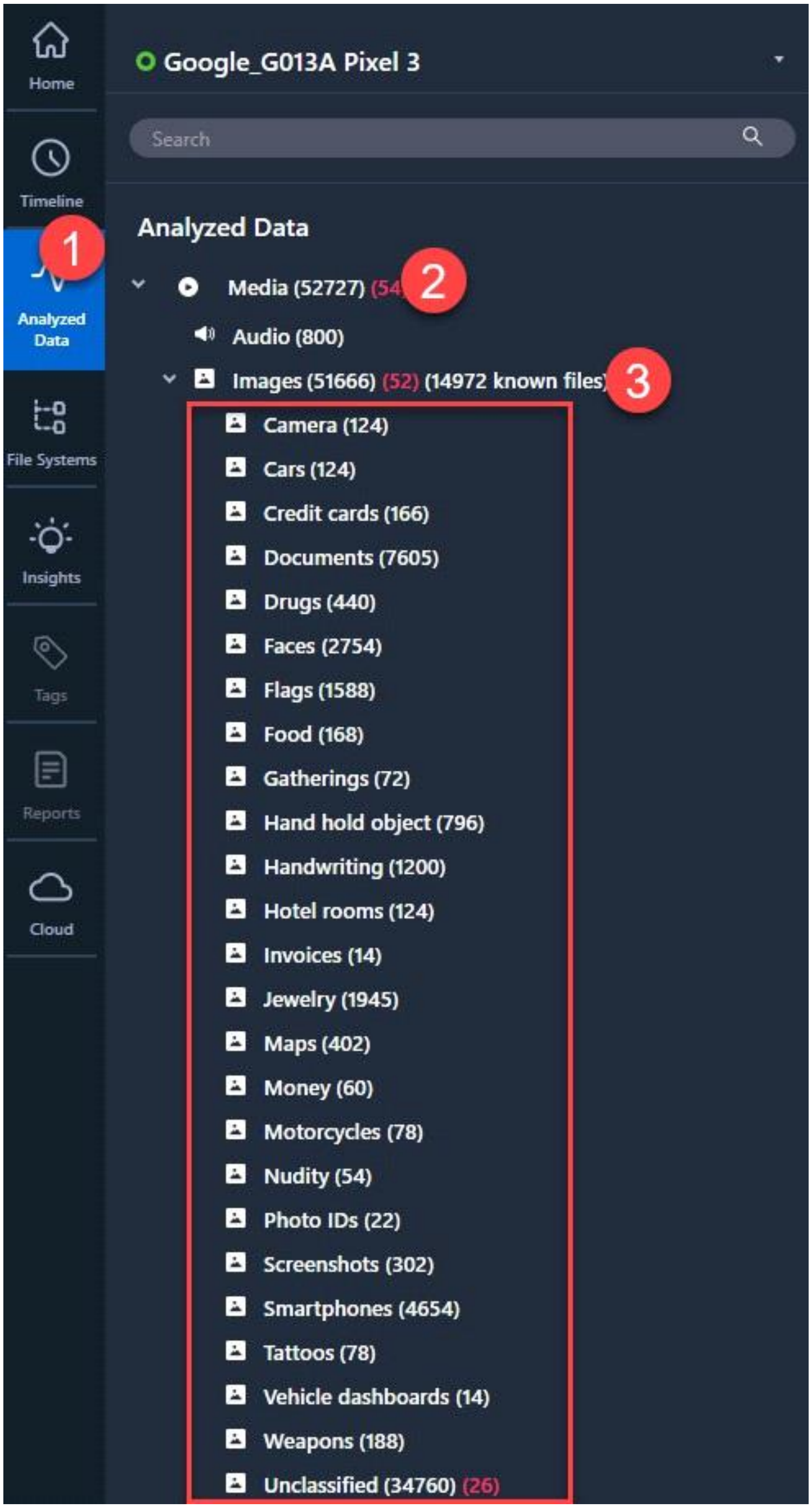
```
ct files._id,
files.media_id as "Media ID",
scene.parent_name as "Scene Parent Name",
scene.scene_name as "Scene Name",
files._data as "File Path",
files._size as "File Size (bytes)",
--File size is recorded in bytes and converts to  Mebibytes to most accurately reflect the file size displayed on the device.
datetime(files.datetaken,'unixepoch','localtime') as "Date Taken",
datetime(files.date_added,'unixepoch','localtime') as "Date Added",
datetime(files.date_modified,'unixepoch','localtime') as "Date Modified",
files.mime_type as "MIME Type",  files._display_name as "Display Name",
files.bucket_display_name as "Bucket Display Name",
files.duration/1000.0 as "Duration (secs)",
CASE
when files.is_trashed = 0 then "No" when files.is_trashed = 1 then "Yes" end as "Is Trashed",
files.volume_name as
```

```
"Volume
Name",
files.latitude
as "Latitude",
files.longitude
as
"Longitude",
location.country_c
ode as "Country
Code",
location.country_
name as "Country
Name",
location.admin_ar
ea as "Admin
Area",
location.locality
as "Locality",
location.addr as
"Address",
location.street_na
me as "Street
Name",
location.street_nu
mber as "Street
Number",
location.postal_co
de as "Postal
Code",
files.video_codec
_info as "Video
Codec",
files.audio_codec
_info as "Audio
Codec", CASE
when
files.is_clo
ud = 1
then "Yes"
when
files.is_clo
ud = 0
then "No"
end as "Is
Cloud",
scene.scen
e_score as
"Scene
Score"
from files
left join scene on
scene.sec_media_id=file
s._id left join location on
```

location.latitude=files.latitude

FOR585 | Smartphone Forensic Analysis In-Depth

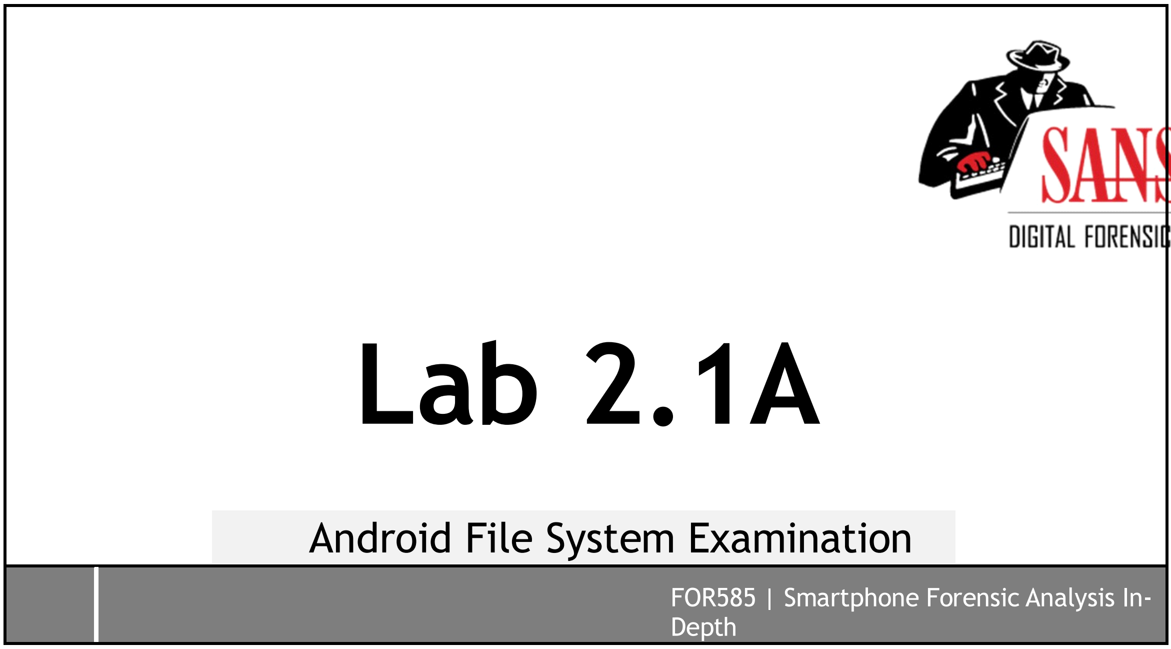
Media Classification is a popular feature for those who work Child Sexual Abuse (CSA) cases, but it's relevant to other cases as well. In AXIOM, you can select media classification for CSA images during processing. In Physical Analyzer, you can run media classification during processing or after the extraction loads. This feature was released in 7.40 and should be used for triage purposes. As we have stressed in this course, the commercial tools produce results that may need to be validated. For a capability like media classification, it's important that the parsing reports correctly or as correctly as possible. This means you will get false positives. However, you want to make sure you don't get a false negative. Meaning that you searched for faces and then later (through validation) find a face of interest that was missed by the tool. Bugs like this should be reported. Once in the images, you can change your confidence levels to filter out the false positives.



Google_G013A Pixel 3

Analyzed Data

- Media (52727) (54
  - Audio (800)
  - Images (51666) (52) (14972 known files)
    - Camera (124)
    - Cars (124)
    - Credit cards (166)
    - Documents (7605)
    - Drugs (440)
    - Faces (2754)
    - Flags (1588)
    - Food (168)
    - Gatherings (72)
    - Hand hold object (796)
    - Handwriting (1200)
    - Hotel rooms (124)
    - Invoices (14)
    - Jewelry (1945)
    - Maps (402)
    - Money (60)
    - Motorcycles (78)
    - Nudity (54)
    - Photo IDs (22)
    - Screenshots (302)
    - Smartphones (4654)
    - Tattoos (78)
    - Vehicle dashboards (14)
    - Weapons (188)
    - Unclassified (34760) (26)

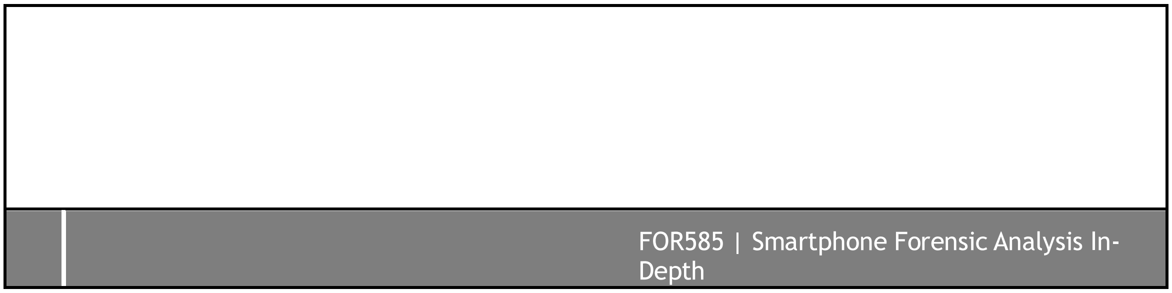


Lab 2.1A

Android File System Examination

FOR585 | Smartphone Forensic Analysis In-Depth

This page intentionally left blank.



FOR585 | Smartphone Forensic Analysis In-Depth

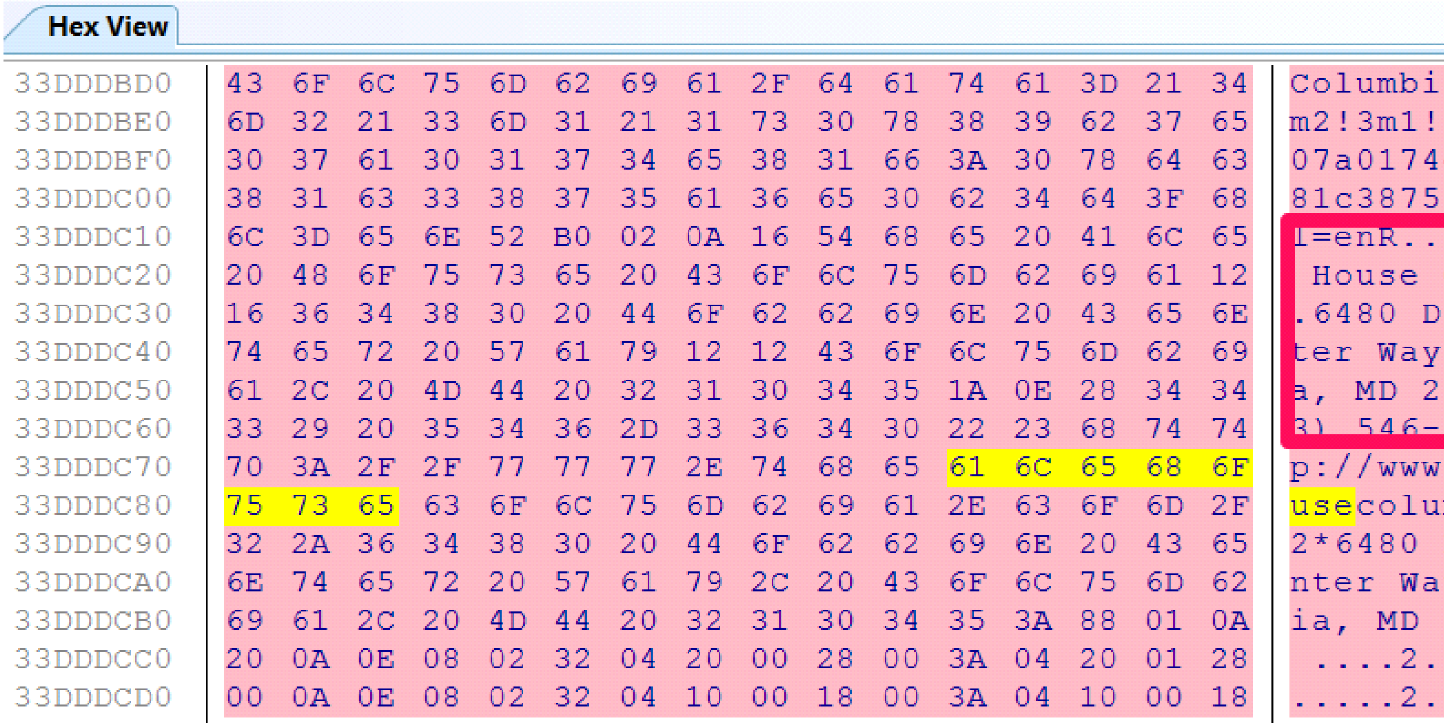
One of the first things you should verify in regard to location information is if the user has location services enabled on their device. These settings are saved in the googlesettings.db in USERDATA/data/com.google.android.gsf/databases/. Here, if you see the value of 1 for "use_location_for_services," then you can state that the user has location information enabled on their device.

com.google.android.apps.maps

- Database: gmm_storage.db
- Hex: Contains search history

FOR585 | Smartphone Forensic Analysis In-Depth

This database doesn't seem to be parsed by the tools and may need to be examined manually in Hex. All files found in the apps directory should be examined to avoid missing data. You will find that BLOBs may cause issues when looking at a database, and these locations are stored in BLOBs. More to come on BLOBs in the course in Sections 4 and 5. In addition to the searched locations, turn-by-tun directions are stored in the BLOBs.



```
33DDDBD0  43 6F 6C 75 6D 62 69 61 2F 64 61 74 61 3D 21 34  Columbi
33DDDBE0  6D 32 21 33 6D 31 21 31 73 30 78 38 39 62 37 65  m2!3m1!
33DDDBF0  30 37 61 30 31 37 34 65 38 31 66 3A 30 78 64 63  07a0174
33DDDC00  38 31 63 33 38 37 35 61 36 65 30 62 34 64 3F 68  81c3875
33DDDC10  6C 3D 65 6E 52 B0 02 0A 16 54 68 65 20 41 6C 65  l=enR..
33DDDC20  20 48 6F 75 73 65 20 43 6F 6C 75 6D 62 69 61 12   House
33DDDC30  16 36 34 38 30 20 44 6F 62 62 69 6E 20 43 65 6E  .6480 D
33DDDC40  74 65 72 20 57 61 79 12 12 43 6F 6C 75 6D 62 69  ter Way
33DDDC50  61 2C 20 4D 44 20 32 31 30 34 35 1A 0E 28 34 34  a, MD 2
33DDDC60  33 29 20 35 34 36 2D 33 36 34 30 22 23 68 74 74  3) 546-
33DDDC70  70 3A 2F 2F 77 77 77 2E 74 68 65 61 6C 65 68 6F  p://www
33DDDC80  75 73 65 63 6F 6C 75 6D 62 69 61 2E 63 6F 6D 2F  usecolu
33DDDC90  32 2A 36 34 38 30 20 44 6F 62 62 69 6E 20 43 65  2*6480
33DDDCA0  6E 74 65 72 20 57 61 79 2C 20 43 6F 6C 75 6D 62  nter Wa
33DDDCB0  69 61 2C 20 4D 44 20 32 31 30 34 35 3A 88 01 0A  ia, MD
33DDDCC0  20 0A 0E 08 02 32 04 20 00 28 00 3A 04 20 01 28   ....2.
33DDDCD0  00 0A 0E 08 02 32 04 10 00 18 00 3A 04 10 00 18  .....2.
```

Location information and mapping can be found in several locations on the Android device. Applications use and store location information in various locations and normally can be found within the application folder. The version of the Android device affects the location information and where it is stored. The example for USERDATA/data/com.google.android.apps.maps/databases should be found on all Android devices.

However, the databases may vary based on use and the one below hasn't been seen since Android 5 but may exist. It is important that you examine all databases associated to Google Maps, not just the ones shown in the next few slides.

The main database of interest for location maps includes the da_destination_history. The table of interest is destination_history. The fields of interest in this table include:
Time =
Stored in
UNIX
Epoch
dest_lat =
Destination
latitude
dest_lng =
Destination
longitude
dest_title = Business name,
street name, point of interest
dest_address = Mapped address

For the search_history.db file in USERDATA/data/com.google.android.apps.maps, location history and suggestions based on searches and locations can be recovered. The tables of interest are history and suggestions. The fields of interest in this table include:
display
query =
suggeste
d
location
data1 =
suggeste
d
location

Make sure you realize that suggestions are not searches typed by the user, but those provided as a suggestion by the device.

- Timestamp indicates when the navigation started

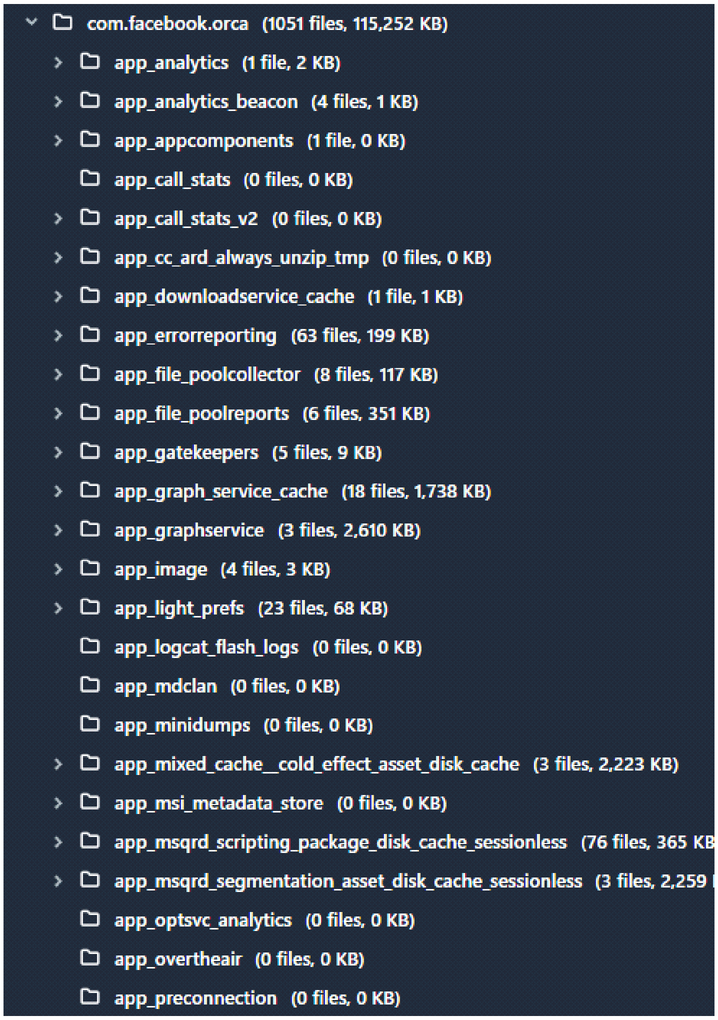
Text to Speech audio files are present in Android 10, 11, & 12. The files are stored in USERDATA/data/com.google.android.apps.maps/app_tts-temp/tts-%UNIXEPOCH%.

NOTE: The timestamp in file name is indicative of when the navigation was started.

This file contains audio files for turn-by-turn directions. The Audio is Google Assistant's T-b-T (turn-by-turn) and appears to only keep the last set of turn-by-turn audio. This type of information can be used to place the user at a location at a specific time!

For more information see blog post https://kibaffo33.data.blog/2021/12/30/at-the-roundabout-take-the-second- exit/.

- Social media geo-tagging
  - Facebook
  - Google+
  - Instagram
  - Snapchat
  - Twitter
  - And more
- Consider what traces are left behind when the user “checks in” and tags a location
- What about third-party apps?



```
com.facebook.orca  (1051 files, 115,252 KB)
    app_analytics  (1 file, 2 KB)
    app_analytics_beacon  (4 files, 1 KB)
    app_appcomponents  (1 file, 0 KB)
    app_call_stats  (0 files, 0 KB)
    app_call_stats_v2  (0 files, 0 KB)
    app_cc_ard_always_unzip_tmp  (0 files, 0 KB)
    app_downloadservice_cache  (1 file, 1 KB)
    app_errorreporting  (63 files, 199 KB)
    app_file_poolcollector  (8 files, 117 KB)
    app_file_poolreports  (6 files, 351 KB)
    app_gatekeepers  (5 files, 9 KB)
    app_graph_service_cache  (18 files, 1,738 KB)
    app_graphservice  (3 files, 2,610 KB)
    app_image  (4 files, 3 KB)
    app_light_prefs  (23 files, 68 KB)
    app_logcat_flash_logs  (0 files, 0 KB)
    app_mdclan  (0 files, 0 KB)
    app_minidumps  (0 files, 0 KB)
    app_mixed_cache__cold_effect_asset_disk_cache  (3 files, 2,223 KB)
    app_msi_metadata_store  (0 files, 0 KB)
    app_msqrd_scripting_package_disk_cache_sessionless  (76 files, 365 KB
    app_msqrd_segmentation_asset_disk_cache_sessionless  (3 files, 2,259
    app_optsvc_analytics  (0 files, 0 KB)
    app_overtheair  (0 files, 0 KB)
    app_preconnection  (0 files, 0 KB)
```

Third-party applications, especially social media apps, allow for the user to tag locations on the device. This means that the user can tag events, pictures, a status, a birthplace, a wedding site, and more. All these actions are tracked within databases for each application. Some applications encrypt or encode this data, while others leave it in plaintext. These types of artifacts will be covered in-depth in Section 5.

Sometimes the only access you will have to Google Maps on the device is by doing a cloud extraction. We will cover more cloud extraction techniques at the end of this section, but here is a glimpse of what's ahead. We can see what Routes the user took and the type of transportation. These tracks can be viewed in Maps by selecting the eye icon under "Show track."

- Start with History and dive deeper from there

The first step in knowing what you are up against in regard to browsers in use on the Android you are examining is to review the installed applications. This can be done by examining the "Installed Apps" in Physical Analyzer or by using Magnet AXIOM to see a breakdown for each browser in addition to installed applications. Make sure you do not overlook the native browser that came with your device. Additionally, you should keyword search for "localstorage" to ensure you don't miss WebKit data. WebKit data will be of interest in this course and in your smartphone investigations. The labs are going to lead you in that direction, so enjoy! Browser artifacts are covered in-depth in Section 5 of this course, but you are going to see browser artifacts in the upcoming labs, so it must be mentioned here.

Applications are not required to save and store data in the USERDATA/data directory, but most comply.[1] Again, the user can elect on where to save the application, which overrides the decision to keep the application information in the USERDATA/data directory.

Each application stored in the USERDATA/data directory has an application folder reflecting the application name. For example, the application Facebook is stored as USERDATA/data/com.facebook.katana. Within the application folder, several folders may exist. These include:

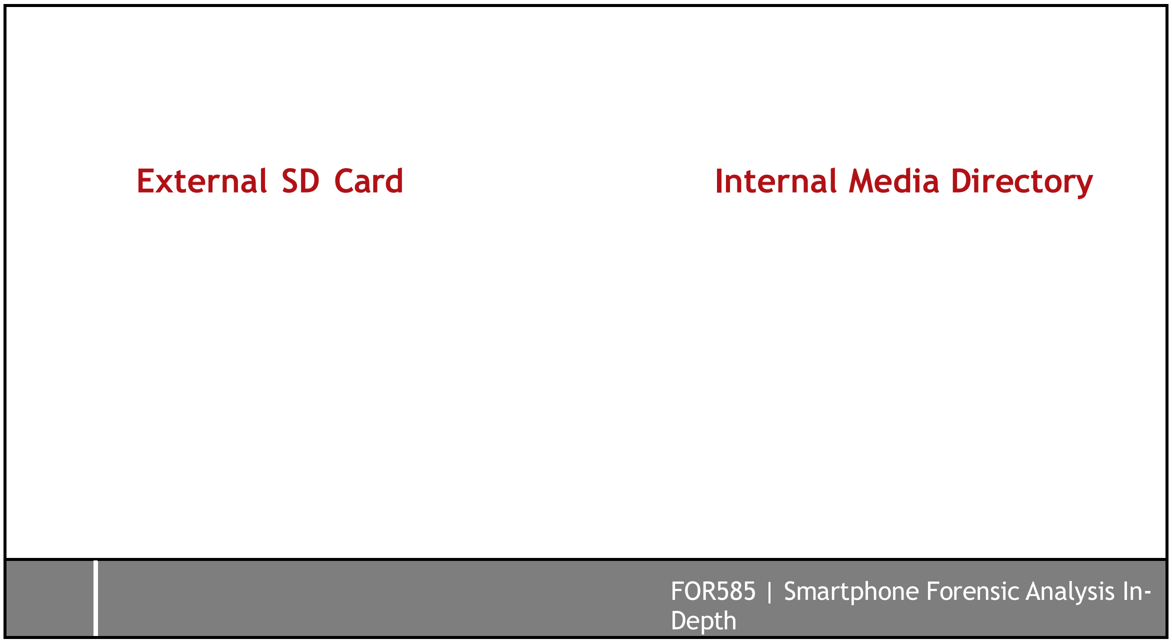
- Lib
- Files
- Cache
- Databases
- Shared Pref

The Cache and Databases folders contain most of the user content we want to examine. However, because Android devices are open source and applications are not restricted, each folder and file should be examined for relevance.

Reference:

- Andrew Hoog, *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*

(Waltham, MA: Elsevier, 2011).



External SD Card

Internal Media Directory

FOR585 | Smartphone Forensic Analysis In-Depth

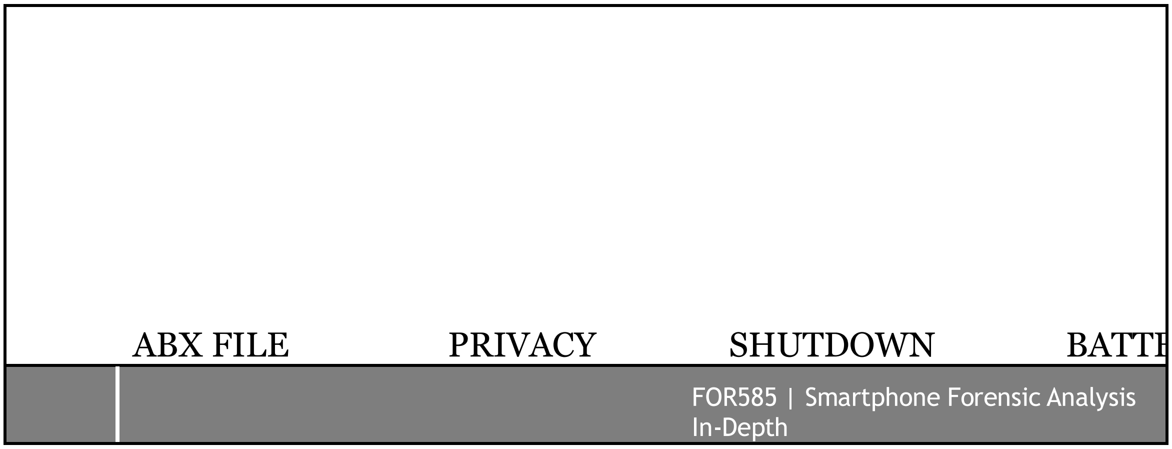
While most applications have directories in the USERDATA/data directory, some store unique data (file attachments, video, databases, and more) on the SD card or the internal USERDATA/media directory. We cannot stress how important it is to thoroughly search the entire device for all remnants of application data. As you saw with the Zedge example, the data can exist anywhere.

The SD card (/mnt or NONAME) contains multiple directories that may be different on each device you examine. Look carefully and dive into any directory that contains application data of interest. Be ready to parse a lot of this data manually. Directories that are commonly found here and that should be examined include:

- Android
- Download
- Media
- <Application directory>

The USERDATA/media/0 directory is essentially an SD card that is stored internally on the device. Directories that are commonly found here and that should be examined include:

- Android
- DCIM
- Download
- Pictures
- * **-** Yep, all of the other directories that contain the data you are most likely trying to piece to that database of interest.



ABX FILE PRIVACY SHUTDOWN BATTE

FOR585 | Smartphone Forensic Analysis In-Depth

This slide intentionally left blank.

Android 12 replacement for standard XML

FOR585 | Smartphone Forensic Analysis In-Depth

Android 12 brought changes to the game of forensics and introduced not only new files and file locations, but a new format for XML. ABX is the new format introduced in Android 12 and it is different than the binary XML sometimes seen as AndroidManifest.xml. The file extension is still XML, however the file header is "ABX" + ABX version (version seen in Android 12 is Version 2), as shown in the screenshot.

These files may not be human readable. For Android 12 devices, all XML files in the /USERDATA/system/ directory path are in this ABX format. This format has also been seen in various other locations (e.g., Recent Tasks XML files). For more information, read Alex Caithness' blog, found here: https://www.cclsolutionsgroup.com//post/android-abx-binary- HYPERLINK "http://www.cclsolutionsgroup.com//post/android-abx-binary-xml"xml

NOTE: ALEAPP supports ABX parsing and Physical Analyzer 7.52 will support it as well.

- What do your tools parse?

- Examine installed applications
- Examine parsed browser artifacts and applications
- Search the parsed databases for keywords of interest
- Triage parsed locations, but don't trust until you validate
- Determine what requires manual examination

- Conduct a keyword search for usernames/accounts

- Examine all tables and preference files in application
directories

- Examine the SD card or media directory

- Search properly in your tools to ensure nothing is overlooked
- Get ready to carve, manually parse, and decode data

Bottom line—user data on a mobile device is overwhelming. Use a tool that can help you get started and then focus on specific artifacts that need to be manually examined or verified.

SANS DFIR

DIGITAL FORENSICS & INCIDENT RESPO

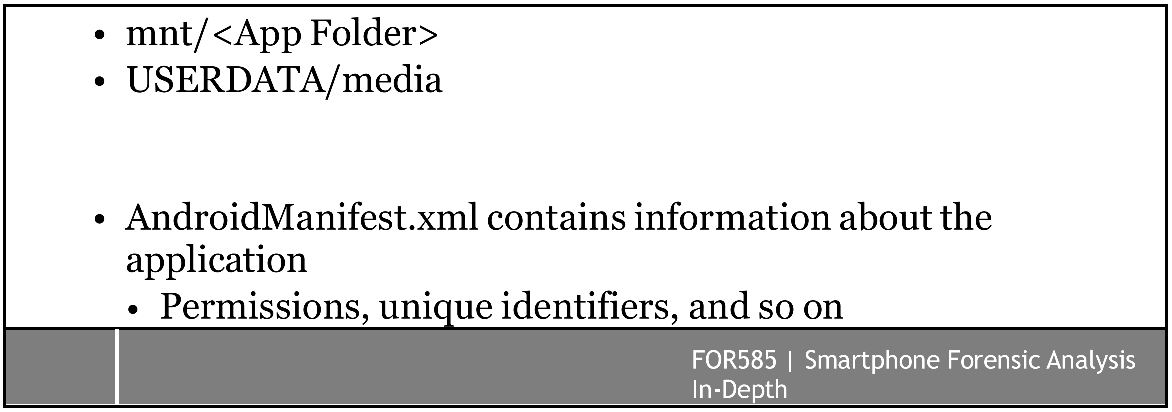
Lab 2.1B

Android File System Examination

This page intentionally left blank.

This page intentionally left blank.

- Application Insights, App Genie, Dynamic App Finder, Fuzzy Model Plug-in, etc.
- USERDATA/data/<App Folder>



- mnt/<App Folder>
- USERDATA/media

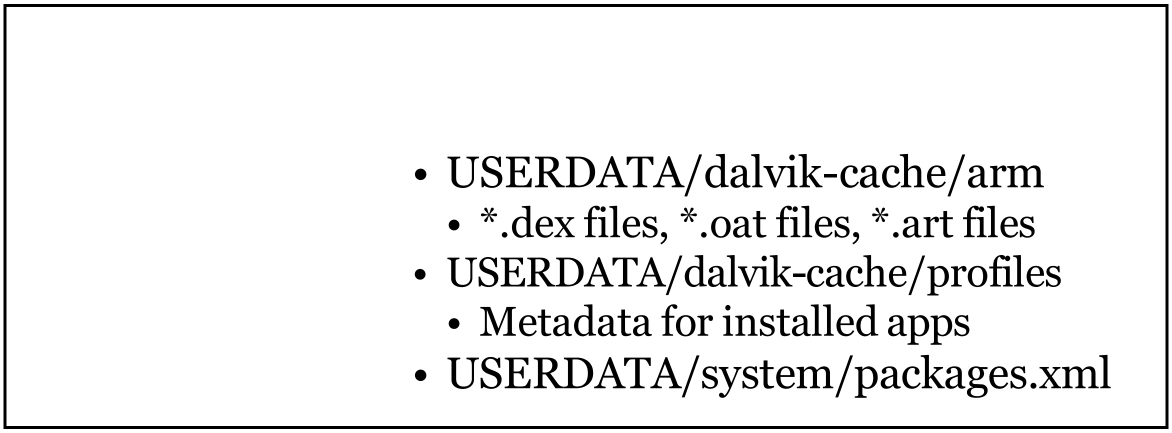
- AndroidManifest.xml contains information about the application
  - Permissions, unique identifiers, and so on

FOR585 | Smartphone Forensic Analysis In-Depth

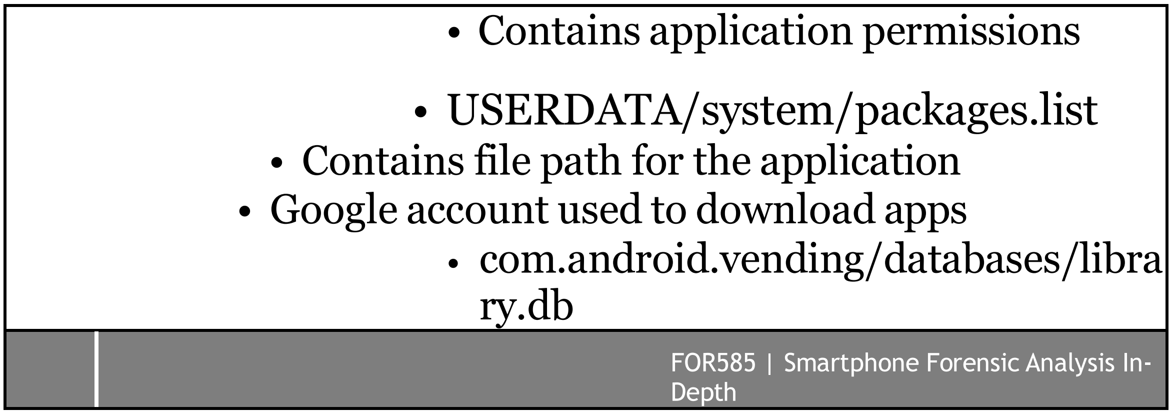
Each installed application should be triaged or quickly examined for relevance in the investigation. Remember that your forensic tool most likely does not parse all data from applications on the device. Your job is to ensure that data is not overlooked. The best way to do this is to examine the installed applications on the device.
Often, a tool pulls the names of user-installed applications that are not custom to the device. This is a good starting point.

From there, examine the application folders on the SD card in the /mnt or NONAME partition, which contains a folder for each application. If the device cannot accept an SD card, examine the USERDATA/media directory. The device USERDATA/data directories must also be examined for relevance to the investigation. Consider applications used for communication and multimedia sharing as a starting point. The section on third-party applications delves deeper into this topic in Section 5 of this course.

A physical keyword search can be conducted for *.apk that provides hits for most applications present on the device. Most .apk files contain an AndroidManifest.xml file, which contains essential information about the application. This includes shared preferences, the unique application UID, GID, and more. The AndroidManifest.xml file can be examined by unpacking the .apk file, as you will learn in the Mobile Malware section (Section 4) of this course.



- USERDATA/dalvik-cache/arm
  - *.dex files, *.oat files, *.art files
- USERDATA/dalvik-cache/profiles
  - Metadata for installed apps
- USERDATA/system/packages.xml



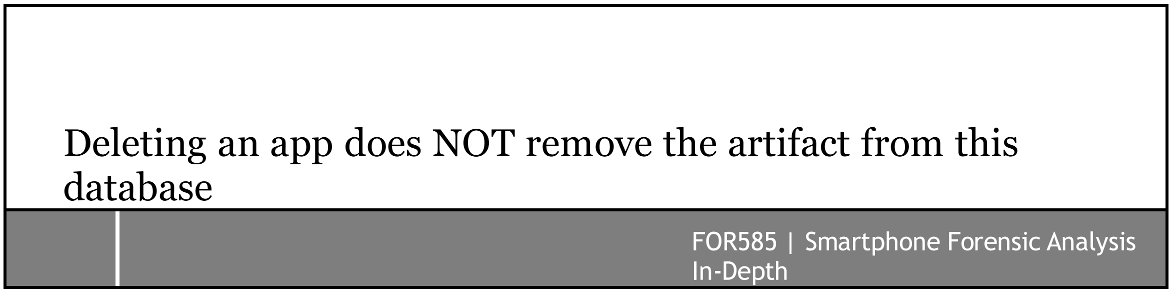
- Contains application permissions
- USERDATA/system/packages.list
  - Contains file path for the application
- Google account used to download apps
  - com.android.vending/databases/library.db

The USERDATA/dalvik-cache directory should be examined for .dex files, .oat files, and .art files. In this directory, you may find both an arm and profiles directory that will contain the files of interest. The .dex files are compiled Android application code files. The .oat files are optimized .dex files leveraging “ahead-of time” features. This type of file speeds up the application load time. The .art files are Android Runtime files.

Applications that currently exist or existed on the device may leave these files behind. If an application was installed and then deleted, traces may reside in these locations. The USERDATA/system directory contains two files that should be examined for application permissions and application metadata, respectively:

- packages.xml
- packages.list

Finally, if you want to know who downloaded and installed the application, the com.android.vending/databases/library.db will be helpful. Make sure to refer to the cheat sheet within your course media files for more locations to examine. There are tons of them!

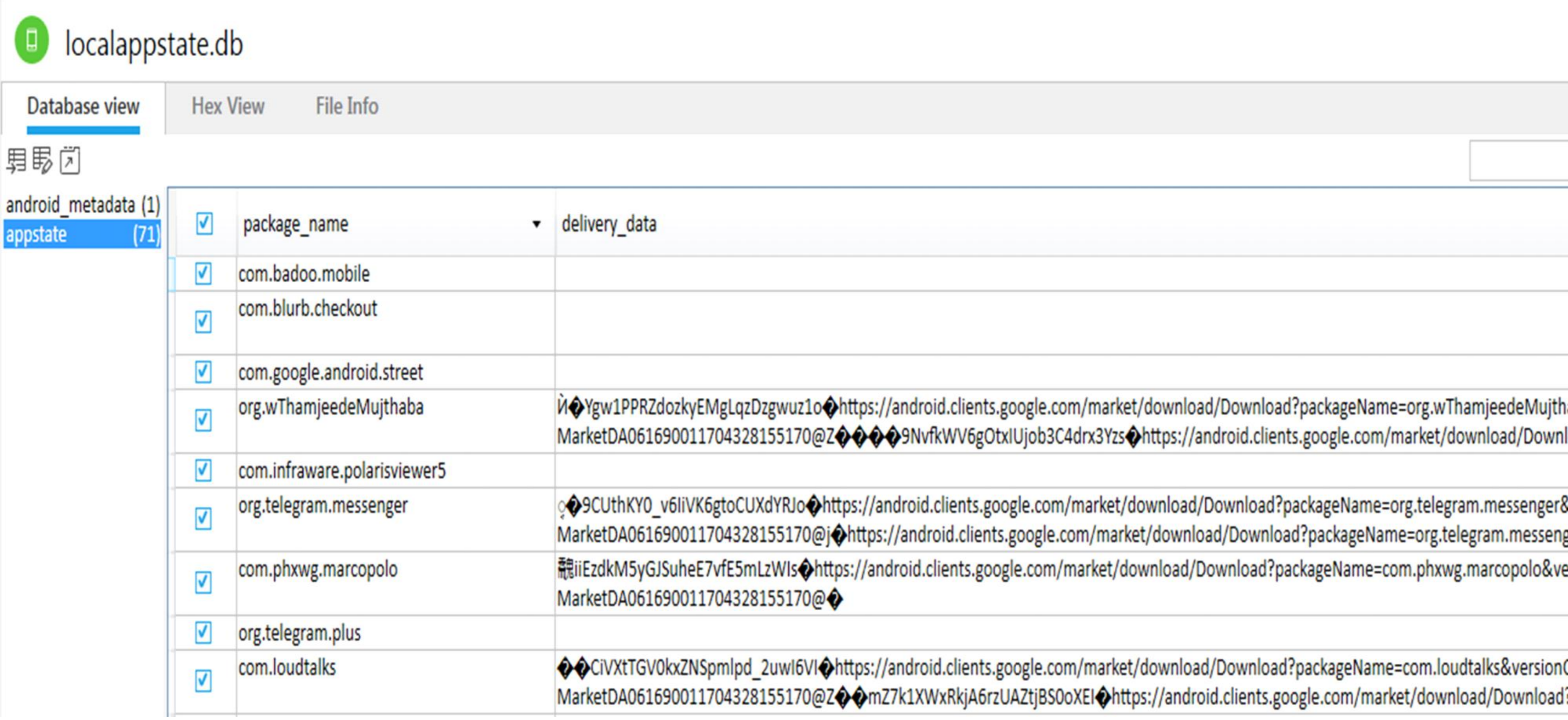


Deleting an app does NOT remove the artifact from this database

The localappstate.db contains information relating to the applications installed, how they were downloaded (delivered to the device), dates and times of interest, and more. When considering if a user intentionally removed and reinstalled an application, the dates and times can be used to help your examination. If the delivery date is after the first download date, it may have been deleted and reinstalled.

A query that may help you handle this database is shared below. As always, modify these queries to work for you and your investigation. NOTE: The delivery data column of the appstate table contains blob data that pertains to the download. (Where did the download come from, and how was it delivered to the device?) This file is located at USERDATA/data/com.android.vending/databases/localappstate.db.

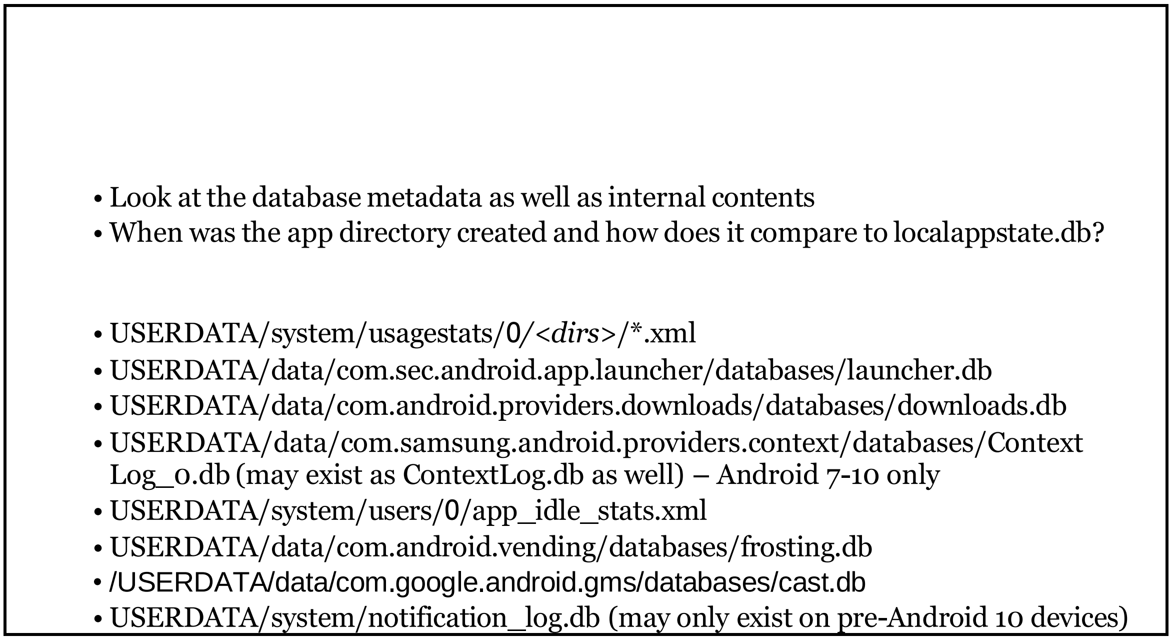
```
SELECT
package_name,
auto_update
As
"1=AutoUpda
te Set",
delivery_data,
DateTime(delivery_data_timestamp_ms / 1000,
'UNIXEPOCH') As "Delivery Date",
DateTime(first_download_ms / 1000, 'UNIXEPOCH') As
"First Download Date", account,
t
i
t
l
e
,

l
a
s
t
_
n
o
t
i
f
i
e
d
_
v
e
r
s
i
o
n
,
datetime(last_update_timestamp_ms/ 1000, 'UNIXEPOCH') As
"Last Update", datetime(install_request_timestamp_ms/ 1000,
'UNIXEPOCH') As "Install Request Date" From appstate
```



localappstate.db

☑	package_name	delivery_data
☑	com.badoo.mobile	
☑	com.blurb.checkout	
☑	com.google.android.street	
☑	org.wThamjeedeMujthaba	Й�Ygw1PPRZdozkyEMgLqzDzgwuz1o�https://android.clients.google.com/market/download/Download?packageName=org.wThamjeedeMujth MarketDA06169001170432815517O@Z����9NvfkWV6gOtxIUjob3C4drx3Yzs�https://android.clients.google.com/market/download/Downl
☑	com.infraware.polarisviewer5	
☑	org.telegram.messenger	ҫ�9CUthKY0_v6IiVK6gtoCUXdYRJo�https://android.clients.google.com/market/download/Download?packageName=org.telegram.messenger& MarketDA06169001170432815517O@j�https://android.clients.google.com/market/download/Download?packageName=org.telegram.messeng
☑	com.phxwg.marcopolo	�royiiEzdkM5yGJSuheE7vfE5mLzWIs�https://android.clients.google.com/market/download/Download?packageName=com.phxwg.marcopolo&ve MarketDA06169001170432815517O@�
☑	org.telegram.plus	
☑	com.loudtalks	��CiVXtTGV0kxZNSpmlpd_2uwI6VI�https://android.clients.google.com/market/download/Download?packageName=com.loudtalks&version(MarketDA06169001170432815517O@Z��mZ7k1XWxRkjA6rzUAZtjBS0oXEI�https://android.clients.google.com/market/download/Download?

Android applications store snapshots in the USERDATA/data/<application> directory, but these are controlled by the application and not the user. Access to these snapshots may require root access. While application information can be recovered from this location, another interesting location for recently used application snapshots, which are created when a user minimizes an app or something kicks them away from that screen, is USERDATA/system/recent_images/*.png. (NOTE: you may find that this exists in system_ce or system_de depending on the device.). Here, the examiner will find screenshots of recently used applications that were either interrupted by another action, minimized, or just navigated away from by the user. In this example, the screenshot on the left show graphics and a video in the users Gallery that were being viewed. If the user deletes a photo or video, this screenshot will not be modified. Keep this in mind for child exploitation type cases where the user may have forgotten to clean up graphics from the entire device. The screenshot on the right is the perfect example of what privacy features look like for some applications. If the application is secure, the screen is supposed to "go blank" when it is no longer in use. Here, Telegram Plus did the job and we cannot see any user data in this screenshot. You may also find recent activity in USERDATA/system_recent_tasks, which also tracks recent activity suspended by the Android device (application interrupted or moved into the background). You will notice that the name of the recent_image (###_task_thumbnail.png) will match the name of the recent_tasks (###_task.xml). These items may be cleaned up when the application is closed by the user.



- Look at the database metadata as well as internal contents
- When was the app directory created and how does it compare to localappstate.db?

- USERDATA/system/usagestats/0/*<dirs>*/*.xml
- USERDATA/data/com.sec.android.app.launcher/databases/launcher.db
- USERDATA/data/com.android.providers.downloads/databases/downloads.db
- USERDATA/data/com.samsung.android.providers.context/databases/ContextLog_0.db (may exist as ContextLog.db as well) – Android 7-10 only
- USERDATA/system/users/0/app_idle_stats.xml
- USERDATA/data/com.android.vending/databases/frosting.db
- /USERDATA/data/com.google.android.gms/databases/cast.db
- USERDATA/system/notification_log.db (may only exist on pre-Android 10 devices)

- USERDATA/data/com.google.android.apps.wellbeing/databases/app_ usage – Digital Wellbeing

FOR585 | Smartphone Forensic Analysis In-Depth

Applications can be installed on a device and then never used. Proving use is sometimes the hardest part of a smartphone investigation. The timestamps in each application directory should reflect when the application was installed and used. Application usage traces may also exist in the following files:

- USERDATA/system/usagestats/0/<various directories>/*.xml (multiple files exist)
- USERDATA/data/com.sec.android.app.launcher/databases/launcher.db
- USERDATA/data/com.samsung.android.providers.context/databases/ContextLog_0.db
- USERDATA/system/users/0/app_idle_stats.xml
- USERDATA/data/com.android.vending/databases/frosting.db
- USERDATA/system/notification_log.db
- USERDATA/data/com.google.android.apps.wellbeing/databases/app_usage
- /USERDATA/data/com.google.android.gms/databases/cast.db - Contains info about discovered Cast devices
    - Table = DeviceInfo
    - Friendly name
    - Model
    - LAN IP address

Although the data in these folders does not directly reflect only application information, remnants may be found to support your other findings. One database of interest is the downloads.db. This database is located at USERDATA/data/com.android.providers.downloads/databases. The table of interest is “downloads,” which will contain the name of the .jar, method for downloading, “hints” on what the application was, where the download was stored, and more. Some items in this table may be encrypted. Another database that contains applications that are removed from the device is the launcher.db located at USERDATA/data/com.sec.android.app.launcher/databases. The tables “appOrder” and “favorites” contain application information even after the application was deleted from the device. The last four files shown in the paths above will be discussed in the upcoming slides.

Another file of interest is the app_idle_stats.xml located at USERDATA/system/users/0. We recommend searching for the file name to ensure the file isn’t overlooked on devices that have Secure Boot or multiple users. If this file is available, it provides information on application usage and tracks how long the application has been idle. The time format shown above is Android chronometer time format, which essentially is like a stopwatch tracking time.[1]

Reference:

- https://for585.com/chronometer

- Text when the text is put on the clipboard (i.e., copied)
- In cases where screenshots are taken, path of the picture is copied to the clipboard
- Stored in HTML

This is found on Samsung devices and is stored in USERDATA/semclipboard/. The files are stored as “clips” and contain content of the clipboard. The clip files have been observed to contain text when the text is put on the clipboard (copied) or in cases where screenshots are taken, the path of the picture is copied to the clipboard. The “clips” are stored in HTML.

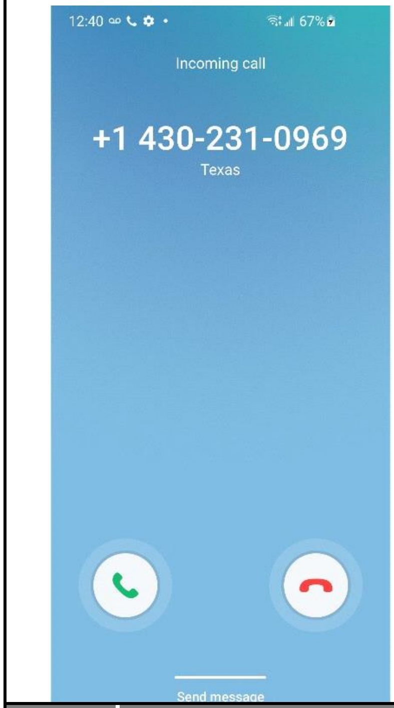
Relevant fields:

- mTimestamp
- mValue (when copied content is text or some other string)
- mHTML

- mInitBaseValue (path of picture when copied content is a screenshot)

When screenshots are taken the folder will also contain a copy of the picture and a thumbnail version of the picture. The clip files are Java ArrayLists, which do not render well in text editors/hex viewers. Physical Analyzer can render the whole file. ALEAPP will pull timestamp and clipboard contents only.

These artifacts can be specifically beneficial for passwords the user may have copied/pasted into apps, webforms, text message, etc. The data in these files is non-volatile.



CONTAINS RUNTIME EXISTS FOR ANDROID XML FORMAT FOR
ABX FORMAT FOR PERMISSIONS FOR 10, 11, &

EACH
APP
(DANGER
OUS)

FOR585 | Smartphone Forensic Analysis In-Depth

The android-permissions.xml contains runtime (dangerous) permissions for each app. This file is present in Android 10, 11, & 12. The paths are:

- Android 10 --> USERDATA/system/users/0/
- Android 11 & 12 --> USERDATA/misc_de/0/apexdata/com.android.permission/

The files are stored as regular XML (non-ABX) Android 10 & 11. and ABX in Android 12. These are the permissions that have been granted by the user to the app (not necessarily the ones the app requested via AndroidManifest.xml). Imagine the impact of this file in malware investigations. Josh Hickman wrote a useful blog which can be found here: https://thebinaryhick.blog/2021/01/26/androids-dangerous-permissions/.

FOR585 | Smartphone Forensic Analysis In-Depth

Digital Wellbeing is an Android application that tracks application usage. This application is now installed by default. Most devices running Android 10 and higher have Digital Wellbeing. According to Barak Goldberg, the Pixel is one of the only devices to have this application on Android 9 - 12.[1] Everything from starting an app, stopping it, pushing it to background and bringing it into focus are tracked in the USERDATA/data/com.google.android.apps.wellbeing/databases/app_usage. The user leverages this file to see where they are spending time and to configure how frequently and when they are “bothered” by their installed applications. Digital wellbeing also keeps track of web history on Chrome. Web history is not captured by default; user has to enable this feature.

Josh Hickman released research and his blog on Digital Wellbeing, which has been implemented into ALEAPP. This blog can be read here:

https://thebinaryhick.blog/2020/02/22/walking-the-android-timeline- using-androids-digital-wellbeing-to-timeline-android-activity/. Physical Analyzer and AXIOM also parse this artifact.

This directory also stores the following databases, which may be of interest:

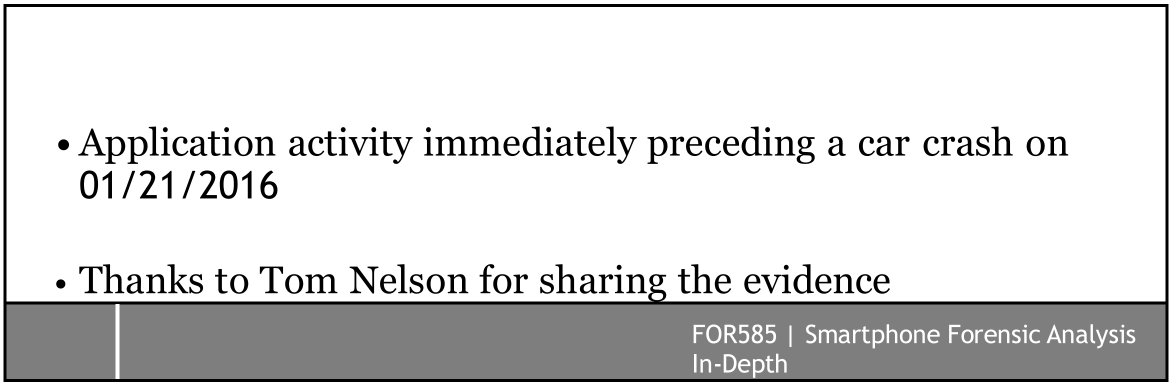
- bedtime
- Sleep_detection
- App_config

Two different locations between the Pixel (vanilla Android) and Samsung

- Pixel: USERDATA/data/com.google.android.apps.wellbeing
- Samsung: USERDATA/data/com.samsung.android.forest
- Other OEMs may have their own variation of this function

- Time limits
  - Samsung = ~7 days
  - Pixel = ~30 days

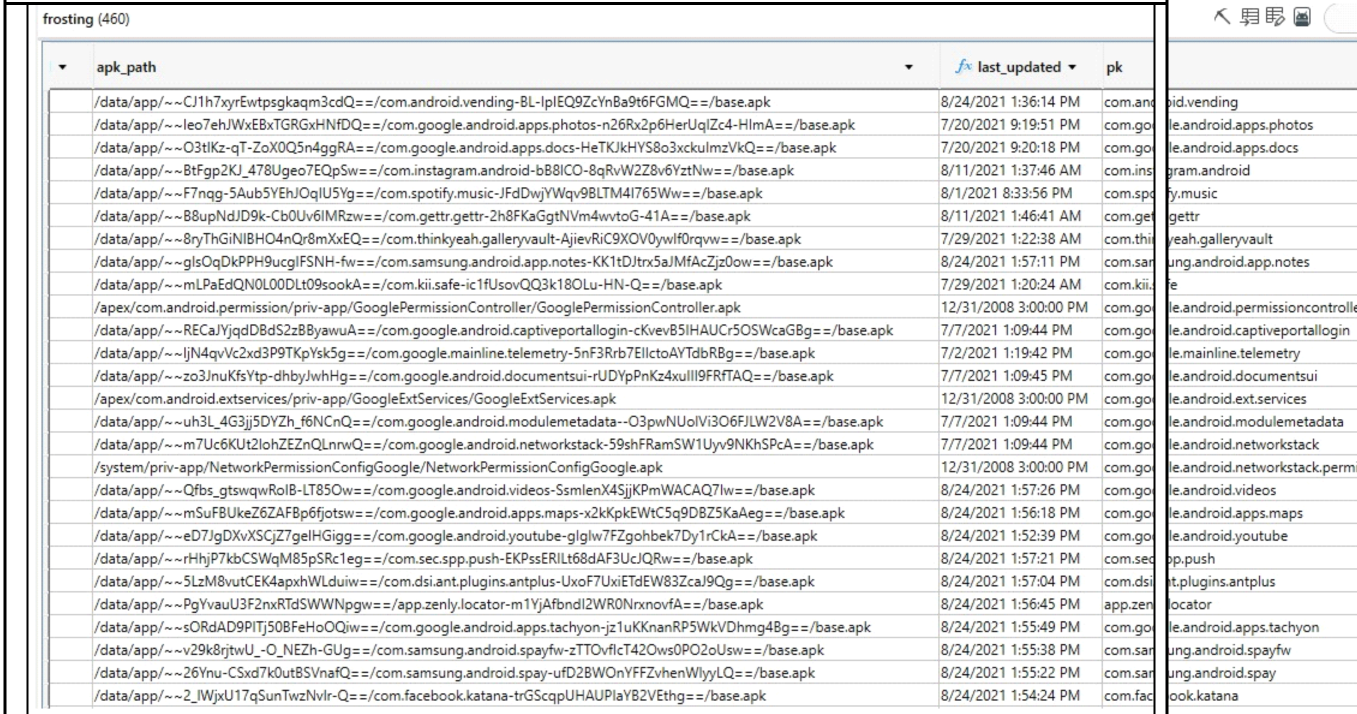
Reference:
[1] https://for585.com/wellbeing



- Application activity immediately preceding a car crash on 01/21/2016
- Thanks to Tom Nelson for sharing the evidence

FOR585 | Smartphone Forensic Analysis In-Depth

Tom Nelson of the Deschutes County Sheriff's office worked a car crash investigation that occurred on January 21, 2016. The driver's phone was examined to determine if he was distracted while driving. The device was a Samsung SCH-i545. The file shown here was located at USERDATA/data/com.samsung.android.providers.context/databases/ContextLog_0.db. This file exists on Samsung devices running at least OS 7. If you stumble upon this file, examine every table for relevance to your investigation. It's great at piecing together application usage. NOTE, some of the screenshot is blocked, as it was provided directly from Mr. Nelson and contained case information. This file is like KnowledgeC for iOS devices.



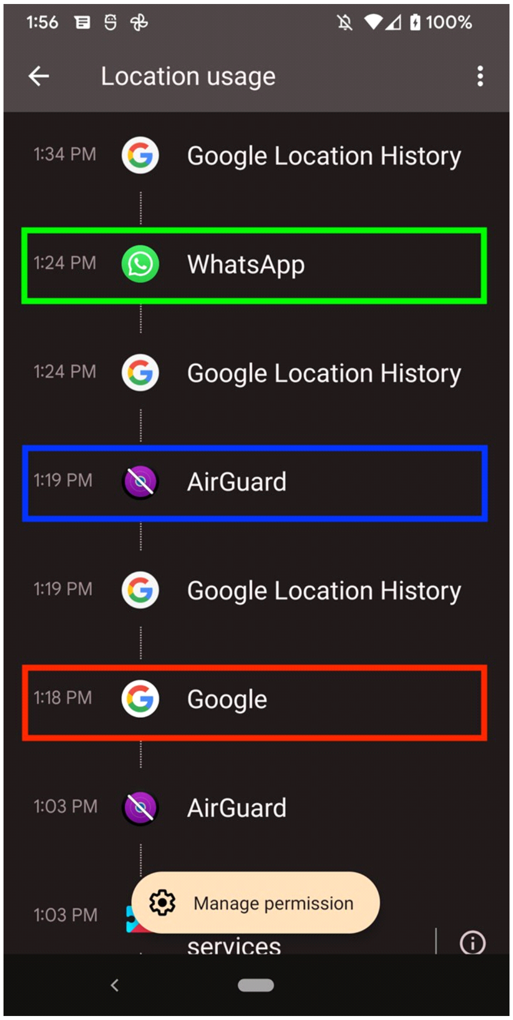
frosting (460)

	apk_path	last_updated	pk
	/data/app/~~CJ1h7xyrEwtpsgkaqm3cdQ==/com.android.vending-BL-IpIEQ9ZcYnBa9t6FGMQ==/base.apk	8/24/2021 1:36:14 PM	com.and oid.vending
	/data/app/~~leo7ehJWxEBxTGRGxHNfDQ==/com.google.android.apps.photos-n26Rx2p6HerUqIZc4-HImA==/base.apk	7/20/2021 9:19:51 PM	com.goo le.android.apps.photos
	/data/app/~~O3tIKz-qT-ZoX0Q5n4ggRA==/com.google.android.apps.docs-HeTKJkHYS8o3xckulmzVkQ==/base.apk	7/20/2021 9:20:18 PM	com.goo le.android.apps.docs
	/data/app/~~BtFgp2KJ_478Ugeo7EQpSw==/com.instagram.android-bB8ICO-8qRvW2Z8v6YztNw==/base.apk	8/11/2021 1:37:46 AM	com.ins gram.android
	/data/app/~~F7nqg-5Aub5YEhJOqIU5Yg==/com.spotify.music-JFdDwjYWqv9BLTM4I765Ww==/base.apk	8/1/2021 8:33:56 PM	com.spo fy.music
	/data/app/~~B8upNdJD9k-Cb0Uv6IMRzw==/com.gettr.gettr-2h8FKaGgtNVm4wvtoG-41A==/base.apk	8/11/2021 1:46:41 AM	com.get gettr
	/data/app/~~8ryThGiNIBHO4nQr8mXxEQ==/com.thinkyeah.galleryvault-AjievRiC9XOV0ywIf0rqvw==/base.apk	7/29/2021 1:22:38 AM	com.thi yeah.galleryvault
	/data/app/~~glsOqDkPPH9ucgIFSNH-fw==/com.samsung.android.app.notes-KK1tDJtrx5aJMfAcZjz0ow==/base.apk	8/24/2021 1:57:11 PM	com.sar ung.android.app.notes
	/data/app/~~mLPaEdQN0L00DLt09sookA==/com.kii.safe-ic1fUsovQQ3k18OLu-HN-Q==/base.apk	7/29/2021 1:20:24 AM	com.kii. fe
	/apex/com.android.permission/priv-app/GooglePermissionController/GooglePermissionController.apk	12/31/2008 3:00:00 PM	com.goo le.android.permissioncontrolle
	/data/app/~~RECaJYjqdDBdS2zBByawuA==/com.google.android.captiveportallogin-cKvevB5IHAUCr5OSWcaGBg==/base.apk	7/7/2021 1:09:44 PM	com.goo le.android.captiveportallogin
	/data/app/~~ljN4qvVc2xd3P9TKpYsk5g==/com.google.mainline.telemetry-5nF3Rrb7EllctoAYTdbRBg==/base.apk	7/2/2021 1:19:42 PM	com.goo le.mainline.telemetry
	/data/app/~~zo3JnuKfsYtp-dhbyJwhHg==/com.google.android.documentsui-rUDYpPnKz4xuIll9FRfTAQ==/base.apk	7/7/2021 1:09:45 PM	com.goo le.android.documentsui
	/apex/com.android.extservices/priv-app/GoogleExtServices/GoogleExtServices.apk	12/31/2008 3:00:00 PM	com.goo le.android.ext.services
	/data/app/~~uh3L_4G3jj5DYZh_f6NCnQ==/com.google.android.modulemetadata--O3pwNUolVi3O6FJLW2V8A==/base.apk	7/7/2021 1:09:44 PM	com.goo le.android.modulemetadata
	/data/app/~~m7Uc6KUt2IohZEZnQLnrwQ==/com.google.android.networkstack-59shFRamSW1Uyv9NKhSPcA==/base.apk	7/7/2021 1:09:44 PM	com.goo le.android.networkstack
	/system/priv-app/NetworkPermissionConfigGoogle/NetworkPermissionConfigGoogle.apk	12/31/2008 3:00:00 PM	com.goo le.android.networkstack.permi
	/data/app/~~Qfbs_gtswqwRoIB-LT85Ow==/com.google.android.videos-SsmlenX4SjjKPmWACAQ7Iw==/base.apk	8/24/2021 1:57:26 PM	com.goo le.android.videos
	/data/app/~~mSuFBUkeZ6ZAFBp6fjotsw==/com.google.android.apps.maps-x2kKpkEWtC5q9DBZ5KaAeg==/base.apk	8/24/2021 1:56:18 PM	com.goo le.android.apps.maps
	/data/app/~~eD7JgDXvXSCjZ7geIHGigg==/com.google.android.youtube-gIglw7FZgohbek7Dy1rCkA==/base.apk	8/24/2021 1:52:39 PM	com.goo le.android.youtube
	/data/app/~~rHhjP7kbCSWqM85pSRc1eg==/com.sec.spp.push-EKPssERILt68dAF3UcJQRw==/base.apk	8/24/2021 1:57:21 PM	com.sec p.push
	/data/app/~~5LzM8vutCEK4apxhWLduiw==/com.dsi.ant.plugins.antplus-UxoF7UxiETdEW83ZcaJ9Qg==/base.apk	8/24/2021 1:57:04 PM	com.dsi t.plugins.antplus
	/data/app/~~PgYvauU3F2nxRTdSWWNpgw==/app.zenly.locator-m1YjAfbndI2WR0NrxnovfA==/base.apk	8/24/2021 1:56:45 PM	app.zen locator
	/data/app/~~sORdAD9PITj50BFeHoOQiw==/com.google.android.apps.tachyon-jz1uKKnanRP5WkVDhmg4Bg==/base.apk	8/24/2021 1:55:49 PM	com.goo le.android.apps.tachyon
	/data/app/~~v29k8rjtwU_-O_NEZh-GUg==/com.samsung.android.spayfw-zTTOvflcT42Ows0PO2oUsw==/base.apk	8/24/2021 1:55:38 PM	com.sar ung.android.spayfw
	/data/app/~~26Ynu-CSxd7k0utBSVnafQ==/com.samsung.android.spay-ufD2BWOnYFFZvhenWlyyLQ==/base.apk	8/24/2021 1:55:22 PM	com.sar ung.android.spay
	/data/app/~~2_lWjxU17qSunTwzNvlr-Q==/com.facebook.katana-trGScqpUHAUPIaYB2VEthg==/base.apk	8/24/2021 1:54:24 PM	com.fac ook.katana

The database found at USERDATA/data/com.android.vending/databases/frosting.db is extremely helpful at proving an application existed on the device. Here we will see the application name, application path, and last update timestamp and data, which may provide additional data about the application. The query below will help you parse this file. This query is also in your course notebook.

```
s
e
l
e
c
t

p
k
,
apk_path,
datetime(last_updated/1000,'UNIXEPOCH','localtime
') AS "Last Updated", data
from frosting
```

- Tracks which applications are using location, camera, and microphone
- Tracks the last 24 hours of usage
- Forensics on this capability may vary as the devices store data differently depending on manufacturer
  - Get ready to research!
- Time seen in this view on the device is local time
- Great for Malware investigations



1:34 PM Google Location History

1:24 PM WhatsApp

1:24 PM Google Location History

1:19 PM AirGuard

1:19 PM Google Location History

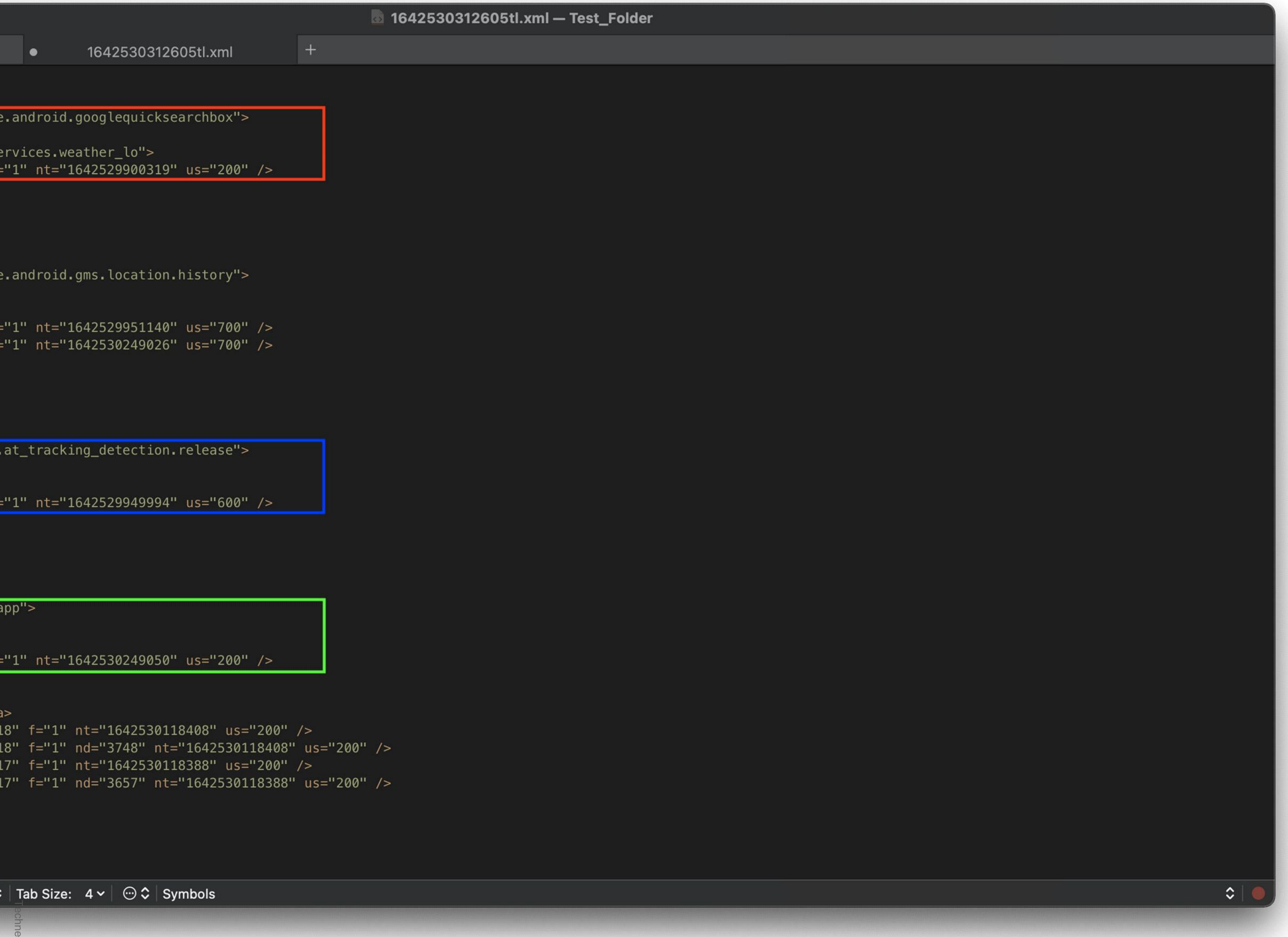
1:18 PM Google

1:03 PM AirGuard

1:03 PM services

The Android 12 Privacy Dashboard is a new feature that was found to be present in the Pixel and Samsung devices. A special thanks to Sahil Dudani, from Virginia Tech for assisting Josh Hickman in his research here. https://thebinaryhick.blog/2022/01/22/snooping-on- android-12s-privacy-dashboard/

The Privacy Dashboard is found at USERDATA/system/appops/discrete. The Privacy Dashboard is designed to track Location, camera, and microphone usage by native and third-party apps. This is similar to some aspects of TCC.db found on iOS devices. This will be covered in section 3 of this course. The Privacy Dashboard keeps the last 24 hours of usage and is stored in the ABX format.



```
e.android.googlequicksearchbox">

ervices.weather_lo">
="1" nt="1642529900319" us="200" />





e.android.gms.location.history">


="1" nt="1642529951140" us="700" />
="1" nt="1642530249026" us="700" />





.at_tracking_detection.release">


="1" nt="1642529949994" us="600" />





app">


="1" nt="1642530249050" us="200" />


a>
18" f="1" nt="1642530118408" us="200" />
18" f="1" nd="3748" nt="1642530118408" us="200" />
17" f="1" nt="1642530118388" us="200" />
17" f="1" nd="3657" nt="1642530118388" us="200" />
```

Package tag values (pn) are the package name and tag values (op) have been observed as follows:

- op=1 is Location
- op=26 is Camera
- op=27 is Microphone

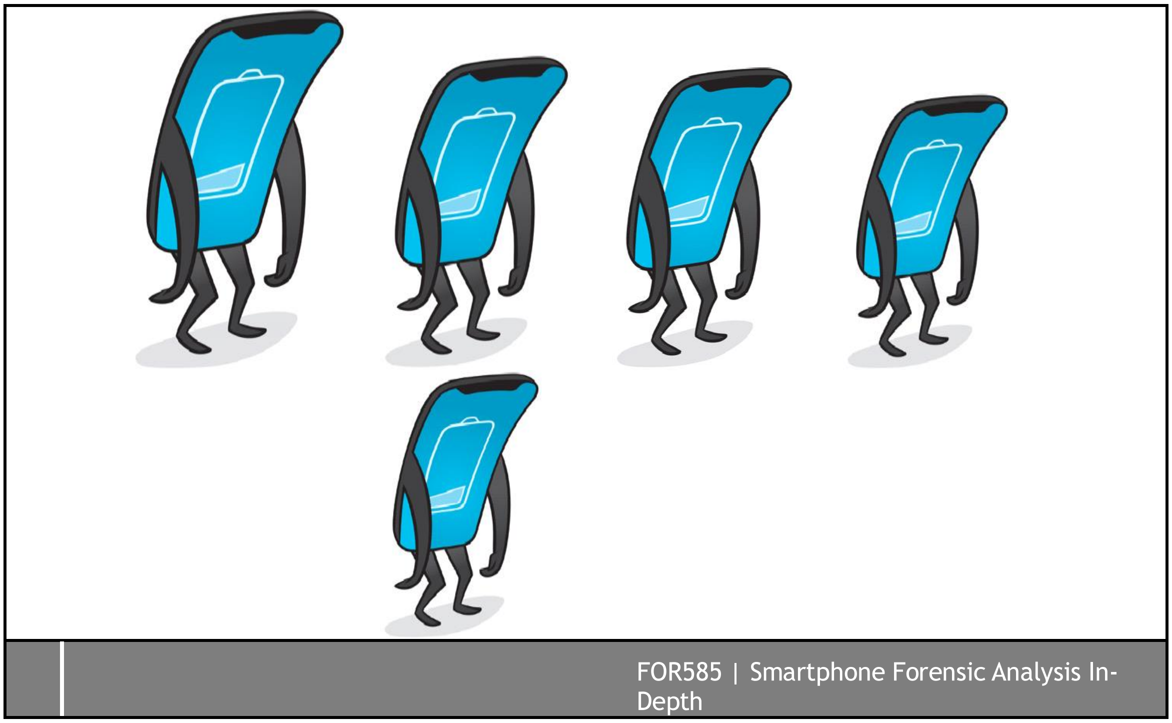
Other values worth noting:

- "nt" tag values are likely noteOps values, i.e., time at which the operation (location, camera, microphone usage) was performed by the package.
- "nd" tag values represent operation duration in milliseconds. For example, if Camera is used to take a 2- minute video, the op entry for microphone usage will show a "nd" value of approximately two minutes in milliseconds.
- "at" tag values are attribution tags. The tags provide context. In the right screenshot above, an "at" value of "services.weather_lo" is seen. This is a reflection of my hitting the weather icon on the phone homescreen. That is powered by Google Quick Search Box. Not all entries will have "at" values.

An app can have multiple operations in one usage. For example, the camera can simultaneously use the camera and location (to append exif data) in one pn entry. So, an examiner should expect to find an entry for camera, along with op=1 and op=26 and associated nt timestamps. This type of information will be extremely helpful in malware investigations as well as proving application usage for a specific user logged into and Android device. Keep in mind, further work would be needed by the examiner as background activity is logged here.

When a user deletes an application from his device, is the data truly gone? Is this data recoverable? It honestly depends on the device and the version of Android running. Best practices encourage the examiner to examine the databases provided by the tool. This means to go to the Databases section in Physical Analyzer and Oxygen or examine the file system and find application directories of interest. The image file should also be loaded into AXIOM or IEF where a full search can be run across the entire image to ensure that the data is not present on the device. Should you not have a commercial tool, do the same in Autopsy.

The SD card and/or USERDATA/media directory should be examined for traces of files associated with the application. Deleting an application does not mean the data stored in the SD card is removed. The CACHE partition may contain a vast amount of data as well. As a last-ditch effort, Google could be contacted to provide the user data stored in the cloud. Of course, this requires proper legal authority or consent.



FOR585 | Smartphone Forensic Analysis In-Depth

Most wouldn't consider what is killing the battery as something that may aid a forensic investigation. Think about your device and what drains your battery—application usage and streaming! Here we have located files of interest that track what is draining the battery.

- USERDATA/system/batterystats.bin
- USERDATA/data/com.google.android.gms/shared_prefs/Batterystats.xml
- USERDATA/data/com.google.android.gms/files/BatterystatsDumpsysTask.gz
- USERDATA/system/batterystats*.xml

The file batterystats.bin may show loss of battery power while an application was launched and in use. All of these files track what is killing the battery. Dive in and enjoy!

- May contain more than that!

FOR585 | Smartphone Forensic Analysis In-Depth

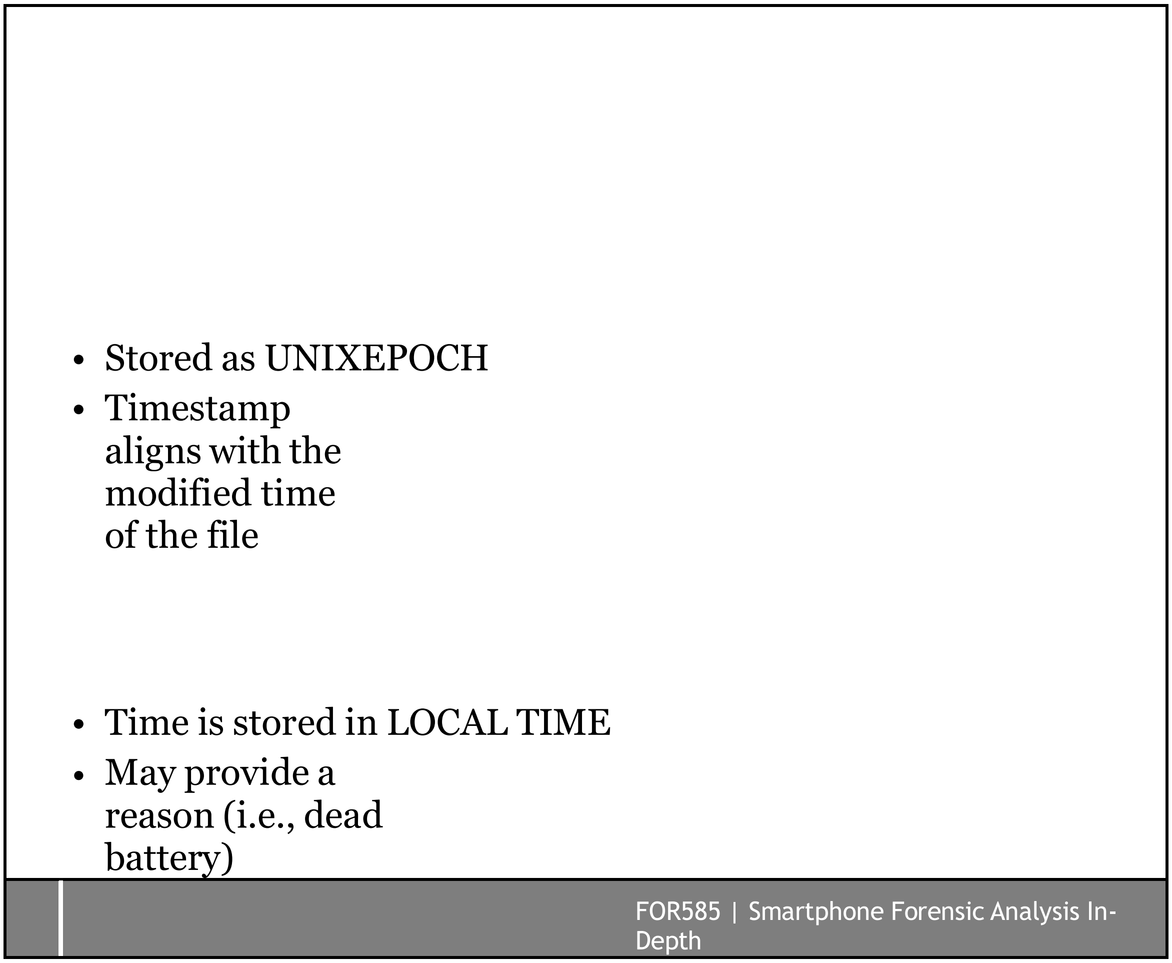
Battery Usage found in Android 12 is different than Battery Stats found on older devices. Battery Usage supposedly tracks battery usage for the past 24 hours (according to UI). NOTE, in testing, it was noted that the data showed up to 36 hours' worth of data.

USERDATA/data/com.google.android.settings.intelligence/databases
/battery-usage-db-v4 Table of interest:
BatteryState

SQL Query:

```
SELECT
BatteryState.appLabel AS "App Name",
BatteryState.bootTimestamp AS "Milliseconds (ms)
Since Last Boot", datetime
(BatteryState.timestamp/1000, 'unixepoch') AS
"Timestamp (UTC)",
BatteryState.foregroundUsageTimeInMs AS "Foreground
Usage Since Last Boot (ms)",
BatteryState.backgroundUsageTimeInMs AS "Background
Usage Since Last Boot (ms)" FROM
BatteryState
```

The last boot time can be calculated by subtracting the value of column "bootTimestamap" from column "timestamp". NOTE: the column "zoneId" does contain timezone information, however, it is not clear whether this is timezone of the time settings, timezone of the device when the timestamp is created, or something else. ALEAPP is currently parsing this database. A great reference blog was written by Kevin Pagano: https://www.stark4n6.com/2021/12/application-battery-usage-via-settings.html.



- Stored as UNIXEPOCH
- Timestamp aligns with the modified time of the file

- Time is stored in LOCAL TIME
- May provide a reason (i.e., dead battery)

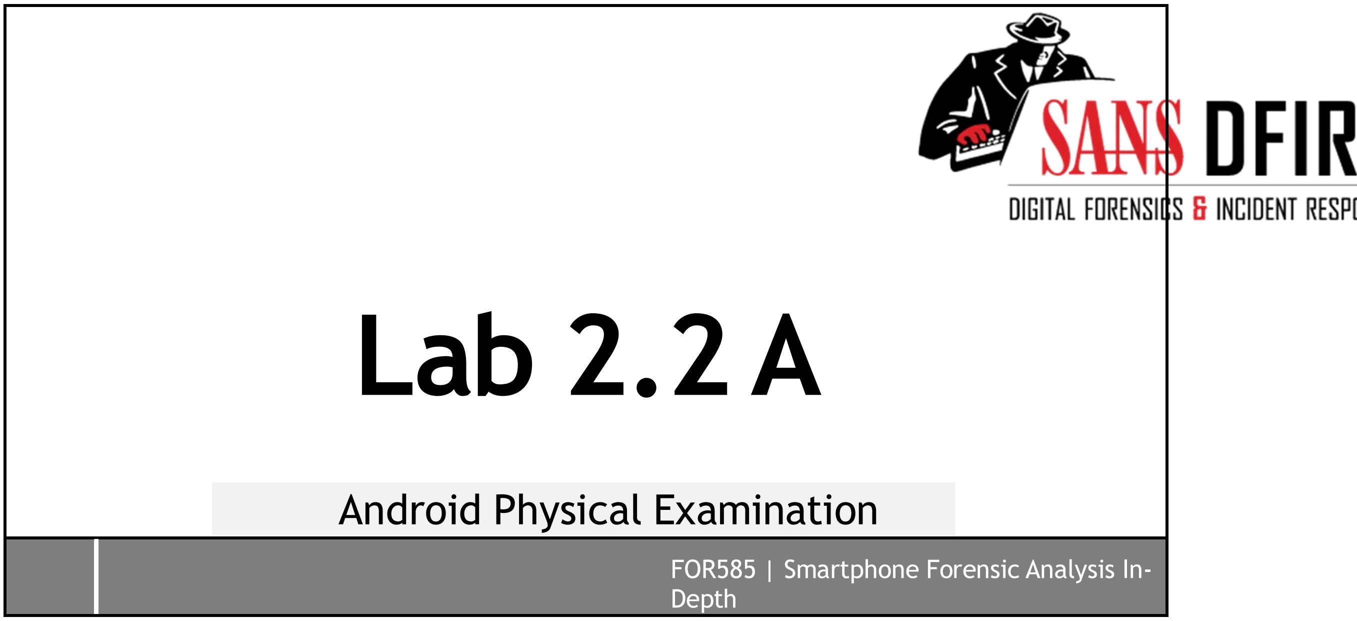
Shutdown checkpoints are stored in USERDATA/system/shutdown-checkpoints. The files are named “checkpoint-UNIXEPOCH”, and the files are in plain text. The timestamp in file name aligns with Modified time of file. The shutdown checkpoints keep a log of when the phone was shutdown/rebooted. Something worth noting is that the time is stored in local time and can also provide a reason for the shutdown (e.g., dead battery). Kevin Pagano wrote a blog on this: https://www.stark4n6.com/2022/01/shutdown-checkpoints-in- android-12.html

At the time of writing this, ALEAPP was one of the only tools to parse this artifact.

ALEAPP is a tool brought to you by Alexis Brignoni. The official ALEAPP blog can be found here: https://abrignoni.blogspot.com/2020/02/aleapp-android-logs-events-and-protobuf.html. There are many contributors to ALEAPP and it's always progressing. ALEAPP is essentially a compilation of research for collaborators and researchers in the DFIR community. It handles more than just running queries against SQLite databases and provides a fast solution to triage, validate, and verify your findings. ALEAPP has integrated a lot of work from the community to include most artifacts mentioned in this section for Android 12.

To run ALEAPP, you can simply navigate to the ALEAPP directory and type the following:

```
python aleappGUI.py
```



Lab 2.2 A

Android Physical Examination

This page intentionally left blank.

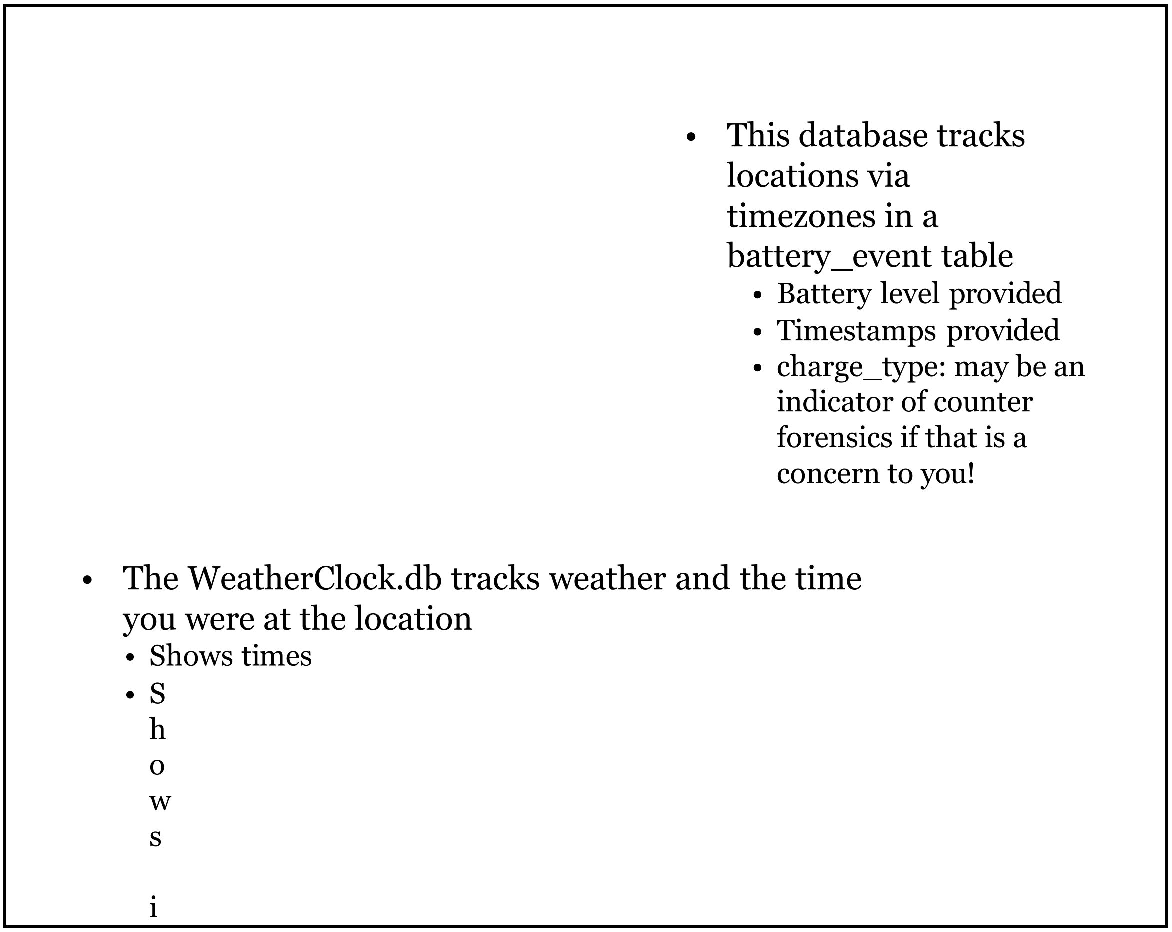
- cache.cell and cache.wifi
  - https://github.com/packetlss/android-locdump

One of the first things you should verify in regard to location information is if the user has location services enabled on their device. These settings are saved in the googlesettings.db in USERDATA/data/com.google.android.gsf/databases/. Here, if you see the value of 1 for “use_location_for_services,” then you can state that the user has location information enabled on their device.

Once we have established that, the floodgates have opened. Everything we discussed in earlier sections can be applied, but what if you still aren’t finding what you need. What if the user cleared their location history?
Other files may help you. Files like cache.cell and cache.wifi may aid if you have a rooted device or full access to the device via physical acquisition.

Browser data may include location information that the user doesn’t realize. Here we see that the user has a last name of Mahalik and is staying in room 226. This was autosaved by Chrome but could exist in any browser installed on the Android. Chances are this is the user logging into the hotel’s Wi-Fi, so I would examine the timestamps and attempt to associate that to a Wi-Fi connection to cement the user being in that location at a specific date and time.

The database found in /data/system/wifigeofence.db is only on Samsung devices and keeps a record of BSSID's, last connection times, and lat/long.



- This database tracks locations via timezones in a battery_event table
  - Battery level provided
  - Timestamps provided
  - charge_type: may be an indicator of counter forensics if that is a concern to you!

- The WeatherClock.db tracks weather and the time you were at the location
  - Shows times
  - S
    h
    o
    w
    s

    i

fitwas fetched for the user

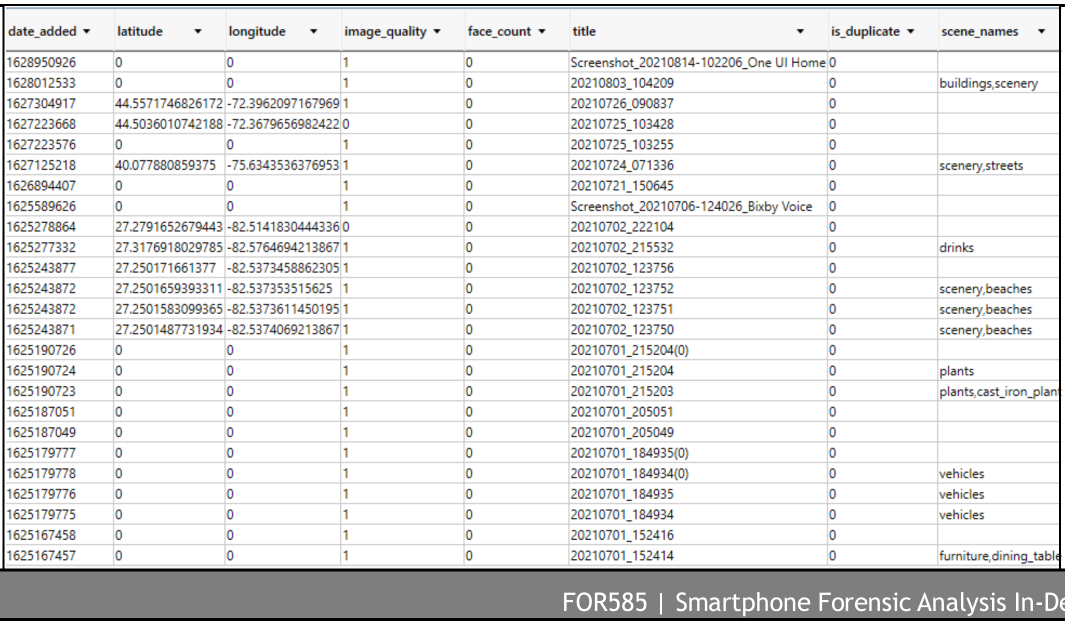
- So much more!

These files may not be located on all Android devices, but when you locate them, they are magical. The turbo.db database was extracted from an Android running Oreo (Android v8). You never know what you may stumble upon that will help you place the device in a location or a time zone at a period in time.

A blog by Kevin Pagano references turbo.db artifacts: https://www.stark4n6.com/2020/12/charging-battery-with-turbo- HYPERLINK "http://www.stark4n6.com/2020/12/charging-battery-with-turbo-db.html"db.html.

The WeatherClock database can be found at /com.sec.android.daemonapp/db/weatherClock. The database will store the most recent location for which the user visited, and the WAL will store other recent visits.

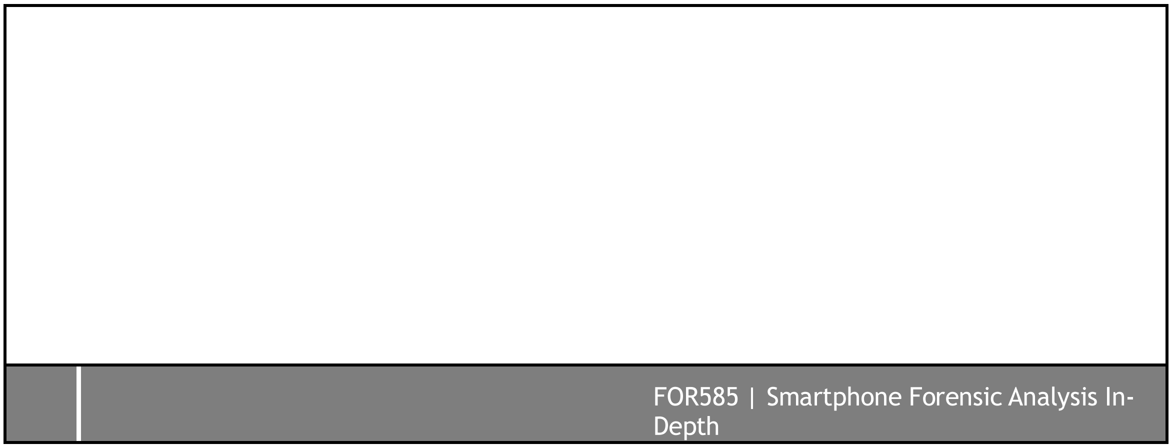
Jared Barnhart, Heather Mahalik, and Ian Whiffin dove into researching location artifacts on iOS and Android and rating them on what you can trust based upon timestamp and location accuracy. The full webinar is available on Cellebrite's website. https://cellebrite.com/en/episode-15-ibeg-to-dfir-location-data-on-ios-and-android-devices/. Refer to the cheat sheet in your section 2 folder in Dropbox. Some of these databases are parsed by the commercial tools and ALEAPP. Always make sure you verify that they are being parsed for all relevant information when your investigation involved location artifacts.



date_added	latitude	longitude	image_quality	face_count	title	is_duplicate	scene_names
1628950926	0	0	1	0	Screenshot_20210814-102206_One UI Home	0	
1628012533	0	0	1	0	20210803_104209	0	buildings,scenery
1627304917	44.5571746826172	-72.3962097167969	1	0	20210726_090837	0	
1627223668	44.5036010742188	-72.3679656982422	0	0	20210725_103428	0	
1627223576	0	0	1	0	20210725_103255	0	
1627125218	40.077880859375	-75.6343536376953	1	0	20210724_071336	0	scenery,streets
1626894407	0	0	1	0	20210721_150645	0	
1625589626	0	0	1	0	Screenshot_20210706-124026_Bixby Voice	0	
1625278864	27.2791652679443	-82.5141830444336	0	0	20210702_222104	0	
1625277332	27.3176918029785	-82.5764694213867	1	0	20210702_215532	0	drinks
1625243877	27.250171661377	-82.5373458862305	1	0	20210702_123756	0	
1625243872	27.2501659393311	-82.537353515625	1	0	20210702_123752	0	scenery,beaches
1625243872	27.2501583099365	-82.5373611450195	1	0	20210702_123751	0	scenery,beaches
1625243871	27.2501487731934	-82.5374069213867	1	0	20210702_123750	0	scenery,beaches
1625190726	0	0	1	0	20210701_215204(0)	0	
1625190724	0	0	1	0	20210701_215204	0	plants
1625190723	0	0	1	0	20210701_215203	0	plants,cast_iron_plant
1625187051	0	0	1	0	20210701_205051	0	
1625187049	0	0	1	0	20210701_205049	0	
1625179777	0	0	1	0	20210701_184935(0)	0	
1625179778	0	0	1	0	20210701_184934(0)	0	vehicles
1625179776	0	0	1	0	20210701_184935	0	vehicles
1625179775	0	0	1	0	20210701_184934	0	vehicles
1625167458	0	0	1	0	20210701_152416	0	
1625167457	0	0	1	0	20210701_152414	0	furniture,dining_table

When it comes to location artifacts you can trust on Android, dme.db is one of the leaders. This database is found at /data/data/com.samsung.storyservice/databases/dme.db. The purpose of this file is for stories and what you want to share. It is designed to record items in the photo gallery including GPS coordinates of where the photo/video was taken.

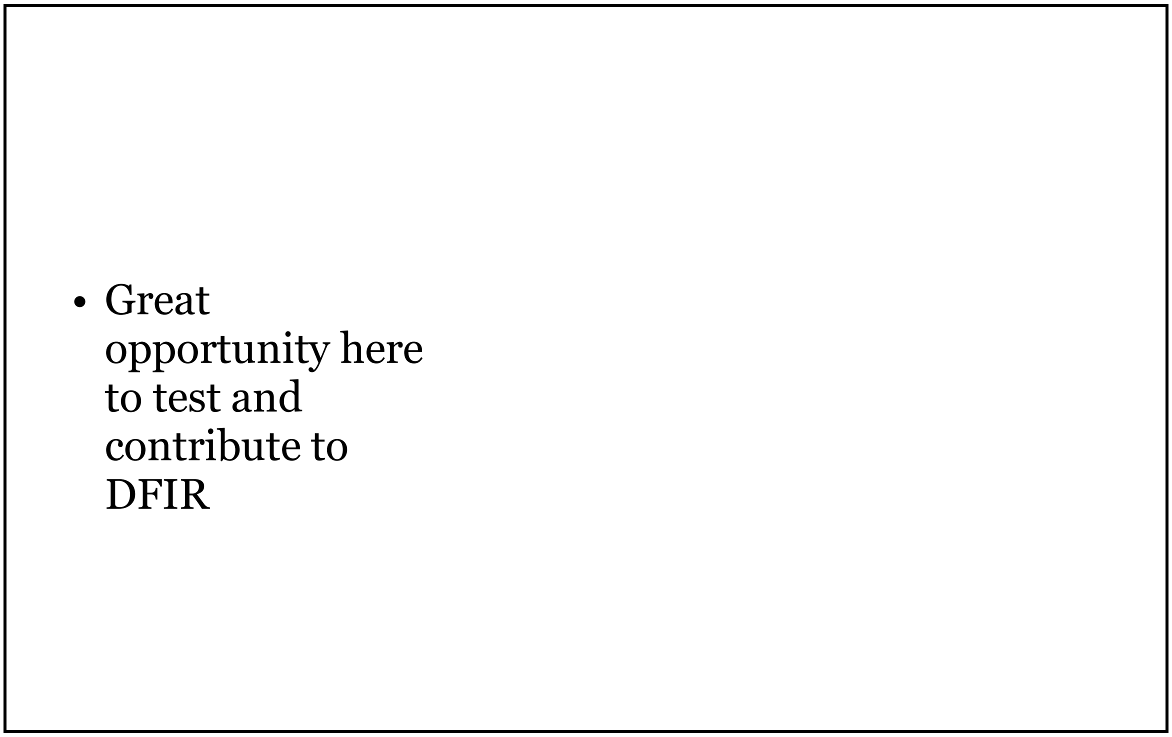
Reference:
[1] https://for585.com/locationdata - Cellebrite’s I Beg to DFIR presentation on location artifacts



FOR585 | Smartphone Forensic Analysis In-Depth

The iwc_dump.txt is located at /data/log/wifi/iwc/iwc_dump.txt and is designed with wireless networks, tracks the frequency of connections, and tracks the lost locations. During testing, Heather found that the locations, timestamps, and frequency were extrememly accurate. This information is powerful as it puts the user at a location as well as leaving a location at certain points in time.

Reference:
[1] https://for585.com/locationdata - Cellebrite's I Beg to DFIR presentation on location artifacts

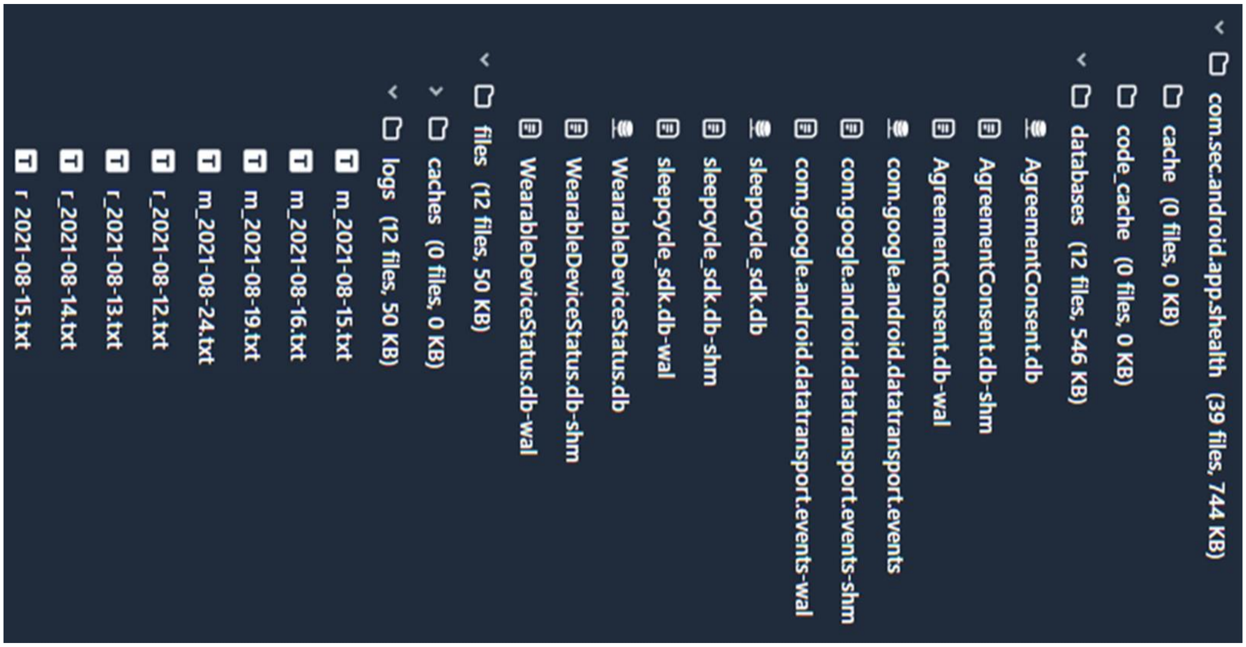


- Great opportunity here to test and contribute to DFIR

This file presents a research opportunity. The locations found in /data/log/dumpstate_app_native.zip/dumpstate_app_native.txt are locations that Heather visited while creating data for a presentation she did with Jared Barnhart and Ian Whiffin. However, the timestamps do not make sense, and the frequency of the scanning is odd. The locations listed are places she visited several times.

Samsung Health is a hassle. There is not a better way to say it, which is most likely why commercial tools are not parsing this data correctly. Files are stored in databases and text files. The text files exist as “r” and “m” for each day of activity. You will find the relevant data in the “r” files, however searching within the file is quite a task. You will find this out shortly in the upcoming lab where a question will lead you down the path of Samsung Health.

We recommend you start with leveraging AXIOM Dynamic App Finder, Physical Analyzer Fuzzy Model Plug-in, and the App Genie. Once you have files of interest from here, you may find you end up in the right location to uncover some exercise.



```
com.sec.android.app.shealth  (39 files, 744 KB)
  cache  (0 files, 0 KB)
  code_cache  (0 files, 0 KB)
  databases  (12 files, 546 KB)
    AgreementConsent.db
    AgreementConsent.db-shm
    AgreementConsent.db-wal
    com.google.android.datatransport.events
    com.google.android.datatransport.events-shm
    com.google.android.datatransport.events-wal
    sleepcycle_sdk.db
    sleepcycle_sdk.db-shm
    sleepcycle_sdk.db-wal
    WearableDeviceStatus.db
    WearableDeviceStatus.db-shm
    WearableDeviceStatus.db-wal
  files  (12 files, 50 KB)
    caches  (0 files, 0 KB)
    logs  (12 files, 50 KB)
      m_2021-08-15.txt
      m_2021-08-16.txt
      m_2021-08-19.txt
      m_2021-08-24.txt
      r_2021-08-12.txt
      r_2021-08-13.txt
      r_2021-08-14.txt
      r 2021-08-15.txt
```

- Heart rate monitoring, etc.

- Tracks even when you aren’t working out
- Setting is OFF by default, so user would have to enable

Garmin wearables have differing capabilities. For example, not all wearables are capable of real-time heart rate monitoring. Garmin can passively detect activities by a user and log the activity without a user actively starting an exercise in the app or on the wearable. This setting is OFF by default. If setting is on, it could log activity, unbeknownst to the user, that could be pertinent, which could include steps, heart rate, location, steps climbed. The type of data logged would be dependent on the capabilities of the wearable. Most of the relevant data available in the databases is stored in JSON. The query below is associated to device notifications.

```
SELECT
datetime(notification_info.postTime/1000,'unixepoch') AS "Notification Time
(UTC)", notification_info.packageName AS "App",
notification_info.title AS
"Notification Title",
notification_info.message
AS "Notification" FROM
notification_info
ORDER BY "Notification Time (UTC)" ASC
```

There is a column "Status" in the database. Observed values are NEW, UPDATED, DISMISSED. I have not had time to test to see how accurate they are.

USERDATA/data/com.garmin.android.apps.connectmobile/databases/gcm_cache.db Tables of interest:

- devices
    - ID Number
    - Product Number
    - Bluetooth MAC address
    - Display & Friendly Names (latter is likely what is seen by user via UI
    - Connection Type
    - Last connection time (UNIX Epoch - will be NULL if never paired with phone, but associated with the Garmin account
    - Software Version
    - URL for stock photo (could help with identifying the actual device)
    - SKU
    - Part number

- json

- The data in this table is, as the name suggests, in json
- Column "concept_name" provides insight into what the json data represents in the associated column ("cached_val")
- Data is updated periodically (last updated timestamp in "saved_timestamp" column)
- NOTE: The data in this table is based upon the capabilities of the wearable. For example, if the wearable is not capable of collecting 24/7 heart rate data, the entry "REAL_TIME_HEART_RATE" may be empty or non-existent.
  - USER_SETTINGS
    - Garmin Connect ID (useful for legal processes)
    - Provided date of birth
    - Gender
    - Handedness (indicates the setting for which hand the wearable is worn)
    - Heart rate format (bpm)
    - Height (in meters - provided by user)
    - Time format (12 or 24 hour)
    - Weight (in grams)
    - Sleep time (in seconds from midnight - wake time and bedtime)

- json_activites
  - Data stored in this table is also in json
  - Tracks activities
  - Activity metrics within the JSON is stored in plain text
  - Includes the following if applicable to the activity AND if the associated wearable is able to capture:
    - Speed
    - Location
    - Elevation
    - Heart Rate
    - Cadence
    - Start/Stop location
    - Nearest Surface Weather Observation Station (searchable on the FAA website)
- cache-database
  - This database appears to temporarily hold data. I have multiple versions of this database, and some have lots of data, others not so much. When it is populated here’s what's is available:
    - sleep_deta
      - Sleep Start/Stop
      - Total time in Seconds
      - Autosleep Start/Stop
        - Not sure how this differs from regular stop/start. Garmin automatically detects sleep - a user doesn't need to initiate it
      - Time spent in sleep stages (in seconds)
        - Deep
        - Light

    - REM
    - Awake
- SpO2 values during sleep
- Respiration

More info can be found in Josh Hickman's blog: https://thebinaryhick.blog/2021/05/22/the-state-of-android- health-data-part-1-garmin/ . A special thanks to Josh for all the research he did on Android Health artifacts.

- Make sure you validate this!

- For this database to appear, the user must be logged into the app
- The username will be appended to the end of the database (i.e., fitness.db.585forensics_gmail.com)

Google Fit was paired with WearOS wearable. The phone application is not a default application, and it must be downloaded from the Play Store. The findings show that the app will automatically log activity without user intervention. For example, Josh walked from his parked car to the front door of the local Target, and his activity was logged as a walk; He was parked ~50 yards from the storefront. Location data may or may not be associated with the activity and appeared to be sporadic. Josh also had instances where his walking was automatically logged just walking around inside of a store if he walked long enough.

Since location is a "Dangerous" Permission, access has to be granted both on the phone app and on a paired WearOS wearable. The data resides in two locations:

USERDATA/data/com.google.android.gms/databases/fitness.db.[GMAIL ACCOUNT NAME]

- This is not automatically present. The user has to sign into the app in order for this to appear on the device.
- For Josh, the file name is **fitness.db.thisisdfir_gmail.com**.
- The ability to add accounts in Google Fit exists, so it is possible more than one database could reside in this location.
- This database contains a lot of the same data seen in the others from **com.google.android.apps.fitness**.
- Entries in this database appear to correspond to the Google Fit Takeout data for the account.
- This file also exists on any WearOS device that is paired to the phone, using the same Google account as the phone, and is using Google Fit. File name and location within WearOS is the same.
- There may be data in this table that was generated elsewhere (another device on which the user has signed into the app).

Table:
Sessions

Activity codes:
7 = Walk
8 = Run
72 = Sleep
NOTE: These are the codes that have been tested. Others may exist.

Query:
 SELECT

```
datetime(Sessions.start_time/1000,'unixepoch') AS "Activity Start Time
(UTC)", datetime(Sessions.end_time/1000,'unixepoch') AS "Activity
Stop Time (UTC)", Sessions.app_package AS "Contributing App",
CASE
WHEN
Sessions.activity=7
THEN "Walking"
WHEN
Sessions.activity=8
THEN "Running"
WHEN
Sessions.activity=72
THEN "Sleeping" END
AS "Activity Type",
Sessions.name AS
"Activity Name",
Sessions.description AS
"Activity Description"
FROM
Sessions
ORDER BY "Activity Start Time (UTC)" ASC
```

USERDATA/data/com.google.android.apps.fitness

- Most of the interesting data is located in ~/files/accounts/%ACCOUNT_NUMBER%/
    - Account numbers start with "1"
- Folder contains a mix of protobuf and database files
- Files of interest:
    - session_database.db
    - Table: session_entries
- Column **id** can contain the start/end times of the activity or the source. In cases of the source, the format is as follows:
- IDENTIFIER:activemode:[ACTIVITY]:Start time (UNIX EPOCH)
- IDENTIFIER:watch-activemode:[ACTIVITY]:Start Time (UNIX EPOCH) (for activity tracked by WearOS wearable)
- Usually, the start time value is the same as that seen in the column "start_time_ms"
- Columns "start_time_ms" and end_time_ms are start/end times in Unix Epoch
- Column **activity_type** contains numerical code for activity:
- Type 8 = Run
- Type 7 = Walk
- Type 72 = Sleep (Sleep)
- **Josh has seen other activity_type values, but these three seem to be consistent.
    - journal_database.db

Table: journal_entries **-** populates the Journal tab in the app UI

Column “id” contains the source of the entry. Entries with “header” mark the beginning of the next day. I had to use two bridging apps: FitToFit (to import Fitbit data) and Health Sync (to import Garmin data).

For WearOS devices, the entries are different. For example:

IDENTIFIER:watch-

activemode:running:[UNIX TIMESTAMP]

Entries for phone-only tracked activities are

different. For example:

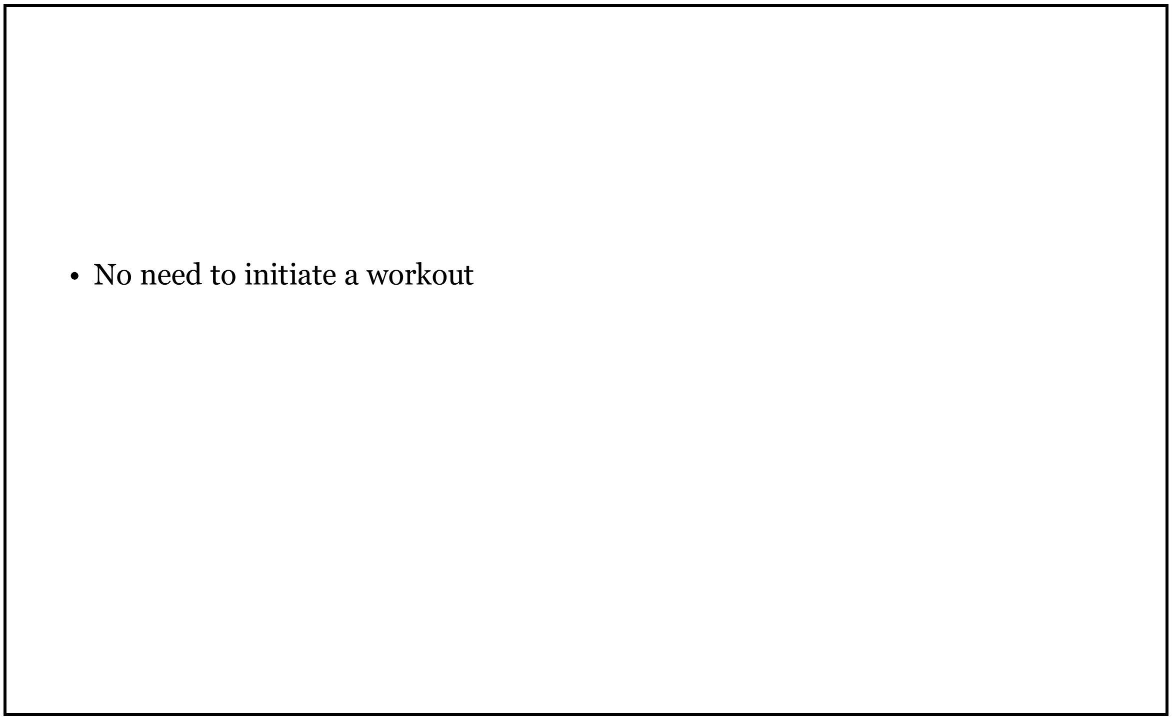
IDENTIFIER:activemode:walking:[UNIX

TIMESTAMP]

The timestamps appear to be the starting time of the activity. The "watch" portion (probably the "WearOS device used" portion) is absent.

Column “journalEntry” contains protobuf. The protobuf describes the entry. Contained within is the title of the journal entry (e.g., “Morning Run”), start and end times, activity type code (see codes from **session_database.db**), length of activity (in milliseconds), source app package (e.g., com.google.android.apps.fitness).

For the bridging apps (FitToFit for Fitbit and Health Sync for Garmin), the blobs for those entries contain the actual source of the activity (Fitbit or Garmin). Again, thank you Josh Hickman for your level of effort on Android Health research.



- No need to initiate a workout

FitBit activity is autodetected. For example, a walk will be logged if the tracker detects it without the user manually initiating an activity. This is also true for sleep.

At the time of writing, ALEAPP is able to parse Fitbit data and AXIOM parses some data, but not as much as ALEAPP. FitBit is also collecting location data in background when the app is not in use (see below). The data is stored in a several database, which are referenced below.

USERDATA/data/com.Fitbit.FitbitMobile/files/User/%ENCODED_USER_ID%/companion_data/platform.db Table of Interest:
console_log

NOTE: The column "message" contain various app messages. Searching the column contents for the word "location" can retrieve URLs that were fetched for weather conditions at the phone's location. The URL contains the lat/long of the phone. There are associated timestamps with the entries. Josh located about 48 hours' worth of log entries like this.

USERDATA/data/com.Fitbit.FitbitMobile/databases/activity_db - contains a summary of logged activity Table of Interest:
ACTIVITY_LOG_ENTRY

Query:

```
SELECT
datetime(ACTIVITY_LOG_ENTRY.LOG_DATE/1000,'unixepoch') AS "Time
of Activity (UTC)", ACTIVITY_LOG_ENTRY.NAME AS "Activity Type",
ACTIVITY_LOG_ENTRY.LOG_TYPE AS "Method of Tracking",
ACTIVITY_LOG_ENTRY.SOURCE_NAME AS "Source of Tracking",
ACTIVITY_LOG_ENTRY.ACTIVE_DURATION AS "Activity Duration
(Seconds)", ACTIVITY_LOG_ENTRY.DISTANCE AS "Distance",
ACTIVITY_LOG_ENTRY.DISTANCE_UNIT AS "Distance Unit of
Measurement", ACTIVITY_LOG_ENTRY.STEPS AS "Steps",
ACTIVITY_LOG_ENTRY.SPEED AS
"Speed (Measurement/hr)", FROM
ACTIVITY_LOG_ENTRY
```

USERDATA/data/com.Fitbit.FitbitMobile/databases/device_database - lists information about devices associated with account
Table of Interest:
core_device

Query:

```
SELECT
core_device.deviceName AS “Device Name”,
core_device.bleMacAddress AS “Bluetooth
MAC Address”, core_devce.deviceType AS
“Device Type”,
datetime(core_device.lasSyncTime/1000, ‘unixepoch’) AS “Device Last Sync
(UTC)”, core_device.batteryPercent AS “Battery Percent At Last Sync”
FROM
core_device
```

USERDATA/data/com.Fitbit.FitbitMobile/databases/exercise_db - stores information about exercises performed. Also contains data that has been sync’d with Fitbit

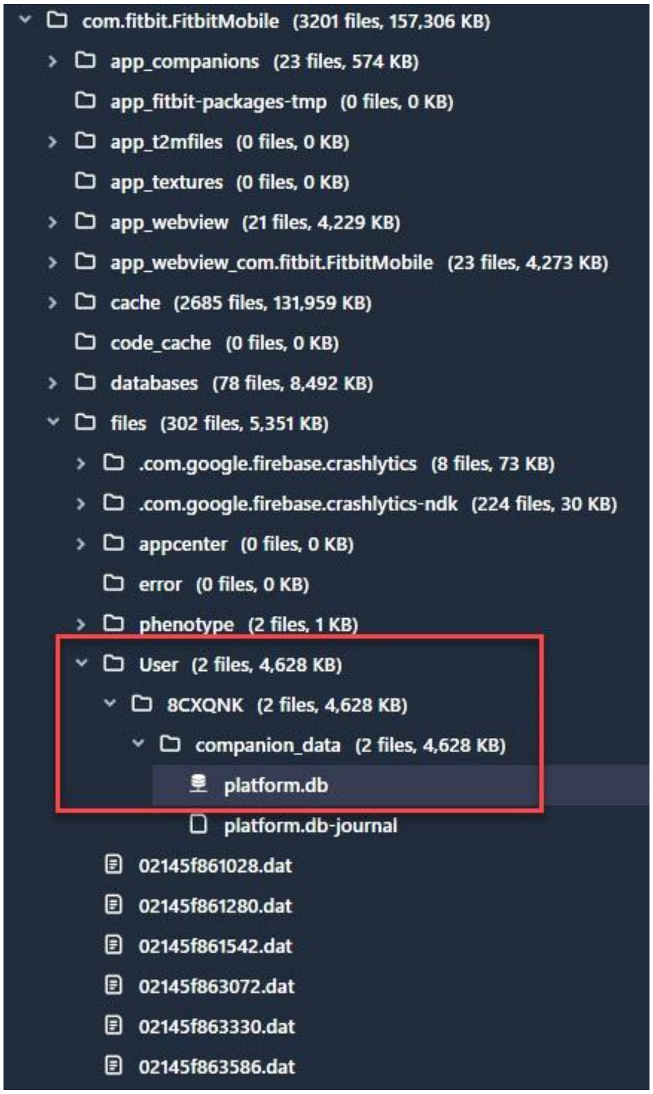
Tables of Interest:
EXERCISE_S
ESSION
EXERCISE_E
VENT

Queries:

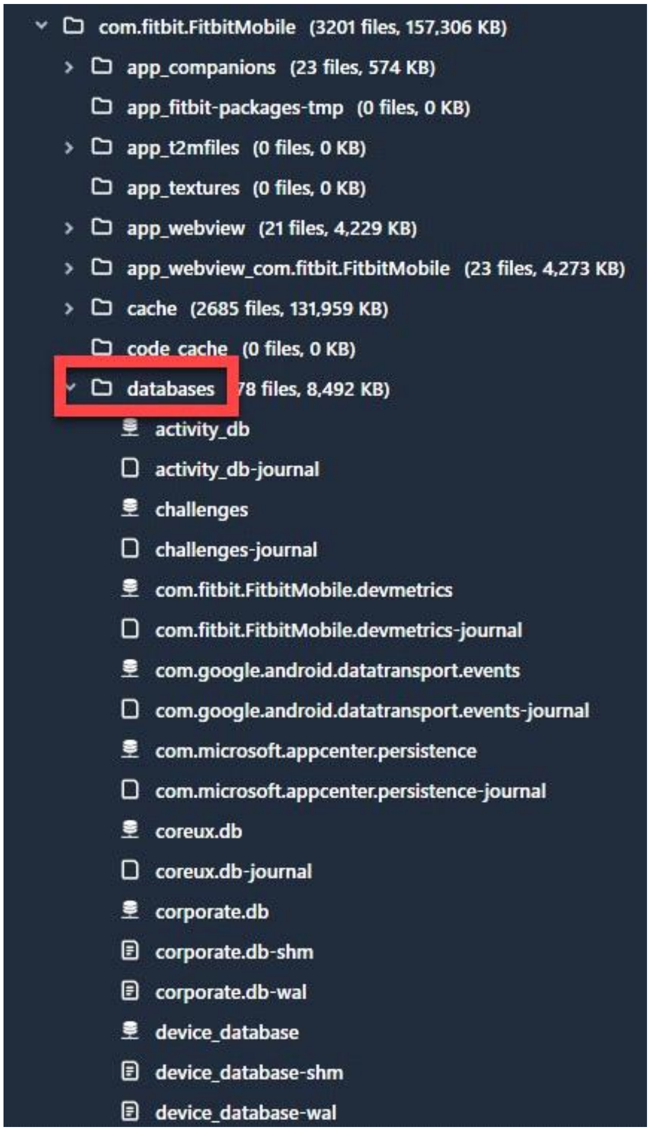
```
SELECT
datetime(EXERCISE_SESSION.START_TIME/1000,’unixepoch’) AS
“Exercise Start Time (UTC)”,
datetime(EXERCISE_SESSION.STOP_TIME/1000,’unixepoch’) AS
“Exercise Stop Time (UTC)”, CASE
WHEN
EXERCISE_SESSION.ACTIVITY_TYPE=90009
THEN ‘Running’ WHEN
EXERCISE_SESSION.ACTIVITY_TYPE=90013
THEN ‘Walking’
EN
D
AS
“Ac
tivit
y
Typ
e”
FR
OM
EX
ER
CIS
E_S
ESS
ION
```

```
SELECT
```

```
Datetime(EXERCISE_EVENT.TIME/1000,’unixepoch’) AS “Time (UTC)”,
EXERCISE_EVENT.SESSION_ID AS “Session ID”,
CASE
WHEN
EXERCISE_SESSION.ACTIVITY_TYPE=90009
THEN ‘Running’ WHEN
EXERCISE_SESSION.ACTIVITY_TYPE=90013
THEN ‘Walking’
END AS
“Activity
Type”,
EXERCISE_E
VENT.LATIT
UDE,
EXERCISE_E
VENT.LONG
ITUDE,
EXERCISE_EVENT.ALTITUDE AS
“Altitude (meters)”,
EXERCISE_EVENT.SPEED AS “Speed
(km/hr)”, EXERCISE_EVENT.ACCURACY
F
R
O
M

E
X
E
R
C
I
S
E
_
E
V
E
N
T
JOIN EXERCISE.SESSION ON EXERCISE_SESSION.UUID=EXERCISE_EVENT.SESSION_ID
```

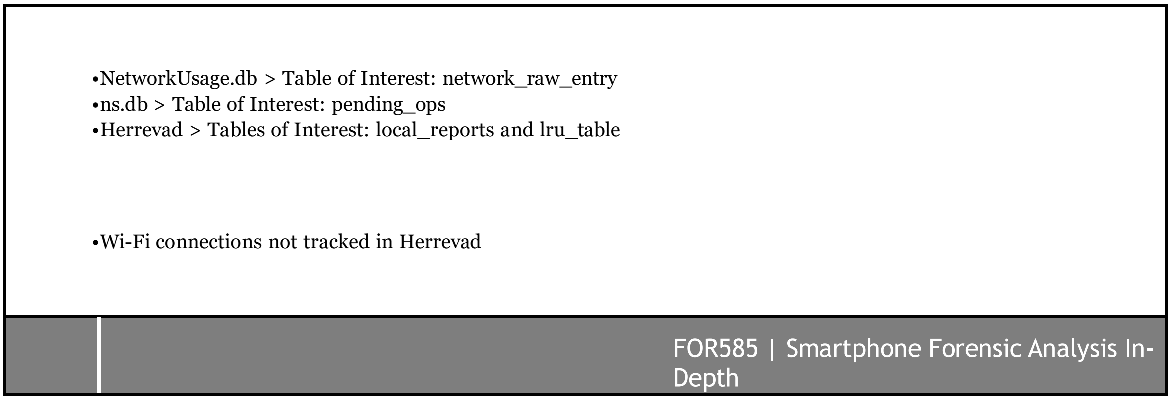


```
com.fitbit.FitbitMobile  (3201 files, 157,306 KB)
  app_companions  (23 files, 574 KB)
  app_fitbit-packages-tmp  (0 files, 0 KB)
  app_t2mfiles  (0 files, 0 KB)
  app_textures  (0 files, 0 KB)
  app_webview  (21 files, 4,229 KB)
  app_webview_com.fitbit.FitbitMobile  (23 files, 4,273 KB)
  cache  (2685 files, 131,959 KB)
  code_cache  (0 files, 0 KB)
  databases  (78 files, 8,492 KB)
  files  (302 files, 5,351 KB)
    .com.google.firebase.crashlytics  (8 files, 73 KB)
    .com.google.firebase.crashlytics-ndk  (224 files, 30 KB)
    appcenter  (0 files, 0 KB)
    error  (0 files, 0 KB)
    phenotype  (2 files, 1 KB)
    User  (2 files, 4,628 KB)
      8CXQNK  (2 files, 4,628 KB)
        companion_data  (2 files, 4,628 KB)
          platform.db
          platform.db-journal
    02145f861028.dat
    02145f861280.dat
    02145f861542.dat
    02145f863072.dat
    02145f863330.dat
    02145f863586.dat
```



```
com.fitbit.FitbitMobile  (3201 files, 157,306 KB)
  app_companions  (23 files, 574 KB)
  app_fitbit-packages-tmp  (0 files, 0 KB)
  app_t2mfiles  (0 files, 0 KB)
  app_textures  (0 files, 0 KB)
  app_webview  (21 files, 4,229 KB)
  app_webview_com.fitbit.FitbitMobile  (23 files, 4,273 KB)
  cache  (2685 files, 131,959 KB)
  code_cache  (0 files, 0 KB)
  databases  (78 files, 8,492 KB)
    activity_db
    activity_db-journal
    challenges
    challenges-journal
    com.fitbit.FitbitMobile.devmetrics
    com.fitbit.FitbitMobile.devmetrics-journal
    com.google.android.datatransport.events
    com.google.android.datatransport.events-journal
    com.microsoft.appcenter.persistence
    com.microsoft.appcenter.persistence-journal
    coreux.db
    coreux.db-journal
    corporate.db
    corporate.db-shm
    corporate.db-wal
    device_database
    device_database-shm
    device_database-wal
```

(NOTE: This query gives all exercise events, along with their location at any given point in time along with the activity type)



- NetworkUsage.db > Table of Interest: network_raw_entry
- ns.db > Table of Interest: pending_ops
- Herrevad > Tables of Interest: local_reports and lru_table

- Wi-Fi connections not tracked in Herrevad

FOR585 | Smartphone Forensic Analysis In-Depth

When trying to determine if the device was in use as part of our investigation, we can simply look at overall datetime stamps, look for application usage, or think outside the box. Here, that is what we are doing. We are trying to see that the SIM card was leveraged and, even better, which number was in use at that time.
Additionally, we can track times associated to SIM card check-ins, which essentially ensures the SIM is still active and usable. If the phone is rebooted and the SIM is not in the device, you will notice the occurrence in check.xml. If you cannot location simcard.dat on the device, make sure to look for the following:

/data/data/com.google.android.gms/databases/MdpSimBasedDatabase
/data/data/com.google.android.gms/databases/constellation.db
Both files contain SIM card information, verified on Android 10. These files are likely to be found on Android 11 and 12.

In Android 11 and 12 replaces the file USERDATA/misc/wifi/softap.conf seen in Android 10 and below with
/USERDATA/apexdata/com.android.wifi/. This file is regular
XML (non-ABX) and contains BSSID and password for
phone tethering capabilities.

The databases stored in the com.google.android.gms/databases directory are used to track network activity to include cellular and Wi-Fi. It’s impressive what we can uncover if we are willing to dig!

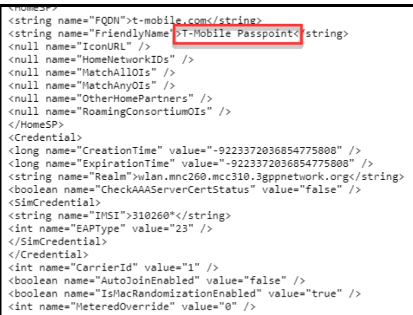
USERDATA/data/com.google.android.gms/databases/

- NetworkUsage.db > Table of Interest: network_raw_entry
- ns.db > Table of Interest: pending_ops
- Herrevad > Tables of Interest: local_reports and lru_table

The ns.db is located at USERDATA/data/com.google.android.gms/databases/ns.db and contains the table **pending_ops**, which tracks application names, the job or task associated with it, and the runtime dates. This can be used to prove the user had an application on a device.

- Tables of Interest: local_reports and lru_table

The database located at
USERDATA/data/com.google.android.gms/databases/herrevad is incredible with what it tracks. The two tables of interest are local_reports and lru_table. Here, the examiner can determine when
Wi-Fi was being used versus Cellular. So many useful artifacts can be extracted from this database. NOTE that this file is only available on Android 9 and earlier devices.



```
<HomeSP>
<string name="FQDN">t-mobile.com</string>
<string name="FriendlyName">T-Mobile Passpoint</string>
<null name="IconURL" />
<null name="HomeNetworkIDs" />
<null name="MatchAllOIs" />
<null name="MatchAnyOIs" />
<null name="OtherHomePartners" />
<null name="RoamingConsortiumOIs" />
</HomeSP>
<Credential>
<long name="CreationTime" value="-9223372036854775808" />
<long name="ExpirationTime" value="-9223372036854775808" />
<string name="Realm">wlan.mnc260.mcc310.3gppnetwork.org</string>
<boolean name="CheckAAAServerCertStatus" value="false" />
<SimCredential>
<string name="IMSI">310260*</string>
<int name="EAPType" value="23" />
</SimCredential>
</Credential>
<int name="CarrierId" value="1" />
<boolean name="AutoJoinEnabled" value="false" />
<boolean name="IsMacRandomizationEnabled" value="true" />
<int name="MeteredOverride" value="0" />
```

WifiConfigStore.xml moved locations in in Android 11 and 12. The paths are listed below.

/USERDATA/misc/wifi - Android 10 and below
/USERDATA/misc/apexdata/com.android.wifi/ - Android 11 and 12
Both files are stored in regular XML (non-ABX). For Android 11 and 12, it still contains Wi-Fi information (BSSID and password). It contains the randomized MAC address, which is seen by Wi-Fi access points. This file may have a timestamp of last connection (NOTE: could be used to put device in a location at a specific time - seen in Samsung A30 running Android 11…this is NOT seen in the Pixel 3 on Android 11 or 12). For Android 12 it contains actual Wi-Fi MAC address.

Android’s version of AirDrop

Nearby Share is Android's version of AirDrop on iOS. This file is stored at /USERDATA/data/com.google.android.gms/shared_prefs/nearbysharing/service/state.xml

- Phone Name
- Unique Identifier
- Google account
- Status (enabled true/false)

NOTE: Outside of logcat, it has been difficult to find any stored information about file transfers. Logcat retention times range anywhere from a few minutes to about 24 hours depending on the phone. Sometimes, you may fine you can't even find the information in logcat if the logs roll over quick enough.

- See https://thebinaryhick.blog/2020/08/22/nearby-share-airdrop-for-android-return-of-the-unsolicited- richard-photograph/

This page intentionally left blank.

Google provides 15 GB of free storage space to users. Keep in mind this is available to even non-Android device users. Thus, Google cloud artifacts are likely to be relevant to most mobile investigations. Make sure you have proper authorization to access cloud data. If you overstep boundaries, you risk having your case thrown out and even ruining your reputation or your career. In order to use these tools, you must have the user's Gmail address and password.

There are several tools available to pull cloud data. We are going to highlight Elcomsoft Cloud eXplorer in the next few slides. You have been provided a license for accessing cloud data. Try it on your own data or try it in a lab this week, assuming you have permission to do so. Keep in mind the user will be alerted if 2FA (two factor authentication) or two-step verification is set. If you are conducting covert operations, do not attempt to do this if you cannot risk the user being alerted!

Once you enter the proper credentials, you will be provided with a screen of what is available for extraction. NOTE, you may have to enter another secure code depending on how the device is set up. We recommend grabbing everything at this point if you are allowed to do so. The amount of information stored here is incredible! Just wait until you pull your first extraction. Locations are incredible if they are accurate. This is something you need to carefully examine for relevance to your investigation. We may have to get crafty to put the user at a location, which we have seen earlier in this section. For example, Wi-Fi artifacts—if the user cleared their network settings on the device, this may enable you to place the device at a location when "something" occurred.

Expect to get TONS of data from Google cloud!

FOR585 | Smartphone Forensic Analysis In-Depth

Here is an example of what Chrome stores for us. In addition to the normal browser history, Chrome keeps track of all the passwords you ask it to "save" for you. Above, we can see several of my usernames and passwords. To see the password, simply click on the eye! Nothing else is needed to view these. The times used shows how frequently I used Chrome to log into something without me having to enter my credentials manually. Scary, right? But amazing for the investigations we work! Remember, humans are creatures of habit. We love to reuse passwords. Keep a running list of passwords for the case you are working, as you never know when they will enable access to other data of interest.

- The user will receive a notification stating that a new device signed into their Google account

  ** This is not recommended if you are conducting covert operations, as you have to assume the user will know you were there!
- If third-party applications are extracted, an SMS for each may be sent to retrieve a code

We cannot stress enough that the security set on these devices and applications will often alert the user of someone or something touching their data or their device. Use caution when conducting covert missions, as you don't want to blow your cover by something so simple as an email alerting the user that their Gmail account was accessed. This message shown here was sent after I logged into my test phone using Elcomsoft. When cloud data was extracted using Oxygen with the user's phone number, an SMS was sent for every application being extracted. Additionally, the user will most likely be logged out of that application and will have to re-authenticate.

Something that was mentioned earlier in the slides and must be mentioned again—iOS, BlackBerry, Windows Phone and even a flip phone user can back up to Google. In this example, you are seeing my Google data. I am currently an iPhone user, so you will notice that my Android icon is grayed out. This is your first clue that this associated data is most likely NOT syncing from an Android. Many other tools will give you similar indicators. Here, we can see some more red flags that should alert us that the user is not backing up from an Android. Things like Calls, Messages, and Wi-Fi being empty could mean that the user doesn't want to back up that information, but that is something we expect to see in these backups. If you dig into User Info and history, you will most likely see an association to my iPhone.

- Always verify you have valid data after you complete an acquisition
- Expect encryption, but don’t let it stop you
  - Do another type of acquisition
- An advanced logical may be your best evidence
- Don’t forget cloud data—ask for it!
- Use more than one tool
- Understand that you MUST know older versions of Android because it takes more than 2.5 years (on average), for users to catch up to a relevant version

This page intentionally left blank.

- Will it reconnect to the network?
- Will you miss a required step?

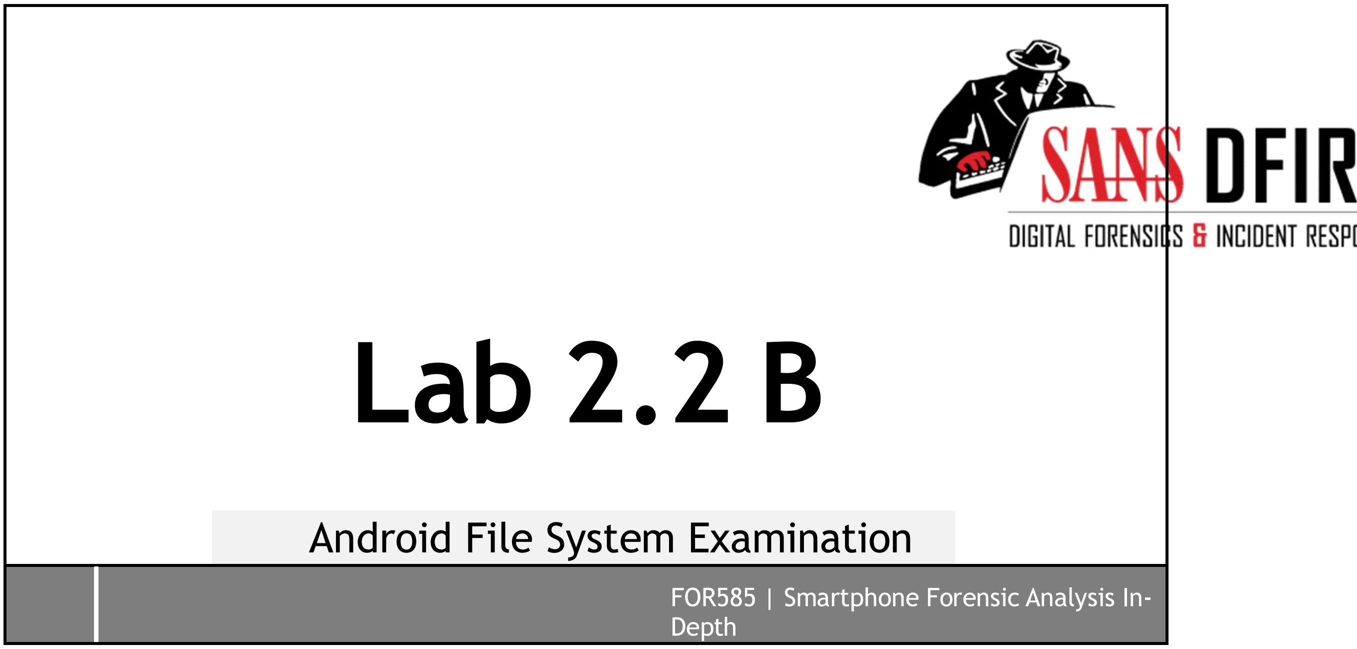
- Don’t rely on your tool to dump it

- Make sure you acquire this externally, if possible

For more details on Android acquisition and volatility, refer to Mattia’s presentation and blog:

https://www.sans.org/presentations/order-of-volatility-in-modern-smartphone-forensics/ https://blog.digital-forensics.it/2021/03/triaging-modern-android-devices-aka.html

We haven't discussed malware in detail yet, but we will in Section 4. This slide is simply a reminder so that when you go back to your lab and wonder how to locate malware on an Android, you can refer back to this! Consider everything we just discussed and apply it to malware. Malware will drain a battery. Malware may provide notifications. It may be downloaded from the browser or clicked on via a link. An android does NOT have to be rooted to get malware, but it's always more vulnerable than a fully patched and up-to-date Android device.



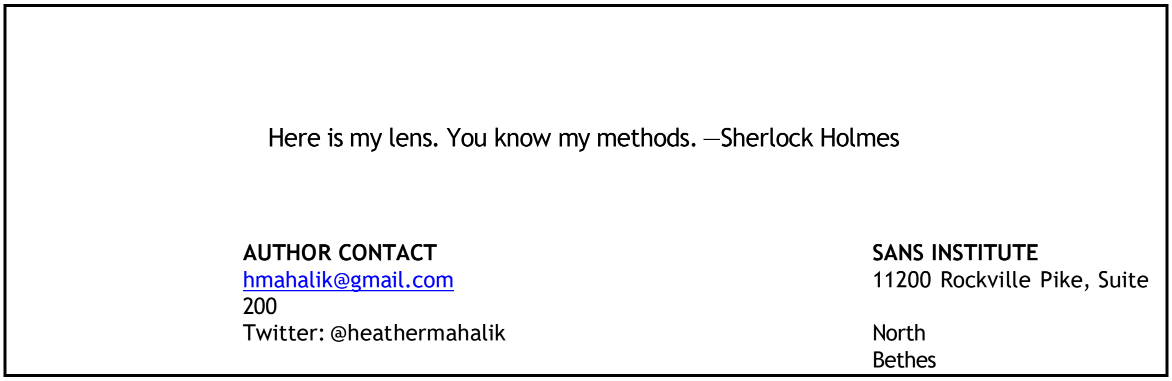
SANS DFIR
DIGITAL FORENSICS & INCIDENT RESPO

Lab 2.2 B

Android File System Examination

FOR585 | Smartphone Forensic Analysis In-Depth

This page intentionally left blank.



Here is my lens. You know my methods. —Sherlock Holmes

AUTHOR CONTACT
hmahalik@gmail.com
200
Twitter: @heathermahalik

SANS INSTITUTE
11200 Rockville Pike, Suite

North
Bethes